Active Learning of Deterministic Transducers with Outputs in Arbitrary Monoids

Quentin Aristote

Active Learning of Deterministic Transducers with Outputs in Arbitrary Monoids

Quentin Aristote
École Normale Supérieure de Paris, PSL University, France
Université Paris Cité, CNRS, Inria, IRIF, F-75013, Paris, France

Abstract

We study monoidal transducers, transition systems arising as deterministic automata whose transitions also produce outputs in an arbitrary monoid, for instance allowing outputs to commute or to cancel out. We use the categorical framework for minimization and learning of Colcombet, Petrişan and Stabile to recover the notion of minimal transducer recognizing a language, and give necessary and sufficient conditions on the output monoid for this minimal transducer to exist and be unique (up to isomorphism). The categorical framework then provides an abstract algorithm for learning it using membership and equivalence queries, and we discuss practical aspects of this algorithm's implementation.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases transducers, monoids, active learning, category theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2024.11

Related Version *Extended Version*: https://ens.hal.science/hal-04172251v2

1 Introduction

Transducers are (possibly infinite) transition systems that take input words over an input alphabet and translate them to some output words over an output alphabet. They are numerous ways to implement them, but here we focus on *subsequential transducers*, i.e deterministic automata whose transitions also produce an output (see Figure 1 for an example). They are used in diverse fields such as compilers [11], linguistics [13], or natural language processing [14].

Two subsequential transducers are considered equivalent when they *recognize* the same *subsequential function*, that is if, given the same input, they always produce the same output. A natural question is thus whether there is a (unique) minimal transducer recognizing a given function (a transducer with a minimal number of states and which produces its ouput as early as possible), and whether this minimal transducer is computable. The answer to both these questions is positive when there exists a finite subsequential transducer recognizing this function: the minimal transducer can then for example be computed through minimization [6].

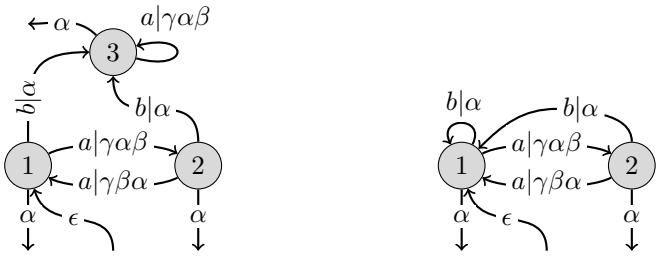
Active learning of transducers

Another method for computing a minimal transducer is to learn it through Vilar's algorithm [21], a generalization to transducers of Angluin's L*-algorithm, which learns the minimal deterministic automaton recognizing a language [1]. Vilar's algorithm thus relies on the existence of an oracle which may answer two types of queries, namely:

- *membership queries*: when queried with an input word, the oracle answers with the corresponding expected output word;
- *equivalence queries*: when queried with a *hypothesis transducer*, the oracle answers whether this transducer recognizes the target function, and, if not, provides a counter-example input word for which this transducer is wrong.

The basic idea of the algorithm is to use the membership queries to infer partial knowledge of the target function on a finite subset of input words, and, when some *closure* and *consistency* conditions are fulfilled, use this partial knowledge to build a hypothesis transducer to submit to the oracle through an equivalence query: the oracle then either confirms this transducer is the right one, or provides a counter-example input word on which more knowledge of the target function should be inferred.

Figure 1 Two transducers: unlike automata, the transitions are also labelled with output words.



(a) A monoidal transducer $\mathcal{A}$. **(b)** A hypothesis transducer built when learning $\mathcal{A}$.

Consider for instance the partial function recognized by the minimal transducer $\mathcal{A}$ of Figure 1a over the input alphabet $A = \{a, b\}$ and output alphabet $\Sigma = \{\alpha, \beta, \gamma\}$. We write this function $\mathcal{L}(\triangleright - \triangleleft) : A^* \to \Sigma^* \sqcup \{\bot\}$, and let $e \in A^*$ and $\epsilon \in \Sigma^*$ stand for the respective empty words over these two alphabets. To learn $\mathcal{A}$, the algorithm maintains a subset $Q \subset A^*$ of prefixes of input words and a subset $T \subset A^*$ of suffixes of input words, and keeps track of the restriction of $\mathcal{L}(\triangleright - \triangleleft)$ to words in $QT \cup QAT$. The prefixes in Q will be made into states of the hypothesis transducer, and two prefixes $q, q' \in Q$ will correspond to two different states if there is a suffix $t \in T$ such that $\mathcal{L}(\triangleright qt \triangleleft) \neq \mathcal{L}(\triangleright q't \triangleleft)$. Informally, closure then holds when for any state $q \in Q$ and input letter $a \in A$ an a-transition to some state $q' \in Q$ can always be built; consistency holds when there is always at most one consistent choice for such a q' and when the newly-built a-transition can be equipped with an output word. The execution of the learning algorithm for the function recognized by $\mathcal{A}$ would thus look like the following.

The algorithm starts with $Q = T = \{e\}$ only consisting of the empty input word. In a hypothesis transducer, we would want $e \in Q$ to correspond to the initial state, and the output value produced by the initial transition to be the longest common prefix $\Lambda(e)$ of each $\mathcal{L}(\triangleright et \triangleleft)$ for $t \in T$, here $\Lambda(e) = \alpha$. But the longest common prefix $\Lambda(a)$ of each $\mathcal{L}(\triangleright at \triangleleft)$ for $t \in T$ is $\gamma\alpha\beta\alpha$, of which $\Lambda(e)$ is not a prefix: it is not possible to make the output of the first a-transition so that following the initial transition and then the a-transition produces a prefix of $\Lambda(a)$! This is a first kind of consistency issue, which we solve by adding a to T, turning $\Lambda(e)$ into the empty output word ϵ and $\Lambda(a)$ into $\gamma\alpha\beta$.

Now $Q = \{e\}$ and $T = \{e, a\}$. The initial transition should go into the state corresponding to e and output $\Lambda(e) = \epsilon$, the final transition from this state should output $\Lambda(e)^{-1}\mathcal{L}(\triangleright e \triangleleft) = \alpha$, the a-transition from this state should output $\Lambda(e)^{-1}\Lambda(a) = \gamma\alpha\beta$, and this a-transition followed by a final transition should output $\Lambda(e)^{-1}\mathcal{L}(\triangleright a \triangleleft) = \gamma\alpha\beta\alpha$. This a-transition should moreover lead to a state from which another a-transition followed by a final transition outputs $\Lambda(a)^{-1}\mathcal{L}(\triangleright aa \triangleleft) = \gamma\beta\alpha^2$: in particular, it cannot lead back to the state corresponding to e, because $\gamma\beta\alpha^2 \neq \gamma\alpha\beta\alpha$. But this state is the only state accounted for by Q, so now we have no candidate for its successor when following the a-transition! This is a closure issue, which we solve by adding a to Q, the corresponding new state then being the candidate successor we were looking for.

Once $Q = T = \{e, a\}$, there are no closure nor consistency issues and we may thus build the hypothesis transducer given by Figure 1b: it coincides with $\mathcal{A}$ on $QA \cup QAT$. Submitting it to the oracle we learn that this transducer is not the one we are looking for, and we get as counter-example the input word bb, which indeed satisfies $\mathcal{L}(\triangleright bb \triangleleft) = \bot$ and yet for which our hypothesis transducer produced the output word α^3: we thus add bb and its prefixes to Q.

With $Q = \{e, a, b, bb\}$ and $T = \{e, a\}$ there is another kind of consistency issue, because the states corresponding to e and b are not distinguished by T ($\Lambda(e)^{-1}\mathcal{L}(\triangleright et \triangleleft) = \Lambda(b)^{-1}\mathcal{L}(\triangleright bt \triangleleft)$ for all $t \in T$) and should thus be merged in the hypothesis transducer, yet this is not the case of their candidate successors when following an additional b-transition ($\mathcal{L}(\triangleright ebe \triangleleft) = \alpha$ yet $\mathcal{L}(\triangleright bbe \triangleleft) = \bot$ is undefined)! This issue is solved by adding b to T, after which there are again no closure nor consistency issues and we may thus build $\mathcal{A}$ as our new hypothesis transducer. The algorithm finally stops as the oracle confirms that we found the right transducer.

Transducers with outputs in arbitrary monoids

In the example above we assumed the output of the transducer consisted of words over the output alphabet $\Sigma = \{\alpha, \beta, \gamma\}$, that is of elements of the free monoid Σ^*. But in some contexts it may be relevant to assume that certain output words can be swapped or can cancel each other out. In other words, transducers may be considered to be *monoidal* and have output not in a free monoid, but in a quotient of a free monoid. An example of a non-trivial family of monoids that should be interesting to use as the output of a transducer is the family of trace monoids, that are used in concurrency theory to model sequences of executions where some jobs are independent of one another and may thus be run asynchronously: transducers with outputs in trace monoids could be used to programatically schedule jobs. Algebraically, trace monoids are just free monoids where some pairs of letters are allowed to commute. For instance, the transducers of Figure 1 could be considered under the assumption that $\alpha\beta = \beta\alpha$, in which case the states 1 and 2 would have the same behavior.

This raises the question of the existence and computability of a minimal monoidal transducer recognizing a function with output in an arbitrary monoid. In [12], Gerdjikov gave some conditions on the output monoid for minimal monoidal transducers to exist and be unique up to isomorphism, along with a minimization algorithm that generalizes the one for (non-monoidal) transducers. This question had also been addressed in [10], although in a less satisfying way as the minimization algorithm relied on the existence of stronger oracles. Yet, to the best of the author's knowledge, no work has addressed the problem of learning minimal monoidal transducers through membership and equivalence queries.

As all monoids are quotients of free monoids, a first solution would of course be to consider the target function to have output in a free monoid, learn the minimal (non-monoidal) transducer recognizing this function using Vilar's algorithm, and only then consider the resulting transducer to have output in a non-free monoid and minimize it using Gerdjikov's minimization algorithm. But this solution is unsatisfactory as, during the learning phase, it may introduce states that will be optimized away during the minimization phase. For instance, learning the function recognized by the transducer $\mathcal{A}$ of Figure 1a with the assumption that $\alpha\beta = \beta\alpha$ would first produce $\mathcal{A}$ itself before having its states 1 and 2 merged during the minimization phase. Worse still, it is possible to find a partial function with output in a finitely generated quotient monoid $\Sigma^*/\sim$ and recognized by a finite monoidal transducer, and yet so that, when this function is considered to have output in Σ^*, Vilar's algorithm may not even terminate if, when answering membership queries, the oracle does not carefully choose the representatives in Σ^* of each equivalence class in $\Sigma^*/\sim$ (this is made formal by Lemma 34 in the appendix; the idea of finding such an example was suggested by an anonymous reviewer whom the author thanks).

A more satisfactory solution would hence do away with the minimization phase and instead use the assumptions on the output monoid during the learning phase to directly produce the minimal monoidal transducer.

Structure and contributions

In this work we thus study the problem of generalizing Vilar's algorithm to monoidal transducers. To this aim, we first recall in Section 2 the categorical framework of Colcombet, Petrişan and Stabile for learning minimal transition systems [7]. This framework encompasses both Angluin's and Vilar's algorithms, as well as a similar algorithm for weighted automata [4, 5]. We use this specific framework because, while others exist, they either do not encompass transducers or require stronger assumptions [3, 19, 20]. In Section 3 we then instantiate this framework to retrieve monoidal transducers as transition systems whose state-spaces live in a certain category (Section 3.2). Studying the existence of specific structures in this category – namely, powers of the terminal object (Section 3.3) and factorization systems (Section 3.4) – we then give conditions for the framework to apply and hence for the minimal monoidal transducers to exist and be computable.

This paper's contributions are thus the following:

- necessary and sufficient conditions on the output monoid for the categorical framework of Colcombet, Petrişan and Stabile to apply to monoidal transducers are given;
- these conditions mostly overlap those of Gerdjikov, but are nonetheless not equivalent: in particular, they extend the class of output monoids for which minimization is known to be possible, although with a possibly worse complexity bound;
- practical details on the implementation of the abstract monoidal transducer-learning algorithm that results from the categorical framework are given;
- in particular, additional structure on the category in which the framework is instantiated provides a neat categorical explanation to both the different kinds of consistency issues that arise in the learning algorithm and the main steps that are taken in every transducer minimization algorithm.

2 Categorical approach to learning minimal automata

In this section we recall (and extend) the definitions and results of Colcombet, Petrişan and Stabile [16, 8]. We assume basic knowledge of category theory [15], but we also focus on the example of deterministic complete automata and on the counter-example of non-deterministic automata. We do not explain it here but the framework also applies to weighted automata [4, 5, 17] and (non-monoidal) transducers (as generalized in Section 3).

2.1 Automata and languages as functors

Let $\mathcal{I}$, the *input category*, be the category freely generated by the diagram $\mathtt{in} \xrightarrow{\triangleright} \mathtt{st} \overset{a}{\circlearrowleft} \xrightarrow{\triangleleft} \mathtt{out}$ where a ranges in the *input alphabet* A: the objects are the vertices of the graph and the morphisms paths between two vertices. $\mathcal{I}$ represents the basic structure of automata as transition systems: $\mathtt{st}$ represents the state-space, $\triangleright$ the initial configuration, each $a : \mathtt{st} \to \mathtt{st}$ the transition along the corresponding letter, and $\triangleleft$ the output values associated to each state. An automaton is then an instantiation of $\mathcal{I}$ in some output category:

▶ **Definition 1** ($(\mathcal{C}, X, Y)$-automaton)**.** *Given an output category $\mathcal{C}$, a $(\mathcal{C}, X, Y)$-automaton is a functor $\mathcal{A} : \mathcal{I} \to \mathcal{C}$ such that $\mathcal{A}(\mathtt{in}) = X$ and $\mathcal{A}(\mathtt{out}) = Y$.*

▶ **Example 2** ((non-)deterministic automata). If $1 = \{*\}$ and $2 = \{\bot, \top\}$, a $(\mathbf{Set}, 1, 2)$-automaton $\mathcal{A}$ is a (possibly infinite) deterministic complete automaton: it is given by a state-set $S = \mathcal{A}(\mathtt{st})$, transition functions $\mathcal{A}(a) : S \to S$ for each $a \in A$, an initial state $s_0 = \mathcal{A}(\triangleright)(*) \in S$ and a set of accepting states $F = \{s \in S \mid \mathcal{A}(\triangleleft)(s) = \top\} \subseteq S$. Similarly, a $(\mathbf{Rel}, 1, 1)$-automaton (where **Rel** is the category of sets and relations between them) is a (possibly infinite) non-deterministic automaton: it is given by a state-set $S = \mathcal{A}(\mathtt{st})$, transition relations $\mathcal{A}(a) \subseteq S \times S$, a set of initial states $\mathcal{A}(\triangleright) \subseteq 1 \times S \cong S$ and a set of accepting states $\mathcal{A}(\triangleleft) \subseteq S \times 1 \cong S$.

▶ **Definition 3** ($(\mathcal{C}, X, Y)$-language). *In the same way, we define a* language *to be a functor $\mathcal{L} : \mathcal{O} \to \mathcal{C}$, where $\mathcal{O}$, the* category of observable inputs*, is the full subcategory of $\mathcal{I}$ on* `in` *and* `out`*. In other words, a language is the data of two objects $X = \mathcal{L}(\mathtt{in})$ and $Y = \mathcal{L}(\mathtt{out})$ in $\mathcal{C}$ (in which case we speak of a $(\mathcal{C}, X, Y)$-*language*), and, for each word $w \in A^*$, of a morphism $\mathcal{L}(\triangleright w \triangleleft) : X \to Y$. In particular, composing an automaton $\mathcal{A} : \mathcal{I} \to \mathcal{C}$ with the embedding $\iota : \mathcal{O} \hookrightarrow \mathcal{I}$, we get the language $\mathcal{L}_\mathcal{A} = \mathcal{A} \circ \iota$* recognized *by $\mathcal{A}$.*

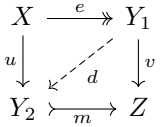
▶ **Example 4** (languages). The language recognized by a $(\mathbf{Set}, 1, 2)$-automaton $\mathcal{A}$ is the language recognized by the corresponding complete deterministic automaton: for a given $w \in A^*$, $\mathcal{L}_\mathcal{A}(\triangleright w \triangleleft)(*) = \top$ if and only if w belongs to the language, and the equality $\mathcal{L}_\mathcal{A}(\triangleright w \triangleleft) = \mathcal{A}(\triangleright w \triangleleft) = \mathcal{A}(\triangleright)\mathcal{A}(w)\mathcal{A}(\triangleleft)$ means that we can decide whether w is in the language by checking whether the state we get in by following w from the initial state is accepting. Such a language could also be seen as a set of relations $\mathcal{L}(\triangleright w \triangleleft) \subseteq 1 \times 1$, where $*$ is related to itself if and only if w belongs to $\mathcal{L}$. The language recognized by a $(\mathbf{Rel}, 1, 1)$-automaton is thus the language recognized by the corresponding non-deterministic automaton.

▶ **Definition 5** (category of automata recognizing a language). *Given a category $\mathcal{C}$ and a language $\mathcal{L} : \mathcal{O} \to \mathcal{C}$, we define the category $\mathbf{Auto}(\mathcal{L})$ whose objects are $(\mathcal{C}, \mathcal{L}(\mathtt{in}), \mathcal{L}(\mathtt{out}))$-automata $\mathcal{A}$ recognizing $\mathcal{L}$, and whose morphisms $\mathcal{A} \to \mathcal{A}'$ are natural transformations whose components on $\mathcal{L}(\mathtt{in})$ and $\mathcal{L}(\mathtt{out})$ are the identity. In other words, a morphism of automata is given by a morphism $f : \mathcal{A}(\mathtt{st}) \to \mathcal{A}'(\mathtt{st})$ in $\mathcal{C}$ such that $\mathcal{A}'(\triangleright) = f \circ \mathcal{A}(\triangleright)$ (it preserves the initial configuration), $\mathcal{A}'(a) \circ f = f \circ \mathcal{A}(a)$ (it commutes with the transitions), and $\mathcal{A}'(\triangleleft) \circ f = \mathcal{A}(\triangleleft)$ (it preserves the output values).*

2.2 Factorization systems and the minimal automaton recognizing a language

▶ **Definition 6** (factorization system). *In a category $\mathcal{C}$, a* factorization system $(\mathcal{E}, \mathcal{M})$ *is the data of a class of $\mathcal{E}$-*morphisms *(represented with $\twoheadrightarrow$) and a class of $\mathcal{M}$-*morphisms *(represented with $\rightarrowtail$) $\mathcal{M}$ such that*

- *every arrow f in $\mathcal{C}$ may be* factored *as $f = m \circ e$ with $m \in \mathcal{M}$ and $e \in \mathcal{E}$;*
- *$\mathcal{E}$ and $\mathcal{M}$ are both stable under composition;*
- *for every commuting diagram as below where $m \in \mathcal{M}$ and $e \in \mathcal{E}$ there is a unique diagonal fill-in $d : Y_1 \to Y_2$ such that $u = d \circ e$ and $v = m \circ d$.*



▶ **Example 7.** In **Set** surjective and injective functions form a factorization system $(\mathrm{Surj}, \mathrm{Inj})$ such that a map $f : X \to Y$ factors through its image $f(X) \subseteq Y$. In **Rel** a factorization system is given by $\mathcal{E}$-morphisms those relations $r : X \to Y$ such that every $y \in Y$ is related

to some $x \in X$ by r, and $\mathcal{M}$-morphisms the graphs of injective functions (i.e. $m \in \mathcal{M}$ if and only if there is an injective function f such that $(x, y) \in m \iff y = f(x)$). A relation $r : X \to Y$ then factors through the subset $\{y \in Y \mid \exists x \in X, (x, y) \in R\} \subseteq Y$.

▶ **Lemma 8** (factorization system on $\mathbf{Auto}(\mathcal{L})$ [16, Lemma 2.8])**.** *Given $\mathcal{L} : \mathcal{I} \to \mathcal{C}$ and $(\mathcal{E}, \mathcal{M})$ a factorization system on $\mathcal{C}$, $\mathbf{Auto}(\mathcal{L})$ has a factorization given by those natural transformations whose components are respectively $\mathcal{E}$- and $\mathcal{M}$-morphisms.*

Because of this last result, we use $(\mathcal{E}, \mathcal{M})$ to refer both to a factorization on $\mathcal{C}$ and to its extensions to categories of automata.

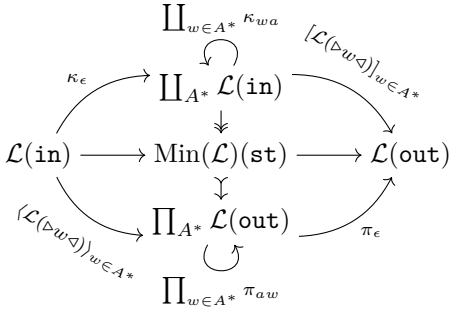
▶ **Definition 9** (minimal object)**.** *When a category $\mathcal{C}$, equipped with a factorization system $(\mathcal{E}, \mathcal{M})$, has both an initial object I and a final object F (for every object X there is exactly one morphism $I \to X$ and one morphism $X \to F$), we define its* $(\mathcal{E}, \mathcal{M})$-minimal object Min *to be the one that $(\mathcal{E}, \mathcal{M})$-factors the unique arrow $I \to F$ as $I \twoheadrightarrow \mathrm{Min} \rightarrowtail F$. For every object X we also define* $\mathrm{Reach}\, X$ *and* $\mathrm{Obs}\, X$ *by the $(\mathcal{E}, \mathcal{M})$-factorizations $I \twoheadrightarrow \mathrm{Reach}\, X \rightarrowtail X$ and $X \twoheadrightarrow \mathrm{Obs}\, X \rightarrowtail F$.*

▶ **Proposition 10** (uniqueness of the minimal object [16, Lemma 2.3])**.** *The minimal object of a category $\mathcal{C}$ is unique up to isomorphism, so that for every other object X,* $\mathrm{Min} \cong \mathrm{Obs}(\mathrm{Reach}\, X) \cong \mathrm{Reach}(\mathrm{Obs}\, X)$*: there are in particular spans $X \leftarrowtail \mathrm{Reach}\, X \twoheadrightarrow \mathrm{Min}$ and co-spans $\mathrm{Min} \rightarrowtail \mathrm{Obs}\, X \twoheadleftarrow X$. It is in that last sense that* Min *is $(\mathcal{E}, \mathcal{M})$-smaller than every other object X, and is thus minimal.*

▶ **Example 11** (initial, final and minimal automata [16, Example 3.1])**.** Since **Set** is complete and cocomplete, the category of $(\mathbf{Set}, 1, 2)$-automata recognizing a language $\mathcal{L} : \mathcal{I} \to \mathbf{Set}$ has an initial, a final and a minimal object. The initial automaton has state-set A^*, initial state $\epsilon \in A^*$, transition functions $\delta_a(w) = wa$ and accepting states the $w \in A^*$ such that w is in $\mathcal{L}$. Similarly, the final automaton has state-set 2^{A^*}, initial state $\mathcal{L}$, transition functions $\delta_a(L) = a^{-1}L$ and accepting states the $L \in 2^{A^*}$ such that ϵ is in L. The minimal automaton for the factorization system of Example 7 thus has the Myhill-Nerode equivalence classes for its states. It is unique up to isomorphism and its $(\mathcal{E}, \mathcal{M})$-minimality ensures that it is the complete deterministic automaton with the smallest state-set that recognizes $\mathcal{L}$: it is in particular finite as soon as $\mathcal{L}$ is recognized by a finite automaton.

On the contrary, there is no good notion of a unique minimal non-deterministic automaton recognizing a regular ($(\mathbf{Rel}, 1, 1)$-) language $\mathcal{L}$. $\mathbf{Auto}(\mathcal{L})$ does have an initial and a final object: the initial automaton is the initial deterministic automaton recognizing $\mathcal{L}$, and the final automaton is the (non-deterministic) transpose of this initial automaton. But there is no factorization system that gives rise to a meaningful minimal object: for instance, the minimal object for the factorization system described in Example 7 has for state-set the set of suffixes of words in $\mathcal{L}$.

Notice how in Example 11 the initial and final $(\mathbf{Set}, 1, 2)$-automata have for respective state-sets A^*, the disjoint union of $|A^*|$ copies of 1, and 2^{A^*}, the cartesian product of $|A^*|$ copies of 2. A similar result holds for non-deterministic automata and generalizes as Theorem 12, itself summarized by the diagram below, where κ and π are the canonical inclusion and projections and $[-]$ and $\langle - \rangle$ are the copairing and pairing of arrows.

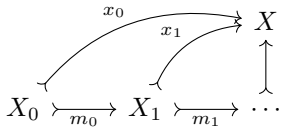


▶ **Theorem 12** ([16, Lemma 3.2]). *Given a countable alphabet A and a language $\mathcal{L} : \mathcal{I} \to \mathcal{C}$,*

- *if $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathtt{in})$ then $\mathbf{Auto}(\mathcal{L})$ has an initial object $\mathcal{A}^{init}(\mathcal{L})$ with $\mathcal{A}^{init}(\mathcal{L})(\mathtt{st}) = \coprod_{A^*} \mathcal{L}(\mathtt{in})$;*
- *dually if $\mathcal{C}$ has all countable powers of $\mathcal{L}(\mathtt{out})$ then $\mathbf{Auto}(\mathcal{L})$ has a final object $\mathcal{A}^{final}(\mathcal{L})$ with $\mathcal{A}^{final}(\mathcal{L})(\mathtt{st}) = \prod_{A^*} \mathcal{L}(\mathtt{out})$;*
- *hence when both of the previous items hold and $\mathcal{C}$ comes equipped with a factorization system $(\mathcal{E}, \mathcal{M})$, $\mathbf{Auto}(\mathcal{L})$ has an $(\mathcal{E}, \mathcal{M})$-minimal object* $\mathrm{Min}\,\mathcal{L}$.

We now have all the ingredients to define algorithms for computing the minimal automaton recognizing a language. But since we will also want to prove the termination of these algorithms, we need an additional notion of finiteness.

▶ **Definition 13** ($\mathcal{E}$-artinian and $\mathcal{M}$-noetherian objects [7, Definition 24]). *In a category $\mathcal{C}$ equipped with a factorization system $(\mathcal{E}, \mathcal{M})$, an object X is said to be* $\mathcal{M}$-noetherian *if every strict chain of $\mathcal{M}$-subobjects is finite: if $(x_n : X_n \rightarrowtail X)_{n\in\mathbb{N}}$ and $(m_n : X_n \rightarrowtail X_{n+1})$ form the commutative diagram*



then only finitely many of the m_n's may not be isomorphisms. Dually, X is $\mathcal{E}$-artinian *if X^{op} is $\mathcal{E}^{\mathrm{op}}$-noetherian in $\mathcal{C}^{\mathrm{op}}$, that is if every strict cochain of $\mathcal{E}$-quotients of X is finite.*

While Colcombet, Petrişan and Stabile do not give complexity results for their algorithm, it is straightforward to do so, hence we extend their definition so that it also measures the size of an object in $\mathcal{C}$.

▶ **Definition 14** (co-$\mathcal{E}$- and $\mathcal{M}$-lengths). *For a fixed $x_0 : X_0 \rightarrowtail X$, we call* $\mathcal{M}$-length *of x_0, written* $\mathrm{length}_{\mathcal{M}}\, x_0$*, the (possibly infinite) supremum of the lengths (the number of pairs of consecutive subobjects) of strict chains of $\mathcal{M}$-subobjects of X that start with x_0. Dually, we call* co-$\mathcal{E}$-length *of an $\mathcal{E}$-quotient $x_0 : X \twoheadrightarrow X_0$ the (possibly infinite) quantity* $\mathrm{colength}_{\mathcal{E}}\, x_0 = \mathrm{length}_{\mathcal{E}^{\mathrm{op}}}\, x_0^{\mathrm{op}}$.

▶ **Example 15.** In **Set**, X is finite if and only if it is Inj-noetherian iff it is Surj-artinian, and in that case for $Y \subseteq X$ we have $\mathrm{colength}_{\mathrm{Surj}}(X \twoheadrightarrow Y) = \mathrm{length}_{\mathrm{Inj}}(Y \rightarrowtail X) = |X - Y|$.

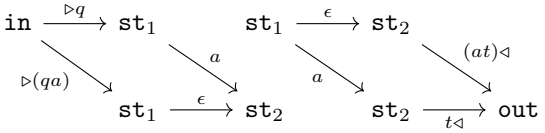
Note that the co-E- and $\mathcal{M}$-lengths need not be equal: see for instance the factorization system we define for monoidal transducers in Section 3.4, for which the co-$\mathcal{E}$- and $\mathcal{M}$-lengths are computed in Lemma 33.

2.3 Learning

In this section, we fix a language $\mathcal{L} : \mathcal{O} \to \mathcal{C}$ and a factorization system $(\mathcal{E}, \mathcal{M})$ of $\mathcal{C}$ that extends to $\mathbf{Auto}(\mathcal{L})$, and we assume that $\mathcal{C}$ has countable copowers of $\mathcal{L}(\mathtt{in})$ and countable powers of $\mathcal{L}(\mathtt{out})$ so that Theorem 12 applies. Our goal is to compute $\operatorname{Min}\mathcal{L}$ with the help of an oracle answering two types of queries: the function $\textsc{Eval}_{\mathcal{L}}$ processes membership queries, and, for a given input word $w \in A^*$, outputs $\mathcal{L}(\triangleright w \triangleleft)$; while $\textsc{Equiv}_{\mathcal{L}}$ processes equivalence queries, and, for a given hypothesis $(\mathcal{C}, \mathcal{L}(\mathtt{in}), \mathcal{L}(\mathtt{out}))$-automaton $\mathcal{A}$, decides whether $\mathcal{A}$ recognizes $\mathcal{L}$, and, if not, outputs a counter-example $w \in A^*$ such that $\mathcal{L}(\triangleright w \triangleleft) \neq (\mathcal{A} \circ \iota)(\triangleright w \triangleleft)$.

For $(\mathbf{Set}, 1, 2)$-automata, if the language is regular this problem is solved using Angluin's L* algorithm [1]. It works by maintaining a set of prefixes Q and of suffixes T and, using $\textsc{Eval}_{\mathcal{L}}$, incrementally building a table $L : Q \times (A \cup \{\epsilon\}) \times T \to 2$ that represents partial knowledge of $\mathcal{L}$ until it can be made into a (minimal) automaton. This automaton is then submitted to $\textsc{Equiv}_{\mathcal{L}}$ when some closure and consistency conditions hold: if the automaton is accepted it must be $\operatorname{Min}\mathcal{L}$, otherwise the counter-example is added to Q and the algorithm loops over. The FunL* algorithm generalizes this to arbitrary $(\mathcal{C}, \mathcal{L}(\mathtt{in}), \mathcal{L}(\mathtt{out}))$, and in particular also encompasses Vilar's algorithm for learning (non-monoidal) transducers, which was described in Section 1 [7].

Instead of maintaining a table, the FunL* algorithm maintains a biautomaton: if $Q \subseteq A^*$ is prefix-closed ($wa \in Q \Rightarrow w \in Q$) and $T \subseteq A^*$ is suffix-closed ($aw \in T \Rightarrow w \in T$), a (Q,T)-*biautomaton* is, similarly to an automaton, a functor $\mathcal{A} : \mathcal{I}_{Q,T} \to \mathcal{C}$, where $\mathcal{I}_{Q,T}$ is now the category freely generated by the graph $\mathtt{in} \xrightarrow{\triangleright q} \mathtt{st}_1 \underset{\epsilon}{\overset{a}{\rightrightarrows}} \mathtt{st}_2 \xrightarrow{t\triangleleft} \mathtt{out}$ where a, q and t respectively range in A, Q and T, and where we also require the diagrams below to commute, the left one whenever $qa \in Q$ and the right one whenever $at \in T$.



A (Q,T)-biautomaton may thus process a prefix in Q and get in a state in $\mathcal{A}(\mathtt{st}_1)$, follow a transition along $A \cup \{\epsilon\}$ to go in $\mathcal{A}(\mathtt{st}_2)$, and output a value for each suffix in T. The category of biautomata recognizing $\mathcal{L}_{Q,T}$ ($\mathcal{L}$ restricted to words in $QT \cup QAT$) is written $\mathbf{Auto}_{Q,T}(\mathcal{L})$. A result similar to Theorem 12 also holds for biautomata [7, Lemma 18], and the initial and final biautomata are then made of finite copowers of $\mathcal{L}(\mathtt{in})$ and finite powers of $\mathcal{L}(\mathtt{out})$ (when these exist). Writing Q/T for the $(\mathcal{E}, \mathcal{M})$-factorization of the canonical morphism $\langle [\mathcal{L}(\triangleright qt \triangleleft)]_{q \in Q} \rangle_{t \in T} : \coprod_Q \mathcal{L}(\mathtt{in}) \to \prod_T \mathcal{L}(\mathtt{out})$, the minimal biautomaton recognizing $\mathcal{L}_{Q,T}$ then has state-spaces $(\operatorname{Min}\mathcal{L}_{Q,T})(\mathtt{st}_1) = Q/(T \cup AT)$ and $(\operatorname{Min}\mathcal{L}_{Q,T})(\mathtt{st}_2) = (Q \cup QA)/T$. The table, represented by the morphism $\langle [\mathcal{L}(\triangleright qt \triangleleft)]_{q \in Q} \rangle_{t \in T}$, may be fully computed using $\textsc{Eval}_{\mathcal{L}}$, and hence so can be the minimal (Q,T)-biautomaton.

A biautomaton $\mathcal{B}$ can then be merged into a hypothesis $(\mathcal{C}, \mathcal{L}(\mathtt{in}), \mathcal{L}(\mathtt{out}))$-automaton precisely when $\mathcal{B}(\epsilon)$ is an isomorphism, i.e. both an $\mathcal{E}$- and an $\mathcal{M}$-morphism (a factorization system necessarily satisfies that $\mathrm{Iso} = \mathcal{E} \cap \mathcal{M}$): this encompasses respectively the closure and consistency conditions that need to hold in the L*-algorithm (and its variants) for the table that is maintained to be merged into a hypothesis automaton.

The FunL* algorithm terminates as soon as $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ is finite, which in this framework is expressed as it being $\mathcal{E}$-artinian and $\mathcal{M}$-noetherian. While Colcombet, Petrişan and Stabile do not give a bound on the actual running time of their algorithm, it would be straightforward to extend their proof to show that the number of updates to Q and T (hence in particular of calls to $\textsc{Equiv}_{\mathcal{L}}$) is linear in the size of $(\operatorname{Min}\mathcal{L})(\mathtt{st})$, itself defined through Definition 14.

Algorithm 1 The FunL*-algorithm.

Input: $\text{Eval}_{\mathcal{L}}$ and $\text{Equiv}_{\mathcal{L}}$
Output: $\text{Min}(\mathcal{L})$

$Q = T = \{\epsilon\}$
loop
  while $\epsilon^{min}_{Q,T}$ is not an isomorphism **do**
    if $\epsilon^{min}_{Q,T}$ is not an $\mathcal{E}$-morphism **then**
      find $qa \in QA$ such that $Q/T \rightarrowtail (Q \cup \{qa\})/T$ is not an $\mathcal{E}$-morphism; add it to Q
    else if $\epsilon^{min}_{Q,T}$ is not an $\mathcal{M}$-morphism **then**
      find $at \in AT$ such that $Q/(T \cup \{at\}) \twoheadrightarrow Q/T$ is not an $\mathcal{M}$-morphism; add it to T
    end if
  end while
  merge $\text{Min}\,\mathcal{L}_{Q,T}$ into $\mathcal{H}_{Q,T}\mathcal{L}$
  if $\text{Equiv}_{\mathcal{L}}(\mathcal{H}_{Q,T}\mathcal{L})$ outputs some counter-example w **then**
    add w and its prefixes to Q
  else
    return $\mathcal{H}_{Q,T}\mathcal{L}$
  end if
end loop

3 The category of monoidal transducers

We now study a specific family of transition systems, monoidal transducers, through the lens of category theory, so as to be able to apply the framework of Colcombet, Petrişan and Stabile. In Section 3.1, we first rapidly recall the notion of monoid. We then define the category of monoidal transducers recognizing a language in Section 3.2, and study how it fits into the framework of Section 2: the initial transducer is given in Corollary 25, conditions for the final transducer to exist are described in Section 3.3, and factorization systems are tackled in Section 3.4.

3.1 Monoids

Let us first recall definitions and notations related to monoids. Most of these are standard in the monoid literature, only coprime-cancellativity (Definition 19) and noetherianity (Definition 20) are uncommon.

▶ **Definition 16** (monoid). *A* monoid $(M, \epsilon_M, \otimes_M)$ *is a set* M *equipped with a binary operation* $\otimes_M$ *(often called the* product*) that is associative* $(\forall \upsilon, \nu, \omega \in M, \upsilon \otimes_M (\nu \otimes_M \omega) = (\upsilon \otimes_M \nu) \otimes_M \omega)$ *and has* ϵ_M *as* unit *element* $(\forall \upsilon \in M, \upsilon \otimes_M \epsilon_M = \epsilon_M \otimes_M \upsilon = \upsilon)$*. When non-ambiguous, it is simply written* $(M, \epsilon, \otimes)$ *or even* M*, and the symbol for the binary operation may be omitted. The* dual *of* $(M, \epsilon, \otimes)$*, written* $(M^{op}, \epsilon^{op}, \otimes^{op})$*, has underlying set* $M^{op} = M$ *and identity* $\epsilon^{op} = \epsilon$*, but symmetric binary operation :* $\forall \upsilon, \nu \in M, \upsilon \otimes^{op} \nu = \nu \otimes \upsilon$.

▶ **Definition 17** (invertibility). *An element* χ *of a monoid* M *is* right-invertible *when there is a* $\chi' \in M$ *such that* $\chi\chi' = \epsilon$*, and* χ' *is then called the* right-inverse *of* χ*.* χ *is* left-invertible *when it is right-invertible in* M^{op}*, and the corresponding right-inverse is called its* left-inverse*. When* χ *is both right- and left-invertible, we say it is* invertible*. In that case its right- and*

left-inverse are equal: this defines its inverse*, written* χ^{-1}*. The set of invertible elements of* M *is written* $M^\times$*. Two families* $(\upsilon_i)_{i\in I}$ *and* $(\nu_i)_{i\in I}$ *indexed by some non-empty set* I *are equal* up to invertibles on the left *when there is some invertible* $\chi \in M$ *such that* $\forall i \in I, \upsilon_i = \chi\nu_i$*.*

▶ **Definition 18** (divisibility)**.** *An element* υ *of a monoid* M left-divides *a family* $\omega = (\omega_i)_{i\in I}$ *of* M *indexed by some set* I *when there is a family* $(\nu_i)_{i\in I}$ *such that* $\forall i \in I, \upsilon\nu_i = \omega_i$*, and we say that* υ *is a* left-divisor *of* ω*.* υ right-divides ω *when it left-divides it in* M^{op}*, and in that case* υ *is called a* right-divisor *of* ω*. A* greatest common left-divisor *(or* left-gcd*) of the family* ω *is a left-divisor of* ω *that is left-divided by all others left-divisors of* ω*. A family* ω *is said to be* left-coprime *if* ϵ_M *is one of its left-gcds, that is if all its left-divisors (or equivalently one of its left-gcds when there is one) are trivial in the sense that they are right-invertible.*

We speak of *greatest* common left-divisors because, while there may be many such elements for a fixed family ω, they all left-divide one another and are thus equivalent in some sense.

▶ **Definition 19** (cancellativity)**.** *A monoid* M *is said to be* left-cancellative *when for any families* $(\upsilon_i)_{i\in I}$ *and* $(\nu_i)_{i\in I}$ *of* M *indexed by some set* I *and for any* $\omega \in M$*,* $\upsilon = \nu$ *as soon as* $\omega\upsilon_i = \omega\nu_i$ *for all* $i \in I$*. If this only implies* $\upsilon_i = \chi\nu_i$ *for some* $\chi \in M^\times$ *that does not depend on* $i \in I$*, we instead say that* M *is* left-cancellative up to invertibles on the left*. Similarly,* M *is said to be* right-coprime-cancellative *when for any* $\upsilon, \nu \in M$ *and any left-coprime family* $(\omega_i)_{i\in I}$ *indexed by some set* I*,* $\upsilon = \nu$ *as soon as* $\upsilon\omega_i = \nu\omega_i$ *for all* $i \in I$*.*

▶ **Definition 20** (noetherianity)**.** *A monoid* M *is* right-noetherian *when for any sequences* $(\upsilon_n)_{n\in\mathbb{N}}$ *and* $(\nu_n)_{n\in\mathbb{N}}$ *of* M *such that* $\nu_n = \nu_{n+1}\upsilon_n$ *for all* $n \in \mathbb{N}$*, there is some* $n \in \mathbb{N}$ *such that* υ_n *is invertible. In this case, we write* $\mathrm{rk}\,\nu$ *for the* rank *of* ν*, the (possibly infinite) supremum of the number of non-invertibles in a sequence* $(\upsilon_n)_{n\in\mathbb{N}}$ *that satisfies* $\nu_n = \nu_{n+1}\upsilon_n$ *for some sequence* $(\nu_n)_{n\in\mathbb{N}}$ *with* $\nu_0 = \nu$*.*

In other words, a monoid is right-noetherian when it has no strict infinite chain of right-divisors (in the definition above, each $\upsilon_n \cdots \upsilon_0$ right-divides ν_0). This condition is the one that will ensure our algorithms terminate (notice the resemblance with Definition 13).

▶ **Example 21.** The canonical example of a monoid is the *free monoid* A^* over an alphabet A, whose elements are words with letters in A, whose product is the concatenation of words and whose unit is the empty word. Notice that the alphabet A may be infinite. The left-divisibility relation is the prefix one, and the left-gcd is the longest common prefix.

The *free commutative monoid* $A^\circledast$ over A has elements the functions $A \to \mathbb{N}$ with finite support, product $(f \otimes g)(a) = f(a)+g(a)$ and unit the zero function $a \mapsto 0$. It is commutative ($f \otimes g = g \otimes f$) hence is its own dual : the divisibility relation is the pointwise order inherited from $\mathbb{N}$ and the greatest common divisor is the pointwise infimum.

These two monoids are examples of *trace monoids* over some A, defined as quotients of A^* by commutativity relations on letters (for $A^\circledast$, all the pairs of letters are required to commute, and for A^* none are). Trace monoids have no non-trivial right- or left-invertible elements, are all left-cancellative, right-coprime-cancellative and right-noetherian, and the rank of a word is simply its number of letters.

Another family of examples is that of *groups*, monoids where all elements are invertible. Again, all groups are left-cancellative, right-coprime-cancellative and right-noetherian.

A final family of monoids of interest is given by $(E, \vee, \bot)$ for E any join-semilattice with a bottom element $\bot$. In these commutative monoids, the divisibility relation is the partial order on E, and the gcd, when it exists, is the infimum. This example shows that a monoid can be coprime-cancellative without being cancellative nor noetherian: this is for instance the case when $E = \mathbb{R}_+$.

3.2 Monoidal transducers as functors

In the rest of this paper we fix a countable *input alphabet* A and an *output monoid* $(M, \epsilon, \otimes)$. To differentiate between elements of A^* and elements of M, we write the former with Latin letters ($a, b, c, \dots$ for letters and $u, v, w, \dots$ for words) and the latter with Greek letters ($\alpha, \beta, \gamma, \dots$ for generating elements and $\upsilon, \nu, \omega, \dots$ for general elements). In particular the empty word over A is denoted e while the unit of M is still written ϵ. We now define our main object of study, M-monoidal transducers.

▶ **Definition 22** (monoidal transducer). *A* monoidal transducer *is a tuple* $(S, (\upsilon_0, s_0), t, (- \odot a)_{a \in A})$ *where* S *is a set of* states*;* $(\upsilon_0, s_0) \in M \times S \sqcup \{\bot\}$ *is the (possibly undefined) pair of the* initialization value *and* initial state*;* $t : S \to M \sqcup \{\bot\}$ *is the partial* termination function*;* $s \odot a \in M \times S \sqcup \{\bot\}$ *for* $a \in A$ *may be undefined, and its two components,* $- \bigcirc a : S \to M \sqcup \{\bot\}$ *and* $- \cdot a : S \to S \sqcup \{\bot\}$*, are respectively called the partial* production function *and the partial* transition function *along* a*.*

▶ **Example 23.** Figure 1a is a graphical representation of a monoidal transducer that takes its input in the alphabet $A = \{a, b\}$ and has output in any monoid that is a quotient of Σ^* with $\Sigma = \{\alpha, \beta, \gamma\}$. Formally, it is given by $S = \{1, 2, 3\}$; $(\upsilon_0, s_0) = (\epsilon, 1)$; $t(1) = t(2) = t(3) = \alpha$; and finally $1 \odot a = (\gamma\alpha\beta, 2)$, $2 \odot a = (\gamma\beta\alpha, 1)$, $3 \odot a = (\gamma\alpha\beta, 3)$, $1 \odot b = 2 \odot b = (\alpha, 3)$ and $3 \odot b = \bot$.

To apply the framework of Section 2 we first need to model monoidal transducers as functors. We thus design a tailored output category that in particular matches the one that instantiates classical transducers when M is a free monoid [16, Section 4]. We write $\mathcal{T}_M$ for the monad on **Set** given by $\mathcal{T}_M X = M \times X + 1 = (M \times X) \sqcup \{\bot\}$ (in Haskell, this monad is the composite of the `Maybe` monad and a `Writer` monad). Its unit $\eta : \mathrm{Id} \Rightarrow \mathcal{T}_M$ is given by $\eta_X(x) = (\epsilon, x)$ and its multiplication $\mu : \mathcal{T}_M^2 \Rightarrow \mathcal{T}_M$ is given by $\mu_X((\upsilon, (\nu, x))) = (\nu\upsilon, x)$, $\mu_X((\upsilon, \bot)) = \bot$ and $\mu_X(\bot) = \bot$. Recall that the Kleisli category $\mathbf{Kl}(\mathcal{T}_M)$ for the monad $\mathcal{T}_M$ has sets for objects and arrows $X \multimap Y$ (notice the different symbol) those functions $f^\dagger : \mathcal{T}_M X \to \mathcal{T}_M Y$ such that $f^\dagger(\bot) = \bot$, $f^\dagger(\upsilon, x) = (\upsilon\nu, y)$ when $f^\dagger(\epsilon, x) = (\nu, y)$ and $f^\dagger(\upsilon, x) = \bot$ when $f(\epsilon, x) = \bot$: in particular, such an arrow is entirely determined by its restriction $f : X \to \mathcal{T}_M Y$, and we will freely switch between these two points of view for the sake of conciseness. The identity on X is then given by the identity function $\mathrm{id}_{\mathcal{T}_M X} = \eta_X^\dagger : \mathcal{T}_M X \to \mathcal{T}_M X$, and the composition of two arrows $X \multimap Y \multimap Z$ is given by the composition of the underlying functions $\mathcal{T}_M X \to \mathcal{T}_M Y \to \mathcal{T}_M Z$.

M-transducers are in one-to-one correspondance with $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-automata, i.e. functors $\mathcal{A} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ such that $\mathcal{A}(\mathtt{in}) = \mathcal{A}(\mathtt{out}) = 1$: $(S, (\upsilon_0, s_0), t, (- \odot a)_{a \in A})$ is modelled by the functor $\mathcal{A} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ given by $\mathcal{A}(\mathtt{st}) = S$, $\mathcal{A}(\triangleright) = * \mapsto (\upsilon_0, s_0) : 1 \to M \times S + 1$, $\mathcal{A}(w) = s \mapsto s \odot w = (((s \odot a_1) \odot^\dagger a_2) \odot^\dagger \cdots) \odot^\dagger a_n : S \to M \times S + 1$ for $w = a_1 \cdots a_n \in A^*$, and $\mathcal{A}(\triangleleft) = t : S \to M + 1 \cong M \times 1 + 1$.

▶ **Definition 24.** *We write* $\mathbf{Trans}_M$ *for the category of* $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$*-automata: the objects are* M*-transducers seen as functors and the morphisms are natural transformations between them. Given a* $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$*-language* $\mathcal{L} : \mathcal{O} \to \mathbf{Kl}(\mathcal{T}_M)$*, we write* $\mathbf{Trans}_M(\mathcal{L})$ *for the subcategory of* M*-transducers* $\mathcal{A} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ *that recognize* $\mathcal{L}$*, i.e. such that* $\mathcal{A} \circ \iota = \mathcal{L}$*.*

Under this correspondance, the language recognized by a transducer $(S, (\upsilon_0, s_0), t, \odot)$ is thus a function $L : A^* \to M + 1$ given by $L(w) = t^\dagger((\upsilon_0, s_0) \odot^\dagger w)$, and a morphism between two transducers $(S_1, (\upsilon_1, s_1), t_1, \odot_1)$ and $(S_2, (\upsilon_2, s_2), t_2, \odot_2)$ is a function $f : S_1 \to M \times S_2 + 1$ such that $f^\dagger(\upsilon_1, s_1) = (\upsilon_2, s_2)$, $t_1(s) = t_2^\dagger(f(s))$ and $f^\dagger(s \odot a) = f^\dagger(s) \odot^\dagger a$.

When $M = B^*$ for some alphabet B, M-transducers coincide with the classical notion of transducers and the minimal transducer is given by Definition 9 [6, 16]. To study the notion of minimal monoidal transducer, it is thus natural to try to follow this framework as well.

3.3 The initial and final monoidal transducers recognizing a function

To apply the framework of Section 2 to $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-automata, we need three ingredients in $\mathbf{Kl}(\mathcal{T}_M)$: countable copowers of 1, countable powers of 1, and a factorization system.

We start with the first ingredient, countable copowers of 1. Since **Set** has arbitrary coproducts, $\mathbf{Kl}(\mathcal{T}_M)$ has arbitrary coproducts as well as any Kleisli category for a monad over a category with coproducts does [18, Proposition 2.2]. Hence Theorem 12 applies:

▶ **Corollary 25** (initial transducer)**.** *For any $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$, $\mathbf{Trans}_M(\mathcal{L})$ has an initial object $\mathcal{A}^{init}(\mathcal{L})$ with state-set $S^{init} = A^*$, initial state $s_0^{init} = e$, initialization value $\upsilon_0^{init} = \epsilon$, termination function $t^{init}(w) = \mathcal{L}(\triangleright w \triangleleft)(*)$ and transition function $w \odot^{init} a = (\epsilon, wa)$. Given any other transducer $\mathcal{A} = (S, (\upsilon_0, s_0), t, \odot)$ recognizing $\mathcal{L}$, the unique transducer morphism $f : \mathcal{A}^{init}(\mathcal{L}) \Longrightarrow \mathcal{A}$ is given by the function $f : A^* \to M \times S + 1$ such that $f(w) = \mathcal{A}(\triangleright w)(*) = (\upsilon_0, s_0) \odot^\dagger w$.*

Similarly, to get a final transducer in $\mathbf{Trans}_M(\mathcal{L})$ for some $\mathcal{L}$, Theorem 12 tells us that it is enough for $\mathbf{Kl}(\mathcal{T}_M)$ to have all countable powers of 1. This is in particular what happens for classical transducers, when M is a free monoid [16, Lemma 4.7]. Hence we study conditions on the monoid M for $\mathbf{Kl}(\mathcal{T}_M)$ to have these powers.

▶ **Theorem 26.** *If M is both left-cancellative up to invertibles on the left and right-coprime-cancellative, and all non-empty countable subsets of M have a unique left-gcd up to invertibles on the right, then $\mathbf{Kl}(\mathcal{T}_M)$ has all countable powers of 1. This is moreover a necessary condition as soon as M is right-noetherian.*

In practice, to build the final transducer we need some additional technical tools hidden in the proof of Theorem 26. Given a countable set I, we consider partial functions $\Lambda : I \to M+1$. We write $\bot^I$ for the nowhere defined function $i \mapsto \bot$ and $(M+1)^I_* = (M+1)^I - \{\bot^I\}$ for the set of partial functions that are defined somewhere. If $I \subseteq J$, $(M+1)^I_*$ may thus be identified with the subset of partial functions of $(M+1)^J_*$ that are undefined on $J - I$. We extend the product $\otimes : M^2 \to M$ of M to a function $M \times (M+1)^I_* \to (M+1)^I_*$ by setting $(\upsilon \otimes \Lambda)(i) = \upsilon \otimes \Lambda(i)$ for $i \in I$ such that $\Lambda(i) \neq \bot$ and $(\upsilon \otimes \Lambda)(i) = \bot$ otherwise.

$\mathbf{Kl}(\mathcal{T}_M)$ has all countable powers of 1 if and only if there are two functions $\mathrm{lgcd} : (M+1)^{\mathbb{N}}_* \to M$ and $\mathrm{red} : (M+1)^{\mathbb{N}}_* \to (M+1)^{\mathbb{N}}_*$ that uniquely decompose a partial function into its left-gcd and the corresponding left-coprime reduced function: they satisfy that

- $\Lambda = \mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda)$ for all $\Lambda \in (M+1)^{\mathbb{N}}_*$;
- $\upsilon\,\mathrm{red}(\Gamma) = \nu\,\mathrm{red}(\Lambda)$ implies $\upsilon = \nu$ and $\mathrm{red}\,\Gamma = \mathrm{red}\,\Lambda$ for all $\Gamma, \Lambda \in (M+1)^{\mathbb{N}}_*$ and $\upsilon, \nu \in M$.

We also write $\mathrm{lgcd}(\bot^{\mathbb{N}}) = \bot$, $\mathrm{red}(\bot^{\mathbb{N}}) = \bot^{\mathbb{N}} = \bot$ and do not distinguish between $(\bot, \bot)$ and $\bot$. For every countable I, lgcd and red can be extended to $(M+1)^I_*$ as I may be embedded into $\mathbb{N}$. We then get $\prod_I 1 = \mathrm{Irr}(I, M) = \{\mathrm{red}(\Lambda) \mid \Lambda \in (M+1)^I_*\} \subseteq (M+1)^I_*$.

▶ **Corollary 27.** *When the functions* lgcd *and* red *satisfying the two conditions above exist, the final transducer $\mathcal{A}^{final}(\mathcal{L})$ recognizing a $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$ exists and has state-set $S^{final} = \mathrm{Irr}(A^*, M)$, initial state $s_0^{final} = \mathrm{red}\,\mathcal{L}$, initialization value $\upsilon_0^{final} = \mathrm{lgcd}\,\mathcal{L}$, termination function $t^{final}(\Lambda) = \Lambda(e)$ and transition functions $\Lambda \odot^{final} a = (\mathrm{lgcd}(a^{-1}\Lambda), \mathrm{red}(a^{-1}\Lambda))$ where we write $(a^{-1}\Lambda)(w) = \Lambda(aw)$ for $a \in A$. Given any other transducer $\mathcal{A} = (S, (\upsilon_0, s_0), t, \odot)$ recognizing $\mathcal{L}$, the unique transducer morphism $f : \mathcal{A} \Longrightarrow \mathcal{A}^{final}(\mathcal{L})$ is given by the function $f : S \to M \times \mathrm{Irr}(A^*, M) + 1$ such that $f(s) = (\mathrm{lgcd}\,\mathcal{L}_s, \mathrm{red}\,\mathcal{L}_s)$ where $\mathcal{L}_s(\triangleright w \triangleleft)(*) = \mathcal{A}(w \triangleleft)(s)$ is the function recognized by $\mathcal{A}$ from the state s.*

▶ **Example 28.** When M is a group it is cancellative (because every element is invertible) and all countable families have a unique left-gcd up to invertibles on the right (ϵ itself) hence Theorem 26 applies and $\mathbf{Trans}_M(\mathcal{L})$ always has a final object. The same is true when M is a trace monoid (the left-gcd then being the longest common prefix, whose existence is guaranteed by [9, Proposition 1.3]).

Conversely, the monoids given by join semi-lattices are not left-cancellative up to invertibles in general. In $\mathbb{R}_+$ for instance, there are ways to define the functions lgcd and red but they may not satisfy that $\mathrm{red}(\upsilon\Lambda) = \mathrm{red}\,\Lambda$ (which should be implied by $\upsilon\,\mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda) = \upsilon\Lambda = \mathrm{lgcd}(\upsilon\Lambda)\,\mathrm{red}(\upsilon\Lambda)$). This is expected, as there may be several non-isomorphic ways to minimize transducers with outputs in these monoids, which is incompatible with the framework of Definition 9.

Theorem 26 provides sufficient conditions that are reminiscent of those developed in [12] for the minimization of monoidal transducers. These conditions are stronger than ours but still similar: the output monoid is assumed to be both left- and right-cancellative, which in particular implies the unicity up to invertibles on the right of the left-gcd whose existence is also assumed. They do only require the existence of left-gcds for finite families (whereas we ask for left-gcds of countable families), which would not be enough for our sake since the categorical framework also encompasses the existence of minimal (infinite) automata for non-regular languages, but in practice our algorithms will only use binary left-gcds as well. We conjecture that, when only those binary left-gcds exist, the existence of a unique minimal transducer is explained categorically by the existence of a final transducer in the category of transducers whose states all recognize functions that are themselves recognized by finite transducers. Where the two sets of conditions really differ is in the conditions required for the termination of the algorithms: where we will require right-noetherianity of M, they require that if some ν left-divides both some ω and $\upsilon\omega$ for some υ, then ν should also left-divide $\upsilon\nu$. This last condition leads to better complexity bounds than right-noetherianity, but misses some otherwise simple monoids that satisfy right-noetherianity, e.g. $\{\alpha, \beta\}^*$ but where we also let α and β^2 commute. Conversely, Gerdjikov's main non-trivial example, the tropical monoid $(\mathbb{R}_+, 0, +)$, is not right-noetherian. It can still be dealt with in our context by considering submonoids (finitely) generated by the output values of a finite transducer's transitions, these monoids themselves being right-noetherian.

3.4 Factorization systems

The last ingredient we need in order to be able to apply the framework of Section 2 is a factorization system on $\mathbf{Trans}_M(\mathcal{L})$. By Lemma 8, it is enough to find one on $\mathbf{Kl}(\mathcal{T}_M)$. When M is a free monoid, [16, Section 4.5], provides such a factorization system for which the minimal $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-automaton recognizing a language is the usual notion of minimal (non-monoidal) transducer. We thus simply generalize this factorization system to arbitrary output monoids. Define the classes of $\mathbf{Kl}(\mathcal{T}_M)$-morphisms Surj, Inj, Inv and Tot as follows: for $f : X \to M \times Y + 1$ and writing $f_1 : X \to M + 1$ for its projection on M and $f_2 : X \to Y + 1$ for its projection on Y, we let $f \in \mathrm{Surj}$ whenever f_2 is surjective on Y, and say f is surjective; let $f \in \mathrm{Inj}$ whenever f_2 is injective when corestricted to Y, and say f is injective; let $f \in \mathrm{Tot}$ whenever it is total, i.e. $f(x) \neq \bot$ for all $x \in X$; and let $f \in \mathrm{Inv}$ whenever it only produces invertible elements, i.e. whenever $f_1(X) \subseteq M^\times + 1$. Then,

▶ **Lemma 29.** $(\mathcal{E}, \mathcal{M}) = (\mathrm{Surj}, \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot})$ *is a factorization system on* $\mathbf{Kl}(\mathcal{T}_\mathcal{M})$.

Theorem 12 and Proposition 10 show that $(\mathcal{E}, \mathcal{M})$ indeed gives rise to a useful notion of minimal transducer.

▶ **Corollary 30.** *When* $\mathbf{Kl}(\mathcal{T}_M)$ *has all countable powers of* 1*, the* $(\mathcal{E},\mathcal{M})$*-minimal transducer recognizing a* $(\mathbf{Kl}(\mathcal{T}_M),1,1)$*-language* $\mathcal{L}$ *is well-defined and has state-set* $S^{min} = \{\mathrm{red}(w^{-1}\mathcal{L}) \mid w \in A^*\} \cap (M+1)_*^{A^*}$*, initial state* $s_0^{min} = \mathrm{red}\,\mathcal{L}$*, initialization value* $v_0^{min} = \mathrm{lgcd}\,\mathcal{L}$*, termination function* $t^{min}(\Lambda) = \Lambda(e)$ *and transition functions* $\mathrm{red}(w^{-1}\mathcal{L}) \odot^{min} a = (\mathrm{lgcd}((wa)^{-1}\mathcal{L}), \mathrm{red}((wa)^{-1}\mathcal{L}))$*. It is characterized by the property that all its states are reachable from the initial state and recognize distinct left-coprime functions.*

▶ **Example 31.** When $M = \{\alpha,\beta,\gamma\}^*$ is the free monoid, the transducer of Figure 1a is minimal for the function it recognizes. But when we allow α and β to commute in M this transducer is not minimal anymore because the states 1 and 2 recognize the same left-coprime function and need to be merged. If we also allow α to commute with γ then the resulting transducer (where the states 1 and 2 have been merged) is again not minimal, because the function recognized by the merged state (and previously by the states 1 and 2) is not left-coprime: it is left-divisible by α.

More interestingly, $(\mathrm{Surj}\cap\mathrm{Inj}\cap\mathrm{Tot},\mathrm{Inv})$ and $(\mathrm{Surj}\cap\mathrm{Inj},\mathrm{Tot}\cap\mathrm{Inv})$ are also factorization systems on $\mathbf{Kl}(\mathcal{T}_M)$, and since $\mathrm{Surj}\cap\mathrm{Inv}\cap\mathrm{Tot} \subset \mathrm{Surj}\cap\mathrm{Inj} \subset \mathrm{Surj}$, together with $(\mathcal{E},\mathcal{M})$ they thus form what is called a *quaternary factorization system*. In practice, this means that the computation of any morphism f in $\mathbf{Kl}(\mathcal{T}_M)$ can be uniquely factored into four orthogonal parts, as $f = f_4 \circ f_3 \circ f_2 \circ f_1$: f_1 is the part that forgets some inputs (it belongs to $\mathrm{Surj}\cap\mathrm{Inj}\cap\mathrm{Inv}$ but need not belong to Tot), f_2 the one that produces non-invertible elements of the output monoid (it belongs to $\mathrm{Surj}\cap\mathrm{Inj}\cap\mathrm{Tot}$ but need not belong to Inv), f_3 the one that merges some inputs together (it belongs to $\mathrm{Surj}\cap\mathrm{Inv}\cap\mathrm{Tot}$ but need not belong to Inj) and f_4 embeds the result into a bigger set (it belongs to $\mathrm{Inv}\cap\mathrm{Inj}\cap\mathrm{Tot}$ but need not belong to Surj).

In particular, for a $(\mathbf{Kl}(\mathcal{T}_M),1,1)$-automaton $\mathcal{A}$ recognizing a language $\mathcal{L}$, the final arrow $\mathrm{Reach}\,\mathcal{A} \multimap \mathcal{A}^{final}(\mathcal{L})$ can be factored in this way into $\mathrm{Reach}\,\mathcal{A} \multimap\!\!\!\!\rightarrow \mathrm{Total}\,\mathcal{A} \multimap\!\!\!\!\rightarrow \mathrm{Prefix}\,\mathcal{A} \multimap\!\!\!\!\rightarrow \mathrm{Min}\,\mathcal{L} \rightarrowtail\!\!\!\!\multimap \mathcal{A}^{final}(\mathcal{L})$, and we retrieve the four main steps of every algorithm that minimizes (monoidal or non-monoidal) transducers [6, 12]. In practice, minimizing $\mathcal{A} = (S,(u_0,s_0),t,\odot)$ indeed involves computing $\mathrm{Reach}\,\mathcal{A}$, $\mathrm{Total}\,\mathcal{A}$, $\mathrm{Prefix}\,\mathcal{A}$ and $\mathrm{Min}\,\mathcal{L}$ one after the other:

- $\mathrm{Reach}\,\mathcal{A}$ has state-set the set of states in S that are reachable from s_0;
- $\mathrm{Total}\,\mathcal{A}$ has state-set S' the set of states in S that recognize a function defined for at least one input word (in particular if $\mathcal{A}$ recognizes $\bot^{A^*}$ then (v_0,s_0) is set to $\bot$);
- $\mathrm{Prefix}\,\mathcal{A} = (S',(v_0\,\mathrm{lgcd}(\mathcal{L}_{s_0}),s_0),t',\odot')$, where $\mathcal{L}_s$ is the function recognized from a state $s \in S$ in $\mathcal{A}$, is obtained from $\mathrm{Total}\,\mathcal{A}$ by setting $t'(s) = \mathrm{lgcd}(\mathcal{L}_s)^{-1}t(s)$ and $s \odot' a = (\mathrm{lgcd}(\mathcal{L}_s)^{-1}(s \circ a)\,\mathrm{lgcd}(\mathcal{L}_{s\cdot a}), s\cdot a)$;
- $\mathrm{Obs}(\mathrm{Reach}\,A) \cong \mathrm{Min}\,\mathcal{L}$ is obtained from $\mathrm{Prefix}\,\mathcal{A}$ by merging two states s_1 and s_2 whenever they recognize functions that are equal up to invertibles on the left in $\mathrm{Prefix}\,\mathcal{A}$, that is when $\mathrm{red}(\mathcal{L}_{s_1}) = \mathrm{red}(\mathcal{L}_{s_2})$ in $\mathcal{A}$.

▶ **Example 32.** Starting from the monoidal transducer $\mathcal{A}$ of Figure 1a considered within the output monoid where we allow α to commute with both β and γ, $\mathrm{Reach}\,\mathcal{A} = \mathcal{A}$ (every state is reachable from the initial state), $\mathrm{Total}\,\mathcal{A} = \mathcal{A}$ (every state produces an output when following at least one input word, e.g. the empty input word), $\mathrm{Prefix}\,\mathcal{A}$ is obtained from $\mathcal{A}$ by pulling the output letter α from the terminal transitions to the initial transition (because α commutes with every output word hence left-divides the functions recognized from each state), and $\mathrm{Min}\,\mathcal{L}$ is then obtained from $\mathrm{Prefix}\,\mathcal{A}$ by merging the states 1 and 2 (they recognize the same function because $\gamma\alpha\beta = \gamma\beta\alpha$).

4 Active learning of minimal monoidal transducers

Let M be a monoid satisfying the conditions of Theorem 26 and consider now a function $A^* \to M+1$ seen as a $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$. Theorem 12 tells us that the minimal M-transducer recognizing $\mathcal{L}$ exists, is unique up to isomorphism and is given by Corollary 30, but does not tell us whether this minimal transducer is computable. For this to hold we need that the product in M, the left-gcd of two elements in M – written $\wedge$ – and the function LeftDivide – that takes as input $\delta, \upsilon \in M$ and outputs a ν such that $\upsilon = \delta\nu$ or fails if there is none – be all computable, and that equality up to invertibles on the left be decidable (and that the corresponding invertible be computable as well). We extend these operations to $M+1$ by means of $u\bot = \bot u = \bot$, $u \wedge \bot = \bot \wedge u = \bot$ and $\textsc{LeftDivide}(\delta, \bot) = \bot$. For the computations to terminate we additionally require that $\operatorname{Min}\mathcal{L}$ have finite state-set and M be right-noetherian, so that $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ is noetherian for the factorization system $(\mathcal{E}, \mathcal{M})$ of Lemma 29:

▶ **Lemma 33.** *An object X of $\mathbf{Kl}(\mathcal{T}_M)$ is $\mathcal{M}$-noetherian if and only if it is a finite set, in which case* $\operatorname{length}_{\mathcal{M}}(m : Y \rightarrowtail\!\!\!\!\circ\!\!\rightarrow X) = |X| - |Y|$*, and it is $\mathcal{E}$-artinian if and only if it is a finite set and either M is right-noetherian or $X = \varnothing$, in which case* $\operatorname{colength}_{\mathcal{E}}(e : X \twoheadrightarrow\!\!\!\!\circ\!\!\rightarrow Y) = |X| - |Y| + \sum_{e(x)=(\upsilon,y)} \operatorname{rk}\upsilon$.

The categorical framework of Section 2 can be extended with an abstract minimization algorithm [2]. With the output category described in Section 3 this instantiates in particular Gerdjikov's algorithm for minimizing monoidal transducers [12], and even shows that the latter is still valid under the conditions discussed in Section 3.3 and terminates as soon as M is right-noetherian. However, we focus here on a second way to compute the minimal transducer recognizing $\mathcal{L}$, namely learning it through membership and equivalence queries, that is relying on a function $\textsc{Eval}_{\mathcal{L}}$ that outputs the value of $\mathcal{L}$ on input words and a function $\textsc{Equiv}_{\mathcal{L}}$ that checks whether the hypothesis transducer is $\operatorname{Min}\mathcal{L}$ or outputs a counterexample otherwise. The FunL* algorithm described in Section 2.3 instantiates such an algorithm, that terminates as soon as $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ is $(\mathcal{E}, \mathcal{M})$-noetherian. We now give a practical description of this categorical algorithm: we explain how to keep track of the minimal biautomaton and how to check whether $\epsilon^{min}_{Q,T}$ is in $\mathcal{E}$ and $\mathcal{M}$. This is summarized by Algorithm 2 in the appendix.

The algorithm for learning the minimal monoidal transducer recognizing $\mathcal{L}$ is very similar to Vilar's algorithm (described in Section 1), the main difference being that the longest common prefix is now the left-gcd and that, in some places, testing for equality is now testing for equality up to invertibles on the left. It maintains two sets Q and T that are respectively prefix-closed and suffix-closed, and tables $\Lambda : Q \times (A \cup \{e\}) \to M+1$ and $R : Q \times (A \cup \{e\}) \times T \to M+1$. They satisfy that, for all $a \in A \cup \{e\}$, $\Lambda(q,a)R(q,a,t) = \mathcal{L}(\triangleright qat \triangleleft)$ and $R(q,a,\cdot)$ is left-coprime, hence $\Lambda(q,a)$ is a left-gcd of $(\mathcal{L}(\triangleright qat\triangleleft))_{t\in T}$. The algorithm then extends Q and T until some closure and consistency conditions are satisfied, and builds a hypothesis transducer $\mathcal{H}(Q,T)$ using Λ and R: its state-set S can be constructed by, starting with $e \in Q$, picking as many $q \in Q$ such that $R(q,e,\cdot)$ is not $\bot^T$ and such that, for any other $q' \in S$, $R(q,e,\cdot)$ and $R(q',e,\cdot)$ are not equal up to invertibles on the left; it then has initial state $e \in Q$, initialization value $\Lambda(e,e)$, termination function $t = q \in S \mapsto R(q,e,e)$ and transition functions given by $q \odot a = (\textsc{LeftDivide}(\Lambda(q,e), \Lambda(q,a))\chi, q')$ for $q, q' \in S$ such that $R(q,a,\cdot) = \chi R(q',e,\cdot)$. The algorithm then adds the counter-example given by $\textsc{Equiv}_{\mathcal{L}}(\mathcal{H}(Q,T))$ to Q and builds a new hypothesis automaton until no counter-example is returned and $\mathcal{H}(Q,T) = \operatorname{Min}\mathcal{L}$.

Closure issues happen when $\epsilon^{min}_{Q,T}$ is not in $\mathcal{E} = \mathrm{Surj}$, that is when there is a $qa \in QA$ such that $R(q, a, \cdot) \neq \chi R(q', e, \cdot)$ for every other $q' \in Q$ and $\chi \in M^\times$, and in that case qa should be added to Q. Consistency issues happen when the $\mathcal{E}$-factor of $\epsilon^{min}_{Q,T}$ is not in $\mathcal{M} = \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot}$, i.e. if it is not in Tot, in Tot but not in $\mathrm{Inv} \cap \mathrm{Tot}$, or in $\mathrm{Inv} \cap \mathrm{Tot}$ but not in $\mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot}$: the quaternary factorization system described in Section 3.4 thus also explains the different kinds of consistency issues we may face. In practice, there is hence a consistency issue if there is an $at \in AT$ such that respectively: either there is a $q \in Q$ such that $R(q, a, t) \neq \bot$ but $R(q, e, T) = \bot^T$; or there is a $q \in Q$ such that $\Lambda(q, e)$ does not left-divide $\Lambda(q, a)R(q, a, t)$; or there are some $q, q' \in Q$ and $\chi \in M^\times$ such that $R(q, e, T) = \chi R(q', e, T)$ but $\textsc{LeftDivide}(\Lambda(q, e), \Lambda(q, a)R(q, a, t)) \neq \chi \textsc{LeftDivide}(\Lambda(q', e), \Lambda(q', a)R(q', a, t))$. In each of these cases at should be added to T.

The number of updates to Q and T, hence in particular of calls to $\textsc{Equiv}_{\mathcal{L}}$, is bounded linearly by the the number of states in $\mathrm{Min}\,\mathcal{L}$ and the sum of the ranks of the left-gcds of the functions recognized by each of these states (although this latter quantity is not necessarily finite). Our algorithm also differs from Vilar's original one in a small additional way: the latter also keeps track of the left-gcds of every $\Lambda(q, \tilde{a})$ where $\tilde{a}$ ranges over $A \cup \{e\}$ and $q \in Q$ is fixed, and checks for consistency issues accordingly. This is a small optimization of the algorithm that does not follow immediately from the categorical framework. In Section 1 we thus actually provided an example run of our version of the algorithm when the output monoid is a free monoid. This also provides example runs of our algorithm for non-free output monoids, as quotienting the output monoid will only remove closure and consistency issues and make the run simpler. For instance letting α commute with β for the transducer of Figure 1a would have removed the closure issue and the need to add a to Q while learning the corresponding monoidal transducer, and letting α also commute with γ would have removed the first consistency issue to arise and the need to add a to T.

5 Summary and future work

In this work, we instantiated Colcombet, Petrişan and Stabile's active learning categorical framework with monoidal transducers. We gave some simple sufficient conditions on the output monoid for the minimal transducer to exist and be unique, which in particular extend Gerdjikov's conditions for minimization to be possible [12]. Finally, we described what the active learning algorithm of the categorical framework instantiated to in practice under these conditions, relying in particular on the quaternary factorization system in the output category.

This work was mainly a theoretical excursion and was not motivated by practical examples where monoidal transducers are used. One particular application that could be further explored is the use of transducers with outputs in trace monoids (and their learning) to programatically schedule jobs, as mentioned in the introduction. We also leave the search for other interesting examples for future work.

Some intermediate results of this work go beyond what the categorical framework currently provides and could be generalized. The use of a quaternary factorization system (or any n-ary factorization system) would split the algorithms into several substeps that should be easier to work with. Here our factorization systems seemed to arise as the image of the factorization system on **Set** through the monad $\mathcal{T}_M$; generalizing this to other monads could provide meaningful examples of factorization systems in any Kleisli category. Finally, we mentioned in Section 3.3 that a problem with the current framework is that it may only account for the minimization of both finite and infinite transition systems at the same time,

and conjectured that we could restrict to only the finite case by working in a subcategory of well-behaved transducers: this subcategory is perhaps an instance of a general construction that has its own version of Theorem 12, so as to still have a generic way to build the initial, final and minimal objects.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987. doi:10.1016/0890-5401(87)90052-6.

2 Quentin Aristote. Functorial approach to minimizing and learning deterministic transducers with outputs in arbitrary monoids, November 2023. URL: https://ens.hal.science/hal-04172251v2.

3 Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra Learning via Duality. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 62–79, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17127-8_4.

4 F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. In M. Bonuccelli, P. Crescenzi, and R. Petreschi, editors, *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 54–62, Berlin, Heidelberg, 1994. Springer. doi:10.1007/3-540-57811-0_6.

5 Francesco Bergadano and Stefano Varricchio. Learning Behaviors of Automata from Multiplicity and Equivalence Queries. *SIAM Journal on Computing*, 25(6):1268–1280, December 1996. doi:10.1137/S009753979326091X.

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

7 Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. Learning automata and transducers: A categorical approach, October 2020. doi:10.48550/arXiv.2010.13675.

8 Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. Learning Automata and Transducers: A Categorical Approach. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2021.15.

9 Robert Cori and Dominique Perrin. Automates et commutations partielles. *RAIRO. Informatique théorique*, 19(1):21–32, 1985. doi:10.1051/ita/1985190100211.

10 Jason Eisner. Simpler and more general minimization for weighted finite-state automata. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology – Volume 1*, NAACL '03, pages 64–71, USA, May 2003. Association for Computational Linguistics. doi:10.3115/1073445.1073454.

11 Charles N. Fischer, Ron K. Cytron, and Richard J. LeBlanc. *Crafting a Compiler*. Crafting a Compiler with C. Addison-Wesley, Boston, 2010.

12 Stefan Gerdjikov. A General Class of Monoids Supporting Canonisation and Minimisation of (Sub)sequential Transducers. In Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira, editors, *Language and Automata Theory and Applications*, Lecture Notes in Computer Science, pages 143–155, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-77313-1_11.

13 Ronald M. Kaplan and Martin Kay. Regular Models of Phonological Rule Systems. *Computational Linguistics*, 20(3):331–378, 1994. URL: https://aclanthology.org/J94-3001.

14 Kevin Knight and Jonathan May. Applications of Weighted Automata in Natural Language Processing. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 571–596. Springer, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-01492-5_14.

15 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1978. doi:10.1007/978-1-4757-4721-8.

16 Daniela Petrişan and Thomas Colcombet. Automata Minimization: A Functorial Approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, March 2020. doi:10.23638/LMCS-16(1:32)2020.

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

18 Jenö Szigeti. On limits and colimits in the Kleisli category. *Cahiers de topologie et géométrie différentielle*, 24(4):381–391, 1983. URL: http://www.numdam.org/item/?id=CTGDC_1983__24_4_381_0.

19 Henning Urbat and Lutz Schröder. Automata Learning: An Algebraic Approach. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 900–914, New York, NY, USA, July 2020. Association for Computing Machinery. doi:10.1145/3373718.3394775.

20 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. Learning Automata with Side-Effects. In Daniela Petrişan and Jurriaan Rot, editors, *Coalgebraic Methods in Computer Science*, Lecture Notes in Computer Science, pages 68–89, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-57201-3_5.

21 Juan Miguel Vilar. Query learning of subsequential transducers. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, Lecture Notes in Computer Science, pages 72–83, Berlin, Heidelberg, 1996. Springer. doi:10.1007/BFb0033343.

A Why Vilar's algorithm is not enough for monoidal transducers

▶ **Lemma 34.** *Let $A = \{a\}$, $\Sigma = \{\alpha, \beta, \gamma\}$, let $\Sigma^*/\!\sim$ be the monoid given by the presentation $\langle \Sigma \mid \alpha\beta = \beta\alpha \rangle$ and let $\pi : \Sigma^* \to \Sigma^*/\!\sim$ be the corresponding quotient. Consider the function $f : A^* \to \Sigma^*/\!\sim$ that maps a^n to $\alpha^n\beta^n\gamma = (\alpha\beta)^n\gamma$.*

f is recognized by a finite transducer with outputs in $\Sigma^*/\!\sim$, yet learning a transducer that recognizes any function $f' : A^* \to \Sigma^*$ such that $f = f' \circ \pi$ with Vilar's algorithm will never terminate if the oracle replies to the membership query for a^n with $\alpha^n\beta^n\gamma \in \Sigma^*$ (which differs from $(\alpha\beta)^n\gamma$ in Σ^* but not in $\Sigma^*/\!\sim$).

Proof. f is recognized by the $(\Sigma^*/\!\sim)$-transducer (Definition 22) with one state s that is initial, initial value $\upsilon_0 = \epsilon$, transition function $a \odot s = (\alpha\beta, s)$ and termination function $t(s) = \gamma$.

Consider now the run of Vilar's algorithm (Algorithm 2 with $M = \Sigma^*$ and $M^\times = \{e\}$) with the oracle answering the membership query for a^n with $f'(a^n) = \alpha^n\beta^n\gamma$. We start with $Q = T = \{e\}$, $\Lambda(e) = \gamma$, $\Lambda(a) = \alpha\beta\gamma$ and $R(e, e) = R(a, e) = \epsilon$, where $\Lambda(w)$ for $w \in Q \cup QA$ is the longest common prefix of the $\{f'(wt) \mid t \in T\}$ and $R(w, t)$ is the suffix such that $f'(wt) = \Lambda(w)R(w, t)$.

Since $\Lambda(e) = \gamma$ is not a prefix of $\Lambda(a) = \alpha\beta\gamma$, there is a consistency issue and we add a to T. Taking $n = 0$, we are now in the configuration C_n given by $Q = Q_n = \{a^k \mid k \leq n\}$, $T = \{e, a\}$, and $\Lambda(a^k) = \alpha^k$, $R(a^k, e) = \beta^k\gamma$ and $R(a^k, a) = \alpha\beta^{k+1}\gamma$ for every $k \leq n + 1$ (since the oracle replies to the membership query for a^k with $\alpha^k\beta^k\gamma$ and to that for $a^k a$ with $\alpha^{k+1}\beta^{k+1}\gamma$).

Suppose now we are in the configuration C_n for some $n \in \mathbb{N}$. Then there is a closure issue, since for all $k \leq n$, $R(a^k, e) = \beta^k\gamma \neq \beta^{n+1}\gamma = R(a^{n+1}, e)$. We thus add a^{n+1} to Q. To compute $\Lambda(a^{n+2})$, $R(a^{n+2}, e)$ and $R(a^{n+2}, a)$, we make a membership query for a^{n+3}: the oracle answers with $f'(a^{n+3}) = \alpha^{n+3}\beta^{n+3}\gamma$. The longest common prefix of

$f'(a^{n+2}) = \Lambda(a^{n+1})R(a^{n+1}, a) = \alpha^{n+2}\beta^{n+2}\gamma$ and $f'(a^{n+2}a) = \alpha^{n+3}\beta^{n+3}\gamma$ is thus α^{n+2}, and the corresponding suffixes are $R(a^{n+2}, e) = \beta^{n+2}\gamma$ and $R(a^{n+2}, a) = \alpha\beta^{n+3}\gamma$: we are now in the configuration C_{n+1}.

Hence the run of the algorithm never terminates and never even reaches an equivalence query, as it must first go through all the configurations C_n for $n \in \mathbb{N}$. ◀

B The learning algorithm

Algorithm 2 The FunL*-algorithm for monoidal transducers.

Input: $\text{Eval}_{\mathcal{L}}$ and $\text{Equiv}_{\mathcal{L}}$
Output: $\text{Min}_M(\mathcal{L})$

$Q = T = \{e\}$
for $a \in A \cup \{e\}$ **do**
  $\Lambda(e, a) = \text{Eval}_{\mathcal{L}}(a)$
  $R(e, a, e) = \epsilon$
end for
loop
  if there is a $qa \in QA$ such that $\forall q' \in Q, \chi \in M^\times, R(q, a, \cdot) \neq \chi R(q', e, \cdot)$ **then**
    add qa to Q
  else if there is an $at \in AT$ such that
  - **either** there is a $q \in Q$ such that $R(q, a, t) \neq \bot$ but $R(q, e, T) = \bot^T$;
  - **or** there is a $q \in Q$ such that $\Lambda(q, e)$ does not left-divide $\Lambda(q, a)R(q, a, t)$;
  - **or** there are $q, q' \in Q$ and $\chi \in M^\times$ such that $R(q, e, T) = \chi R(q', e, T)$ but $\text{LeftDivide}(\Lambda(q, e), \Lambda(q, a)R(q, a, t)) \neq \chi\text{LeftDivide}(\Lambda(q', e), \Lambda(q', a)R(q', a, t))$

  then
    add at to T
  else
    build $\mathcal{H}(Q, T) = (S, (\upsilon_0, s_0), t, \odot)$ given by:
  - $S \subseteq Q$ is built by starting with $e \in S$ and adding as many $q \in Q$ as long as $R(q, e, \cdot) \neq \bot$ and $\forall q' \in S, \chi \in M^\times, R(q, e, \cdot) \neq \chi R(q', e, \cdot)$;
  - $(\upsilon_0, s_0) = (\Lambda(e), e)$
  - $q \odot a = (\text{LeftDivide}(\Lambda(q, e), \Lambda(q, a))\chi, q')$ with $q \in S$, $\chi \in M^\times$ given by $R(q, a, \cdot) = \chi R(q', e, \cdot)$
  - $t(q) = R(q, e, e)$

    if $\text{Equiv}_{\mathcal{L}}(\mathcal{H}_{Q,T}(\mathcal{L}))$ outputs some counter-example w **then**
      add w and its prefixes to Q
    else
      return $\mathcal{H}(Q, T)$
    end if
  end if
  update Λ and R using $\text{Eval}_{\mathcal{L}}$
end loop

If $\mathcal{A}$ is a monoidal transducer, write $|\mathcal{A}|_{\mathtt{st}} = |\mathcal{A}(\mathtt{st})|$ and $\text{rk}(\mathcal{A}) = \sum_{s \in \mathcal{A}(\mathtt{st})} \text{rk}(\text{lgcd}(\mathcal{L}_s))$ (where $\mathcal{L}_s$ is the partial function recognized by $\mathcal{A}$ when $s \in \mathcal{A}(\mathtt{st})$ is chosen to be the initial state).

▶ **Theorem 35.** *Algorithm 2 is correct and terminates as soon as* $\operatorname{Min}\mathcal{L}$ *has finite state-set and* M *is right-noetherian. It makes at most* $3\left|\operatorname{Min}\mathcal{L}\right|_{\mathtt{st}} + \operatorname{rk}(\operatorname{Min}\mathcal{L})$ *updates to* Q *(Lines 8 and 14) and at most* $\operatorname{rk}(\operatorname{Min}\mathcal{L}) + \left|\operatorname{Min}\mathcal{L}\right|_{\mathtt{st}}$ *updates to* T *(Line 10).*