Learning Automata and Transducers: A Categorical Approach

Thomas Colcombet
IRIF, CNRS, Paris, France
https://www.irif.fr/~colcombe/
thomas.colcombet@irif.fr

Daniela Petrişan
IRIF, Université de Paris, France
https://www.irif.fr/~petrisan/
daniela.petrisan@irif.fr

Riccardo Stabile
Università degli Studi di Milano, Dipartimento di Matematica, Italy
riccardo.stabile@yahoo.com

Abstract

In this paper, we present a categorical approach to learning automata over words, in the sense of the L*-algorithm of Angluin. This yields a new generic L*-like algorithm which can be instantiated for learning deterministic automata, automata weighted over fields, as well as subsequential transducers. The generic nature of our algorithm is obtained by adopting an approach in which automata are simply functors from a particular category representing words to a "computation category". We establish that the sufficient properties for yielding the existence of minimal automata (that were disclosed in a previous paper), in combination with some additional hypotheses relative to termination, ensure the correctness of our generic algorithm.

2012 ACM Subject Classification Theory of computation → Algebraic language theory; Theory of computation → Transducers

Keywords and phrases Automata, transducer, learning, category

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.15

Related Version A full version of the paper is available at [14], https://arxiv.org/abs/2010.13675.

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).
Daniela Petrişan: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).
Riccardo Stabile: Supported by the European Commission under the Erasmus+ programme for a five-month study period at Université de Paris.

1 Introduction

Learning automata is a classical subject at the intersection of machine learning and automata theory. It has found a wide range of applications spanning from adaptive model checking, compositional verification to learning network invariants or interface specifications for Java classes. We refer the reader to [18] and the references therein for a survey of such applications.

The most famous learning algorithm for automata is certainly the L*-*algorithm* of Angluin [1]. Its goal is to learn a regular language of words L. For this, the algorithm interacts with a *teacher* (an oracle) who knows L by asking two kinds of queries:

Membership query it can ask whether a given word belongs to L, or

Equivalence query it can provide a *hypothesis automaton* and ask the teacher whether this automaton recognizes L or not. If the answer is no, the teacher is bound to provide a *counter-example word*, witnessing the non-equivalence.

The algorithm stops when the teacher agrees that the hypothesis automaton recognizes the language L. A key property of the $\mathtt{L}^*$-algorithm is that it terminates in time polynomial in the size of the alphabet, of the minimal deterministic automaton for L, and of the longest counter-example. Furthermore, all candidate automata appearing during its execution (and hence in particular the final one) are deterministic, complete, and minimal.

The $\mathtt{L}^*$-algorithm

Let us illustrate the behaviour of this algorithm when it tries to learn the language $\{a\}$ over the alphabet $\Sigma = \{a\}$. At each step, the algorithm maintains two sets of words Q, T, starting with $Q = \{\varepsilon\}$, $T = \{\varepsilon\}$. One can understand Q as a set of words which identify states of the hypothesis automaton under construction. The set T is used in order to discover if words need to be distinguished by the automaton: two words $u, v \in \Sigma^*$ are *T-equivalent* if for all $t \in T$, $ut \in L$ if and only if $vt \in L$.

At the beginning, the algorithm attempts to construct an automaton with as sole state $\varepsilon \in Q$, which has to be initial. In particular, the target of the transition labelled a issued from ε has to be determined. Such a transition should go to a state in Q which has to be T-equivalent to $\varepsilon a = a$. It fails since there are no such states in Q (we say that the pair Q, T fails to have the *closedness property*). The algorithm corrects it by adding the word a to Q. We reach $Q = \{\varepsilon, a\}$, $T = \{\varepsilon\}$. The algorithm now tries to construct an automaton with states $Q = \{\varepsilon, a\}$: this time, it is possible to construct an a-labelled transition from $\varepsilon \in Q$ to $a \in Q$. What should now be the a-labelled transition issued from a? It should be some state $q \in Q$ which is T-equivalent to aa. Luckily, there is one, namely ε. Hence, we succeed in constructing the left hypothesis automaton in Figure 1. The algorithm now

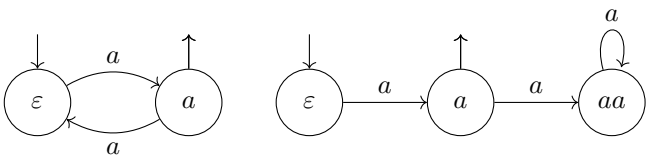


Figure 1 Two successive hypothesis automata.

queries for equivalence of the language of this automaton with the language. This fails since $a(aa)^* \neq L = \{a\}$, and hence the teacher answers in return a counter-example word, say aaa. The algorithm then adds (for reasons that are not detailed here) the prefix aa of aaa to Q, yielding $Q = \{\varepsilon, a, aa\}$, $T = \{\varepsilon\}$. Here, ε and aa are T-equivalent, but constructing an a-labeled transition from ε would yield a, while constructing one from aa would yield aaa, which is T-equivalent to ε. Hence, ε and aa cannot be merged as a same state (we say that Q, T fails to have the *consistency property*). The algorithm compensates it by adding a to T, thus yielding $Q = \{\varepsilon, a, aa\}$ and $T = \{\varepsilon, a\}$. Now, Q, T are both closed and consistent, and the right hypothesis automaton in Figure 1 is constructed. It recognizes $\{a\}$, and thus the teacher agrees and the algorithm terminates. It has constructed the minimal deterministic and complete automaton for the language $L = \{a\}$.

This example witnesses the different steps involved in the algorithm: (a) if (Q, T) is not closed, a word is added to Q, (b) if (Q, T) is not consistent, a word is added to T, and (c) when (Q, T) is both closed and consistent, it is possible to construct a hypothesis automaton

and perform an equivalence query: if this automaton happens to not accept the expected language, the teacher provides a counter-example word from which words to add to Q are constructed. The algorithm functions by performing the operation until the teacher agrees.

The correctness of the algorithm bears many resemblances with the question of minimizing deterministic automata. This can be witnessed in the fact that the $\mathtt{L}^*$-algorithm automatically constructs minimal deterministic and complete automata. It can also be witnessed in the fact that the T-equivalences induce along the run finer and finer partitions of the words that converge eventually to the Myhill-Nerode equivalence, another notion highly connected to minimization questions.

The $\mathtt{L}^*$-algorithm turns out to be extremely robust, and has been extended to various other forms of automata (weighted automata over fields [5, 6], nominal automata [19], omega automata [3], non-deterministic automata [9], alternating automata [2], symbolic automata [15], subsequential transducers [24, 25], transducers of trees [7, 8]). Although with a focus on concrete implementations, Bollig et al. [10] emphasize that "the need for a unifying framework collecting various types of learning techniques is, thus, beyond all questions."

Contributions

The aim of this paper is to present such a unifying framework for learning word automata using the toolkit of category theory. Concretely, we provide an abstract categorical version of Angluin's $\mathtt{L}^*$-algorithm, called $\mathtt{FunL}^*$ (Algorithm 1), we prove its correctness and termination (Theorem 26), and we give three running instantiations for it, namely in the case of deterministic automata, field weighted automata and subsequential transducers.

To this end, we reuse the framework developed in [13] which models automata as functors from an input category $\mathcal{I}$ (describing the structure of the computation) to an output category $\mathcal{C}$. For example, to model word automata, the input category is a fixed three-object category $\mathcal{I}_{A^*}$, that we will recall in Section 2. By varying the category $\mathcal{C}$, this definition captures several forms of automata, and in particular the ones mentioned above. In [13], we present sufficient conditions on $\mathcal{C}$ that guarantee the existence of minimal automata. These conditions are quite mild: $\mathcal{C}$ should have certain products and coproducts, on one hand, and a factorization system, on the other. Apart from these three conditions on the output category, Theorem 26 – which states that our new algorithm computes the minimal automaton for the language to be learned – requires only one additional assumption which ensures termination, namely a 'finiteness' hypothesis (using the notion of noetherianity).

In order to describe our generic $\mathtt{FunL}^*$ algorithm we provide abstract versions of the steps of the $\mathtt{L}^*$-algorithm described above. These are obtained as follows:

- We describe the pair of sets of words (Q, T) using a four-object category $\mathcal{I}_{Q,T}$, introduced in Definition 15. This category is a modification of $\mathcal{I}_{A^*}$, which allows us to obtain a partial view of the language, namely only its values on words of the form qt and qat with $q \in Q, t \in T$ and a a letter in the alphabet.
- Computing the approximations of the Myhill-Nerode equivalence (that is, the T-equivalence relations) roughly corresponds in our generic setting to performing a minimization-like computation. This is achieved using off-the-shelf results from [13] by changing the input category from $\mathcal{I}_{A^*}$ to $\mathcal{I}_{Q,T}$. We obtain a form of minimal "biautomaton" featuring an ε-transition between its two state objects.
- The pair (Q, T) being closed and consistent amounts to the ε-transition of the above biautomaton being an isomorphism between the two state objects. We then say that the pair (Q, T) is $\mathcal{L}$-automatable. Under this assumption, it is meaningful to collapse the two state objects, defining in this way the hypothesis automaton (represented now as a functor $\mathcal{I}_{A^*} \to \mathcal{C}$).

What is interesting about our FunL*-algorithm – compared to previous approaches – is that it highlights the strong relationship between learning and minimizing automata. Each elementary step of the algorithm involves performing a minimization-like computation and leverages the modularity of our previous work [13], this time by varying the input category. In contrast to other category theoretic approaches to learning, FunL* does not rely neither on algebras nor on coalgebras. Instead, we exploit the symmetry of the word automata model. This is reflected by the self-duality of the input category $\mathcal{I}_{A^*}$, which is underpinning the well known duality between observability and reachability.

Finally, a prominent instantiation of the FunL*-algorithm is Vilar's learning algorithm of subsequential transducers [24]. Our notion of $\mathcal{L}$-automatable pairs (Q, T) perfectly instantiates to the conditions considered by Vilar to construct a hypothesis transducer. A coalgebraic modelisation of subsequential transducers was provided in [16], but, to the best of our knowledge, this example is not featured in the category theoretic learning literature.

Related works

We briefly review the (co)algebraic approaches to automata learning that have been proposed in recent years. The paper [17] was the first to recast key ingredients of Angluin's algorithm in a coalgebraic setting. This line of work was continued with the CALF framework of van Heerdt et.al [22], which models automata as triples consisting of an algebra for a functor, an initial map and an output map. In [22, Section 5] a connection between minimization and learning is mentioned and formalized for DFAs. More precisely, the main theorem proving the correctness of the learning algorithm [22, Theorem 16] can be used to show the correctness of the minimization algorithm for DFA, with reachability and observability playing a crucial role. The same authors proposed in [23] a learning algorithm for automata with side-effects. These are extensions of DFAs to automata interpreted in a category of Eilenberg-Moore algebras for a Set monad T – used to represent a certain side effect. For example, the finite powerset monad corresponds to non-determinism and the ensuing automata model serves to represent non-deterministic automata. In order to prove the termination of the learning algorithm, the monad T above is assumed to preserve finite sets. Hence the monad that we use in the present work to model subsequential transducers does not fit in the scope of [23]. Another small difference is that we work within the Kleisli category.

Another category theoretic learning algorithm was proposed in [4] and provides a coalgebraic and duality theoretic foundation for learning bisimilarity quotients of state-based transition systems. The core idea is to use logical formulas as tests, taking stock of dual adjunctions between states and logical theories, formalized as algebra-coalgebra dualities.

The recent paper [21] gives a learning algorithm for automata whose transitions can be encoded both as algebras for a functor F on a category $\mathcal{C}$, and as coalgebras for the right adjoint of F (assumed to exist). The approximations of the Myhill-Nerode equivalence present in the learning algorithm are computed using factorizations of morphisms from approximations of an initial algebra (obtained using an initial chain) to approximations of a final coalgebra (obtained using a final co-chain). Some of the ingredients of this category theoretic algorithm are similar to ours, e.g. the heavy use of factorization systems or the notion of "finite" object in a category, however, there are more assumptions on the underlying category and on the preservation properties of the adjoint functors considered in the (co)-algebraic definition of the automata, see [21, Assumption 3.5] and [21, Assumption 4.1].

Structure of the paper

In Section 2, we present necessary material from [13], which includes in particular the categorical modeling of automata, its instantiation for deterministic finite automata, for field weighted automata and for subsequential transducers, and how to minimize them. This material is key in our description of the algorithm in Section 3. For simplicity, we describe first a slightly simplified version of the algorithm. The optimized version is the presented in [14, Appendix C]. Section 4 concludes.

2 Minimization

In this section, we recall the categorical approach to automata minimization from [12, 13].

2.1 Languages and automata as functors

We first recall the notion of automata as functors. We consider an arbitrary small category $\mathcal{I}$, called the *input category*, and one of its full subcategories $\mathcal{O}$, denoting by i the inclusion functor: $\mathcal{O} \xhookrightarrow{i} \mathcal{I}$. Intuitively, $\mathcal{I}$ represents the the inner computations performed by an automaton, and in particular its internal behaviour, while $\mathcal{O}$ represents the observable behaviour of the automaton and is used to define the language it accepts.

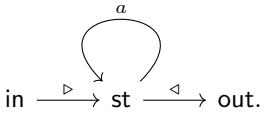
We consider another category $\mathcal{C}$, called the *output category*, which models the output computed by the automaton (e.g., a boolean value, probabilities, words over an alphabet).

▶ **Definition 1.** *A* $\mathcal{C}$-automaton *(or simply an automaton)* $\mathcal{A}$ *is a functor from* $\mathcal{I}$ *to* $\mathcal{C}$*. A* $\mathcal{C}$-language *(or simply a language)* $\mathcal{L}$ *is a functor from* $\mathcal{O}$ *to* $\mathcal{C}$*. A* $\mathcal{C}$*-automaton* $\mathcal{A}$ accepts *a* $\mathcal{C}$*-language* $\mathcal{L}$ *if* $\mathcal{A} \circ i = \mathcal{L}$*.*

We denote by $\mathsf{Auto}(\mathcal{L})$ *the subcategory of the functor category* $[\mathcal{I}, \mathcal{C}]$*:*

- *whose objects are all* $\mathcal{C}$*-automata* $\mathcal{A}$ *accepting* $\mathcal{L}$*;*
- *whose arrows are* $\mathcal{C}$-automata morphisms*, meaning natural transformations* $\alpha \colon \mathcal{A}_1 \Rightarrow \mathcal{A}_2$ *such that* $\alpha \circ id_i = id_{\mathcal{L}}$*.*

In this paper, we will instantiate the input category $\mathcal{I}$ in two ways. The first one, $\mathcal{I}_{A^*}$, is used in [12] to model different forms of *word automata*; we describe it in this section and use it for modeling the three running instantiations. In Section 3, we will consider another input category, $\mathcal{I}_{Q,T}$, which we use in the process of constructing our hypothesis automata.



We define now the input category $\mathcal{I}_{A^*}$ used for describing word automata. Here A is a finite alphabet, fixed for the rest of the paper, and A^* the set of words over it. The input category $\mathcal{I}_{A^*}$ is the category freely generated by the graph on the right, where a ranges over A: That is, $\mathcal{I}_{A^*}$ is the three-object category with arrows spanned by $\triangleright$, $\triangleleft$ and a for all $a \in A$, so that the composition $\mathsf{st} \xrightarrow{w} \mathsf{st} \xrightarrow{w'} \mathsf{st}$ is given by the concatenation ww'. So, for example, the morphisms in $\mathcal{I}_{A^*}$ from in to st are of the form $\triangleright w$ with $w \in A^*$, while the morphisms on the object st are of the form w with $w \in A^*$.

Let $\mathcal{O}_{A^*}$ denote the full subcategory of $\mathcal{I}_{A^*}$ on the objects in and out. Its morphisms are of the form $\mathsf{in} \xrightarrow{\triangleright w \triangleleft} \mathsf{out}$ for $w \in A^*$.

Hereafter, by a language we mean a functor from $\mathcal{O}_{A^*}$ to $\mathcal{C}$ and by an automaton we mean a functor from $\mathcal{I}_{A^*}$ to $\mathcal{C}$. If $\mathcal{L}(\mathsf{in}) = X$ and $\mathcal{L}(\mathsf{out}) = Y$, a language $\mathcal{L}$ will be referred to as a $(\mathcal{C}, X, Y)$-*language*; if $\mathcal{A}(\mathsf{in}) = X$ and $\mathcal{A}(\mathsf{out}) = Y$, an automaton $\mathcal{A}$ will be called a $(\mathcal{C}, X, Y)$-*automaton*. We provide three *running instantiations* of the output category $\mathcal{C}$ and of the objects X and Y, in order to model deterministic automata, field weighted automata and subsequential transducers.

▶ **Example 2** (Deterministic automata)**.** A deterministic and complete automaton is a $(\mathsf{Set}, 1, 2)$-automaton. Indeed, we can see a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Set}$ with $\mathcal{A}(\mathsf{in}) = 1$ and $\mathcal{A}(\mathsf{out}) = 2$ as a deterministic automaton by interpreting
- $\mathcal{A}(\mathsf{st})$ as its set of states,
- $\mathcal{A}(\triangleright)\colon 1 \to \mathcal{A}(\mathsf{st})$ as choosing the initial state,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \to \mathcal{A}(\mathsf{st})$ as the transition map for the letter $a \in A$,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \to 2$ as the characteristic map of the subset of accepting states.

▶ **Example 3** (Weighted automata over a field)**.** Let $\mathbb{K}$ be a field and let Vec denote the corresponding category of $\mathbb{K}$-vector spaces and linear transformations. A *weighted automaton over the field* $\mathbb{K}$ (in the sense of [20]) is a $(\mathsf{Vec}, \mathbb{K}, \mathbb{K})$-automaton. Indeed, a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Vec}$ with $\mathcal{A}(\mathsf{in}) = \mathbb{K}$ and $\mathcal{A}(\mathsf{out}) = \mathbb{K}$ is seen as a weighted automaton over $\mathbb{K}$ by interpreting
- $\mathcal{A}(\mathsf{st})$ as the vector space spanned by its states,
- $\mathcal{A}(\triangleright)\colon \mathbb{K} \to \mathcal{A}(\mathsf{st})$ as the linear transformation mapping the unit of $\mathbb{K}$ to the initial vector,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \to \mathcal{A}(\mathsf{st})$ as the linear transformation transition for the letter $a \in A$,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \to \mathbb{K}$ as the output linear transformation.

▶ **Example 4** (Subsequential transducers)**.** The aim is to represent what we call *transductions* in this paper, which are partial maps from A^* to B^*, where B is some fixed output alphabet. Roughly, a subsequential transducer [11] is a deterministic automaton which, at each step, while reading an input letter from A, either has no transition or has a unique transition which changes deterministically the state and outputs a word from B^*. In this paper, we define *subsequential transducers* as $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata [13], for a definition of $\mathsf{Kl}(\mathcal{T})$ that we give now.

The **output category** $\mathsf{Kl}(\mathcal{T})$**.** Let $\mathcal{T}$ be the monad defined by $\mathcal{T}X = B^* \times X + 1$ and let $\mathsf{Kl}(\mathcal{T})$ denote the Kleisli category for $\mathcal{T}$. Concretely, the objects of $\mathsf{Kl}(\mathcal{T})$ are sets, while its morphisms, denoted by negated arrows, are of the form $f\colon X \nrightarrow Y$ for a function $f\colon X \to \mathcal{T}Y$, that is, a partial function from X to $B^* \times Y$. We write $\perp$ for the element of the singleton 1 and think of it as the *undefined* element. Given $f\colon X \nrightarrow Y$ and $g\colon Y \nrightarrow Z$, their composite $g \circ f\colon X \nrightarrow Z$ is defined on $x \in X$ by (uv, z), when $f(x) = (u, y) \in B^* \times Y$ and $g(y) = (v, z) \in B^* \times Z$ (with uv denoting the concatenation of u and v in B^*) and $f(x) = \perp$ in all other cases. Note that with this definition, a transduction can be identified in an obvious manner with a map from A^* to arrows of the form $1 \nrightarrow 1$.

We now recall that $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata are equivalent to *subsequential transducers* in the sense of Choffrut's definition [11]. Indeed, we can see a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Kl}(\mathcal{T})$ with $\mathcal{A}(\mathsf{in}) = 1$ and $\mathcal{A}(\mathsf{out}) = 1$ as a subsequential transducer by interpreting
- $\mathcal{A}(\mathsf{st})$ as the set of states,
- $\mathcal{A}(\triangleright)\colon 1 \nrightarrow \mathcal{A}(\mathsf{st})$ as either choosing an initial state together with an initial output in B^* or having an undefined initial state,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \nrightarrow \mathcal{A}(\mathsf{st})$ as the transition map for the letter a which associates to a given state either undefined or a pair consisting of an output word in B^* and a successor state,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \nrightarrow 1$ as the final map which associates to a state either its output in B^* or undefined when it is non-accepting.

2.2 Minimization of automata

Now we describe what it means to be minimal in a category (Definition 5) together with an abstract result of existence of such an object (Lemma 6). We then provide sufficient material for our three running instantiations to be covered.

Let $\mathcal{K}$ be a category endowed with a factorization system $(\mathcal{E}, \mathcal{M})$. We write $\twoheadrightarrow$ for arrows belonging to $\mathcal{E}$ and we will call them *$\mathcal{E}$-quotients*; we write $\rightarrowtail$ for arrows belonging to $\mathcal{M}$ and we will call them *$\mathcal{M}$-subobjects*.

▶ **Definition 5.** *Consider two objects X, Y of $\mathcal{K}$. We say that X $(\mathcal{E}, \mathcal{M})$-*divides *$Y$ whenever X is an $\mathcal{E}$-quotient of an $\mathcal{M}$-subobject of Y, that is, we have a span of the form:*

$$X \twoheadleftarrow \cdot \rightarrowtail Y\,.$$

*An object Z in $\mathcal{K}$ is $(\mathcal{E}, \mathcal{M})$-*minimal *if it $(\mathcal{E}, \mathcal{M})$-divides all the objects in $\mathcal{K}$.*

As shown in the following lemma, having an initial and a final object turns out to be a sufficient condition for the minimal object to exist and be unique up to isomorphism.

▶ **Lemma 6.** *Let $\mathcal{K}$ be a category endowed with an initial object I, a final object F and a factorization system $(\mathcal{E}, \mathcal{M})$. Let* Min *be the factorization of the unique arrow from I to F:*

$$I \twoheadrightarrow \mathsf{Min} \rightarrowtail F.$$

Then Min *is $(\mathcal{E}, \mathcal{M})$-minimal.*

We apply this lemma when $\mathcal{K}$ is instantiated with a category of automata $\mathsf{Auto}(\mathcal{L})$.

▶ **Corollary 7.** *If the category $\mathsf{Auto}(\mathcal{L})$ has an initial automaton $\mathcal{A}_{init}(\mathcal{L})$, a final automaton $\mathcal{A}_{final}(\mathcal{L})$ and a factorization system, then the minimal automaton $\mathsf{Min}(\mathcal{L})$ for the language $\mathcal{L}$ is obtained via the following factorization: $\mathcal{A}_{init}(\mathcal{L}) \twoheadrightarrow \mathsf{Min}(\mathcal{L}) \rightarrowtail \mathcal{A}_{final}(\mathcal{L})$.*

Notice that this notion of minimization is parametric in the factorization system. In all our examples, we obtain a suitable factorization system on $\mathsf{Auto}(\mathcal{L})$ from one on $\mathcal{C}$, as follows.

▶ **Lemma 8.** *If a category $\mathcal{C}$ has a factorization system $(\mathcal{E}, \mathcal{M})$, then the category $\mathsf{Auto}(\mathcal{L})$ 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 natural transformations with components in $\mathcal{E}$ and $\mathcal{M}_{\mathsf{Auto}(\mathcal{L})}$ consists of all natural transformations with components in $\mathcal{M}$.*

▶ **Example 9** (factorization systems)**.** There exists a factorization system in our three running examples. For Set, this is the well known factorization system (Surjections, Injections). Similarly in Vec, (Surjective linear maps, Injective linear maps) is a factorization system.

In the case of $\mathsf{Kl}(\mathcal{T})$, the factorization system does not follow from general arguments. We define now the factorization system $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ for $\mathsf{Kl}(\mathcal{T})$. Given a morphism $f\colon X \nrightarrow Y$ in $\mathsf{Kl}(\mathcal{T})$, we write $\pi_1(f)\colon X \to B^* + \{\bot\}$ and $\pi_2(f)\colon X \to Y + \{\bot\}$ for the projections: if $f(x) = \bot$ then $\pi_1(x) = \pi_2(x) = \bot$, otherwise $f(x) = (\pi_1(f)(x), \pi_2(f)(x))$.

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

By [13, Lemma 4.8], $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ is a factorization system.

We specialize the result of Corollary 7 to the case of word automata $\mathcal{I}_{A^*} \to \mathcal{C}$. Due to the special shape of the category $\mathcal{I}_{A^*}$, we can compute the initial and the final automata, provided the output category satisfies some mild assumptions, recalled in Lemmas 10 and 12.

▶ **Lemma 10.** *Fix a language* $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$*. If the category* $\mathcal{C}$ *has countable copowers of* $\mathcal{L}(\mathsf{in})$*, the* initial automaton $\mathcal{A}_{init}(\mathcal{L})$ *exists and is given by the following data:*

- $\mathcal{A}_{init}(\mathcal{L})(\mathsf{st}) = \coprod_{A^*} \mathcal{L}(\mathsf{in})$*;*
- $\mathcal{A}_{init}(\mathcal{L})(\triangleright)\colon \mathcal{L}(\mathsf{in}) \to \coprod_{A^*} \mathcal{L}(\mathsf{in})$ *is given by the coproduct injection corresponding to* ε*, for this reason we will denote this map by* ε*;*
- $\mathcal{A}_{init}(\mathcal{L})(a)\colon \coprod_{A^*} \mathcal{L}(\mathsf{in}) \to \coprod_{A^*} \mathcal{L}(\mathsf{in})$ *is given on the* w*-component* $\mathcal{L}(\mathsf{in})$ *by the coproduct injection corresponding to* wa*;*
- $\mathcal{A}_{init}(\mathcal{L})(\triangleleft)\colon \coprod_{A^*} \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ *is the coproduct of the morphisms* $\mathcal{L}(\triangleright w \triangleleft)\colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ *with* $w \in A^*$*, that is, it computes the value of the language on a given word, for this reason we will also denote this map by* $\mathcal{L}?$*.*

▶ **Example 11.** Since the categories Set, Vec and $\mathsf{Kl}(\mathcal{T})$ have all copowers, the initial automaton for a given language can be easily computed in these cases as an instance of the above lemma. We recall the details for Set and $\mathsf{Kl}(\mathcal{T})$.

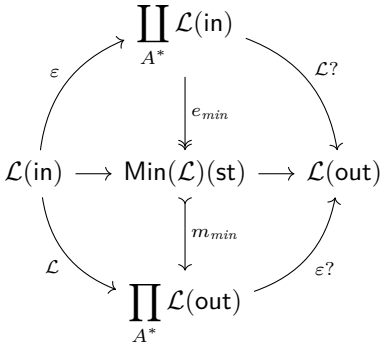
- Given a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathsf{Set}$, the initial deterministic automaton accepting $\mathcal{L}$ is described above in the left diagram. Its state space is the set of all words, with ε being the initial one. A word is accepted if and only if it belongs to the language.
- For a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathsf{Kl}(\mathcal{T})$, the initial subsequential transducer accepting $\mathcal{L}$ is as depicted in the right diagram. Its state space is A^*, the initial state is $\varepsilon \in A^*$ with initial output $\varepsilon \in B^*$. For an input letter $a \in A$, the corresponding transition maps w to wa and produces output $\varepsilon \in B^*$. Finally, the map $\mathcal{L}?$, which is in fact a function from A^* to $B^* + 1$, associates to a word w the value of the language at w, that is, the value computed by $\mathcal{L}(\triangleright w \triangleleft)$.

▶ **Lemma 12.** *Fix a language* $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$*. If the category* $\mathcal{C}$ *has countable powers of* $\mathcal{L}(\mathsf{out})$*, the* final automaton $\mathcal{A}_{final}(\mathcal{L})$ *exists and is given by the following data:*

- $\mathcal{A}_{final}(\mathcal{L})(\mathsf{st}) = \prod_{A^*} \mathcal{L}(\mathsf{out})$*;*
- $\mathcal{A}_{final}(\mathcal{L})(\triangleright)\colon \mathcal{L}(\mathsf{in}) \to \prod_{A^*} \mathcal{L}(\mathsf{out})$ *is the product of the morphisms* $\mathcal{L}(\triangleright w \triangleleft)\colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ *with* $w \in A^*$*, for this reason we will also denote this map by* $\mathcal{L}$*;*
- $\mathcal{A}_{final}(\mathcal{L})(a)\colon \prod_{A^*} \mathcal{L}(\mathsf{out}) \to \prod_{A^*} \mathcal{L}(\mathsf{out})$ *is the product over* $w \in A^*$ *of the* aw*-projections* $\prod_{A^*} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$*;*
- $\mathcal{A}_{final}(\mathcal{L})(\triangleleft)\colon \coprod_{A^*} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$ *is given by the* ε*-projection, for this reason we will also denote this map by* $\varepsilon?$*.*

▶ **Example 13** (the final automata in Set, Vec and $\mathsf{Kl}(\mathcal{T})$)**.** Since the categories Set and Vec have all products, the final automaton for a given language can be computed using Lemma 12. We illustrate this for Set and $\mathsf{Kl}(\mathcal{T})$.

- Given a language $\mathcal{L} \colon \mathcal{O}_{A^*} \to \mathsf{Set}$, the final deterministic automaton accepting $\mathcal{L}$ is described above in the left diagram. Its state space is the set of all languages over the alphabet A. The initial state is the language $\mathcal{L}$ itself. A language is an accepting state if and only if it contains ε. Given a language K, while reading letter a, the automaton goes to the residual language $a^{-1}K = \{u \in K \mid au \in K\}$.
- Somewhat suprisingly, $\mathsf{Kl}(\mathcal{T})$-automata also fit in the scope of Lemma 12, as we can prove that the object $\mathsf{Irr}(A^*, B^*)$ (which we will define next) is the power of A^*-many copies of 1 in $\mathsf{Kl}(\mathcal{T})$. Define first, given a transduction K, $\mathsf{lcp}(K)$ to be undefined if K is nowhere defined, and the longest common prefix of the words in $\{K(u) \mid u \in A^*\}$ otherwise. A transduction K is *irreducible* if $\mathsf{lcp}(K) = \varepsilon$. We denote by $\mathsf{Irr}(A^*, B^*)$ the set of irreducible transductions. For all K not nowhere defined, we put $\mathsf{red}(K)$ to be the only irreducible transduction such that $K(u) = \mathsf{lcp}(K)\mathsf{red}(K)(u)$, i.e. the transduction in which the prefix $\mathsf{lcp}(K)$ has been stripped away from all outputs. For K nowhere defined, let $\mathsf{red}(K)$ be also nowhere defined.
  We can describe now the final automaton for a transduction $\mathcal{L}$ as an automaton that has irreducible transductions as states. The initial map is the constant map equal to $(\mathsf{lcp}(\mathcal{L}), \mathsf{red}(\mathcal{L}))$ (or undefined if $\mathcal{L}$ is nowhere defined). When reading the letter a from state K, the automaton jumps to $\mathsf{red}(K(a-))$ in which $K(a-)$ is such that $K(a-)(u) = K(au)$ (or undefined if $K(a-)$ is nowhere defined). The final map sends an irreducible transduction to $K(\varepsilon)$.



Combining Lemmas 8, 10 and 12 with Corollary 7, we obtain the following result.

▶ **Theorem 14.** *Let $\mathcal{C}$ be a category with a factorization system $(\mathcal{E}, \mathcal{M})$ and let $\mathcal{L} \colon \mathcal{O}_{A^*} \to \mathcal{C}$ be a language. Suppose $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathsf{in})$ and all countable powers of $\mathcal{L}(\mathsf{out})$. The minimal $\mathcal{C}$-automaton $\mathsf{Min}(\mathcal{L})$ accepting $\mathcal{L}$ is obtained via the factorization in the commuting diagram to the right.*

3 The basic FunL* algorithm

In this section, we provide our generic FunL*-algorithm for learning word automata. Just as in Angluin's algorithm, there are a teacher and a learner. Throughout this section we fix the alphabet A, the output category $\mathcal{C}$ and its factorization systems $(\mathcal{E}, \mathcal{M})$, all known to both

teacher and learner. The *teacher* knows a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$, hereafter called the *target language*. The learner wants to find this language, the output of the algorithm being the minimal automaton $\mathsf{Min}(\mathcal{L})$ accepting $\mathcal{L}$. The learner can ask two kinds of queries, which can be thought as high-level generalizations of Angluin's original ones in the special case of deterministic automata (see [1]).

- *Evaluation queries*: given a certain word w, what is $\mathcal{L}(\triangleright w \triangleleft)$?
- *Equivalence queries*: does a certain automaton accept the target language? If it does not, what is a counterexample for it not doing that?

Let $\mathcal{A}$ be an automaton which is incorrect, that is, such that $\mathcal{A} \circ i \neq \mathcal{L}$, $\mathcal{L}$ being the target language; a word w is said to be a *counterexample* if $\mathcal{A} \circ i(\triangleright w \triangleleft) \neq \mathcal{L}(\triangleright w \triangleleft)$. In other words, a counterexample witnesses the incorrectness of a certain automaton proposed by the learner.

In order to formulate the generic algorithm, we still need to generalize the notions of table and hypothesis automaton from Angluin's original algorithm. We do this in Section 3.1. We provide the generic algorithm and prove its correctness and termination in Section 3.2.

3.1 Hypothesis automata

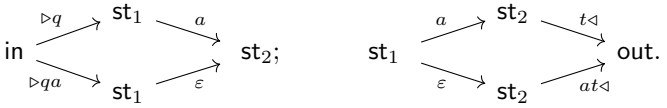
Just as in Angluin's $\mathtt{L}^*$-algorithm, the learner keeps in memory a pair (Q, T) of subsets of A^* such that Q is prefix-closed, i.e. it contains the prefixes of all its elements, while T is suffix-closed, the same for the suffixes; in particular, $\varepsilon \in Q \cap T$. Using the evaluation queries, the learner produces an approximation of $\mathsf{Min}(\mathcal{L})$, explicitly a hypothesis automaton, to be introduced in Definition 22.

It turns out that the category $\mathsf{Auto}(\mathcal{L})$ does not suffice to capture the whole learning process. At a given stage of the algorithm, the learner has access, via evaluation queries, only to a part of $\mathcal{L}$: specifically, he knows the values of $\mathcal{L}(\triangleright q t \triangleleft)$ and $\mathcal{L}(\triangleright q a t \triangleleft)$, where $q \in Q$, $t \in T$ and $a \in A$. This leads us to consider a restriction of the language $\mathcal{L}$ to a subcategory of $\mathcal{O}_{A^*}$ whose arrows are of the form $\triangleright q t \triangleleft$ or $\triangleright q a t \triangleleft$ as above. To produce a hypothesis automaton consistent with this partial view of $\mathcal{L}$, we would also need to adapt the input category. A first attempt would be to discard some of the arrows of $\mathcal{I}_{A^*}$ from in to st, respectively from st to out. Explicitly, we would like to keep only the arrows of the form $\triangleright q\colon \mathsf{in} \to \mathsf{st}$ for the state words $q \in Q$ and, respectively, $t \triangleleft\colon \mathsf{st} \to \mathsf{out}$ for the test words $t \in T$. However, this is not feasible: we would also need the transition maps $a\colon \mathsf{st} \to \mathsf{st}$, and via composition we would generate, for example, all arrows $\triangleright w\colon \mathsf{in} \to \mathsf{st}$. The solution is to "dissociate" the state object st in $\mathcal{I}_{A^*}$ and consider a four-state input category.

▶ **Definition 15.** *The input category $\mathcal{I}_{Q,T}$ is the free category generated by the graph*

$$\mathsf{in} \xrightarrow{\triangleright q} \mathsf{st}_1 \underset{\varepsilon}{\overset{a}{\rightrightarrows}} \mathsf{st}_2 \xrightarrow{t \triangleleft} \mathsf{out}$$

for all $q \in Q$, $a \in A$, $t \in T$ and with ε a fixed symbol (informally representing the empty word) such that the following coherence diagrams *commute for all $a \in A$, for all $q \in Q$ such that $qa \in Q$, and for all $t \in T$ such that $at \in T$:*



Furthermore, let $\mathcal{O}_{Q,T}$ denote the full subcategory of $\mathcal{I}_{Q,T}$ on the objects in *and* out.

The two coherence diagrams in the definition of $\mathcal{I}_{Q,T}$, as well as the prefix-closure of Q and the suffix-closure of T, ensure that we have a functor

$$i^*\colon \mathcal{I}_{Q,T} \to \mathcal{I}_{A^*}$$

which merges st_1 and st_2 sending both of them to st, maps $\varepsilon\colon \mathsf{st}_1 \to \mathsf{st}_2$ to the identity on st and maps all the other morphisms of $\mathcal{I}_{Q,T}$ to the homonymous ones in $\mathcal{I}_{A^*}$.

▶ **Lemma 16.** *The functor $i^*\colon \mathcal{I}_{Q,T} \to \mathcal{I}_{A^*}$ is well defined and, furthermore, $\mathcal{O}_{Q,T}$ is a subcategory of $\mathcal{O}_{A^*}$. That is, we have the following commuting diagram:*

$$\begin{array}{ccc} \mathcal{O}_{Q,T} & \hookrightarrow & \mathcal{O}_{A^*} \\ \downarrow & & \downarrow \\ \mathcal{I}_{Q,T} & \xrightarrow{i^*} & \mathcal{I}_{A^*}. \end{array}$$

The partial knowledge of the language $\mathcal{L}$ the learner has access to at this given stage of the algorithm is captured by the restriction $\mathcal{L}_{Q,T}$ of $\mathcal{L}$ to $\mathcal{O}_{Q,T}$:

$$\mathcal{L}_{Q,T}\colon \mathcal{O}_{Q,T} \hookrightarrow \mathcal{O}_{A^*} \xrightarrow{\mathcal{L}} \mathcal{C}.$$

Hence, to a pair (Q,T) we can associate the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$ obtained by instantiating in Definition 1 the input category $\mathcal{I}$ with $\mathcal{I}_{Q,T}$ and its observable behaviour subcategory with $\mathcal{O}_{Q,T} \hookrightarrow \mathcal{I}_{Q,T}$.

▶ **Definition 17.** *We call a functor $\mathcal{B}$ in $\mathsf{Auto}(\mathcal{L}_{Q,T})$ a (Q,T)-*biautomaton *$\mathcal{B}$ or a $\mathcal{C}_{Q,T}$-*biautomaton*, if we want to underline the dependence on $\mathcal{C}$. We say that $\mathcal{B}$ is* consistent *with the $\mathcal{C}$-language $\mathcal{L}$.*

In the $\mathtt{L}^*$-algorithm, the learner constructs a table associated to each pair of subsets (Q,T). This is done essentially by computing the quotient of the state words in Q by an approximation $\sim_T$ of the Myhill-Nerode equivalence for a language L given by: $w \sim_T v$ iff for all $t \in T$ we have $wt \in L \Leftrightarrow vt \in L$. This leads us to consider as a generalization of the notion of *table* the *minimal biautomaton* $\mathsf{Min}(\mathcal{L}_{Q,T})$ in the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$. In order to compute it, we use Corollary 7. To this end, we first exhibit explicitly the initial and the final objects of $\mathsf{Auto}(\mathcal{L}_{Q,T})$, assuming that the output category $\mathcal{C}$ has got certain products and coproducts.

We will use the following notation. Given two subsets R and S of A^*, let RS denote the set $\{xy\colon x \in R, y \in S\}$.

▶ **Lemma 18.** *Assume $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathsf{in})$. The initial $\mathcal{C}_{Q,T}$-biautomaton is the functor $\mathcal{A}_{init}(\mathcal{L}_{Q,T})\colon \mathcal{I}_{Q,T} \to \mathcal{C}$ described in the next diagram*

$$\mathcal{L}(\mathsf{in}) \xrightarrow{\triangleright q_{init}} \coprod_{Q} \mathcal{L}(\mathsf{in}) \underset{\varepsilon_{init}}{\overset{a_{init}}{\rightrightarrows}} \coprod_{Q \cup QA} \mathcal{L}(\mathsf{in}) \xrightarrow{t \triangleleft_{init}} \mathcal{L}(\mathsf{out}),$$

where, explicitly:

- $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_1) = \coprod_{Q} \mathcal{L}(\mathsf{in})$ *and* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_2) = \coprod_{Q \cup QA} \mathcal{L}(\mathsf{in})$*;*
- $\triangleright q_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(\triangleright q)$ *is the coproduct injection j_q of $\mathcal{L}(\mathsf{in})$ into $\coprod_{Q} \mathcal{L}(\mathsf{in})$;*
- $\varepsilon_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(\varepsilon)$ *is the canonical inclusion between the two coproducts;*

- $a_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(a)$ *is obtained via the universal property as the coproduct over* $q \in Q$ *of the canonical injections* $j_{qa} \colon \mathcal{L}(\mathsf{in}) \to \coprod_{Q \cup QA} \mathcal{L}(\mathsf{in})$*;*
- $t\triangleleft_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(t\triangleleft)$ *is obtained via the universal property as the coproduct over* $w \in Q \cup QA$ *of the morphims* $\mathcal{L}(\triangleright wt\triangleleft) \colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$*.*

Dually, we can describe the final $\mathcal{C}_{Q,T}$-biautomaton as follows.

▶ **Lemma 19.** *Assume* $\mathcal{C}$ *has all countable powers of* $\mathcal{L}(\mathsf{out})$*. The final* $\mathcal{C}_{Q,T}$*-biautomaton is the functor* $\mathcal{A}_{final}(\mathcal{L}_{Q,T}) \colon \mathcal{I}_{Q,T} \to \mathcal{C}$ *described in the next diagram*

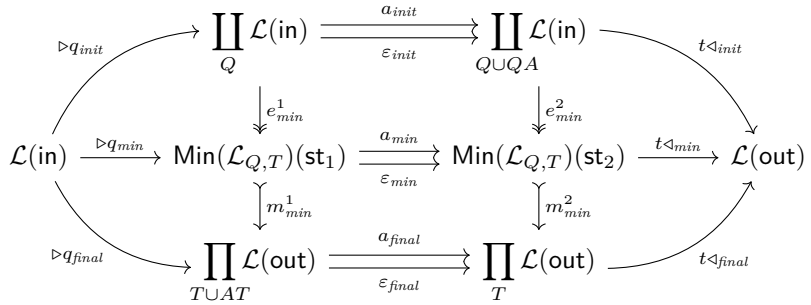
$$\mathcal{L}(\mathsf{in}) \xrightarrow{\triangleright q_{final}} \prod_{T \cup AT} \mathcal{L}(\mathsf{out}) \underset{\varepsilon_{final}}{\overset{a_{final}}{\rightrightarrows}} \prod_{T} \mathcal{L}(\mathsf{out}) \xrightarrow{t\triangleleft_{final}} \mathcal{L}(\mathsf{out}),$$

where, explicitly:

- $\mathcal{A}_{final}(\mathcal{L}_{Q,T})(\mathsf{st}_1) = \prod_{T \cup AT} \mathcal{L}(\mathsf{out})$ *and* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_2) = \coprod_{T} \mathcal{L}(\mathsf{out})$*;*
- $\triangleright q_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(\triangleright q)$ *is obtained via the universal property as the product over* $w \in T \cup AT$ *of the morphisms* $\mathcal{L}(\triangleright qw\triangleleft) \colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$*;*
- $\varepsilon_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(\varepsilon)$ *is the canonical restriction between the two products;*
- $a_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(a)$ *is obtained via the universal property of* $\prod_{T} \mathcal{L}(\mathsf{out})$ *as the product over* $t \in T$ *of the canonical projections* $\pi_{at} \colon \prod_{T \cup AT} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$*;*
- $t\triangleleft_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(t\triangleleft)$ *is the projection* $\pi_t \colon \prod_{T} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$*.*

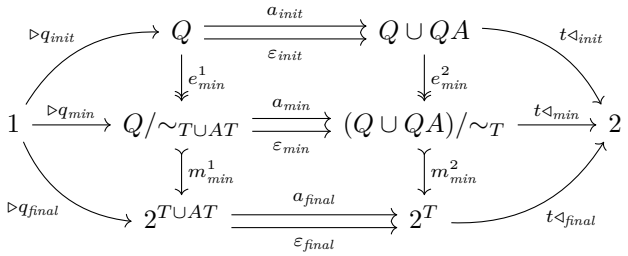
Combining Corollary 7 with Lemmas 8, 18 and 19, we obtain the minimal biautomaton $\mathsf{Min}(\mathcal{L}_{Q,T})$ in $\mathsf{Auto}(\mathcal{L}_{Q,T})$.

▶ **Theorem 20.** *Assume that* $\mathcal{C}$ *is equipped with a factorization system* $(\mathcal{E}, \mathcal{M})$ *and has countable copowers of* $\mathcal{L}(\mathsf{in})$ *and countable powers of* $\mathcal{L}(\mathsf{out})$*. Then the minimal* $\mathcal{C}_{Q,T}$*-biautomaton* $\mathsf{Min}(\mathcal{L}_{Q,T})$ *is obtained as the unique up to isomorphism factorization of the unique morphism from* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})$ *to* $\mathcal{A}_{final}(\mathcal{L}_{Q,T})$*.*



Notice that the arrows $\triangleright q_{min}$, a_{min}, ε_{min} and $t\triangleleft_{min}$ are obtained using the diagonal fill-in property of the factorization system. Let us now see how this theorem instantiates in the case of deterministic automata.

▶ **Example 21.** Assume the target language $\mathcal{L}$ is a $(\mathsf{Set}, 1, 2)$-language, so the learner wants to learn the minimal deterministic automaton accepting $\mathcal{L}$. For a given couple (Q, T), the minimal biautomaton is obtained as the following factorization.

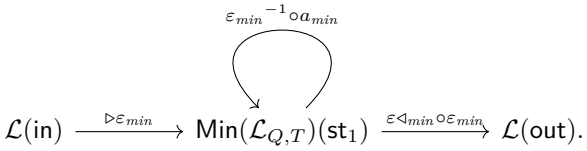


Hence, the first set of states of the minimal biautomaton is the set Q quotiented by the $T \cup AT$-approximation $\sim_{T \cup AT}$ of the Myhill-Nerode equivalence. The second set of states is the quotient of $Q \cup QA$ by $\sim_T$. These kinds of quotients are also needed in the classical Angluin's algorithm, when building the table corresponding to the couple (Q, T).

Let us now understand when the map ε_{min} is an isomorphism, that is, in this case, a bijection. We can verify that ε_{min} being a surjection is equivalent to the table in L*-algorithm being closed, that is, for all $q \in Q$ and $a \in A$ there exists $q' \in Q$ such that $q' \sim_T qa$. On the other hand, ε_{min} being an injection is equivalent to the consistency of the table from Angluin's L*-algorithm. It means that if q and q' are such that $q \sim_T q'$ then $q \sim_{T \cup AT} q'$.

If the table from Angluin's algorithm is generalized via the minimal biautomaton $\mathsf{Min}(\mathcal{L}_{Q,T})$, the above example suggests that the conditions that make possible the generation of a hypothesis automaton from a table can be stated at this abstract level by requiring the morphism ε_{min} be an isomorphism. In this way, we can identify $\mathsf{Min}(\mathcal{L})(\mathsf{st}_1)$ and $\mathsf{Min}(\mathcal{L})(\mathsf{st}_2)$ to obtain the state space of the hypothesis automaton.

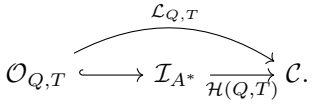
▶ **Definition 22.** *If the map ε_{min} is an isomorphism, we say that (Q, T) is* $\mathcal{L}$-automatable. *The* hypothesis automaton *$\mathcal{H}(Q, T)$ associated to a $\mathcal{L}$-automatable couple (Q, T) is the $\mathcal{C}$-automaton with state space $\mathsf{Min}(\mathcal{L}_{Q,T})(\mathsf{st}_1)$ described on the generator arrows of $\mathcal{I}_{A^*}$ by*



The uniqueness up to isomorphism of the hypothesis automaton $\mathcal{H}(Q, T)$ is an easy consequence of the uniqueness up to isomorphism of the minimal biautomaton in $\mathsf{Auto}(\mathcal{L}_{Q,T})$.

It is important to remark that, when passing from a biautomaton to an automaton, the consistency with the language is preserved, in the sense of the lemma below.

▶ **Lemma 23.** *Let (Q, T) be an $\mathcal{L}$-automatable couple and let $\mathcal{H}(Q, T)$ be its associated hypothesis automaton. Then the next diagram commutes:*



3.2 The learning algorithm

We now have all the necessary ingredients to state the FunL*-algorithm. Our algorithm takes as input a target language $\mathcal{L}$ and outputs its minimal automaton, provided some mild assumptions listed in Theorem 26 are satisfied. We start by instantianting the couple (Q, T) by $(\varepsilon, \varepsilon)$. As long as this couple is not $\mathcal{L}$-automatable, further words are added to the

subsets Q and T to force ε_{min} to become an isomorphism. Once this is achieved, we obtain a hypothesis automaton. If this automaton does not recognize the target language, then the provided counterexample and its prefixes are added to Q, in order to let the learner progress in learning.

While the role played by equivalence queries is self-evident, notice that evaluation queries are necessary in order to build up the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$ and analyse its minimal automaton.

Algorithm 1 The basic FunL*learning algorithm.

```
input  : minimally adequate teacher of the target language L
output : Min(L)
Q := T := {ε}
repeat
    while (Q, T) is not L-automatable do
        if ε_min ∉ E then
            add QA to Q
        end
        if ε_min ∉ M then
            add AT to T
        end
    end
    ask an equivalence query for the hypothesis automaton H(Q, T)
    if the answer is no then
        add the provided counterexample and all its prefixes to Q
    end
until the answer is yes;
return H(Q, T)
```

In order for this generic algorithm to work, we need several mild assumptions on the output category $\mathcal{C}$ and on the target language. First, in order to compute the hypothesis automaton we need the existence of the minimal automaton in the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$. For this reason, we will assume the hypothesis of Theorem 20, pertaining to the existence of certain powers, certain copowers and a factorization system. Furthermore, in order to ensure the termination of our algorithm, a noetherianity condition is required on the language $\mathcal{L}$, akin to the regularity of the language in the L*-algorithm. This notion, also used in [21], can be understood as a finiteness assumption as shown in Example 25.

▶ **Definition 24.** *An object X of $\mathcal{C}$ is called* $(\mathcal{E},\mathcal{M})$-noetherian *when the following conditions hold.*

- *There does not exist an infinite co-chain of $\mathcal{E}$-quotients of X as in the left commutative diagram below and such that the arrows $e_1, e_2 \ldots \in \mathcal{E}$ are not isomorphisms.*
- *There does not exist an infinite chain of $\mathcal{M}$-subobjects of X as in the right commutative diagram below and such that the arrows $m_1, m_2 \ldots \in \mathcal{M}$ are not isomorphisms.*

▶ **Example 25.** Let us see now what noetherianity means for the factorization systems of our running instantiations (Example 9). It is easy to see that in Set, an object X is (Surjections, Injections)-noetherian if and only if it is finite in the usual sense. Similarly,

an object of Vec is (Surjective linear maps, Injective linear maps)-noetherian if and only if it is a finite dimension vector space. With a bit more thoughts, one can establish that an object X of $\mathsf{Kl}(\mathcal{T})$ is $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$-noetherian if and only if it is finite.

In order to guarantee the termination of our algorithm, we require the $(\mathcal{E}, \mathcal{M})$-noetherianity of the state space of the minimal automaton of the target language. This is a natural condition, generalizing the regularity of the target language in the $\mathtt{L}^*$-algorithm. If $(\mathcal{E}, \mathcal{M})$-noetherian objects are closed under $\mathcal{E}$-quotients and $\mathcal{M}$-subobjects – as it is the case in all our examples – we could also replace this hypothesis by assuming the existence of an automaton with $(\mathcal{E}, \mathcal{M})$-noetherian state space which accepts the target language.

▶ **Theorem 26.** *We consider a target language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$ such that:*

- *the output category $\mathcal{C}$ is endowed with a factorization system $(\mathcal{E}, \mathcal{M})$;*
- *$\mathcal{C}$ has all copowers of $\mathcal{L}(\mathsf{in})$ and all powers of $\mathcal{L}(\mathsf{out})$;*
- *the state space $\mathsf{Min}(\mathcal{L})(\mathsf{st})$ of the minimal automaton for $\mathcal{L}$ is $(\mathcal{E}, \mathcal{M})$-noetherian.*

Then the $\mathtt{FunL}^*$-algorithm terminates, eventually producing the minimal automaton $\mathsf{Min}(\mathcal{L})$ accepting the target language.

The proof of this theorem relies on a careful analysis of the factorizations

$$\coprod_Q \mathcal{L}(\mathsf{in}) \longrightarrow\!\!\!\!\!\rightarrow \Im_{Q,T} \rightarrowtail \prod_T \mathcal{L}(\mathsf{out})$$

of the canonical maps $\coprod_Q \mathcal{L}(\mathsf{in}) \to \prod_T \mathcal{L}(\mathsf{out})$ obtained by taking the coproduct over $q \in Q$ of the product over $t \in T$ of $\mathcal{L}(\triangleright qt \triangleleft)$. We can prove that the state spaces of the biautomata featured while running the algorithm are precisely of the form $\Im_{Q,T}$, while the state space of the minimal automaton accepting $\mathcal{L}$ is $\Im_{A^*,A^*}$.

We prove that the **while** loop terminates in [14, Proposition 35]. In [14, Lemma 36] we show that only finitely many counterexamples can be added, hence the algorithm terminates. Finally, the fact that the outcome automaton is minimal is shown in [14, Lemma 38].

Next, we see how the $\mathtt{FunL}^*$-algorithm instantiates to the case of subsequential transducers. We need to understand what it means for ε_{min} to be an isomorphism.

▶ **Example 27** (Learning algorithm for subsequential transducers)**.** Assume the target language $\mathcal{L}$ is a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-language, so the learner wants to learn the minimal subsequential transducer accepting $\mathcal{L}$. We need to extend the notions of lcp and red to a generic partial map g whose domain is $T \subseteq A^*$ and whose codomain is B^* as follows: $\mathsf{lcp}(g)$ is undefined if g is nowhere defined, and denotes the longest common prefix of the words in $\{g(u) \mid u \in T\}$ otherwise; analogously, $\mathsf{red}(g)\colon T \to B^* \cup \{\bot\}$ is nowhere defined if g is nowhere defined, and is the only partial map such that $g(u) = \mathsf{lcp}(g)\mathsf{red}(g)(u)$ otherwise. Thinking of the language to learn as a transduction $f\colon A^* \to B^* + \{\bot\}$, let's define the following equivalence relation for all $q_1, q_2 \in Q$: $q_1 \sim_T q_2$ if and only if $\mathsf{red}(f(q_1-)|_T)(t) = \mathsf{red}(f(q_2-)|_T)(t)$ for all $t \in T$, $f(q-)|_T$ being the restriction of $f(q-)$ to T. For a couple (Q, T), ε_{min} in $\mathsf{Auto}(\mathcal{L}_{Q,T})$ turns out to be the map $Q/{\sim_{T \cup AT}} \twoheadrightarrow (Q \cup QA)/{\sim_T}, [q] \mapsto (\mathsf{lcp}(f(q-)|_{T \cup AT})^{-1}\mathsf{lcp}(f(q-)|_T), [q])$, the first set of states being Q quotiented by $\sim_{T \cup AT}$, the second set of states being the quotient of $Q \cup QA$ by $\sim_T$. Let's understand the word a class $[q]$ is mapped to: with $\mathsf{lcp}(f(q-)|_T)$, we mean the lcp of the function $f(q-)$ restricted to the domain T, that is, the longest common prefix of the subset $\{f(qt) | t \in T\}$; with $\mathsf{lcp}(f(q-)|_{T \cup AT})^{-1}\mathsf{lcp}(f(q-)|_T)$, we mean the word $\mathsf{lcp}(f(q-)|_T)$ from which $\mathsf{lcp}(f(q-)|_{T \cup AT})$ (one of its prefixes, as it is the longest common prefix of a bigger set of words) has been stripped away; when one of the lcps is undefined, $[q]$ is mapped to undefined.

ε_{min} is an isomorphism if and only if $\pi_2(\varepsilon_{min})$ is a bijection and $\pi_1(\varepsilon_{min})$ is the constant function mapping every $x \in X$ to ε. We can verify that $\pi_2(\varepsilon_{min})$ being a surjection is equivalent to the condition that for all $q \in Q$ and $a \in A$ there exists $q' \in Q$ such that $q' \sim_T qa$, whereas $\pi_2(\varepsilon_{min})$ being an injection is equivalent to the condition that if q and q' are such that $q \sim_T q'$, then $q \sim_{T \cup AT} q'$. Finally, the condition on $\pi_1(\varepsilon_{min})$ is true if and only if $\mathsf{lcp}(f(q-)|_{T \cup AT}) = \mathsf{lcp}(f(q-)|_T)$. Remarkably, these three naturally arising conditions turn out to be equivalent to the ones required in Vilar's learning algorithm for subsequential transducers (see [24]).

Every time the while cycle runs, our algorithm adds either all words QA to Q or all words AT to T: this is not strictly necessary. We show next that it is sufficient to add just one properly chosen single word $qa \in QA$ or $at \in AT$, preserving the correctness of the algorithm. The canonical inclusion $\coprod_Q \mathcal{L}(\mathsf{in}) \to \coprod_{Q \cup \{qa\}} \mathcal{L}(\mathsf{in})$ induces a canonical morphism between the factorizations $\Im_{Q,T} \rightarrowtail \Im_{Q \cup \{qa\},T}$. Similarly, the canonical restriction $\prod_{T \cup \{at\}} \mathcal{L}(\mathsf{out}) \to \prod_T \mathcal{L}(\mathsf{out})$ induces a canonical morphism between the factorizations $\Im_{Q,T} \twoheadleftarrow \Im_{Q,T \cup \{at\}}$, which will be featured in the optimized algorithm.

▶ **Theorem 28.** *Algorithm 1 can be optimized by replacing lines 5 and 8 respectively by:*

- add to Q a $qa \in QA$ s.t. $\Im_{Q,T} \rightarrowtail \Im_{Q \cup \{qa\},T}$ is not an isomorphism*;*
- add to T an $at \in AT$ s.t. $\Im_{Q,T} \twoheadleftarrow \Im_{Q,T \cup \{at\}}$ is not an isomorphism*.*

4 Conclusion and future work

In this paper, we described the abstract algorithm FunL*, a categorical version of Angluin's L*-algorithm for learning word automata. The focus was on providing a minimalistic category theoretic framework for learning, with as few assumptions as possible, emphasizing along the way the deep connection between learning and minimization.

So far, FunL* does not cover instances of the L*-like algorithms such as nominal automata, or automata/transducers over trees. A natural continuation is to develop these generalizations. Another aspect to understand abstractly is the complexity of this algorithm in terms of number of evaluation and equivalence queries.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput., 75, 87-106*, 1987.

2 Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *IJCAI*, pages 3308–3314. AAAI Press, 2015.

3 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.

4 Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra learning via duality. In *FoSSaCS*, volume 11425 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2019.

5 Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. In *CIAC*, volume 778 of *Lecture Notes in Computer Science*, pages 54–62. Springer, 1994.

6 Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

7 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning rational functions. In *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2012.

8 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning top-down tree transducers with regular domain inspection. In *ICGI*, volume 57 of *JMLR Workshop and Conference Proceedings*, pages 54–65. JMLR.org, 2016.

9 Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009. URL: `http://ijcai.org/Proceedings/09/Papers/170.pdf`.

10 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. libalf: The automata learning framework. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 360–364, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

11 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.

12 Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. In *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, pages 8:1–8:16, 2017. `doi:10.4230/LIPIcs.CALCO.2017.8`.

13 Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, 2020. URL: `https://lmcs.episciences.org/6213`.

14 Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. Learning automata and transducers: a categorical approach. *CoRR*, 2020. `arXiv:2010.13675`.

15 Samuel Drews and Loris D'Antoni. Learning symbolic automata. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 173–189, 2017.

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

17 Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, volume 8464 of *Lecture Notes in Computer Science*, pages 384–406. Springer, 2014.

18 Martin Leucker. Learning meets verification. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, pages 127–151, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In *POPL*, pages 613–625. ACM, 2017. URL: `http://dl.acm.org/citation.cfm?id=3009879`.

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

21 Henning Urbat and Lutz Schröder. Automata learning: An algebraic approach. In *LICS*, pages 900–914. ACM, 2020.

22 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. CALF: categorical automata learning framework. In *CSL*, volume 82 of *LIPIcs*, pages 29:1–29:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

23 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. Learning automata with side-effects. In *CMCS*, 2020.

24 Juan Miguel Vilar. Query learning of subsequential transducers. In *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, pages 72–83, 1996. `doi:10.1007/BFb0033343`.

25 Juan Miguel Vilar. Improve the learning of subsequential transducers by using alignments and dictionaries. In *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11-13, 2000, Proceedings*, pages 298–311, 2000. `doi:10.1007/978-3-540-45257-7_24`.