Automata and minimization[1]

Thomas Colcombet, IRIF, CNRS
Daniela Petrişan, IRIF, CNRS

Already in the seventies, strong results illustrating the intimate relationship between category theory and automata theory have been described and are still investigated. In this column, we provide a uniform presentation of the basic concepts that underlie minimization results in automata theory. We then use this knowledge for introducing a new model of automata that is an hybrid of deterministic finite automata and automata weighted over a field. These automata are very natural, and enjoy minimization result by design.

The presentation of this paper is indeed categorical in essence, but it assumes no prior knowledge from the reader. It is also non-conventional in that it is neither algebraic, nor co-algebraic oriented.

1. INTRODUCTION

In this column, we attempt to give a simple, user-friendly, description of how category theory sheds an interesting light on some aspects of automata theory and in particular concerning the existence of minimal recognizers.

Seen from distance, an automaton is a machine that

processes an input, respecting its *structure* (word, tree, infinite word or tree, trace, ...)	and	outputs a quantity in some *universe of output values* (Boolean values, probabilities, vector space, words, ...)

These two aspects, structure of the input and universe of outputs play an essentially orthogonal role, and provide a good organization scheme in the description of the landscape of automata. Though sometimes interacting, these two axes deserve to be understood in an independent way. An important unification step for understanding the structure of the input axis has been described by Bojańczyk [Bojańczyk 2015] thanks to the use of 'monads' from category theory. Our focus is on the universe of output values axis.

In this paper, we focus our attention to the universe of outputs, and the related notion of state space. Here, once more, category theory offers a neat understanding of phenomena. We assume no knowledge of category theory from the reader.

THEOREM 1.1. *Given an automaton $\mathcal{A}$ in a class C, there exists another automaton $\mathcal{M}$ in C which is algebraically minimal*[2] *while having the same semantics.*

[1]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 developed.

[2]It is a common approximation to say that an automaton is minimal if its number of states is minimal. We emphasize that this is not the notion we consider by using the terminology algebraically minimal.

This theorem is very well known for deterministic (and complete) automata as presented in the seminal work of Rabin and Scott [Rabin and Scott 1959]. It also known from Schützenberger's work on automata weighted over a field [Schützenberger 1961] (that we will present in more detail). Other similar results involve automata over trees [Brainerd 1968], deterministic transducers [Choffrut 1979; Choffrut 2003], syntactic monoids [Schützenberger 1965], syntactic $\omega/\diamond/\circ$-semigroups [Perrin and Pin 1995; Bedon 1996; Bedon et al. 2010; Carton et al. 2011; Colcombet and A. V. 2015], stabilisation monoids [Colcombet 2009; Kuperberg 2011; Colcombet 2013], syntactic semirings for languages [Polák 2001], syntactic forest algebras [Bojańczyk and Walukiewicz 2008], syntactic nominal monoids [Bojańczyk 2011], and so on. However, it also fails for many other classes, starting with non-deterministic automata or deterministic automata over infinite inputs.

A goal of this paper is to give a neat description of what this approach means at an abstract level, and why it sometimes works, and sometimes not. This will be also the occasion to describe new forms of automata, namely hybrid-set-vector automata that were not known in the literature, that extend both deterministic finite automata and automata weighted over a field, while preserving the property of admitting algebraically minimal automata (the computation of which being still open). This is an example of the process of using a category-theoretic approach for defining new devices (here automata) that enjoy strong properties by design.

Structure of the column

In Section 2, we present the very natural notion of an automaton in a category, the examples of deterministic automata and vector space automata, as well as the minimalistic concepts of category theory that are required for this definition to make sense. In Section 3 we explain the concepts that are behind results of minimization, and in particular the one of factorization. In Section 4 we present the hybrid-set-vector automata. In Section 5 we discuss the connection between automata and category theory and some of the literature on this topic.

2. AUTOMATA IN A CATEGORY AND THEIR SEMANTICS

In this section, after studying classical examples, we introduce the notion of an automaton in a category. This presentation does not contain any new material, but departs from the literature in that it doesn't adopt the algebraic or the coalgebraic presentation of these objects. We hope that the resulting presentation is simpler, requires less background, and more faithfully follows the spirit of standard automata theory.

Before pursuing this description, we need to understand what is the semantics of an automaton. Let us start with these two examples:

DETERMINISTIC AUTOMATA

A *deterministic automaton* (finite or infinite), or simply a $\mathtt{Set}$-*automaton* is a tuple

$$\mathcal{A} = \langle Q, A, i, f, \delta \rangle$$

in which Q is a *set of states*, A is the *input alphabet*, $i \colon 1 \to Q$ is the *initial map* (where 1 is some one element set, let us say $\{0\}$), $f \colon Q \to 2$ is the *final map* (where 2 is some two element set, let us say $\{0,1\}$), and $\delta_a \colon Q \to Q$ is the *transition map for the letter* a for all $a \in A$.

VECTOR SPACE AUTOMATA

Let us consider vector spaces over a base field $\mathbb{K}$. A *vector space automaton*, or simply a $\mathtt{Vec}$-*automaton*, is a tuple

$$\mathcal{A} = \langle Q, A, i, f, \delta \rangle$$

in which Q is a *vector space of configurations* , A is the *input alphabet*, $i \colon \mathbb{K} \to Q$ is the *initial map*, $f \colon Q \to \mathbb{K}$ is the *final map*, and $\delta_a \colon Q \to Q$ is the linear *transition map for letter* $a \in A$.

Given a word $u = a_1 \dots a_n \in A^*$, the Set-automaton $\mathcal{A}$ *accepts* the map

$$[\![\mathcal{A}]\!](u) = f \circ \delta_{a_n} \circ \cdots \circ \delta_{a_1} \circ i \ .$$

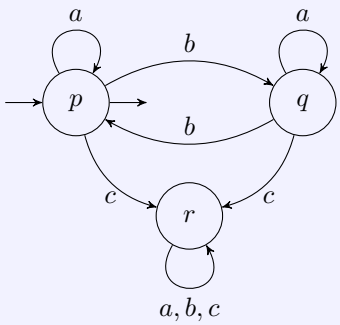
Since a map from 1 to 2 can take only two possible values: the constant 0 and the constant 1, $[\![\mathcal{A}]\!]$ can be understood as associating to each input word u either 1 (and we say that *the word is accepted*), or 0 (and we say that *the word is rejected*). Hence, it *computes a language*. We recognize here, in a barely disguised wording, the standard definition of a DFA.

Example 2.1. Consider the language $L_{\rm set}$ over the finite alphabet $A = \{a, b, c\}$ defined by:

$$L_{\rm set} = \{u \in A^* \mid |u|_b \text{ is even and } |u|_c = 0\} \ .$$

An automaton $\mathcal{A}_{\rm set}$ for this L is as follows:

— the set of states is $Q = \{p, q, r\}$,
— the initial map selects $p \in Q$,
— the final map maps p to 1, and q, r to 0.
— the transition maps are described below in the standard way



Given a word $u = a_1 \dots a_n \in A^*$, the Vec-automaton $\mathcal{A}$ *accepts* the linear map

$$[\![\mathcal{A}]\!](u) = f \circ \delta_{a_n} \circ \cdots \circ \delta_{a_1} \circ i \ .$$

Since a linear map from $\mathbb{K}$ to $\mathbb{K}$ is of the form $x \mapsto ax$ for some $a \in \mathbb{K}$, $[\![\mathcal{A}]\!]$ can be understood as associating to each input word u a scalar $a \in \mathbb{K}$. This is a variation around the idea of an *automaton weighted over a field* of Schützenberger [Schützenberger 1961][3].

Example 2.1. Consider the following map $L_{\rm vec}$ which to a word u associates the linear map $L_{\rm vec}(u)\colon \mathbb{R} \to \mathbb{R}$ defined by:

$$L_{\rm vec}(u)(x) = \begin{cases} 2^{|u|_a} x & \text{if } |u|_b \text{ is even} \\ & \text{and } |u|_c = 0, \\ 0 & \text{otherwise} \end{cases}$$

An automaton $\mathcal{A}_{\rm vec}$ for this L is as follows:

— the vector space of configurations is $\mathbb{R}^2$,
— the initial map maps x to $(x, 0)$,
— the final map maps (x, y) to x,
— the transition map for a maps (x, y) to $(2x, 2y)$,
— the transition map for b maps (x, y) to (y, x),
— the transition map for c maps (x, y) to $(0, 0)$.

It is easy to check that this vector space automaton x accept $L_{\rm vec}$.

[3]The two differences being that in our case, (a) the vector spaces can be of infinite dimension, and (b) there is no need for choosing a basis for Q, as it is done in the original definition.

Inspecting the two above definitions of automata, it is obvious that these can be unified. Category theory is certainly the proper language for such a unification. Let us give as a starter some very elementary definitions concerning categories.

WHAT IS A CATEGORY?

A *category* has essentially two parts:

(objects, arrows) ,

where the *objects* are denoted $X, Y, \ldots$, and each *arrow*, denoted $f\colon X \to Y$, goes from a *source object* X to a *target object* Y.

Typical categories are:

`Set` = (sets, functions between sets),	`Vec` = (vector spaces, linear maps),
`Pos` = (ordered sets, order preserving maps),	`Aff` = (affine spaces, affine maps),
`Rel` = (sets, relations between sets),	`Grp` = (groups, group morphisms),
`Top` = (topological spaces, continuous maps),	...

Furthermore, properly defined categories have to contain the following extra pieces of information:

(a) for all objects X, there is an arrow $\mathrm{Id}_X\colon X \to X$ called the *identity of* X, and
(b) given arrows $f\colon X \to Y$, $g\colon Y \to Z$, there exists a *composite arrow* $g \circ f\colon X \to Z$.

The complete the definition, the composition of arrows has to be associative, and the identity has to act as a neutral element for it (on the left and on the right).

Finally, define an arrow $f\colon X \to Y$ to be an *isomorphism* if there exists an arrow $g\colon Y \to X$ such that $g \circ f = \mathrm{Id}_X$ and $f \circ g = \mathrm{Id}_Y$. If such an isomorphism exists, then X and Y are *isomorphic*.

All these properties are obvious in the above examples, with the natural notion of identity arrow and composition of arrows. In what follows the categories that we used are essentially `Set` and `Vec`, in which all the categorical notions that we are interested in have a natural meaning.

We are now ready to describe what is a (word) language in a category and an automaton in a category.

Definition 2.2. Let us fix an alphabet A, a category $\mathcal{C}$, and two of its objects I and F. A $(\mathcal{C}, I, F)$-*language* L is a map that associates to each word $u \in A^*$ an arrow $L(u)\colon I \to F$ in $\mathcal{C}$.

A $(\mathcal{C}, I, F)$-*automaton* is a tuple:

$$\mathcal{A} = \langle Q, A, i, f, \delta \rangle$$

in which Q is an object from $\mathcal{C}$ called the *state object*, A is the *input alphabet*, $i\colon I \to Q$ is an arrow of $\mathcal{C}$ called the *initial arrow* $f\colon Q \to F$ is an arrow of $\mathcal{C}$ called the *final arrow* and $\delta(a)\colon Q \to Q$ is an arrow of $\mathcal{C}$ for all letters $a \in A$, called the *transition arrows*.

Given a word $u = a_1 \ldots a_n \in A^*$, a $(\mathcal{C}, I, F)$-automaton $\mathcal{A}$ *recognizes* the $(\mathcal{C}, I, F)$-language $[\![\mathcal{A}]\!]$ defined for all $u \in A^*$ by:

$$[\![\mathcal{A}]\!](u) = f \circ \underbrace{\delta(a_n) \circ \cdots \circ \delta(a_1)}_{\delta^*(u)} \circ i \ .$$

Given a $(\mathcal{C}, I, F)$-language L, an *automaton for* L is a $(\mathcal{C}, I, F)$-automaton that recognizes L.

DETERMINISTIC AUTOMATA

A deterministic automaton is nothing but a $(\mathtt{Set}, 1, 2)$-automaton.

VECTOR SPACE AUTOMATA

A vector space automaton is nothing but a $(\mathtt{Vec}, \mathbb{K}, \mathbb{K})$-automaton.

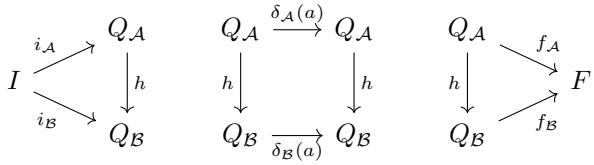
As usual when adopting a category theoretic approach, it is not sufficient to know what are the objects we are interested in, but we need also to know how these are

related through arrows. In our case, the arrows are the morphisms of automata that we introduce now.

Definition 2.3 (*category of automata for a language*). A *morphism of* $(\mathcal{C}, I, F)$*-automata* from $\mathcal{A} = \langle Q_{\mathcal{A}}, A, i_{\mathcal{A}}, f_{\mathcal{A}}, \delta_{\mathcal{A}} \rangle$ to $\mathcal{B} = \langle Q_{\mathcal{B}}, A, i_{\mathcal{B}}, f_{\mathcal{B}}, \delta_{\mathcal{B}} \rangle$, is an arrow of $\mathcal{C}$ $h \colon Q_{\mathcal{A}} \to Q_{\mathcal{B}}$ such that for all letters $a \in A$,

$$h \circ i_{\mathcal{A}} = i_{\mathcal{B}}, \qquad h \circ \delta_{\mathcal{A}}(a) = \delta_{\mathcal{B}}(a) \circ h, \qquad \text{and} \quad f_{\mathcal{A}} = f_{\mathcal{B}} \circ h,$$

or said differently, such that the following three diagrams commute:



(1)

Given a $(\mathcal{C}, I, F)$-language L, define the *category of automata for* L to be the category which has as objects the $(\mathcal{C}, I, F)$-automata that recognize L, and as arrows the morphisms of automata. We denote it by:

$$\mathtt{Auto}_L \ .$$

Note that whenever there is a morphisms of automata between two automata, then both have to recognize the same language.

3. ALGEBRAIC MINIMIZATION OF AUTOMATA, AND FACTORIZATIONS IN A CATEGORY

In this section, we explain some features of the category of automata for a language that make minimization of automata possible. There are essentially three required properties: (1) the existence of an initial automaton for the language, and (2) symmetrically, the existence of a final automaton for the language, and (3) the fact that the category of automata for the language has a factorization system. We begin our description with this last point.

3.1. Divisibility and factorization

The standard definition is that a deterministic automaton $\mathcal{M}$ is said *algebraically minimal* if for all other deterministic automata $\mathcal{A}$ for the same language, $\mathcal{M}$ divides $\mathcal{A}$ with the definition:

$$\text{“}\mathcal{B} \textit{ divides } \mathcal{A} \qquad \text{if} \qquad \mathcal{B} \text{ is the quotient of a subautomaton of } \mathcal{A} \text{ .”}$$

Hence we need to understand what is a quotient and what is a subautomaton. Both notions are related to the one of morphisms of automata: indeed, a *quotient* is the image of the automaton under a ‘surjective morphism’, and a *subautomaton* is an automaton which is sent into the other one under an ‘injective morphism’.

The notion of ‘surjectivity’ and ‘injectivity’ is the subject of the notion of factorizations in a category that we recall here.[3] An accessible and comprehensive reference for all matters concerning factorization systems is [Adámek et al. 1990].

[3]Usually, an emphasis is put on the fact that quotients correspond to ‘regular epis’, and subobjects to ‘monomorphism’. We try to avoid these case specific considerations, and concentrate here on the properties that arrows should have to be considered as ‘surjective like’ and ‘injective like’: namely that the two classes form a factorization system.

Definition 3.1. For two classes of arrows $\mathcal{E}$ and $\mathcal{M}$, we say that $(\mathcal{E}, \mathcal{M})$ forms a *factorization system* if the following conditions hold, where we denote the arrows in $\mathcal{E}$ by two-headed arrows $\twoheadrightarrow$ and the arrows in $\mathcal{M}$ by $\rightarrowtail$.

— The arrows that are both in $\mathcal{E}$ and $\mathcal{M}$ are exactly the isomorphisms.
— The $\mathcal{E}$-arrows are closed under composition.
— The $\mathcal{M}$-arrows are closed under composition.
— For every arrow $f\colon X \to Y$, there exists an $\mathcal{E}$-arrow $e\colon X \twoheadrightarrow Z$ and a $\mathcal{M}$-arrow $m\colon Z \rightarrowtail Y$ such that

$$f = m \circ e\ .$$

This composition is called the *factorization of* f. We also refer to the object Z as the *factorization of* f.
— For every arrow $e\colon X \twoheadrightarrow T$ in $\mathcal{E}$, $g\colon T \to Y$, $f\colon X \to S$ and $m\colon S \rightarrowtail Y$ in $\mathcal{M}$ such that $g \circ e = m \circ f$, there exists one and exactly one arrow $d\colon T \to S$ such that $d \circ e = f$ and $m \circ d = g$. In other words, if the following square commutes, then there exists a unique diagonal arrow such that the resulting diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{\ e\ } & T \\ {\scriptstyle f}\downarrow & \swarrow_{d} & \downarrow{\scriptstyle g} \\ S & \xrightarrow[\ m\ ]{} & Y \end{array} \tag{2}$$

This property is usually called the *diagonal property* and the unique morphism d is called a *diagonal fill*.

Note that the combination of the above properties make the factorization of an arrow unique up to isomorphisms: indeed, if $m \circ e = m' \circ e' = f$ are two factorizations of an arrow f through Z, respectively Z', then by the diagonal property there exist unique arrows $d\colon Z \to Z'$ and $d'\colon Z' \to Z$, so that $d \circ e = e'$, $d' \circ e' = e$, $m' \circ d = m$ and $m \circ d' = m'$. It readily follows that d and d' are isomorphisms inverse to each other. For example, both $d' \circ d$ and Id_Z are diagonal fills for the square

$$\begin{array}{ccc} X & \xrightarrow{\ e\ } & Z \\ {\scriptstyle e}\downarrow & \nearrow_{\mathrm{Id}_Z} & \downarrow{\scriptstyle m} \\ Z & \xrightarrow[\ m\ ]{} & Y\ , \end{array} \tag{3}$$

and hence, by uniqueness of the diagonal fill, we have $d' \circ d = \mathrm{Id}_Z$. This is a very standard argument in category theory.

Given a factorization system $(\mathcal{E}, \mathcal{M})$, an *$\mathcal{E}$-quotient* (or simply a *quotient* if $\mathcal{E}$ is clear from the context) of an object X is an arrow $e\colon X \twoheadrightarrow Y$ that belongs to $\mathcal{E}$. For ease of language, it also happens that Y itself is called a *quotient* of X. Similarly, an *$\mathcal{M}$-subobject* of an object X (or simply a *subobject* if $\mathcal{M}$ is clear from the context) is an arrow $m\colon Y \rightarrowtail X$ with m in $\mathcal{M}$.

There may be several pairs of classes of arrows $(\mathcal{E}, \mathcal{M})$ that yield a factorization system in the same category. For instance, in all categories, we can take $\mathcal{E}$ to be the isomorphisms and $\mathcal{M}$ to be all arrows (or the other way round), though this does not give us a very interesting notion... This is the reason why quotients and subobjects are notions relative to the choice of a factorization system. It is nevertheless true that in a lot of situations, a satisfying choice is to chose $\mathcal{E}$ to be the class of 'regular epis', and $\mathcal{M}$ to be the class of 'monomorphisms'; in particular, this works for categories of

algebraic structures such as `Set`, `Grp`, `Vec` or `Aff`. It also works well as for topological spaces `Top`.

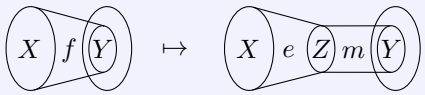
It is good to see some examples.

IN THE CATEGORY OF SETS

In the category of sets, a natural $(\mathcal{E},\mathcal{M})$-factorization is obtained by taking $\mathcal{E}$ to be the class of surjections and $\mathcal{M}$ the class the injections. The important thing is that every map $f\colon X \to Y$ can be decomposed as a composite of an injective map m and a surjection e:

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

The codomain of e is the image of f, as can be seen in the next picture.



Moreover, the diagonal property holds. Indeed, interpreting diagram (4) in the category of sets, we define d as follows. For $t \in T$ we put $d(t) = f(x)$ for some $x \in e^{-1}(t)$. Such an x exists since e is surjective, and, moreover, the definition of $d(t)$ does not depend on its choice, since m is injective. Indeed, for any other $x' \in e^{-1}(t)$ we have $f(x) = f(x')$ since m is injective and $m(f(x)) = m(f(x')) = g(t)$.

IN THE CATEGORY OF VECTOR SPACES

In the category of vector spaces, we define $\mathcal{E}$ as the class of surjective linear maps and $\mathcal{M}$ as the class of injective linear maps. Analogously to the `Set` case, one obtains an $(\mathcal{E},\mathcal{M})$-factorization.

Furthermore, the notion of factorization naturally yields the one of rank. Indeed if we decompose a linear map $f\colon X \to Y$ as the composite:

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

then the dimension of Z is exactly the rank of the linear map f.

A reason why the definition of a factorization system is so important is that it is extremely robust: in particular it naturally 'extends component-wise to functor categories', (we will briefly discuss functor categories and their relevance for automata in this context in Section 5.2). In our case, this robustness appears in the following lemma, which follows a standard categorical line of proof:

LEMMA 3.2. *Let $(\mathcal{E},\mathcal{M})$ be a factorization system for the category $\mathcal{C}$. Then $(\mathcal{E}_{\mathrm{Auto}},\mathcal{M}_{\mathrm{Auto}})$ forms a factorization system for the category $\mathtt{Auto}_L$ for all $(\mathcal{C},I,F)$-languages L, where*

— *$\mathcal{E}_{\mathrm{Auto}}$ consists of these morphisms of automata that happen to belong to $\mathcal{E}$, and*
— *$\mathcal{M}_{\mathrm{Auto}}$ consists of these morphisms of automata that happen to belong to $\mathcal{M}$.*

3.2. Initial and final automata

Apart from factorizations, the other ingredient that is required for having minimal automata is the existence of an initial automaton and of a final automaton in the category of automata for a language.

Definition 3.3. An object I in a category is *initial* if for all objects X there exists a unique arrow $I \to X$. Similarly, an object F is *final* if for all objects X there exists a unique arrow $X \to F$.

Initial and final objects, when they exist, are unique up to isomorphism. We shall see now that in our two running examples of categories of automata, the initial and final objects do exist.

DETERMINISTIC AUTOMATA

Let L be a $(\mathtt{Set}, 1, 2)$-language, i.e. $L(u)$ is a map from 1 to 2.

The *initial automaton for* L is the $(\mathtt{Set}, 1, 2)$-automaton such that:

— The set of states is A^*.
— The initial map $1 \to A^*$ selects ε.
— The final map sends a state $u \in A^*$ to $L(u) \in 2$.
— The transition map for a letter a is $\delta(a)(u) = ua$.

It is easy to check that this automaton recognizes the language L, hence it belongs to $\mathtt{Auto}_L$. A closer inspection reveals that automaton is in fact initial in the category $\mathtt{Auto}_L$.

The *final automaton for* L is the $(\mathtt{Set}, 1, 2)$-automaton such that:

— The states are the $(\mathtt{Set}, 1, 2)$-languages, i.e. the maps from A^* to maps from 1 to 2.
— The initial map sends 1 to L.
— The final map sends state R to $R(\varepsilon)$.
— The transition map for letter a sends the state R to $a^{-1}(R)$ which maps each word u to $R(au)$.

Once more, this automaton recognizes the language L, hence it belongs to $\mathtt{Auto}_L$. Again, a closer inspection reveals that automaton is in fact final in the category $\mathtt{Auto}_L$.

VECTOR SPACE AUTOMATA

Let $L(u)$ be a linear map from $\mathbb{K}$ to $\mathbb{K}$ for all words u. The *initial automaton for* L is such that: is the $(\mathtt{Vec}, \mathbb{K}, \mathbb{K})$-automaton such that:

— The state space is the vector space with basis $(e_u)_{u \in A^*}$.
— The initial map sends $x \in \mathbb{K}$ to xe_ε.
— The final map sends e_u to $L(u)(1_\mathbb{K})$.
— The transition map for the letter a sends e_u to e_{ua}.

This vector space automaton recognises the $(\mathtt{Vec}, \mathbb{K}, \mathbb{K})$-language L. A closer inspection shows that it is in fact initial with this property.

The *final automaton for* L is such that

— The state space is the vector space $\mathbb{K}^{A^*}$ of all functions from A^* to $\mathbb{K}$.
— The initial map sends $1_\mathbb{K} \in \mathbb{K}$ to the function mapping $u \in A^*$ to $L(u)(1_\mathbb{K})$.
— The final map sends $h \in \mathbb{K}^{A^*}$ to $h(\varepsilon)$.
— The transition map for the letter a sends $h \in \mathbb{K}^{A^*}$ to $\lambda u.h(au) \in \mathbb{K}^{A^*}$.

This automaton recognizes the language L, hence it belongs to $\mathtt{Auto}_L$. Yet this time, a closer inspection reveals that automaton is in fact final in the category $\mathtt{Auto}_L$.

In fact, there are some cases—like in the above examples—in which the existence of such initial and final automata for a language exist for easy category theoretic arguments. This is witnessed by the following lemma (which we include here for completeness, although we do no provide the very classical definitions of power and copower in this column, see [**?**]):

LEMMA 3.4. *If the countable copower of I exists in $\mathcal{C}$, then for all $(\mathcal{C}, I, F)$-languages L the category $\mathtt{Auto}_L$ has an initial object, called the* initial automaton for L. *If the countable power of F exists in $\mathcal{C}$, then for all $(\mathcal{C}, I, F)$-languages L the category $\mathtt{Auto}_L$ has a final object, called the* final automaton for L.

In particular, in the running examples, we have:

DETERMINISTIC AUTOMATA

Let L be a $(\mathtt{Set}, 1, 2)$-language, i.e. $L(u)$ is a map from 1 to 2.

The set of states A^* of the initial automaton for L is the copower (or coproduct) of A^*-many copies of 1.

The set of states 2^{A^*} of the final automaton for L is the power (or product) of A^*-many copies of 2.

VECTOR SPACE AUTOMATA

Let $L(u)$ be a linear map from $\mathbb{K}$ to $\mathbb{K}$ for all words u.

The vector space of configurations of the initial automaton for L is the copower (or coproduct) of A^*-many copies of $\mathbb{K}$, that is, the direct sum $\bigoplus_{u \in A^*} \mathbb{K}$ of A^*-many copies of $\mathbb{K}$.

The vector space of configurations of the final automaton for L is the power (or product) of A^*-many copies of $\mathbb{K}$, that is, the direct product $\prod_{u \in A^*} \mathbb{K}$ of A^*-many copies of $\mathbb{K}$.

3.3. Minimal automaton and minimization

At last, we are able to provide a general description of why there exists a minimal automaton for a language, and what is the general procedure for minimizing a given automaton.

In fact, the notion of a minimal automaton is now generic, it is a notion that works whenever there is an initial object, a final object, and some factorization system $(\mathcal{E}, \mathcal{M})$.

Consider a factorization system $(\mathcal{E}, \mathcal{M})$ for a category $\mathcal{A}$ and two of its objects X, Y. Let us say that:

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

Let us note immediately that in general this notion of $(\mathcal{E},\mathcal{M})$-divisibility may not be transitive[4]. 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, is our case, when the category has both initial and a final object, we can state the following minimization lemma:

LEMMA 3.5. *Let $\mathcal{A}$ be a category with initial object I and final object F and let $(\mathcal{E}, \mathcal{M})$ be a factorization system for $\mathcal{A}$. Define for every object X:*

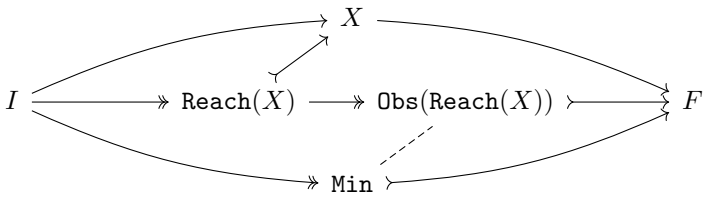
— $\mathtt{Min}$ *to be the factorization of the only arrow from I to F,*
— $\mathtt{Reach}(X)$ *to be the factorization of the only arrow from I to X, and*
— $\mathtt{Obs}(X)$ *to be the factorization of the only arrow from X to F.*

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:

[4]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.



Using the definition of Reach and 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 an factorization system, 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 Min. In a symmetric way, $\texttt{Reach}(\texttt{Obs}(X))$ is also isomorphic to Min. □

COROLLARY 3.6. *Let L be a $(\mathcal{C}, I, F)$-language for which $\mathcal{C}$ has a factorization system $(\mathcal{E}, \mathcal{M})$ such that $\texttt{Auto}_L$ contains both an initial automaton $\mathcal{I}$ and a final automaton $\mathcal{F}$. Then:*

— *The $(\mathcal{C}, I, F)$-automaton $\mathcal{S}$ for L that is at the middle point of an $(\mathcal{E}_{\text{Auto}},\mathcal{M}_{\text{Auto}})$-factorization of the only automata morphism from $\mathcal{I}$ to $\mathcal{F}$ is called the* syntactic automaton for L.
— *The syntactic automaton for L $\mathcal{S}$ $(\mathcal{E}_{\text{Auto}},\mathcal{M}_{\text{Auto}})$-divides every automaton for L.*
— *For all automata $\mathcal{A}$ for L, $\mathcal{S}$ is isomorphic to both $\texttt{Reach}(\texttt{Obs}(\mathcal{A}))$ and $\texttt{Obs}(\texttt{Reach}(\mathcal{A}))$.*

The process of starting from an automaton, and applying to it Reach then Obs (in any order) is called '*minimization*. Note that implementing Reach and Obs in an effective way is a problem that may prove difficult on its own, and we do not elaborate on this aspect.

DETERMINISTIC AUTOMATA

It is well known that for all languages $L \subseteq A^*$, there exists a minimal deterministic automaton for it, that furthermore is finite if and only L is regular. Indeed, if L is accepted by a finite automaton $\mathcal{A}$, then, by Corollary 3.6 the syntactic automaton for L divides $\mathcal{A}$, hence its state space must be finite since it is the quotient of a subset of a finite set.

VECTOR SPACE AUTOMATA

Similarly, it is well known that for all languages $L\colon A^* \to \mathbb{K}$, there exists a minimal vector space automaton for it, that furthermore is finite dimensional if and only L is regular. Indeed, if L is accepted by a finite dimensional vector space automaton $\mathcal{A}$, then the syntactic automaton is a quotient of $\mathcal{A}$, hence its state space must be finite dimensional since it is the quotient of a subspace of a finite dimensional space.

4. A NOVEL FORM OF AUTOMATA: HYBRID-SET-VECTOR AUTOMATA

In this section, we describe a new form of automata, which we can call hybrid-set-vector automata, and which extend both deterministic finite automata and vector space automata, while still possessing syntactic automata. We will see how this minimization is obtained along the same lines described in this column so far.

4.1. An intuition

Let us consider the vector space automaton $\mathcal{A}_{\mathrm{vec}}$ from Example 2.1 accepting the map L_{vec}, which corresponds to the weighted language $A^* \to \mathbb{R}$ mapping u to $2^{|u|_a}$ when u does not contain any c and has even number of b's, and to 0 otherwise.

Let us think for a moment on how one would "implement" the function L_{vec} as an online device that would get letters as input, and would modify its internal state accordingly. Would we implement concretely the automaton of Example 2.1 directly? Probably not, since there is a more economic[5] way to obtain the same result: we can maintain 2^m where m is the number of a's seen so far, together with one bit for remembering whether the number of b's is even or odd. Such an automaton would start with 1 in its unique real valued register. Each time an a is met, the register is doubled, each time b is met, the bit is reversed, and when c is met, the register is set to 0. At the end of the input word, the automaton would output 0 or the value of the register depending on the current value of the bit.

If we consider the configuration space that we use in this encoding, we use $\mathbb{R} \uplus \mathbb{R}$ instead of $\mathbb{R} \times \mathbb{R}$. Essentially, the set of vectors spanned by applying in arbitrary order the linear transformations δ_a, δ_b and δ_c from Example 2.1 to the vector $(1, 0) \in \mathbb{R}^2$ is the infinite set of vectors described in the above diagram. Of course, in the category of vector spaces this set spans the whole $\mathbb{R}^2$. Yet, in this example this set lies on the "union" of two one dimensional spaces. Can we define an automaton model that would be able to faithfully implement this example?

4.2. A first generalization: disjoint unions of vector spaces.

A way to achieve this is to interpret the generic notion of automata in the category of finite disjoint unions of vector spaces (*duvs*). One way to define such a *finite disjoint unions of vector spaces* is to use a finite set N of '*indices*' $p, q, r \ldots$, and to each index p associate a vector space V_p, possibly with different dimensions. The *corresponding set* is:

$$\{(p, \vec{v}) \mid p \in N,\ \vec{v} \in V_p\}\ .$$

A 'map' between duvs represented by (N, V) and (N', V') is then a pair $h : N \to N'$ together with a linear map f_p from V_p to $V'_{h(p)}$ for all $p \in N$. It can be seen as mapping each $(p, \vec{v}) \in N \times V_p$ to $(h(p), f_p(\vec{v}))$. Call this a *duvs map*. Such duvs maps are composed in a natural way. This defines a category, and hence we can consider *duvs automata* which are automata with a duvs for its state space, and transitions implemented by duvs maps.

For instance, we can pursue with the computation of L_{vec} and provide a duvs automaton

$$\mathcal{A}^{\mathtt{duvs}} = (Q^{\mathtt{duvs}}, i^{\mathtt{duvs}}, f^{\mathtt{duvs}}, \delta^{\mathtt{duvs}})$$

where

$$Q^{\mathtt{duvs}} = \{(s, x) \mid s \in \{\mathtt{even}, \mathtt{odd}\},\ x \in \mathbb{K}\}$$

(considered as a disjoint union of vector spaces with set of indices $\{\mathtt{even}, \mathtt{odd}\}$ and all associated vector spaces $V_{\mathtt{even}} = V_{\mathtt{odd}} = \mathbb{K}$). The maps can be conveniently defined as

[5] Under the assumption that maintaining a real is more costly than maintaining a bit.

follows:

$$i^{\mathtt{duvs}}(x) = (\mathtt{even}, x) \qquad (\mathtt{even}, x) = (\mathtt{even}, 2x) \qquad \delta_a^{\mathtt{duvs}}(\mathtt{odd}, x) = (\mathtt{odd}, 2x)$$
$$f^{\mathtt{duvs}}(\mathtt{even}, x) = x \qquad \delta_b^{\mathtt{duvs}}(\mathtt{even}, x) = (\mathtt{odd}, x) \qquad \delta_b^{\mathtt{duvs}}(\mathtt{odd}, x) = (\mathtt{even}, x)$$
$$f^{\mathtt{duvs}}(\mathtt{odd}, x) = 0 \qquad \delta_c^{\mathtt{duvs}}(\mathtt{even}, x) = (\mathtt{even}, 0) \qquad \delta_c^{\mathtt{duvs}}(\mathtt{odd}, x) = (\mathtt{odd}, 0)$$

This automaton computes the language L_{vec}. Automata over finite disjoint unions of vector spaces generalize both deterministic finite state automata (using only 0-dimensional vector spaces), and vector space automata (using only one index). In this particular example, it can also be seen as a semi-direct product of a two state machine with a purely vector space automaton, (but this remark fails when the spaces V_p have different dimensions.) However, is it the joint generalization that we hoped for? The answer is no...

4.3. Failure to minimize.

We could think that the above automaton $\mathcal{A}^{\mathtt{duvs}}$ is minimal. However, it involved some arbitrary decisions when defining it. This can be seen in the fact that when $\delta_c^{\mathtt{duvs}}$ is applied, we chose to not change the index (and set to null the real value): this is arbitrary, and we could have exchanged `even` and `odd`, or fixed it arbitrarily to `even`, or to `odd`. All these *variants* would be equally valid as far as computing L_{vec} is concerned.

Let us provide some high level intuitions, invoking some standard automata-theoretic concepts. The first remark is that every configuration in $Q^{\mathtt{duvs}}$ is 'reachable' in this automaton: indeed $(\mathtt{even}, x) = i^{\mathtt{duvs}}(x)$ and $(\mathtt{odd}, x) = \delta_b^{\mathtt{duvs}} \circ i^{\mathtt{duvs}}(x)$ for all $x \in \mathbb{K}$. Hence there is no hope to improve the automaton $\mathcal{A}^{\mathtt{duvs}}$ or one of its variants by some form of 'restriction to its reachable configurations'. Only 'quotienting of configurations' remains. However, (using the only reasonable definition of quotient in duvs), one can show that none among $\mathcal{A}^{\mathtt{duvs}}$ and its variants is the quotient of another. More precisely, if we keep in mind the Myhill-Nerode equivalence, what we would like to do is to merge the configurations $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$ since these are observationally equivalent:

$$f^{\mathtt{duvs}} \circ \delta_u^{\mathtt{duvs}}(\mathtt{even}, 0) = 0 = f^{\mathtt{duvs}} \circ \delta_u^{\mathtt{duvs}}(\mathtt{odd}, 0) \qquad \text{for all words } u \in A^*.$$

However, if we try to merge using a duvs map the configurations $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$, we eventually obtain an automaton with one index associated to a one-dimensional vector space. This would in fact be a vector space automaton, and we already mentioned that such an automaton cannot compute L_{vec}. Overall, there is no minimal duvs automaton for L_{vec}.

4.4. The category of gluings of vector spaces

The subject of this section is to introduce hybrid-set-vector automata, and for this we need to describe the category that they use: the category of gluings of vector spaces

Indeed, after the failure to minimize of the last section, the only reasonable thing to do is to try to merge $(\mathtt{even}, 0)$ and $(\mathtt{odd}, 0)$, but nothing else (because no other pairs of distinct states are observationally equivalent). This is made possible thanks to a change of category, in which 'gluings' can be performed.

A direct definition

The first solution is to do it naturally: a gluing of vector spaces would be a disjoint union of vector spaces enriched with some 'equivalence' representing how the different components of the duvs have to be glued. Formally, a *gluing of vector spaces* (N, V) consists of

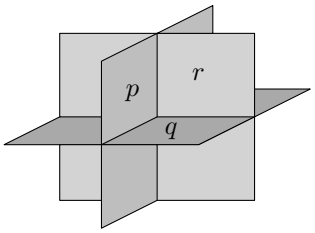
— a set of indices N and vector spaces $(V_p)_{p\in N}$, thus forming a disjoint union of vector spaces, together with
— a *gluing equivalence* $\sim_{\text{glue}}$ which is an equivalence relation on the corresponding set

$$|(N,V)| = \{(p,\vec{v}) \mid p \in N,\ \vec{v} \in V_p\}$$

such that for all distinct indices $p, q \in N$,
  — the equivalence classes of $\sim_{\text{glue}}$ restricted to each $\{p\} \times V_p$ are singletons, and
  — the equivalence classes of $\sim_{\text{glue}}$ restricted to $\{p\} \times V_p \cup \{q\} \times V_q$ that are of size 2 form a linear bijection between a subspace of V_p and a subspace of V_q.

An example consists of three copies of $\mathbb{R}^2$, say p, q, r such that furthermore

$$\begin{aligned} (p,x,0) &\sim_{\text{glue}} (q,0,x)\ , \\ (q,x,0) &\sim_{\text{glue}} (r,0,x)\ , \\ \text{and} \qquad (r,x,0) &\sim_{\text{glue}} (p,0,x)\ . \end{aligned}$$



This could be roughly described as the picture to the right.

The definition of *maps of gluings of vector spaces* is then the one of maps of disjoint union of vector spaces, but that would furthermore be required to preserve $\sim_{\text{glue}}$.[6] For instance, the map g which for all $x, y \in \mathbb{R}$ is defined by:

$$\begin{aligned} g(p,x,y) &= (q,y,x) \\ g(q,x,y) &= (r,y,x) \\ g(r,x,y) &= (p,y,x)\ , \end{aligned}$$

does preserve the structure of the above gluing equivalence, and hence is a valid map of gluings of vector spaces.

Note that a map of gluings of vector spaces $f\colon S \to T$ induces a map $|f|\colon |S| \to |T|$. (In fact, this translation is a functor from the category of gluings of vector spaces to the category of sets)

Definition 4.1. The gluings of vector spaces together with the maps between them form a category called the *category of gluings of vector spaces*. We denote it $\texttt{Glue}(\texttt{Vec})$ (we shall give some more explanations about this notation).

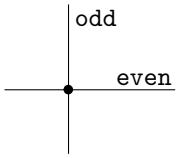
An *hybrid-set-vector automaton* is then simply an automaton in the category of gluings of vector spaces.

Example 4.2. The hybrid-set-vector automaton for L_{vec} that we are interested in can now be described formally:

— the state object is a gluing of vector spaces that consists of two copies of the vector space $\mathbb{R}$, indexed as $\texttt{even}$ and $\texttt{odd}$ that are glued at 0, i.e.,

$$(\texttt{even},0) \sim_{\text{glue}} (\texttt{odd},0)\ .$$

This could be depicted as in the right figure, in which the gluing equivalence is emphasized using a black dot.



[6] In fact, we are making an approximation here, since several such maps should be considered as equivalent. For instance, in our example, the map of gluing of spaces (corresponding to the transition of letter c in $\mathcal{A}^{\text{duvs}}$) that sends $(\texttt{even},x) \mapsto (\texttt{even},0)$ and $(\texttt{odd},x) \mapsto (\texttt{odd},0)$ is equivalent to the one that sends $(\texttt{even},x) \mapsto (\texttt{even},0)$ and $(\texttt{odd},x) \mapsto (\texttt{even},0)$, simply because $(\texttt{odd},0) \sim_{\text{glue}} (\texttt{even},0)$.

— The initial map sends x to $(\texttt{even}, x)$.
— The final map sends $(\texttt{even}, x)$ to x and $(\texttt{odd}, x)$ to 0.
— The transition map for letter a sends $(\texttt{even}, x)$ to $(\texttt{even}, 2x)$ and $(\texttt{odd}, x)$ to $(\texttt{odd}, 2x)$.
— The transition map for letter b sends $(\texttt{even}, x)$ to $(\texttt{odd}, x)$ and $(\texttt{odd}, x)$ to $(\texttt{even}, x)$.
— The transition map for letter c sends $(\texttt{even}, x)$ to $(\texttt{even}, 0)$ and $(\texttt{odd}, x)$ to $(\texttt{even}, 0)$.

The category of gluings of vector spaces can be seen as a joint extension of the category of sets and the category of vector spaces. This is the reason for the name "hybrid-set-vector automaton". This remark is made formal now:

LEMMA 4.3. *The category of gluings of vector spaces restricted to gluings of 0-dimension vector spaces is equivalent*[7] *to the category of sets.*
The category of gluings of vector spaces restricted to gluings of vector spaces with one index only is equivalent to the category of vector spaces.

Thanks to the above lemma, we obtain that:

— deterministic automata, which are $(\texttt{Set}, 1, 2)$-automata, are also hybrid-set-vector automata; namely the ones in which the definition of the state object does only involve 0-dimension vector spaces),
— vector space automata, which are $(\texttt{Vec}, \mathbb{K}, \mathbb{K})$-automata, are also hybrid-set-vector automata; namely the one that have only one index in the definition of their state object.

A categorical approach to gluing: the category $\texttt{Glue}(\texttt{Vec})$

The few lines that follow require some background in category theory. However, these are not necessary for understanding the rest of the paper.

Another approach for defining gluings of vector spaces is to consider $\texttt{Vec}$ as a generic category $\mathcal{C}$, and define in categorical terms the 'gluings of objects in $\mathcal{C}$'. We name it $\texttt{Glue}(\mathcal{C})$.

In fact, the reader used to categorical construction knows the standard approach for gluing objects in a category, based on the concept of a free colimit. In this view, the category of gluings of vector spaces can be seen as a subcategory of the free cocompletion of $\texttt{Vec}$. Informally, an element of the free cocompletion of $\texttt{Vec}$ is an (equivalence class of) 'diagrams' that describe a set of objects of $\mathcal{C}$ and give constraints on how these should be 'glued together'. However, this generic description is much less constrained that the one we have described in the previous definition of the gluing of vector spaces. For instance, it is possible using free colimits to take a copy of $\mathbb{R}^2$ and 'glue' together the axis $\mathbb{R}(0,1)$ and $\mathbb{R}(1,0)$ (say using $(0,x) \sim (x,0)$). Such a construction would yield a formal object that is different than the gluings of vector spaces we are interested in.

Thus, our definition of $\texttt{Glue}(\mathcal{C})$ does only use the diagrams that we consider 'meaningful'. It can be (informally) stated as follows:

Definition 4.4. $\texttt{Glue}(\mathcal{C})$ is the free cocompletion of $\mathcal{C}$ restricted to the diagrams that have a cocone in $\mathcal{C}$, all the arrows of which are monos (or more generally in $\mathcal{M}$ for a suitable class $\mathcal{M}$).

The advantage of this approach is that it can be used with other categories; for instance for constructing the category of *gluings of affine spaces* (i.e. $\texttt{Glue}(\texttt{Aff})$) or even *gluing of sets* (i.e. $\texttt{Glue}(\texttt{Set})$). These categories have interest on their own, that we do not develop in this column.

[7] *Equivalent* is the proper notion of 'isomorphism' for categories. Technically, it is an isomorphism 'up to isomorphisms of the objects'.

4.5. The minimization of hybrid-set-vector automata

In the previous section, we introduced the concept of hybrid-set-vector automata; these automata live in the category of gluings of vector spaces. Our motivation was to explain how these can be minimized. In fact, we shall see that maybe these are not exactly the automata we are interested in.

Let us recall that we had identified three ingredients for the existence of minimal automata for a language: the existence of an initial automaton, the existence of a final automaton, and the existence of a factorization system. Let us review what is the status of the category of hybrid-set-vector automata for a language with respect to these three points.

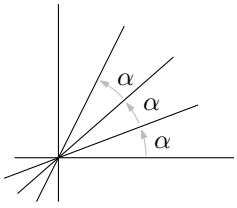
The following lemma can be proved using a more generic argument for handling initial and final automata:

LEMMA 4.5. *The category of hybrid-set-vector automata for a language has an initial and a final object.*[8]

The more interesting part concerns the third ingredient required for having minimal automata: the existence of a factorization system for $\mathtt{Glue}(\mathtt{Vec})$. Indeed, it has one, but the issue is that... this is not what we really are looking for, as shown by the following example:

Example 4.6. Consider the language which to a word $u \in a^*$ associates the value $\cos(\alpha|u|)$ for some α which is not a rational multiple of π. This can be recognized by a vector space automaton as follows:

— the vector space of configurations is $\mathbb{R}^2$,
— the initial map maps x to $(x, 0)$,
— the final map maps (x, y) to x, and
— the transition map for the letter a performs a rotation of α radian of the plane $\mathbb{R}^2$: it maps (x, y) to $(\cos(\alpha)x - \sin(\alpha)y, \sin(\alpha)x + \cos(\alpha)y)$:



(This automaton can also be seen as a hybrid-set-vector automaton.) Now, the $\mathtt{Glue}(\mathtt{Vec})$-automaton obtained by minimizing (in fact, simply by restriction to the reachable states), consists of countably many copies of the line $\mathbb{R}$, all glued at 0. More precisely,

— the state object is the gluing of vector spaces that has indices $\mathbb{N}$, each vector space V_n is $\mathbb{R}$, and the gluing equivalence merges all the 0 points: $(m, x) \sim_{\text{glue}} (n, y)$ if $m = n$ and $x = y$, or if $x = y = 0$,
— the initial map maps $\mathbb{R}$ to the vector space of index 0, i.e. it maps every $x\mathbb{R}$ to $(0, x)$,
— the final map maps (m, x) to $\cos(\alpha m)x$,
— and the transition maps sends (m, x) to $(m + 1, x)$.

[8] In fact, $\mathtt{Glue}(\mathcal{C})$ has all coproducts (and as a consequence all copowers), and furthermore all colimits that $\mathcal{C}$ has. If $\mathcal{C}$ has all colimits and products, then $\mathtt{Glue}(\mathcal{C})$ also has all products (and hence all powers). If $\mathcal{C}$ has all limits and colimits then $\mathtt{Glue}(\mathcal{C})$ also has them.

It happens that this automaton is the minimal one (This automaton would still be correct if α would be a rational multiple of π, but in this case, it could not be minimal). What we see here is that this automaton is merely storing the "current rotation" in the index part of the gluing of vector spaces of configurations.

The above example shows that minimizing without further caution leads to a problem. Indeed, we started from a perfectly valid dimension 2 vector space used for recognizing a language, and after minimizing it it became an automaton that uses a countable union of vector spaces as configurations: something that is more difficult to handle effectively. The answer to this problem lies in a assumption that was left unspoken so far. We are interested only in automata involving 'finite gluings of finite dimension vector spaces' only, because these are the automata that can be used algorithmically. We get the following definition:

Definition 4.7. A gluing of vector spaces is *effective* if it has a finite index, and all the vector spaces involved in its definition are of finite dimension. In the same way, the hybrid-set-vector automata that have an effective state object are also named *effective*.

At the categorical level, this means to define $\mathtt{Glue}_{\mathsf{fin}}(\mathcal{C})$, and construct this way the category $\mathtt{Glue}_{\mathsf{fin}}(\mathtt{Vec}_{\mathsf{fin}})$ where $\mathtt{Vec}_{\mathsf{fin}}$ is the category:

$$\mathtt{Vec}_{\mathsf{fin}} = (\text{finite dimension vector spaces}, \text{linear maps}),$$

with the natural notion of composition of linear maps and identity map.

To complete the picture, it is necessary to explain how to minimize in the world of the effective hybrid-set-vector automata. However, following the line of descriptions seen so far, there is a problem:

— We need to use hybrid-set-vector automata in their general form, since the initial and the final automaton, which are essential parts in the minimization arguments, are never effective (unless the input alphabet is empty...).
— However, we want the 'minimal automaton' that we construct as a result of a factorization to be effective.

The way to resolve this conflict is to modify in a simple and natural, yet unconventional to our knowledge, way the notion of factorization system. We substitute to it the notion of "factorization system through". The heart of this definition is to consider a subcategory $\mathcal{S}$ (that we think of as the category of small/manageable objects) of a larger category $\mathcal{C}$:

Definition 4.8. Consider a category $\mathcal{C}$, a full subcategory $\mathcal{S}$ of $\mathcal{C}$. Call *$\mathcal{S}$-small* an arrow $f\colon X \to Y$ of $\mathcal{C}$ that factors through $\mathcal{S}$, i.e. such that $f = h \circ g$ with $g\colon X \to Z$ and $h\colon Z \to Y$ for some Z object of $\mathcal{S}$.

Example 4.9. Consider the category $\mathcal{C}$ of $(\mathtt{Set}, 1, 2)$-automata, which are the deterministic automata, and its subcategory $\mathcal{S}$ of $(\mathtt{Set}_{\mathsf{fin}}, 1, 2)$-automata (where $\mathtt{Set}_{\mathsf{fin}}$ is the subcategory of finite sets), which is the category of finite deterministic automata. Then for a language L the only morphism from the initial automaton for L to the final automaton for L is $\mathcal{S}$-small if and only if L is a regular language. Indeed, being $\mathcal{S}$-small in this example means exactly being accepted by a finite deterministic automaton.

If one takes the category $\mathcal{C} = \mathtt{Vec}$ and its subcategory $\mathcal{S} = \mathtt{Vec}_{\mathsf{fin}}$, then a linear map in $\mathcal{C}$ is $\mathcal{S}$-small if and only if it is of finite rank.

We now introduce the refined notion of factorization through. It essentially formalizes what it is to be a factorization system that factorizes only $\mathcal{S}$-small arrows, and further do it in $\mathcal{S}$. Formally, the definition is only a slight variation around the one for a factorization system.

Definition 4.10. Let $\mathcal{S}$ be a subcategory of a category $\mathcal{C}$, and $\mathcal{E}_\mathcal{S}, \mathcal{M}_\mathcal{S}$ be two classes of arrows such that:

— all arrows in $\mathcal{E}_\mathcal{S}$ end in $\mathcal{S}$, and
— all arrows in $\mathcal{M}_\mathcal{S}$ start in $\mathcal{S}$,

then $(\mathcal{E}_\mathcal{S}, \mathcal{M}_\mathcal{S})$ forms a *factorization system through* $\mathcal{S}$ if the following conditions hold, where we denote the arrows in $\mathcal{E}_\mathcal{S}$ by two-headed arrows $\twoheadrightarrow$ and the arrows in $\mathcal{M}_\mathcal{S}$ by $\rightarrowtail$.

— The arrows that are both in $\mathcal{E}_\mathcal{S}$ and $\mathcal{M}_\mathcal{S}$ are exactly the isomorphisms in $\mathcal{S}$.
— The $\mathcal{E}_\mathcal{S}$-arrows are closed under composition.
— The $\mathcal{M}_\mathcal{S}$-arrows are closed under composition.
— For all $\mathcal{S}$-small arrows $f\colon X \to Y$, there exists some object Z of $\mathcal{S}$, an $\mathcal{E}_\mathcal{S}$-arrow $e\colon X \twoheadrightarrow Z$ and a $\mathcal{M}_\mathcal{S}$-arrow $m\colon Z \rightarrowtail Y$ such that

$$f = m \circ e\ .$$

This composition is called the *factorization of* f *through* $\mathcal{S}$. We also refer to the object Z as the *factorization of* f *through* $\mathcal{S}$.
— For all arrows $e\colon X \twoheadrightarrow T$ in $\mathcal{E}_\mathcal{S}$, $g\colon T \to Y$, $f\colon X \to S$ and $m\colon S \rightarrowtail Y$ in $\mathcal{M}_\mathcal{S}$ such that $g \circ e = m \circ f$, there exists one and exactly one arrow $d\colon T \to S$ (of $\mathcal{S}$) such that $d \circ e = f$ and $m \circ d = g$. In other words, if the following square commutes, then there exists a unique diagonal arrow such that the resulting diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{e} & T \\ {\scriptstyle f}\downarrow & \overset{d}{\swarrow} & \downarrow{\scriptstyle g} \\ S & \xrightarrow[m]{} & Y \end{array} \tag{4}$$

As can be expected, this property is also called the *diagonal property*, and the unique morphism d is called a *diagonal fill*.

In fact, what happens is that all the proofs that we have done so far, and in particular the existence of a minimal object, Lemma 3.5, can be adapted to this variation of the notion of factorization system.

For instance, Lemma 3.5, becomes:

LEMMA 4.11. *Let $\mathcal{A}$ be a category with initial object I and final object F and let $(\mathcal{E}_\mathcal{S}, \mathcal{M}_\mathcal{S})$ be a factorization system though a subcategory $\mathcal{S}$ for $\mathcal{A}$. Assume furthermore that the only arrow from I to F is $\mathcal{S}$-small, and define for all objects X of $\mathcal{S}$:*

— $\texttt{Min}_\mathcal{S}$ *to be the factorization through $\mathcal{S}$ of the only arrow from I to F,*
— $\texttt{Reach}_\mathcal{S}(X)$ *to be the factorization through $\mathcal{S}$ of the only arrow from I to X (note that this arrow is $\mathcal{S}$-small since X is an object of $\mathcal{S}$), and*
— $\texttt{Obs}_\mathcal{S}(X)$ *to be the factorization through $\mathcal{S}$ of the only arrow from X to F (note that it is $\mathcal{S}$-small since X is an object of $\mathcal{S}$).*

Then

— $\texttt{Min}_\mathcal{S}$ *is ($\mathcal{E}_\mathcal{S}$,$\mathcal{M}_\mathcal{S}$)-minimal (for the natural definition of it), and*
— $\texttt{Min}_\mathcal{S}$, $\texttt{Obs}_\mathcal{S}(\texttt{Reach}_\mathcal{S}(X))$ *and* $\texttt{Reach}_\mathcal{S}(\texttt{Obs}_\mathcal{S}(X))$ *are isomorphic for all objects X of $\mathcal{S}$.*

Now, almost all the landscape is ready. We have a class of automata, the hybrid-set-vector automata. It has an initial and a final object according to Lemma 4.5. We have also identified the subcategory of effective hybrid-set-vector automata for the language. It remains to tell what are the classes $\mathcal{E}_\mathcal{S}$ and $\mathcal{M}_\mathcal{S}$ that we use for 'factorizing through' (Lemma 4.11).

Definition 4.12. The *effective monos* are the arrows $m\colon X \to Y$ in the category of hybrid-set-vector automata where X is effective, and $|m|$ is injective, i.e., the map is injective when all structure has been forgotten.

The *effective extremal epis* are the arrows $e\colon X \to Y$ with Y effective, and such that whenever $e = m \circ f$ for some effective mono m, then m is an isomorphism.

Under this definition, we obtain the expected factorization system through.

LEMMA 4.13. *(Effective extremal epis, effective monos) forms a factorization system of the category of hybrid-set-vector automata through the subcategory of effective hybrid-set-vector automata.*

If we briefly summarize what we have, we get the following statement.

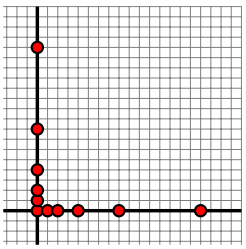
THEOREM 4.14. *For all languages accepted by an effective hybrid-set-vector automaton, there exists a (effective extremal epis, effective monos)-minimal equivalent one.*

In particular, the effective hybrid-set-vector automaton of Example 4.2 is minimal as in the above theorem.

Though we do not provide more detail here on why this result holds (not to mention the effectivity of the constructions), let us emphasize that it relies crucially on the following statement:

> In a finite dimension vector space E, for every set $X \subseteq E$, there exists a finite union of vector spaces $F \subseteq E$ such that $X \subseteq F$ and which is minimal for inclusion.

This statement is not difficult to establish. At any rate, it gives a good intuition of what is happening.
At the very beginning, we have introduced the vector space automaton $\mathcal{A}_{\text{vec}}$ that accepts the language L_{vec}. Its state space is $\mathbb{R}^2$. However, starting, say, from $(1, 0)$, we can draw all the configurations that are reachable by applications of the transition maps associated to the letters. We obtain the picture to the right.



Now, we can apply the above statement: there is a least finite union of subspaces that covers the red points. Indeed, it consists of the union $\mathbb{R}(0, 1) \cup \mathbb{R}(1, 0)$. If now, we forget the fact that these two dimension 1 vector spaces are subspaces of $\mathbb{R}^2$, then what remains is a union of two copies of $\mathbb{R}$ that are glued at 0. This is how one obtains the automaton of Example 4.2.

5. DISCUSSION: AUTOMATA AND CATEGORY THEORY

The idea of using category theory to provide a unifying framework for automata theory and linear systems of control theory goes back all the way to the late sixties, and ever since there has been a substantial body of research on this topic. Perhaps it is not a coincidence, that Samuel Eilenberg, one of founders of the field of category theory is also the author of the influential automata theory books [Eilenberg 1974; Eilenberg 1976], which axiomatise (without using category theory!) the algebraic approach to automata theory and provide among others his celebrated variety theorem.

On the other hand, cutting-edge research in automata theory relies at times on combinatorial aspects, which seem difficult, if not impossible, to explain in a generic setting. The purpose of this column is to convince people coming from the combinatorial background, and who have not been previously exposed to category theory, that the

conceptual view point is not just an idle exercise in abstract nonsense, and that sometimes it can lead to new interesting combinatorial problems.

As a disclaimer, it was not our intention to provide an exhaustive survey of the interplay between automata and category theory, nor of the huge literature on this very topic. But we feel necessary to highlight some contributions and perspectives.

Early days: Machines in a category. Already in the late 60s, Eilenberg and Wright [Eilenberg and Wright 1967] gave a generalised notion of language recognizability in categories of algebras and formulated their results in terms of Lawvere's algebraic theories, the back then fresh approach to categorical algebra. A couple of years later, Arbib and Manes further advanced the category-theoretic unification of sequential machines and linear systems of control theory in a series of seminal papers [Arbib and Manes 1975; Arbib and Manes 1974a; Arbib and Manes 1974b]. One of their contributions was the connection between minimization and factorization systems, as well as an account of the duality between reachability and observability in this setting, building on the work of Kalman on linear systems [Kalman 1963]. In parallel, Goguen [Goguen 1972] also developed a theory of minimal realization in the setting of monoidal closed categories. A nice survey of these early developments can be found in [Arbib and Manes 1980].

Automata as algebras for a functor. Arbib and Manes advanced the view of sequential machines and linear systems as algebras for a functor, although they adopted a different terminology in those early papers (algebras for a functor were called dynamorphisms). In this setting, the transition map is captured as an algebra for a functor F, that is a map of the form $\delta\colon FQ \to Q$. For example, for deterministic finite automata on a finite alphabet A, we would use the functor $F\colon \mathtt{Set} \to \mathtt{Set}$ given by $FX = A \times X$. This approach was further developed in the work of the Prague seminar on General Mathematical Structures by Věra Trnková, Jiři Adámek, Jan Reiterman, Václav Koubek, see for example the book [Adámek and Věra 1989] and the references therein. These works explore free algebras for finitary functors, existence and universality of minimal realisations, as well descriptions of languages via rational operations. In particular [Adámek and Věra 1989, Theorem III.2.14], similar in spirit to the developments we presented in Section 3.3, establishes sufficient conditions for the existence of minimal realization via factorization systems for automata modeled as algebras for coadjoint functors preserving epimorphisms.

Yet, automata are not entirely modelled as algebras. The initial state can be incorporated in the type of the functor, but specifying the final states is outside the realm of algebra. For example, for deterministic finite automata, one can consider the functor given by $FX = 1 + A \times X$. Formally a deterministic finite automaton is a map $[i, \delta]\colon 1 + A \times Q \to Q$, plus the characteristic function of the subset of final states $f\colon Q \to 2$.

Automata as coalgebras for a functor. Alternatively, one could model deterministic finite automata as maps of the form $\langle f, \gamma\rangle\colon Q \to 2 \times Q^A$ obtained by pairing the characteristic function of the subset of accepting states and the map $\gamma\colon Q \to Q^A$, obtained from the transition map δ via currying. The map $\langle f, \gamma\rangle$ is an example of a coalgebra for the functor $G\colon \mathtt{Set} \to \mathtt{Set}$, defined by $GX = 2 \times X^A$, however, in this framework, it is the initial state which is left out.

The view of automata (and more generally of systems) as coalgebras was put forward in the work of Rutten and Jacobs, see [Jacobs and Rutten 1997] and [Rutten 2000]. However, the roots of coalgebras in computer science go back to the work of Aczel and his insights into the coinductive nature of Milner-Park notion of bisimulation from concurrency theory. The coalgebraic view of automata, and the connections

to other fields of computer science proved surprisingly useful. An example of a success story is the work of Bonchi and Pous [Bonchi and Pous 2013], which gives an efficient algorithm for deciding language equivalence for non-deterministic finite automata, using enhancements of the coinductive proof techniques (the so-called up-to techniques) and drawing on previous developments from concurrency theory.

The coalgebraic method (along with the numerous contributions in this research area) was described in the *Semantics Column* of a previous SIGLOG news issue [Silva 2015] and was illustrated by giving a category-theoretic account of Brzozowski's minimization algorithm. Which brings us to...

Minimization. A dual narrative: algebra vs. coalgebra. Automata minimization was understood both algebraically at different levels of generality, (as in the work of Arbib and Manes, Adámek and Trnková, etc.), as well as coalgebraically, see for example [Adámek et al. 2012; Bonchi et al. 2012]. Notably, [**?**] carries out an elegant study of minimization algorithms for linear weighted automata from a coalgebraic viewpoint.

The interplay between these two views of automata, both as algebras and as coalgebras, was fully exploited in [Bonchi et al. 2014] to give a category-theoretic account of Brzozowski's minimization algorithm. The paper [Rot 2016] showcases the connection between two approaches to minimization (either by partition refinement or reverse-determinisation)—one involving an initial algebra construction, the other a final coalgebra construction. The deep connection between minimization and duality theory were also investigated in [Bezhanishvili et al. 2012].

We should mention in passing that duality theory plays a fundamental role in language theory on aspects related to recognition (see [**?**; **?**]), and, coincidentally, this is also featured in this issue, in the *Complexity* column. These works were the starting point of the ERC project *DuaLL* which made this collaboration possible. In the same spirit, the recent paper [**?**] explores other ideas from category and duality theory for tackling problems in language theory.

5.1. Beyond that point

In this column, we have attempted to show how category theory gives an insight into the nature of automata and the questions of minimization. These facts are well known for decades, though we adopted a presentation which we believe to be simpler and more direct. There are many continuations of this description that could be followed from that point, and space does not permit it.

5.2. Automata as functors

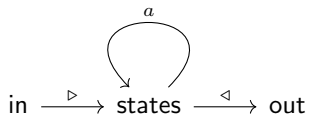
For the category-theoretic minded readers, we would like to emphasise some aspects of the approach described here. In this subsection we will assume more category-theoretic background on the part of the reader.

Although we haven't mentioned this explicitly thus far, our view of automata is neither algebraic, nor coalgebraic, but a "combination" of the two. Formally, we view automata as functors

$$\mathcal{A} \colon \mathcal{I} \to \mathcal{C}$$

where $\mathcal{I}$ is a category of inputs and $\mathcal{C}$ is the category which specifies the universe of output values. For example, for word automata, the category $\mathcal{I}$ has three objects $\mathsf{in}, \mathsf{states}$ and out, and for each $w \in A^*$, arrows $\triangleright w \colon \mathsf{in} \to \mathsf{states}$, $w \triangleleft \colon \mathsf{states} \to \mathsf{out}$ and $w \colon \mathsf{states} \to \mathsf{states}$, generated from the arrows pictured below, so that $w' \circ w$ is defined

as the concatenation ww'.



DETERMINISTIC AUTOMATA

A deterministic automaton is obtained by instantiating $\mathcal{C}$ to $\mathtt{Set}$ and considering functors that map in to 1 and out to 2.

VECTOR SPACE AUTOMATA

A vector space automaton is obtained by instantiating $\mathcal{C}$ to $\mathtt{Set}$ and considering functors that map both in and out to $\mathbb{K}$.

In this approach, a language on words can be seen as a functor $L\colon \mathcal{O} \to \mathcal{C}$ from the full subcategory $\mathcal{O}$ of $\mathcal{I}$ on objects in and out

$$\mathsf{in} \xrightarrow{\triangleright w \triangleleft} \mathsf{out}\,,$$

the arrows of which are $\triangleright w \triangleleft\colon \mathsf{in} \to \mathsf{out}$ for all $w \in A^*$. We denote by $\iota\colon \mathcal{O} \to \mathcal{I}$ the inclusion of the category $\mathcal{O}$ in $\mathcal{I}$. An automaton $\mathcal{A}$ accepts the language L if

$$\mathcal{A} \circ \iota = L$$

DETERMINISTIC AUTOMATA

A language accepted by a deterministic automaton is a functor

$$L\colon \mathcal{O} \to \mathtt{Set}$$

mapping in to 1 and out to 2.

VECTOR SPACE AUTOMATA

A language accepted by a vector space automaton is a functor

$$L\colon \mathcal{O} \to \mathtt{Vec}$$

mapping both in and out to $\mathbb{K}$.

It is easy to see that in these cases, we retrieve the running examples discussed in Section 2. If the category $\mathcal{C}$ has countable products and coproducts, then the existence of the initial and final automaton accepting a given language can also be explained in terms of more generic category-theoretic constructions (left and right Kan extensions). In the process of writing this paper, we discovered that a similar approach based on Kan extensions, was considered in an old (and seemingly forgotten) paper of Bainbridge [Bainbridge 1974].

In this framework, we can obtain new automata models by varying the input and the output categories, $\mathcal{I}$, respectively $\mathcal{O}$. For example, the hybrid-set-vector automata of Section 4 are obtained by instantiating $\mathcal{C}$ with $\mathtt{Glue}(\mathtt{Vec})$ and $\mathtt{Glue}_{\mathsf{fin}}(\mathtt{Vec}_{\mathsf{fin}})$.

Syntactic algebras. In a recent paper [Bojańczyk 2015], Bojańczyk considered languages recognised by monads and described syntactic algebras in this setting. By tuning the input category $\mathcal{I}$ so that the algebraic structure at issue is hard-wired in the morphisms of $\mathcal{I}$, we obtain a unifying view of syntactic algebras and minimization.

Tree automata. For handling tree automata, it is necessary to possess a way to describe 'maps' of several arguments. Of course in the category $\mathtt{Set}$ of sets we can just use the cartesian product. But in $\mathtt{Vec}$ one has to use the tensor product instead. More generally, the right setting for this is to use monoidal categories. These are categories equipped with a 'bifunctor' $\otimes$ called the *tensor product* that satisfies sufficiently many properties for allowing to aggregate several objects into one in a 'category-meaningful

way'. The results of minimization as described in this column can be mimicked in this generalised context. In particular, hybrid-set-vector automata extend to this tree automata.

Enriched forms of automata. Another extension that is important to consider is when the alphabet also possesses some structure. For instance, one could imagine that the alphabet is infinite and computations are meant to be permutation-invariant, as in nominal automata [Bojańczyk et al. 2014]. Again, we can choose the category $\mathcal{I}$ so that the additional structure (e.g. permutation-invariance) is captured by its structure. In this way one can retrieve minimization results for the resulting automata model.

REFERENCES

Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. 2012. A Coalgebraic Perspective on Minimization and Determinization. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'12)*. Springer-Verlag, Berlin, Heidelberg, 58–73. DOI:http://dx.doi.org/10.1007/978-3-642-28729-9_4

Jiří (ing) Adámek, Horst Herrlich, and George E. Strecker. 1990. *Abstract and concrete categories : the joy of cats*. Wiley, New York. http://opac.inria.fr/record=b1087598 A Wiley-Interscience publication.

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

Michael A. Arbib and Ernest G. Manes. 1974a. Basic concepts of category theory applicable to computation and control. In *Category Theory Applied to Computation and Control (Lecture Notes in Computer Science)*, Vol. 25. Springer, 1–34.

Michael A. Arbib and Ernest G. Manes. 1974b. A categorist's view of automata and systems. In *Category Theory Applied to Computation and Control (Lecture Notes in Computer Science)*, Vol. 25. Springer, 51–64.

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

Michael A. Arbib and Ernest G. Manes. 1980. Machines in a category. *Journal of Pure and Applied Algebra* 19 (1980), 9 – 20. DOI:http://dx.doi.org/10.1016/0022-4049(80)90090-0

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

Nicolas Bedon. 1996. Finite automata and ordinals. (1996), 119–144.

Nicolas Bedon, Alexis Bès, Olivier Carton, and Chloe Rispal. 2010. Logic and Rational Languages of Words Indexed by Linear Orderings. *Theory Comput. Syst.* 46, 4 (2010), 737–760. DOI:http://dx.doi.org/10.1007/s00224-009-9222-6

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

Mikołaj Bojańczyk. 2011. Data Monoids. In *STACS 2011: 28th International Symposium on Theoretical Aspects of Computer Science (LIPIcs)*, Thomas Schwentick and Christoph Dürr (Eds.), Vol. 9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 105–116.

Mikołaj Bojańczyk. 2015. Recognisable Languages over Monads. In *DLT (Lecture Notes in Computer Science)*, Vol. 9168. Springer, 1–13.

Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. 2014. Automata theory in nominal sets. *Logical Methods in Computer Science* 10, 3 (2014).

Mikołaj Bojańczyk and Igor Walukiewicz. 2008. Forest algebras. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*. 107–132.

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

Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. 2012. Brzozowski's Algorithm (Co)Algebraically. In *Logic and Program Semantics (Lecture Notes in Computer Science)*, Vol. 7230. Springer, 12–23.

Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence. In *POPL*. ACM, 457–468.

Walter S. Brainerd. 1968. The Minimalization of Tree Automata. *Information and Control* 13, 5 (1968), 484–491. DOI:http://dx.doi.org/10.1016/S0019-9958(68)90917-0

Olivier Carton, Thomas Colcombet, and Gabriele Puppis. 2011. Regular Languages of Words over Countable Linear Orderings. In *ICALP 2011 (2): Automata, Languages and Programming - 38th International Colloquium*, Luca Aceto, Monika Henzinger, and Jiri Sgall (Eds.), Vol. 6756. 125–136.

Christian Choffrut. 1979. A Generalization of Ginsburg and Rose's Characterization of G-S-M Mappings. In *Automata, Languages and Programming, 6th Colloquium, Graz, Austria, July 16-20, 1979, Proceedings (Lecture Notes in Computer Science)*, Hermann A. Maurer (Ed.), Vol. 71. Springer, 88–103. DOI:http://dx.doi.org/10.1007/3-540-09510-1_8

Christian Choffrut. 2003. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.* 292, 1 (2003), 131–143. DOI:http://dx.doi.org/10.1016/S0304-3975(01)00219-5

Thomas Colcombet. 2009. The theory of stabilisation monoids and regular cost functions. In *ICALP 2009 (2): Automata, Languages and Programming, 36th International Colloquium*, Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas (Eds.), Vol. 5556. 139–150.

Thomas Colcombet. 2013. Regular cost functions, Part I: logic and algebra over words. 9, 3 (2013), 47.

Thomas Colcombet and Sreejith A. V. 2015. Limited Set Quantifiers over Countable Linear Orders. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015 (Lecture Notes in Computer Science)*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.), Vol. 9135. Springer, 146–158. DOI:http://dx.doi.org/10.1007/978-3-662-47666-6_12

Samuel Eilenberg. 1974. *Automata, Languages, and Machines*. Vol. Volume A. Academic Press, Inc., Orlando, FL, USA.

Samuel Eilenberg. 1976. *Automata, Languages, and Machines*. Vol. Volume B. Academic Press, Inc., Orlando, FL, USA.

Samuel Eilenberg and Jesse B. Wright. 1967. Automata in general algebras. *Information and Control* 11, 4 (1967), 452 – 470. DOI:http://dx.doi.org/10.1016/S0019-9958(67)90670-5

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

Bart Jacobs and Jan Rutten. 1997. A Tutorial on (Co)Algebras and (Co)Induction. *EATCS Bulletin* 62 (1997), 62–222.

R. E. Kalman. 1963. Mathematical Description of Linear Dynamical Systems. *Journal of the Society for Industrial and Applied Mathematics Series A Control* 1, 2 (Jan. 1963), 152–192. DOI:http://dx.doi.org/10.1137/0301010

Denis Kuperberg. 2011. Linear temporal logic for regular cost functions. In *STACS 2011: 28th International Symposium on Theoretical Aspects of Computer Science (LIPIcs)*, Thomas Schwentick and Christoph Dürr (Eds.), Vol. 9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 627–636.

Dominique Perrin and Jean-Éric Pin. 1995. Semigroups and automata on infinite words. In *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*, J. Fountain (Ed.). Kluwer academic publishers, 49–72.

Libor Polák. 2001. Syntactic Semiring of a Language. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*. 611–620. DOI:http://dx.doi.org/10.1007/3-540-44683-4_53

Michael O. Rabin and Dana Scott. 1959. Finite automata and their decision problems. *IBM J. Res. and Develop.* 3 (April 1959), 114–125.

Jurriaan Rot. 2016. Coalgebraic Minimization of Automata by Initiality and Finality. *Electronic Notes in Theoretical Computer Science* 325 (2016), 253 – 276. DOI:http://dx.doi.org/10.1016/j.entcs.2016.09.042

J.J.M.M. Rutten. 2000. Universal coalgebra: a theory of systems. *Theoretical Computer Science* 249, 1 (2000), 3 – 80. DOI:http://dx.doi.org/10.1016/S0304-3975(00)00056-6

M.P. Schützenberger. 1961. On the definition of a family of automata. *Information and Control* 4, 2 (1961), 245 – 270. DOI:http://dx.doi.org/10.1016/S0019-9958(61)80020-X

Marcel-Paul Schützenberger. 1965. On finite monoids having only trivial subgroups. *Information and Control* 8 (1965), 190–194.

Alexandra Silva. 2015. A Short Introduction to the Coalgebraic Method. *ACM SIGLOG News* 2, 2 (April 2015), 16–27. DOI:http://dx.doi.org/10.1145/2766189.2766193