Functorial approach to minimizing and learning deterministic transducers with outputs in arbitrary monoids

Quentin Aristote

Functorial approach to minimizing and learning deterministic transducers with outputs in arbitrary monoids

Quentin Aristote

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. We also extend the framework with a categorical algorithm for minimizing transition systems, whose instantiation retrieves the algorithm for minimizing monoidal transducers but also extends the class of output monoids for which this algorithm is valid.

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 [16], linguistics [19], or natural language processing [20].

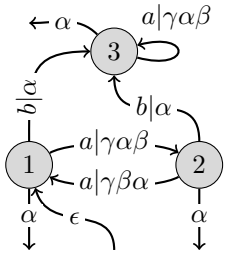
Two subsequential transducers are considered equivalent when they *recognize* the same *subsequential function*, that is if, given the same input, they always produce the same output. A natural question is thus whether there is a (unique) minimal transducer recognizing a given function (a transducer 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 the function: the minimal transducer can then for example be computed through minimization [9].

Active learning of transducers. Another method for computing a minimal transducer is to learn it through Vilar's algorithm [29], 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:

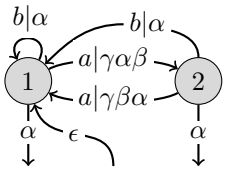
- *membership queries*: when queried with an input word, the oracle answers with the corresponding expected output word;
- *equivalence queries*: when queried with a *hypothesis transducer*, the oracle answers whether this transducer recognizes the target function, and, if not, provides a counter-example 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 two subsets $Q, T \subset A^*$ of prefixes and 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 they will be distinguished according to the output words that should be produced when starting from these states and reading a suffix in T. Informally, closure then holds when for any state $q \in Q$ and input letter $a \in A$ there is a candidate state $q' \in A$ that we can choose to build an a-transition from q to q'; consistency holds when for any $q \in Q$ and $a \in A$ all the states that can be chosen in such a way can be merged, 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 $QT \cup QAT$. Submitting it to the oracle we learn that this transducer is not the one we were 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.

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 that 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 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 independant of one another and may thus be run asynchronously: transducers with outputs in trace monoids could be used to programatically schedule some 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 [17], 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 [15], 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 $\Sigma^*/\sim$ (the idea of finding such an example was suggested by an anonymous reviewer whom the author thanks):

Lemma 1.1. *Let $A = \{a\}$, $\Sigma = \{\alpha, \beta, \gamma\}$, let $\Sigma/\sim$ be the monoid Σ^* with the additional assumption that $\alpha\beta = \beta\alpha$ 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 $\Sigma/\sim$).

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, related work 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 [10]. This framework encompasses both Angluin's and Vilar's algorithms, as well as a similar algorithm for

weighted automata [5, 6]. We use this specific framework because, while others exist, they either do not encompass transducers or require stronger assumptions [3, 27, 28].

In Section 3 we extend this framework with an abstract minimization algorithm. This is not the first categorical account of minimization nor the one that instantiates to the most efficient algorithms [14, 30], but it is the only one that matches the functorial framework of Colcombet, Petrişan and Stabile.

In Section 4 we then instantiate this framework to retrieve monoidal transducers as transition systems whose state-spaces live in a certain category (Section 4.2). Studying the existence of specific structures in this category — namely, powers of the terminal object (Section 4.3) and factorization systems (Section 4.4) — 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:

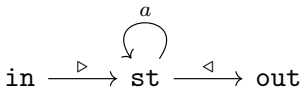
- the framework of Colcombet, Petrişan and Stabile is extended with a categorical minimization algorithm;
- necessary and sufficient conditions on the output monoid for the this framework to apply to monoidal transducers are given;
- these conditions mostly overlap those of Gerdjikov, but are nonetheless not equivalent: in particular, they extend the class of output monoids for which minimization is known to be possible, although without ensuring a similar complexity;
- practical details on the implementation of the abstract monoidal transducer-learning algorithm that results from the categorical framework are given;
- 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 automata minimization and learning

In this section we recall (and extend) the definitions and results of Colcombet, Petrişan and Stabile [11, 23]. We assume basic knowledge of category theory [21], but we also focus on the example of deterministic complete automata, on the counter-example of non-deterministic automata, and mention the examples of weighted automata and (classical) transducers, the latter being detailed and generalized in Section 4.

2.1. Automata and languages as functors

Definition 2.1 ($(\mathcal{C}, X, Y)$-automaton)**.** An automaton is seen as a functor from the *input category* $\mathcal{I}$ to an *output category* $\mathcal{C}$. The input category is the one freely generated by the diagram below, where a ranges in the *input alphabet* A: its objects are the vertices of the diagram and its morphisms are the paths between two vertices.



We then say that a functor $\mathcal{A} : \mathcal{I} \to \mathcal{C}$ is a $(\mathcal{C}, X, Y)$-*automaton* when $\mathcal{A}(\mathtt{in}) = X$ and $\mathcal{A}(\mathtt{out}) = Y$.

The input category represents the basic structure of automata as transition systems: $\mathtt{st}$ represents the state-space, $\triangleright$ the initial configuration, a the transitions along the corresponding letters, and $\triangleleft$ the output values associated to each state.

Example 2.2 (automata and transducers)**.** 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 $\mathbf{Rel}$ 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 $\mathcal{A}(\triangleleft) \subseteq S \times 1 \cong S$.

If $\mathbb{K}$ is a field we may see $\mathbb{K}$-weighted automata as functors $\mathcal{A}$ from $\mathcal{I}$ to the category of $\mathbb{K}$-vector-spaces, $\mathbb{K}\mathbf{Vec}$, such that $\mathcal{A}(\mathtt{in}) = \mathcal{A}(\mathtt{out}) = \mathbb{K}$, and if B is an output alphabet, we may see deterministic transducers as functors from $\mathcal{I}$ to the Kleisli category of free algebras for the monad $\mathcal{T}_{B^*} X = B^* \times X + 1$, $\mathbf{Kl}(\mathcal{T}_{B^*})$, such that $\mathcal{A}(\mathtt{in}) = \mathcal{A}(\mathtt{out}) = 1$. This last example will be detailed and generalized in Definition 4.9.

Definition 2.3 ($(\mathcal{C}, X, Y)$-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 $\mathtt{in}$ and $\mathtt{out}$. 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 2.4 (languages for classical automata)**.** 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.

Similarly, when $\mathcal{C}$ is $\mathbb{K}\mathbf{Vec}$ or $\mathbf{Kl}(B^* \times - + 1)$ the corresponding notions of languages are those respectively recognized by weighted automata and transducers.

Definition 2.5 (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 2.6 (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 $e \in \mathcal{E}$, $\mathcal{E} \cap \mathcal{M}$ is exactly the class Iso of isomorphisms of $\mathcal{C}$, $\mathcal{E}$ and $\mathcal{M}$ are both stable under composition, and for every commuting diagram

$$\begin{array}{ccc} X & \xrightarrow{e} & Y_1 \\ {\scriptstyle u}\downarrow & \swarrow_{d} & \downarrow{\scriptstyle v} \\ Y_2 & \xrightarrow[m]{} & Z \end{array}$$

where $m \in \mathcal{M}$ and $e \in \mathcal{E}$ there is a unique $d : Y_1 \to Y_2$ such that $u = d \circ e$ and $v = m \circ d$.

Example 2.7. (some factorization systems) In **Set** and $\mathbb{K}$**Vec**, surjective and injective functions and linear transformations form factorization systems $(\mathrm{Surj}, \mathrm{Inj})$ such that a map $f : X \to Y$ factors through its image $f(X) \subseteq Y$. In the Kleisli category for the monad $X \mapsto B^* \times X + 1$, we may define similarly a notion of surjection and injection that gives a meaningful factorization system. Again, this last example will be detailed and generalized in Section 4.4.

Finally, 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 $\{y \in Y \mid \exists x \in X, (x, y) \in R\} \subseteq Y$. We write this factorization system $(\mathrm{Surj}, \mathrm{Det} \cap \mathrm{Inj})$ (for surjections and deterministic injections).

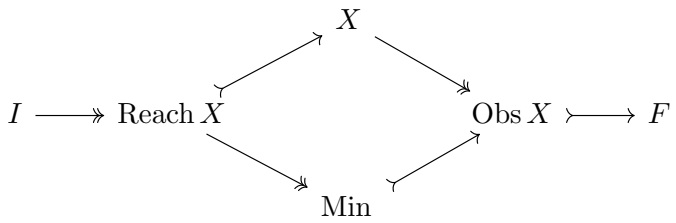
Lemma 2.8 (factorization system on **Auto**$(\mathcal{L})$ [23, Lemma 2.8])**.** *Given $\mathcal{L} : \mathcal{I} \to \mathcal{C}$ and $(\mathcal{E}, \mathcal{M})$ a factorization system on $\mathcal{C}$,* **Auto**$(\mathcal{L})$ *has a factorization given by those natural transformations whose components 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 2.9 (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 $I \twoheadrightarrow \text{Min} \rightarrowtail F$. For every object X we also define $\text{Reach}\, X$ and $\text{Obs}\, X$ by the $(\mathcal{E}, \mathcal{M})$-factorizations $I \twoheadrightarrow \text{Reach}\, X \rightarrowtail X$ and $X \twoheadrightarrow \text{Obs}\, X \rightarrowtail F$: when the morphism $I \to X$ is in $\mathcal{E}$ we say X is *reachable* and when the morphism $X \to F$ we say it is *observable*.

Proposition 2.10 ($(\mathcal{E}, \mathcal{M})$-minimality [23, Lemma 2.3])**.** *The minimal object of a category $\mathcal{C}$ is unique up to isomorphism, and is $(\mathcal{E}, \mathcal{M})$-minimal in the sense that it* $(\mathcal{E}, \mathcal{M})$-divides *every other object X, meaning we have commuting diagrams*



In particular, $\text{Min} \cong \text{Obs}(\text{Reach}\, X) \cong \text{Reach}(\text{Obs}\, X)$.

Example 2.11 (initial, final and minimal automata and transducers [23, Example 3.1])**.** Since **Set** is complete and cocomplete, the category of $(\mathbf{Set}, 1, 2)$-automata recognizing a language $\mathcal{L} : I_{A^*} \to \mathbf{Set}$ 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 2.7 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.

For the same reason, the category of $(\mathbb{K}\mathbf{Vec}, \mathbb{K}, \mathbb{K})$-automata recognizing a language $\mathcal{L} : \mathcal{I} \to \mathbb{K}\mathbf{Vec}$ also has a minimal object that corresponds to the minimal $\mathbb{K}$-weighted automaton recognizing $\mathcal{L}$ [25]. The example of transducers seen as automata over the Kleisli category for the monad $X \mapsto B^* \times X + 1$ is more involved: this category has all coproducts, but not necessarily products. Yet, it happens that 1 does have countable powers, and hence minimization is still possible for $(\mathbf{Kl}(\mathcal{T}), 1, 1)$-automata : with the right factorization system, we retrieve Choffrut's minimal transducer [9].

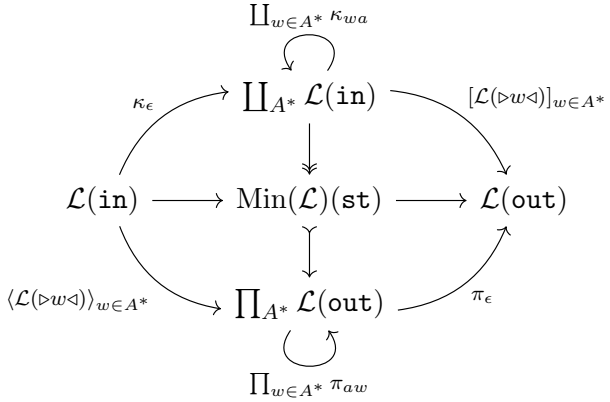
On the contrary, there is no good notion of a unique minimal non-deterministic automaton recognizing a regular ($(\mathbf{Rel}, 1, 1)$-)language $\mathcal{L}$. $\mathbf{Auto}(\mathcal{L})$ does have an initial and a final object: the initial automaton is the initial deterministic automaton recognizing $\mathcal{L}$, 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 2.7 has for state-set the set of suffixes of words in $\mathcal{L}$.

Notice how in Example 2.11 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 weighted automata and transducers, and generalizes as Theorem 2.12.

Theorem 2.12 ([23, Lemma 3.2])**.** *Given a language $\mathcal{L} : \mathcal{I} \to \mathcal{C}$ with A countable,*

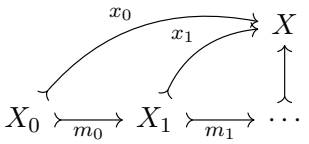
- *if $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathtt{in})$ then $\mathbf{Auto}(\mathcal{L})$ has an initial object $\mathcal{A}^{init}(\mathcal{L})$ with $\mathcal{A}^{init}(\mathcal{L})(\mathtt{st}) = \coprod_{A^*} \mathcal{L}(\mathtt{in})$;*
- *dually if $\mathcal{C}$ has all countable 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 $\mathcal{C}$ is equipped with a factorization system $(\mathcal{E}, \mathcal{M})$, $\mathbf{Auto}(\mathcal{L})$ has an $(\mathcal{E}, \mathcal{M})$-minimal object* $\operatorname{Min} \mathcal{L}$.

This is summarized by the diagram below, where κ and π are the canonical inclusion and projections.



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 2.13 ($\mathcal{E}$-artinian and $\mathcal{M}$-noetherian objects [10, 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 2.14 (co-$\mathcal{E}$- and $\mathcal{M}$-length)**.** 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 length (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 2.15. 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 $\mathrm{colength}_{\mathrm{Surj}}(X \twoheadrightarrow Y) = \mathrm{length}_{\mathrm{Inj}}(Y \rightarrowtail X) = |X - Y|$. Similarly, in $\mathbb{K}$**Vec** X has finite dimension if and only if it is Inj-noetherian iff it is Surj-artinian, and in that case for $Y \subseteq X$ we have $\mathrm{colength}_{\mathrm{Surj}}(X \twoheadrightarrow Y) = \mathrm{length}_{\mathrm{Inj}}(Y \rightarrowtail X) = \dim X - \dim Y$.

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 4.4, for which the co-$\mathcal{E}$- and -$\mathcal{M}$-length are computed in Lemmas 5.1 and 5.2.

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 2.12 applies.

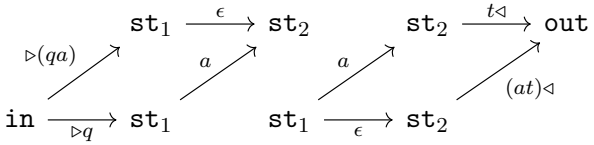
Our goal is to compute $\mathrm{Min}\,\mathcal{L}$ with the help of two oracles: $\textsc{Eval}_{\mathcal{L}}$, when queried with a word $w \in A^*$, outputs $\mathcal{L}(\triangleright w \triangleleft)$; while $\textsc{Equiv}_{\mathcal{L}}$, when queried with a $(\mathcal{C}, \mathcal{L}(\mathtt{in}), \mathcal{L}(\mathtt{out}))$-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 $\textsc{Equiv}_{\mathcal{L}}$: if it is accepted it must be $\mathrm{Min}\,\mathcal{L}$, otherwise the counter-example is added to Q and the algorithm loops over. The FunL* algorithm (Algorithm 1) 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 [10].

Instead of maintaining a table, it 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 diagram

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

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 $qa \in QA$ and the right one whenever $at \in AT$.



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 2.12 also holds for biautomata [10, 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: this encompasses respectively the closure and consistency conditions that need to hold in the L*-algorithm and its variants for the table that is maitained to be merged into a hypothesis automaton.

Theorem 2.16 ([10, Theorem 26])**.** *Algorithm 1 is correct. If* $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *is* $\mathcal{M}$*-noetherian and* $\mathcal{E}$*-artinian, the algorithm also terminates.*

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 2.14).

3. An algorithm for minimizing $(\mathcal{C},X,Y)$-automata

We now fix a $(\mathcal{C},X,Y)$-language $\mathcal{L}$ and a factorization system $(\mathcal{E},\mathcal{M})$ of $\mathcal{C}$, and assume that $\mathcal{C}$ has all the countable copowers of $\mathcal{L}(\mathtt{in})$ and all the countable powers of $\mathcal{L}(\mathtt{out})$ so that Theorem 2.12 applies and $\mathbf{Auto}(\mathcal{L})$ has initial and final objects $\mathcal{A}^{init}$ and $\mathcal{A}^{final}$.

While Colcombet, Petrişan and Stabile's framework provide a generic algorithm for learning $\operatorname{Min}\mathcal{L}$ (which we recalled in Section 2.3), it does not give a minimization algorithm, that is a way to compute $\operatorname{Min}\mathcal{L}$ given another automaton $\mathcal{A}$ recognizing $\mathcal{L}$: Proposition 2.10 only hints that $\operatorname{Min}\mathcal{L}$ can be computed in two steps, by first computing Reach and then Obs (or conversely).

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 qa 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 at 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

In this section we thus describe such an algorithm, and reuse the notations and arguments they used for their learning algorithm, making some of their key lemmas more explicit in our proofs. In Section 3.1 we give a basic version of the algorithm that does not require any more assumptions than those of the framework of Section 2. In Section 3.2 we show how it can be further detailed given some additional assumptions on the output category and the factorization system. Finally, in Section 3.3 we show how our algorithm precisely encompasses several well-known algorithms for minimizing different kinds of automata.

3.1. The basic algorithm

Let thus $\mathcal{A}$ be a $(\mathcal{C}, X, Y)$-automaton recognizing $\mathcal{L}$. Since $\mathcal{I}$ is self-dual, this automaton is entirely described by the dual $(\mathcal{C}^{\text{op}}, X^{\text{op}}, Y^{\text{op}})$-automaton $\mathcal{A}^{\text{op}} : \mathcal{I} \to \mathcal{C}^{\text{op}}$. But $(\mathcal{M}^{\text{op}}, \mathcal{E}^{\text{op}})$ is a factorization system in $\mathcal{C}^{\text{op}}$ and the dual of the initial object in $\mathcal{C}$ is the final object in $\mathcal{C}^{\text{op}}$, hence

Lemma 3.1 ([23, Remark 2.5])**.** *The dual of the* $(\mathcal{E}, \mathcal{M})$*-factorization* $\text{Obs}\,\mathcal{A}$ *in* $\mathbf{Auto}(\mathcal{L})$ *is the* $(\mathcal{M}^{\text{op}}, \mathcal{E}^{\text{op}})$*-factorization* $\text{Reach}\,\mathcal{A}^{\text{op}}$ *in* $\mathbf{Auto}(\mathcal{L}^{\text{op}})$*.*

Remark 3.2. Note that Lemma 3.1 does not mean that, for a fixed category $\mathcal{C}$ along with a factorization system $(\mathcal{E}, \mathcal{M})$ and the two objects X and Y, having an algorithm that computes the $(\mathcal{E}, \mathcal{M})$-factorization Reach for $(\mathcal{C}, X, Y)$-automata is equivalent to

having one that computes the $(\mathcal{E},\mathcal{M})$-factorization Obs for $(\mathcal{C},X,Y)$-automata as well. Indeed, $\mathcal{C}^{\mathrm{op}}$ and $\mathcal{C}$ itself may differ greatly, and even if $\mathcal{C}$ were self-dual (hence $\mathcal{C}=\mathcal{C}^{\mathrm{op}}$), $\mathcal{E}$ and $\mathcal{M}^{\mathrm{op}}$ or X and Y^{op} could still differ, meaning that computing the $(\mathcal{E},\mathcal{M})$- and $(\mathcal{M}^{\mathrm{op}},\mathcal{E}^{\mathrm{op}})$-factorizations would not necessarily be equivalent. For instance, **Rel** is self-dual but, as Example 3.13 shows, $\mathrm{Reach}\,\mathcal{A}$ may be computed for automata $\mathcal{A}:\mathcal{I}\to\mathbf{Rel}$ while $\mathrm{Obs}\,\mathcal{A}$ may not.

What this result says however is that getting a *generic* algorithm that computes Obs (one that is parametric in $\mathcal{C}$) is thus equivalent to getting one that computes Reach: we focus on the latter.

Definition 3.3. Given a set $Q\subseteq A^*$, write $e_Q:\coprod_Q\mathcal{L}(\mathtt{in})\twoheadrightarrow Q/\mathcal{A}(\mathtt{st})$ and $m_Q:Q/\mathcal{A}(\mathtt{st})\rightarrowtail\mathcal{A}(\mathtt{st})$ for the factorization of the arrow $[\mathcal{A}(\triangleright q)]_{q\in Q}:\coprod_Q\mathcal{L}(\mathtt{in})\to\mathcal{A}(\mathtt{st})$ in $\mathcal{C}$.

Notice how for $Q=A^*$ we get by Theorem 2.12 that $A^*/\mathcal{A}(\mathtt{st})=(\mathrm{Reach}\,\mathcal{A})(\mathtt{st})$, which is exactly the object we want to compute (along with its embedding into $\mathcal{A}(\mathtt{st})$, given by m_{A^*}). But we also want to compute the transition arrows on this object:

Lemma 3.4. *(similar to [10, Lemma 31]) For any $w\in A^*$ and $Q'\subseteq A^*$ such that $Qw=\{qw\mid q\in Q\}\subseteq Q'$, there is a unique morphism $\kappa^{Q'}_{Q\cdot w}\big/\mathcal{A}(w):Q/\mathcal{A}(\mathtt{st})\to Q'/\mathcal{A}(\mathtt{st})$ making the two squares in the diagram below commute. Moreover, if $uv=w$ and $Qw\subseteq Q'v\subseteq Q''$, then $\kappa^{Q''}_{Q'\cdot v}\big/\mathcal{A}(v)\circ\kappa^{Q'}_{Q\cdot u}\big/\mathcal{A}(u)=\kappa^{Q''}_{Q\cdot w}\big/\mathcal{A}(w)$.*

$$\begin{array}{ccccc}
\coprod_{Q'}\mathcal{L}(\mathtt{in}) & \xrightarrow{e_{Q'}} & Q'/\mathcal{A}(\mathtt{st}) & \xrightarrowtail{m_{Q'}} & \mathcal{A}(\mathtt{st}) \\
{\scriptstyle \kappa^{Q'}_{Q\cdot w}=\coprod_{q\in Q}\kappa_{qw}}\uparrow & & {\scriptstyle \kappa^{Q'}_{Q\cdot w}/\mathcal{A}(w)}\uparrow & & \uparrow{\scriptstyle \mathcal{A}(w)} \\
\coprod_{Q}\mathcal{L}(\mathtt{in}) & \xrightarrow[e_Q]{} & Q/\mathcal{A}(\mathtt{st}) & \xrightarrowtail[m_Q]{} & \mathcal{A}(\mathtt{st})
\end{array}$$

We fix the notation $\kappa^{Q'}_{Q\cdot w}\big/\mathcal{A}(w)$ for the canonical morphisms of Lemma 3.4, and simply write $\kappa^{Q'}_{Q}\big/\mathcal{A}(\mathtt{st})=\kappa^{Q'}_{Q\cdot\epsilon}\big/\mathcal{A}(\epsilon)$ when w is the empty word. In particular, we have that $(\mathrm{Reach}\,\mathcal{A})(a)=\kappa^{A^*}_{A^*\cdot a}\big/\mathcal{A}(a)$.

Of course, we can't just compute $A^*/\mathcal{A}(\mathtt{st})$ and its associated morphisms as a factorization of the morphism from $\coprod_{A^*}\mathcal{A}(\mathtt{in})$, as this coproduct is infinite: in **Set** with $\mathcal{A}(\mathtt{in})=1$, it corresponds to the set A^* itself. Instead our strategy will be to find some finite subset Q that describes $\mathrm{Reach}\,\mathcal{A}$ entirely, i.e. such that we have an isomorphism $\kappa^{A^*}_{Q}\big/\mathcal{A}(\mathtt{st}):Q/\mathcal{A}(\mathtt{st})\cong A^*/\mathcal{A}(\mathtt{st})$: for classical automata this isomorphism means exactly that the Q-reachable states — those that are reachable from the initial state by following transitions corresponding to words in Q — are precisely the reachable states.

Informally, to find such a subset we start with $Q=\varnothing$ and add words to it until the $\kappa^{Q\cup\{qa\}}_{Q}\big/\mathcal{A}(\mathtt{st})$ are isomorphisms for all $q\in Q$ and $a\in A$: in **Set** this means following the transitions until we cannot reach an unvisited state from a state we have previously

visited. We show that if these morphisms are all isomorphisms then so is $\kappa_Q^{Q \cup QA} \big/ \mathcal{A}(\mathtt{st})$ and then that so is $\kappa_Q^{A^*} \big/ \mathcal{A}(\mathtt{st})$.

Finally, notice that if $Q \subseteq Q'$ then $\kappa_Q^{Q'} \big/ \mathcal{A}(\mathtt{st})$ is already an $\mathcal{M}$-morphism (by [8, Proposition 14.9(1)], because $\mathcal{A}(\epsilon) = \mathrm{id}$ is an $\mathcal{M}$-morphism): in **Set**, the Q-reachable states are in particular Q'-reachable. Hence to check that $\kappa_Q^{Q \cup \{qa\}} \big/ \mathcal{A}(\mathtt{st})$ is an isomorphism we only really need to check that it is an $\mathcal{E}$-morphism (since $\mathcal{E} \cap \mathcal{M}$ is exactly the class of isomorphisms): in **Set**, that the set of Q-reachable states surjects onto the set of $(Q \cup \{qa\})$-reachable states, i.e. that the state reached when following qa is Q-reachable and has already been visited.

In practice, this procedure leads to Algorithm 2 where the partition of A on line 6 is always chosen to be the partition into singletons.

Algorithm 2 A categorical algorithm for computing Reach

Input: an automaton $\mathcal{A} : \mathcal{I} \to \mathcal{C}$
Output: Reach $\mathcal{A}$

1: set $todo \leftarrow \{\{\epsilon\}\}$
2: set $Q_{done} \leftarrow \varnothing$
3: **while** there is a $Q \in todo$ **do**
4: remove Q from $todo$
5: **if** $\kappa_{Q_{done}}^{Q_{done} \sqcup Q} \big/ \mathcal{A}(\mathtt{st}) : Q_{done}/\mathcal{A}(\mathtt{st}) \rightarrowtail Q_{done} \sqcup Q/\mathcal{A}(\mathtt{st})$ is not an $\mathcal{E}$-morphism **then**
6: choose a partition P of A and set $todo \leftarrow todo \sqcup \{Q\tilde{A} \mid \tilde{A} \in P\}$
7: set $Q_{done} \leftarrow Q_{done} \sqcup Q$
8: **end if**
9: **end while**
10: **return** the automaton

$$\mathcal{A}(\mathtt{in}) \xrightarrow[e_{Q_{done}} \circ \kappa_\epsilon]{} Q_{done}/\mathcal{A}(\mathtt{st}) \xrightarrow[\mathcal{A}(\triangleleft) \circ m_{Q_{done}}]{} \mathcal{A}(\mathtt{out})$$

$$\left(\kappa_{Q_{done}}^{Q_{done} \cup Q_{done} a} \big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ \kappa_{Q_{done} \cdot a}^{Q_{done} \cup Q_{done} a} \big/ \mathcal{A}(a)$$

Remark 3.5. Note that the order in which the elements of the to-do list are treated in Algorithm 2 does not have any impact on the correctness and complexity results. In practice, for some arbitrary $Q \subseteq A^*$ it may also be more efficient to compute $(Q_{done} \cup Qa)/\mathcal{A}(\mathtt{st})$ in a batch instead of computing each $(Q_{done} \cup \{qa\})/\mathcal{A}(\mathtt{st})$ for $q \in Q$ separately: this happens for instance in Example 3.14. Hence why we allow choosing a partition of A on line 6: splitting A into anything coarser than singletons means filling the to-do list with sets of words that contain more than just one word.

Theorem 3.6. *If $\mathcal{A}(\mathtt{st})$ is $\mathcal{M}$-noetherian, Algorithm 2 is correct, terminates, and the condition on line 5 is satisfied at most* $\mathrm{length}_{\mathcal{M}}(m_\varnothing : \varnothing/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st}))$ *times.*

3.2. Computing the factorizations incrementally

A shortcoming of Algorithm 2 is that it does not give us a generic way to compute the intermediate factorizations $Q/\mathcal{A}(\mathtt{st})$ incrementally: in **Set**, it does not tell us that the set of $(Q \cup \{qa\})$-reachable states can be computed by finding the state reached by following q, following an additional a-transition, and adding the resulting state to the set of Q-reachable states. Fortunately, this can also be expressed categorically, although with an additional assumption on the output category $\mathcal{C}$.

Lemma 3.7. *For every $Q, Q' \subseteq A^*$, if the coproduct $Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st})$ exists, $[m_Q, m_{Q'}] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$ $(\mathcal{E}, \mathcal{M})$-factors into*

$$m_{Q \sqcup Q'} \circ \left[\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \, , \, \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \twoheadrightarrow (Q \sqcup Q')/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$$

Lemma 3.8. *For every $Q \subseteq A^*$, $\mathcal{A}(a) \circ m_Q$ $(\mathcal{E}, \mathcal{M})$-factors into $m_{Qa} \circ \kappa_{Q \cdot a}^{Qa} \Big/ \mathcal{A}(\mathtt{st}) : Q/\mathcal{A}(\mathtt{st}) \twoheadrightarrow Qa/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$.*

When the binary coproducts of the $Q/\mathcal{A}(\mathtt{st})$ exist, Algorithm 2 may thus be detailed into Algorithm 3. Notice how instead a keeping track of a set of words we have already considered and a set of words that we have to consider next (Q_{done} and Q_{todo} in Algorithm 2), we now only directly keep track of the subobjects corresponding to these words: $m_{Q_{done}} : Q_{done}/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$ is the already-visited subobject of $\mathcal{A}(\mathtt{st})$, while each $(m_Q, \tilde{A})$ in the to-do list corresponds to a subobject m_Q of $\mathcal{A}(\mathtt{st})$ along with a subset of transitions from this subobject that we have to consider next. We also keep track of $e_{Q_{done}} \circ \kappa_\epsilon$, which represents the initial transition into the already-visited subobject $m_{Q_{done}}$.

Corollary 3.9. *Assume that for every finite $Q, Q' \subseteq A^*$, the coproduct $Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st})$ exists.*

Then, if $\mathcal{A}(\mathtt{st})$ is $\mathcal{M}$-noetherian, Algorithm 3 is correct, terminates, and the condition on line 12 is satisfied at most $\mathrm{length}_{\mathcal{M}}(m_\epsilon : \{\epsilon\}/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st}))$ *times.*

Algorithm 3 is already much closer than Algorithm 2 to the algorithm that computes the reachable states of a classical automata, but there is still a procedure that could be further detailed: on line 12, Algorithm 3 only asks us to check whether $\kappa_{Q_{done}}^{Q_{done} \sqcup Q\tilde{A}} \Big/ \mathcal{A}(\mathtt{st})$ is an $\mathcal{E}$-morphism, that is whether the set of Q_{done}-reachable states surjects onto that of $(Q_{done} \sqcup Q\tilde{A})$-reachable states, but it does not tell us that to check this it is enough to just check whether the $Q\tilde{A}$-reachable states had already been visited. Again, this can be expressed categorically with an additional assumption on the factorization system $(\mathcal{E}, \mathcal{M})$.

Lemma 3.10. *If $\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) : Q/\mathcal{A}(\mathtt{st}) \rightarrowtail (Q \sqcup Q')/\mathcal{A}(\mathtt{st})$ is an isomorphism then*

$$m_{Q'} = m_Q \circ \left(\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right)^{-1} \circ \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$$

Algorithm 3 A detailed version of Algorithm 2

Input: an automaton $\mathcal{A} : \mathcal{I} \to \mathcal{C}$

Output: Reach $\mathcal{A}$

compute $e_{\{\epsilon\}} : \mathcal{A}(\mathtt{in}) \twoheadrightarrow \{\epsilon\}/\mathcal{A}(\mathtt{st})$ and $m_{\{\epsilon\}} : \{\epsilon\}/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$

choose a partition P of A and set $todo \leftarrow \{(m_\epsilon, \tilde{A}) \mid \tilde{A} \in P\}$

set $m_{Q_{done}} \leftarrow m_{\{\epsilon\}}$

set $e_{Q_{done}} \circ \kappa_\epsilon \leftarrow e_{\{\epsilon\}}$

while there is a pair $(m_Q, \tilde{A}) \in todo$ **do**

  remove $(m_Q, \tilde{A})$ from $todo$

  for $a \in \tilde{A}$ **do**

    compute $m_{Qa} : Qa/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$ by factoring $\mathcal{A}(a) \circ m_Q$ as in Lemma 3.8

  end for

  compute $m_{Q\tilde{A}} : Q\tilde{A}\Big/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$ by factoring $[m_{Qa}]_{a\in\tilde{A}}$ as in Lemma 3.7

  compute $\kappa_{Q_{done}}^{Q_{done}\sqcup Q\tilde{A}}\Big/\mathcal{A}(\mathtt{st}) : Q_{done}/\mathcal{A}(\mathtt{st}) \rightarrowtail \left(Q_{done} \sqcup Q\tilde{A}\right)\Big/\mathcal{A}(\mathtt{st})$ and $m_{Q_{done}\sqcup Q\tilde{A}} : \left(Q_{done} \sqcup Q\tilde{A}\right)\Big/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$ by factoring $[m_{Q_{done}}, m_{Q\tilde{A}}]$ as in Lemma 3.7

  if $\kappa_{Q_{done}}^{Q_{done}\sqcup Q\tilde{A}}\Big/\mathcal{A}(\mathtt{st})$ is not an $\mathcal{E}$-morphism **then**

    choose a partition P of A and set $todo \leftarrow todo \sqcup \{(m_{Q\tilde{A}}, \tilde{A}') \mid \tilde{A}' \in P\}$

    set $m_{Q_{done}} \leftarrow m_{Q\sqcup Q\tilde{A}}$

    set $e_{Q_{done}} \circ \kappa_\epsilon \leftarrow \kappa_{Q_{done}}^{Q_{done}\sqcup Q\tilde{A}}\Big/\mathcal{A}(\mathtt{st}) \circ e_{Q_{done}} \circ \kappa_\epsilon : \mathcal{A}(\mathtt{in}) \to \left(Q \sqcup Q\tilde{A}\right)\Big/\mathcal{A}(\mathtt{st})$

  end if

end while

return the automaton

$$\left(\kappa_{Q_{done}}^{Q_{done}\cup Q_{done}a}\Big/\mathcal{A}(\mathtt{st})\right)^{-1} \circ \kappa_{Q_{done}a}^{Q_{done}\cup Q_{done}a}\Big/\mathcal{A}(\mathtt{st}) \circ \kappa_{Q_{done}\cdot a}^{Q_{done}a}\Big/\mathcal{A}(a)$$

$$\mathcal{A}(\mathtt{in}) \xrightarrow[e_{Q_{done}}\circ\kappa_\epsilon]{} Q_{done}/\mathcal{A}(\mathtt{st}) \xrightarrow[\mathcal{A}(\triangleleft)\circ m_{Q_{done}}]{} \mathcal{A}(\mathtt{out})$$

with $\kappa_{Q_{done}\cdot a}^{Q_{done}a}\Big/\mathcal{A}(a) : Q_{done}/\mathcal{A}(\mathtt{st}) \twoheadrightarrow Q_{done}a/\mathcal{A}(\mathtt{st})$ computed by factoring $\mathcal{A}(a) \circ m_{Q_{done}}$ as in Lemma 3.8

Conversely, in the context of Lemma 3.7, assume also that all split epimorphisms are in $\mathcal{E}$ (if $e \circ s = \mathrm{id}$ for some e, s then $e \in \mathcal{E}$). If $m_{Q'}$ factors as $m_Q \circ f$ for some $f : Q/\mathcal{A}(\mathtt{st}) \to (Q \sqcup Q')/\mathcal{A}(\mathtt{st})$ then $\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism.

Corollary 3.11. *When $\mathcal{E}$ contains all split epimorphisms, it is enough to check on line 12 of Algorithm 3 that $m_{Q\tilde{A}} : Q\tilde{A} \Big/ \mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$ doesn't factor through $m_{Q_{done}} : Q_{done}/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$. In particular, line 11 can be moved inside the if statement on line 12.*

3.3. Examples

Example 3.12 (Graph traversal)**.** Fix a $(\mathbf{Set}, 1, 2)$-automaton $\mathcal{A}$ with finite state-set. Since **Set** is cocomplete and $\mathcal{E} = \mathrm{Surj}$ is exactly the class of split epimorphisms (surjections are exactly those functions that have a right-inverse), we may use Algorithm 3 (by Corollary 3.9) with the partitions of A chosen to be the partition into singletons, along with the optimization of Corollary 3.11: the condition on line 12 is then satisfied at most as many times as there are states in $\mathcal{A}$.

The resulting algorithm is exactly the standard graph traversal algorithm for computing the reachable states of an automaton, as described for instance in in [24, Exercise 1.18]: an $\mathcal{M}$-subobject $m_Q : Q/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$ is indeed a subset of the states of the automaton. The algorithm thus keeps track of a subset of already visited states, $m_{Q_{done}}$, and a to-do list of states and transitions to consider next (the factorization of a morphism $1 \to \mathcal{A}(\mathtt{st})$ yields a singleton). The algorithm then goes through the to-do list: for a given pair of a state and a transition $(m_{\{q\}}, a)$ that needs to be considered, it computes the successor of the state along the transition (yielding $m_{\{qa\}}$). It then checks whether this new state already belongs to the subset of already visited states by checking whether $m_{\{qa\}}$ factors through $m_{Q_{done}}$: when this is not the case, $m_{\{qa\}}$ is added to the subset of already visited states as well as to the to-do list, once for every transition. $\kappa_{Q_{done}}^{Q_{done} \sqcup \{qa\}} \Big/ \mathcal{A}(\mathtt{st})$ is the inclusion of the old subset of already visited states into the new one, and helps us keep track of which state was the initial state. Once the to-do list is empty, the algorithm returns the automaton with set of states the subset of states that were visited, and inherits its initial and final states as well as its transition functions from those of $\mathcal{A}$.

Example 3.13 (Graph traversal for non-deterministic automata)**.** As explained in Example 2.11, the factorization system $(\mathrm{Surj}, \mathrm{Det} \cap \mathrm{Inj})$ on **Rel** described in Example 2.7 does not produce an interesting notion of minimal automata. But while there is indeed no unique way to merge the non-distinguishable states of a non-deterministic automaton, it is still possible to compute its set of reachable states.

This is encompassed by this factorization system: given a $(\mathbf{Rel}, 1, 1)$-automaton $\mathcal{A}$, $\mathrm{Reach}_{(\mathrm{Surj},\mathrm{Det}\cap\mathrm{Inj})} \mathcal{A}$ is the restriction of $\mathcal{A}$ to its reachable states. Since **Rel** has arbitrary coproducts, $\mathrm{Reach}\, \mathcal{A}$ may be computed using Algorithm 3 (by Corollary 3.9) with the partitions of A chosen to be the partition into singletons, along with the optimization of Corollary 3.11 (as the split epimorphisms in **Rel** are exactly the graphs of the partial

surjective functions, i.e. relations $r \subseteq X \times Y$ for which there is a surjective function $f : X' \to Y$ with codomain some subset $X' \subseteq X$ such that $(x, y) \in r \iff y = f(x)$): the condition on line 12 would then be satisfied at most as many times as there are states in $\mathcal{A}$.

The resulting algorithm is very similar to the one of Example 3.12 for complete deterministic automata, which is also the standard algorithm for computing the reachable states of a non-deterministic automaton. The only difference is that the successors of a state along the transitions labeled by a specific letter are not added as separate subobjects in the to-do list, but only as part of the same subobject. Hence if one of these successors has not been visited yet, the whole subobject of these successors is considered not yet visited: this may bring a lot of redundancy if all the other successors had already been visited.

Example 3.14 (Moore's algorithm)**.** Fix a $(\mathbf{Set}, 1, 2)$-automaton $\mathcal{A}$ with finite state-set. Since **Set** is complete and $\mathcal{M} = \mathrm{Inj}$ is exactly the class of split monomorphisms (injections are exactly those functions that have a right-inverse), we may use the dual of Algorithm 3 (by the dual of Corollary 3.9) with the partitions of A chosen to be the trivial partition $P = \{A\}$, along with the optimization of Corollary 3.11: the dual of the condition on line 12 is then satisfied at most as many times as there are states in $\mathcal{A}$.

The resulting algorithm is similar to Moore's algorithm [22], a standard algorithm for merging the equivalent states of an automaton: an $\mathcal{E}$-quotient of $e_T : \mathcal{A}(\mathtt{st}) \twoheadrightarrow T/\mathcal{A}(\mathtt{st})$ is indeed exactly a partition of the states of the automaton into equivalence classes, two states being equivalent if they accept the same subset of words of T. The algorithm thus keeps track of two such a partitions $e_{A^{\leq n}}$ and e_{A^n}, starting with $n = 0$ which just means that the states are split depending on whether they are final or not. $e_{A^{\leq n}}$ is the tentative partition that gets refined as n grows and more words are taken into account to distinguish the states, while e_{A^n} is used to compute the next refinement of this tentative partition: computing e_{aA^n} by factoring $e_{A^n} \circ \mathcal{A}(a)$ really means distinguish states when their transitions along the letter a do not fall into the same partition of e_{A^n}. Merging the refinements in the tentative partition is then done by computing $e_{A^{\leq n+1}}$ through the factorization of $\langle e_{A^{\leq n}}, e_{A^{n+1}} \rangle$, which means computing the intersection of every two equivalence class of $e_{A^{\leq n}}$ and $e_{A^{n+1}}$ and only keeping those that are not empty. In particular, if $e_{A^{n+1}}$ factors through $e_{A^{\leq n}}$ then $e_{A^{\leq n}}$ is already a refinement of $e_{A^{n+1}}$ and the algorithm can stop, outputting the automaton with state-set the equivalence classes of $e_{A^{\leq n}}$ and initial and final states as well as transition functions inherited from those of $\mathcal{A}$.

Note that the usual presentation of Moore's algorithm would have the next refinement of the tentative partition be computed by pulling $e_{A^{\leq n}}$ instead of e_{A^n} along transitions, but the resulting refinement would be the same. Still, this point of view could be probably be accomodated in our framework with a result similar to Lemma 3.7 but with pushouts instead of coproducts.

Hopcroft's algorithm [18], the other standard partition refinement algorithm for merging non-distinguishable states, could on the contrary probably not be accomodated in a very generic way, because it relies heavily on specificities of **Set**, namely that

$\mathcal{A}(\mathtt{out}) = 2$ and the possibility to compute cardinals of objects.

Example 3.15 (Minimization of weighted automata)**.** Given a field $\mathbb{K}$, fix a $(\mathbb{K}\mathbf{Vec}, \mathbb{K}, \mathbb{K})$-automaton $\mathcal{A}$ with finite-dimensional state-space. Since $\mathbb{K}\mathbf{Vec}$ is cocomplete and $\mathcal{E} = \mathrm{Surj}$ is exactly the class of split epimorphisms (linear surjections are exactly those functions that have a right-inverse), we may use Algorithm 3 (by Corollary 3.9) along with the optimization of Corollary 3.11: the condition on line 12 is then satisfied at most $\dim \mathcal{A}(\mathtt{st})$ times.

The resulting algorithm computes $\mathrm{Reach}\,\mathcal{A}$, i.e. an automaton whose configurations are exactly those that are reachable in $\mathcal{A}$: $v \in (\mathrm{Reach}\,\mathcal{A})(\mathtt{st})$ if and only if there are $\lambda_1, \dots, \lambda_k \in \mathbb{K}$ and $w_1, \dots, w_k \in A^*$ such that $v = \sum_{i=1}^{k} \lambda_i \mathcal{A}(\triangleright w)$. An $\mathcal{M}$-subobject of $\mathcal{A}(\mathtt{st})$ is a subvector space of $\mathcal{A}(\mathtt{st})$ or equivalently a set of independant vectors of $\mathcal{A}(\mathtt{st})$. The algorithm thus maintains $m_{Q_{done}} : Q/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$, a set of independant vectors of $\mathcal{A}(\mathtt{st})$ that form a basis of the vector space of configurations that have already been visited, as well as a to-do list of pairs of a vector $m_{\{q\}} : \{q\}/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$ (the factorization of a morphism $\mathbb{K} \to \mathcal{A}(\mathtt{st})$ has dimension lesser or equal to 1) and a letter a that are to be considered next. When considering $(m_{\{q\}}, a)$, the algorithm first computes the successor vector $m_{\{qa\}}$ along a by applying the matrix $\mathcal{A}(a)$ and then checks whether $m_{\{qa\}}$ factors through $m_{Q_{done}}$ hence solves a linear system to check whether the vector $m_{\{qa\}}$ is a linear combination of those generating $m_{Q_{done}}$. If this is not the case, $(m_{\{qa\}}, a')$ is added to the to-do list for each $a' \in A$ and $m_{\{qa\}}$ is added to the basis of $m_{Q_{done}}$ to form the basis of $m_{Q_{done} \sqcup \{qa\}}$. Once the to-do list is empty, the algorithm returns the automaton with state-space $Q_{done}/\mathcal{A}(\mathtt{st})$ whose initial configuration, transition functions and final values are inherited from those of $\mathcal{A}$ through a change of basis.

Since the algorithm terminates, it only considers finite subsets Q of A^* and thus only manipulates finite-dimensional vector spaces. We may therefore as well consider that all of its computations take place in the category of finite-dimensional vector spaces. But the functor that sends a vector space to its dual and a linear map to its transpose is an isomorphism between this category and its dual which sends $\mathbb{K}$ on itself and swaps Surj and Inj. We are thus in the very special case mentioned in Remark 3.2 where the algorithm that computes the transpose of $\mathrm{Obs}\,\mathcal{A}$ is exactly the one that computes Reach as described above, but applied to the transpose of $\mathcal{A}$.

4. 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 Section 2 as well as the generic minimization algorithm we extended it with in Section 3. In Section 4.1, we first rapidly recall the notion of monoid. We then define the category of monoidal transducers recognizing a function in Section 4.2, and study how it fits into the framework of Sections 2 and 3: the initial transducer is given in Corollary 4.11, conditions for the existence of the final transducer are described in Section 4.3, and factorization systems are tackled in Section 4.4.

4.1. Monoids

Let us first recall definitions relating to monoids, and fix some notations. Most of these are standard in the monoid literature, only coprime-cancellativity (Definition 4.4) and noetherianity (Definition 4.5) are uncommon.

Definition 4.1 (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 ($\forall u, v, w \in M, u \otimes_M (v \otimes_M w) = (u \otimes_M v) \otimes_M w$) and has ϵ_M as *unit* element ($\forall u \in M, u \otimes_M \epsilon_M = \epsilon_M \otimes_M u = u$). 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 : $\forall u, v \in M, u \otimes^{op} v = v \otimes u$.

The dual of a monoid is mainly used here for the sake of conciseness : whenever we define some "left-property", the corresponding "right-property" is defined as the left-property but in the dual monoid. Note that when M is commutative it is its own dual and the left- and right-properties coincide.

Definition 4.2 (invertibility)**.** An element x of a monoid M is *right-invertible* when there is a $y \in M$ such that $xy = \epsilon$, and y is then called the *right-inverse* of x. It is *left-invertible* when it is right-invertible in M^{op}, and the corresponding right-inverse is called its *left-inverse*. When x 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 x^{-1}. The set of invertible elements of M is written $M^\times$.

Two families $(u_i)_{i \in I}$ and $(v_i)_{i \in I}$ indexed by some non-empty set I are equal *up to invertibles on the left* when there is some invertible $x \in M$ such that $\forall i \in I, u_i = x v_i$.

Definition 4.3 (divisibility)**.** An element u of a monoid M *left-divides* a family $w = (w_i)_{i \in I}$ of M indexed by some set I when there is a family $(v_i)_{i \in I}$ such that $\forall i \in I, uv_i = w_i$, and we say that u is a *left-divisor* of w. It *right-divides* it when it left-divides it in M^{op}, and in that case u is called a *right-divisor* of w.

A *greatest common left-divisor* (or *left-gcd*) of the family w is a left-divisor of w that is left-divided by all others left-divisors of w.

A family w is said to be *left-coprime* when it has ϵ as a left-gcd, i.e. when all its left-divisors (or equivalently one of its left-gcds, if there is one) 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 4.4 (cancellativity)**.** A monoid M is said to be *left-cancellative* when for any families $(u_i)_{i \in I}$ and $(v_i)_{i \in I}$ of M indexed by some set I and for any $w \in M$, $u = v$ as soon as $wu_i = wv_i$ for all $i \in I$. If this only implies $u_i = x v_i$ for some $x \in M^\times$, 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 $u, v \in M$ and any left-coprime family $(w_i)_{i \in I}$ indexed by some set I, $u = v$ as soon as $uw_i = vw_i$ for all $i \in I$.

Definition 4.5 (noetherianity)**.** A monoid M is *right-noetherian* when for any sequences $(u_n)_{n\in\mathbb{N}}$ and $(v_n)_{n\in\mathbb{N}}$ of M such that $v_n = v_{n+1}u_n$ for all $n \in \mathbb{N}$, there is some $n \in \mathbb{N}$ such that u_n is invertible.

In this case, we write $\operatorname{rk} v$ for the *rank* of v, the (possibly infinite) supremum of the numbers of non-invertibles in a sequence $(u_n)_{n\in\mathbb{N}}$ that satisfies $v_n = v_{n+1}u_n$ for a sequence $(v_n)_{n\in\mathbb{N}}$ with $v_0 = v$.

In other words, a monoid is right-noetherian when it has no strict infinite chains of right-divisors (in the definition above, each $u_n \cdots u_0$ right-divides v_0). Note that $\operatorname{rk}(uv) \leq \operatorname{rk} u + \operatorname{rk} v$, and $\operatorname{rk} : M \to \mathbb{N}$, and if this is an equality M is said to be *graded*.

Noetherianity will be used to ensure that fixed-point algorithms terminate, hence we will often assume that the monoids we work with satisfy some noetherianity properties. We therefore state two additional lemmas making it easier to work with noetherian monoids. They both assume M to be right-noetherian but the dual results also stand.

Lemma 4.6. *M is right-noetherian if and only if for any sequences $(u_n)_{n\in\mathbb{N}}$ and $(v_n)_{n\in\mathbb{N}}$ of M such that $v_n = v_{n+1}u_n$ for all $n \in \mathbb{N}$, there is some $n \in \mathbb{N}$ such that for all $i \geq n$, u_i is invertible.*

Lemma 4.7. *If M is right-noetherian then all its right- and left-invertibles are invertible.*

Example 4.8. 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 monoid of interest is $(E, \vee, \bot)$ for E any join-semilattice with a bottom element $\bot$. In this commutative monoid, 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 $E = \mathbb{R}_+$.

4.2. 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 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 objects the sets 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 $\mathcal{T}_M X \to \mathcal{T}_M Y \to \mathcal{T}_M Z$.

We now have all the ingredients to define our main object of study, monoidal transducers.

Definition 4.9 (monoidal transducer)**.** The category $\mathbf{Trans}_M$ of *M-transducers* is the category of $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-automata from Definition 2.1. An M-transducer (or *monoidal transducer over M*, or simply *transducer*) is thus a functor $\mathcal{A} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ with $\mathcal{A}(\mathtt{in}) = \mathcal{A}(\mathtt{out}) = 1$, and a *morphism of transducers* is a natural transformation between the corresponding functors whose restriction to $\mathcal{O}$ is the identity. Similarly given a $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$, $\mathbf{Trans}_M(\mathcal{L})$ is the category of monoidal transducers recognizing $\mathcal{L}$.

In practice, a monoidal transducer $\mathcal{A}$ is thus a tuple $(S, (\upsilon_0, s_0), t, (- \odot a)_{a \in A})$ where:

- S is the *state-set*;
- $(\upsilon_0, s_0) = \mathcal{A}(\triangleright) \in M \times S + 1$ is the (possibly undefined) pair of the *initialization value* and *initial state*;
- $t = \mathcal{A}(\triangleleft) : S \to M + 1$ is the partial *termination function*;
- $s \odot a = \mathcal{A}(a)(s) \in M \times S + 1$ may be undefined, and we write $- \cdot a : S \to S + 1$ for the second component of $- \odot a$, the partial *transition function along a*, and $- \bigcirc a : S \to M + 1$ for its first component, the partial *production function along a*

Similarly, the language $\mathcal{L} = \mathcal{A} \circ \iota$ recognized by this monoidal transducer is thus a function $L : A^* \to M + 1$ given by $L(a_1 \cdots a_n) = t^\dagger((\upsilon_0, s_0) \odot^\dagger a_1 \odot^\dagger \cdots \odot^\dagger a_n)$, 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

$f^\dagger(s \odot a) = f^\dagger(s) \odot^\dagger a$. We say that this function is *subsequential* when it is recognized by a transducer with finite state-space.

Example 4.10. Figure 2 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 $\Sigma = \{\alpha, \beta\}$. Formally, it is given by $S = \{1, 2, 3, 4\}$; $(\upsilon_0, s_0) = (\epsilon, 1)$; $t(1) = \alpha$, $t(2) = \bot$, $t(3) = \alpha$ and $t(4) = \epsilon$; and finally $1 \odot a = (\epsilon, 2)$, $1 \odot b = (\beta, 3)$, $3 \odot b = (\beta, 3)$ as well as $s \odot c = \bot$ for any other $s \in S$ and $c \in A$. This transducer recognizes the function given by $L(b^n) = \beta^n\alpha$ (seen in the corresponding quotient monoid, so for instance $L(b^n) = \alpha\beta^n$ when $M = \Sigma^{\circledast}$) for all $n \in \mathbb{N}$ and $L(w) = \bot$ otherwise.

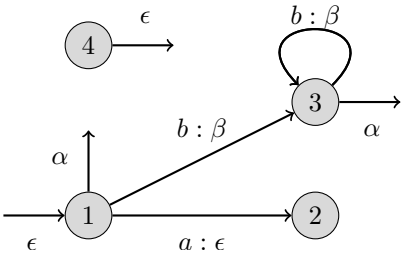


Figure 2: A monoidal transducer $\mathcal{A}$

When $M = B^*$ for some alphabet B, M-transducers coincide with the classical notion of transducers and the minimal transducer is given by Definition 2.9 [9, 23]. To study the notion of minimal monoidal transducer, it is thus natural to try to follow this framework as well.

4.3. The initial and final monoidal transducers recognizing a function

To apply the framework of Sections 2 and 3 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 [26]. Hence Theorem 2.12 applies:

Corollary 4.11 (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 2.12 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 [23]. Hence we study the properties of a monoid M such that $\mathbf{Kl}(\mathcal{T}_M)$ has these powers.

To this means, given a countable set I we consider partial functions $\Lambda : I \to M+1$. We write $\perp^I$ for the nowhere defined function $i \mapsto \perp$ and $(M+1)^I_* = (M+1)^I - \{\perp^I\}$ 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 $\Lambda(i) \neq \perp$ and $(\upsilon \otimes \Lambda)(i) = \perp$ otherwise.The universal property of the product then translates as:

Proposition 4.12. *The following are equivalent:*

(1) $\mathbf{Kl}(\mathcal{T}_M)$ *has all countable powers of* 1*;*

(2) there are two functions $\operatorname{lgcd} : (M+1)^{\mathbb{N}}_* \to M$ *and* $\operatorname{red} : (M+1)^{\mathbb{N}}_* \to (M+1)^{\mathbb{N}}_*$ *such that*

  a) for all $\Lambda \in (M+1)^{\mathbb{N}}_*$, $\Lambda = \operatorname{lgcd}(\Lambda)\operatorname{red}(\Lambda)$*;*

  b) for all $\Gamma, \Lambda \in (M+1)^{\mathbb{N}}_*$ *and* $\upsilon, \nu \in M$*, if* $\upsilon \operatorname{red}(\Gamma) = \nu \operatorname{red}(\Lambda)$ *then* $\upsilon = \nu$ *and* $\operatorname{red}\Gamma = \operatorname{red}\Lambda$*;*

(3) $\mathbf{Kl}(\mathcal{T}_M)$ *has all countable products.*

Moreover when these hold, since any countable set I *embeds into* $\mathbb{N}$, lgcd *and* red *can be extended to* $(M+1)^I_*$. $\operatorname{lgcd}\Lambda$ *is then a left-gcd of* $(\Lambda(i))_{i|\Lambda(i)\neq\perp}$ *and the product of* $(X_i)_{i\in I}$ *for some* $I \subseteq \mathbb{N}$ *is the set of pairs* $(\Lambda, (x_i)_{i\in I})$ *such that* $\Lambda \in \operatorname{red}((M+1)^I_*)$ *and, for all* $i \in I$*,* $x_i \in X_i$ *if* $\Lambda(i) \neq \perp$ *and* $x_i = \perp$ *otherwise. In particular, the* I*-th power of* 1 *is the set of* irreducible *partial functions* $I \to M+1$*:*

$$\prod_I 1 = \operatorname{Irr}(I, M) = \{\operatorname{red}\Lambda \in (M+1)^I_* \mid \Lambda \in (M+1)^I_*\}$$

We also write $\operatorname{lgcd}(\perp^{\mathbb{N}}) = \perp$, $\operatorname{red}(\perp^{\mathbb{N}}) = \perp^{\mathbb{N}} = \perp$ and do not distinguish between $(\perp, \perp)$ and $\perp$. Also note that:

Lemma 4.13. *Conditions (2)a and (2)b are equivalent to saying that*

(4) a) $\langle \operatorname{lgcd}, \operatorname{red} \rangle$ *is injective;*

  b) for all $\Lambda \in (M+1)^{\mathbb{N}}_*$, $\operatorname{lgcd}(\operatorname{red}\Lambda) = \epsilon$ *and* $\operatorname{red}(\operatorname{red}\Lambda) = \operatorname{red}\Lambda$*;*

  c) for all $\Lambda \in (M+1)^{\mathbb{N}}_*$ *and* $\upsilon \in M$, $\operatorname{lgcd}(\upsilon\Lambda) = \upsilon \operatorname{lgcd}(\Lambda)$ *and* $\operatorname{red}(\upsilon\Lambda) = \operatorname{red}(\Lambda)$.

Corollary 4.14. *When the functions* lgcd *and* red *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* $S^{final} = \operatorname{Irr}(A^*, M)$*, initial state* $s_0^{final} = \operatorname{red}\mathcal{L}$*, initialization value* $\upsilon_0^{final} = \operatorname{lgcd}\mathcal{L}$*, termination function* $t^{final}(\Lambda) = \Lambda(e)$ *and transition function* $\Lambda \odot^{final} a = (\operatorname{lgcd}(a^{-1}\Lambda), \operatorname{red}(a^{-1}\Lambda))$ *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* $f : S \to M \times \operatorname{Irr}(A^*, M) + 1$ *such that* $f(s) = (\operatorname{lgcd}\mathcal{L}_s, \operatorname{red}\mathcal{L}_s)$ *where* $\mathcal{L}_s(\triangleright w \triangleleft)(*) = \mathcal{A}(w\triangleleft)(s)$ *is the function recognized by* $\mathcal{A}$ *from the state* s*.*

In practice we will assume that M is right-noetherian to ensure algorithms terminate. It is thus interesting to see what the existence of the powers of 1 implies in this specific case (and in particular, by Lemma 4.7, when M is such that all right- and left-invertibles are invertibles).

Lemma 4.15. *If right- and left-invertibles of M are all invertibles and if $\mathbf{Kl}(\mathcal{T}_M)$ has all countable powers of 1 then M is both left-cancellative up to invertibles on the left and right-coprime-cancellative, and all non-empty countable families of M have a unique left-gcd up to invertibles on the right.*

And conversely, these conditions on M are enough for $\mathbf{Kl}(\mathcal{T}_M)$ to have all countable powers of 1 (even when M is not noetherian), while being easier to show than properly defining the two functions lgcd and red.

Lemma 4.16. *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.*

Example 4.17. When M is a group it is cancellative (because all elements are invertible) and all countable families have a unique left-gcd up to invertibles on the right (ϵ itself) hence Lemma 4.16 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 [13, 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 condition (4)c, more precisely that $\mathrm{red}(\upsilon\Lambda) = \mathrm{red}\,\Lambda$. This is expected, as there may be several non-isomorphic ways to minimize automata with outputs in these monoids, which is incompatible with the framework of Definition 2.9.

Proposition 4.12 is reminiscent of a very similar sufficient condition on a semi-ring for deterministic weighted automata to be minimizable in a unique way [15], namely that only an analogue of the red function should exist. The minimal transducer could then again be obtained as the factorization of the unique arrow $\mathcal{A}^{init}(\mathcal{L}) \to \mathcal{A}^{final}(\mathcal{L})$, except $\mathcal{A}^{final}(\mathcal{L})$ would not be final anymore but only weakly final (for any transducer $\mathcal{A}$ recognizing $\mathcal{L}$ there would be an arrow $\mathcal{A} \to \mathcal{A}^{final}(\mathcal{L})$ but it would not need to be unique), hence our framework would not entail its unicity. However, as further discussed in Section 5.2, these weaker conditions do not really lead to a proper minimization algorithm.

Lemma 4.16 provides sufficient conditions that are reminiscent of those developed in [17] 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 b left-divides both some c and some ac, then b should also left-divide ab. 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 submonoids themselves being right-noetherian.

4.4. Factorization systems

The last ingredient we need in order to be able to apply the framework of Sections 2 and 3 is a factorization system on $\mathbf{Trans}_M(\mathcal{L})$. By Lemma 2.8, it is enough to find a factorization system on $\mathbf{Kl}(\mathcal{T}_M)$.

When M is a free monoid, define $\mathcal{E} = \mathrm{Surj}$ to be the class of those functions $f : X \to M \times Y + 1$ that are surjective on Y and $\mathcal{M} = \mathrm{Inj} \cap \mathrm{Eps} \cap \mathrm{Tot}$ to be the class of those functions $f : X \to M \times Y + 1$ that are total ($f \in \mathrm{Tot}$), injective when corestricted to Y ($f \in \mathrm{Inj}$), and only produce the empty word ($f \in \mathrm{Eps}$). Then the $(\mathcal{E}, \mathcal{M})$-minimal transducer recognizing a function is the one defined by Choffrut [9, 23]: in particular, the fact that the minimal transducer $(\mathcal{E}, \mathcal{M})$-divides all other transducers means (thanks to the surjectivity of $\mathcal{E}$-morphisms and the injectivity of $\mathcal{M}$-morphisms) that it has the smallest possible state-set and produces its outputs as early as possible.

It is thus natural to try and extend this factorization system to $\mathbf{Kl}(\mathcal{T}_M)$ for arbitrary M. It is not enough by itself because isomorphisms may produce invertible elements that may be different from ϵ: $\mathrm{Iso} \subsetneq \mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Eps} \cap \mathrm{Tot}$, yet we need the intersection $\mathcal{E} \cap \mathcal{M}$ to be Iso. $\mathcal{M}$-morphisms must thus be able to produce invertible elements as well. Formally, define therefore Surj, Inj, Tot and Inv as follows. For $f : X \to M \times Y + 1$, write $f_1 : X \to M + 1$ 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, $f \in \mathrm{Inj}$ whenever f_2 is injective when corestricted to Y, $f \in \mathrm{Tot}$ whenever $f(x) \neq \bot$ for all $x \in X$ and $f \in \mathrm{Inv}$ whenever $f_1(x)$ is either $\bot$ or in $M^\times$. The point is that when replacing Eps with Inv, we get back that

Lemma 4.18. *In* $\mathbf{Kl}(\mathcal{T}_M)$, $\mathrm{Iso} = \mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot}$, *and these four classes are all closed under composition (within themselves).*

The different ways in which we may distribute these four classes into the two classes $\mathcal{E}$ and $\mathcal{M}$ leads to not just one but three interesting factorization systems:

Proposition 4.19. $(\mathcal{E}_1, \mathcal{M}_1) = (\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv}, \mathrm{Tot})$, $(\mathcal{E}_2, \mathcal{M}_2) = (\mathrm{Surj} \cap \mathrm{Inj}, \mathrm{Inv} \cap \mathrm{Tot})$ *and* $(\mathcal{E}_3, M_3) = (\mathrm{Surj}, \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot})$ *are all factorization systems in* $\mathbf{Kl}(\mathcal{T}_M)$.

The factorization system we choose to define the minimal M-transducer is $(\mathcal{E}_3, \mathcal{M}_3) = (\mathrm{Surj}, \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot})$, because it generalizes the factorization system that defines the minimal transducer (with output in a free monoid). It will be our main factorization system, and as such from now on we reserve the notation $(\mathcal{E}, \mathcal{M})$ for it.

Theorem 2.12 and Proposition 2.10 show that $(\mathcal{E}, \mathcal{M})$ indeed gives rise to a useful notion of minimal transducer.

Corollary 4.20. *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.*

Why are $(\mathcal{E}_1, \mathcal{M}_1)$ and $(\mathcal{E}_2, \mathcal{M}_2)$ also interesting then ? They do not give rise to useful notions of minimality, but they show that the computation of Obs can be split into substeps. Indeed, since $\mathcal{E}_1 \subset \mathcal{E}_2 \subset \mathcal{E}_3$ (and equivalently $\mathcal{M}_1 \supset \mathcal{M}_2 \supset \mathcal{M}_3$), $(\mathcal{E}_i, \mathcal{M}_i)_{1 \leq i \leq 3}$ is a *quaternary factorization system*:

Corollary 4.21. *For every* $\mathbf{Kl}(\mathcal{T}_M)$*-arrow* $f : X \multimap Y$*, there is a unique (up to isomorphisms) factorization of* f *into*

$$X \xrightarrow{f_1}_{1} Z_1 \xrightarrow{f_2}_{2} Z_2 \xrightarrow{f_3}_{2} Z_3 \xrightarrow{f_4} Y$$

such that $f_1 \in \mathcal{E}_1$*,* $f_2 \in \mathcal{E}_2 \cap \mathcal{M}_1$*,* $f_3 \in \mathcal{E}_3 \cap \mathcal{M}_2$ *and* $f_4 \in \mathcal{M}_3$*.*

Note how we respectively write $\multimap\!\!\!\!\twoheadrightarrow_1$, $\multimap\!\!\!\!\twoheadrightarrow_2$, $\overset{}{\underset{1}{\rightarrowtail\!\!\!\!\multimap}}$ and $\overset{}{\underset{2}{\rightarrowtail\!\!\!\!\multimap}}$ for arrows in $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{M}_1$ and $\mathcal{M}_2$ (but stick with $\multimap\!\!\!\!\twoheadrightarrow$ and $\rightarrowtail\!\!\!\!\multimap$ for arrows in $\mathcal{E} = \mathcal{E}_3$ and $\mathcal{M} = \mathcal{M}_3$).

Intuitively, this results means that the computation of any f can be factored into four parts: first forgetting some inputs (f_1 belongs to $\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv}$ but need not belong to Tot), then producing non-invertible elements of the output monoid (f_2 belongs to $\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Tot}$ but need not belong to Inv), then merging some inputs together (f_3 belongs to $\mathrm{Surj} \cap \mathrm{Inv} \cap \mathrm{Tot}$ but need not belong to Inj) and finally embedding the result into a bigger set (f_4 belongs to $\mathrm{Inv} \cap \mathrm{Inj} \cap \mathrm{Tot}$ but need not belong to Surj).

In particular, the $\mathcal{E}$-quotient $\mathrm{Reach}\,\mathcal{A} \multimap\!\!\!\!\twoheadrightarrow \mathrm{Obs}(\mathrm{Reach}\,\mathcal{A})$ factors as follows.

Definition 4.22. Given an M-transducer $\mathcal{A}$ recognizing the $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$, define $\mathrm{Total}\,\mathcal{A}$ and $\mathrm{Prefix}\,\mathcal{A}$ to be the $(\mathcal{E}_1, \mathcal{M}_1)$- and $(\mathcal{E}_2, \mathcal{M}_2)$-factorizations of the final arrow $\mathrm{Reach}\,\mathcal{A} \multimap \mathcal{A}^{final}(\mathcal{L})$:

$$\mathrm{Reach}\,\mathcal{A} \xrightarrow[1]{} \mathrm{Total}\,\mathcal{A} \xrightarrow[2]{} \mathrm{Prefix}\,\mathcal{A} \xrightarrow[2]{} \mathrm{Min}\,\mathcal{L} \rightarrowtail \mathcal{A}^{final}(\mathcal{L})$$

In practice, if $\mathcal{A} = (S, (u_0, s_0), t, \odot)$,

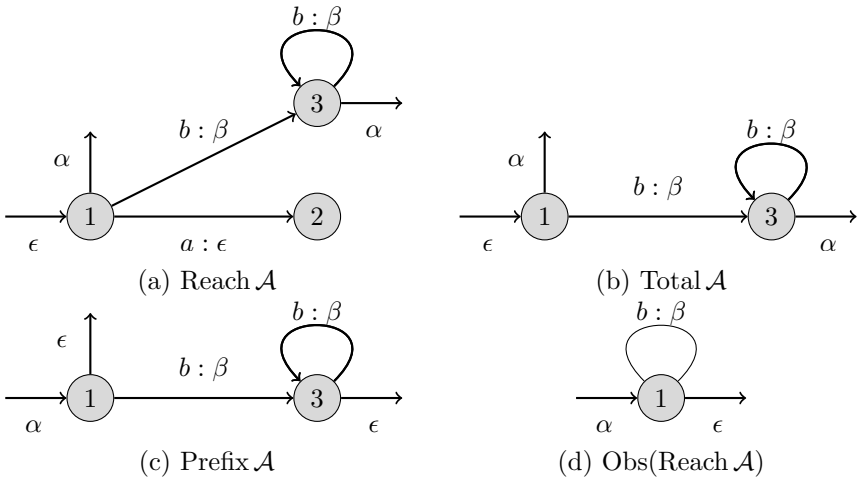
- $\mathrm{Reach}\,\mathcal{A}$ has state-set the set of states in S that are reachable from s_0;

- Total $\mathcal{A}$ has state-set S' the set of states in S that recognize a function defined for at least one word (in particular if $\mathcal{A}$ recognizes $\bot^{A^*}$ then (u_0, s_0) is set to $\bot$);
- Prefix $\mathcal{A} = (S', (u_0 \operatorname{lgcd}(\mathcal{L}_{s_0}), s_0), t', \odot')$, where $\mathcal{L}_s$ is the function recognized from a state $s \in S$ in $\mathcal{A}$, is obtained from Total $\mathcal{A}$ by setting $t'(s) = \operatorname{lgcd}(\mathcal{L}_s)^{-1} t(s)$ and $s \odot' a = (\operatorname{lgcd}(\mathcal{L}_s)^{-1}(s \bigcirc a) \operatorname{lgcd}(\mathcal{L}_{s \cdot a}), s \cdot a)$;
- $\operatorname{Min} \mathcal{L} \cong \operatorname{Obs}(\operatorname{Reach} \mathcal{A})$ is obtained from 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 Prefix $\mathcal{A}$, that is when $\operatorname{red}(\mathcal{L}_{s_1}) = \operatorname{red}(\mathcal{L}_{s_2})$ in $\mathcal{A}$.

In particular, these four steps (computing Reach $\mathcal{A}$, Total $\mathcal{A}$, Prefix $\mathcal{A}$ and finally Obs $\mathcal{A}$) match exactly the four steps into which all the algorithms for minimizing (possibly monoidal) transducers are decomposed [4, 7, 9, 15, 17].

Example 4.23. For the transducer $\mathcal{A}$ of Figure 2 seen as a transducer with output in the free commutative monoid $\Sigma^{\circledast}$, the corresponding minimal transducer $\operatorname{Min} \mathcal{L}$ is computed step-by-step in Figure 3.

Notice in particular how in Figure 3b both the functions recognized by the states 1 and 3 are left-divisible by α hence α is pulled back to the initialization value in Figure 3c. This would not have happened had M been the free monoid Σ^*, and the corresponding minimal transducer would have been different.



(a) Reach $\mathcal{A}$ (b) Total $\mathcal{A}$

(c) Prefix $\mathcal{A}$ (d) Obs(Reach $\mathcal{A}$)

Figure 3: Increasingly small transducers recognizing the same function as the transducer of Figure 2 when $M = \Sigma^{\circledast}$

5. Algorithms on monoidal transducers

Assume now that M is right-noetherian, left-cancellative up to invertibles on the left and right-coprime-cancellative, and that non-empty countable families of elements of M

all have a left-gcd that is unique up to invertibles on the right. Thanks to Lemma 4.16, Theorem 2.12 then tells us that the minimal M-transducer recognizing a $(\mathbf{Kl}(\mathcal{T}_M), 1, 1)$-language $\mathcal{L}$ always exists (and it is given by Corollary 4.20), but it does not tell us how to compute it (in finite time). This section thus discusses two algorithms for computing the minimal transducer, respectively instantiating Algorithms 1 and 2: the first one takes as input two oracles $\textsc{Eval}_{\mathcal{L}}$ and $\textsc{Equiv}_{\mathcal{L}}$ and learns $\operatorname{Min}\mathcal{L}$ (Section 5.1), while the second one takes as input any finite transducer recognizing $\mathcal{L}$ and transforms it into $\operatorname{Min}\mathcal{L}$ (Section 5.2). The right-noetherianity of M guarantees that these algorithms will terminate when the minimal transducer has finite state-set:

Lemma 5.1. *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* $\operatorname{length}_{\mathcal{M}}(m : Y \rightarrowtail X) = |X| - |Y|$.

Lemma 5.2. *An object X of $\mathbf{Kl}(\mathcal{T}_M)$ 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* $\operatorname{colength}_{\mathcal{E}}(e : X \multimap\!\!\!\!\rightarrow Y) = |X| - |Y| + \operatorname{rk} e$ *where* $\operatorname{rk} e = \sum_{e(x)=(v,y)} \operatorname{rk} v$.

Given a function $\Lambda \in (M+1)^I_*$, we write $\operatorname{rk}\Lambda = \operatorname{rk}(\operatorname{lgcd}\Lambda)$ and, finally, given a transducer $\mathcal{A}$, we then write $\operatorname{rk}\mathcal{A} = \sum_{s\in(\operatorname{Total}\mathcal{A})(\mathtt{st})} \operatorname{rk}\mathcal{L}_s$, $|\mathcal{A}|_{\mathtt{st}}$ for the cardinal of the state-set of $\mathcal{A}$, and $|\mathcal{A}|_{\rightarrow}$ for the number of transitions $s \odot a \neq \bot$ in $\mathcal{A}$. We will use these notions to express the complexity of our algorithms.

These two algorithms are modular in M: they only require to be given functions computing basic operations and may then be implemented the same no matter the structure of M. Hence we also assume that we are given access to the following oracles: one that checks for equality in M; $\otimes : M+1 \to M+1 \to M+1$ that computes the product in $M+1$ (with $v\bot = \bot v = \bot$); $\wedge : M+1 \to M+1 \to M+1$ that computes a left-gcd of two elements of $M+1$ (with $v \wedge \bot = \bot \wedge v = \bot$); $\textsc{LeftDivide}$ that takes as input $\delta, v \in M$ and computes a $\nu \in M$ such that $v = \delta\nu$ (or outputs $\nu = \bot$ when $v = \bot$) or fails when there is no such ν; and $\textsc{UpToInvLeft}$ that takes as input two families $v, \nu \in M^I$ indexed by a finite set I and outputs a $\chi \in M^\times$ such that $v = \chi\nu$ if such a χ exists, and $\bot$ otherwise. Note that when M is right-cancellative this last oracle may be derived from another oracle that only tests for equality up to invertibles of two elements of M (and not two families), and when M has no non-trivial invertible elements these two oracles just test for equality.

5.1. Learning

In this section, we thus first describe an algorithm that, given a subsequential function $\mathcal{L}$, learns $\operatorname{Min}\mathcal{L}$ using two oracles: $\textsc{Eval}_{\mathcal{L}}$ takes as input a word $w \in A^*$ and computes $\mathcal{L}(\triangleright w \triangleleft)$; and $\textsc{Equiv}_{\mathcal{L}}$ takes as input a transducer $\mathcal{A}$, decides whether it recognizes $\mathcal{L}$, and if not outputs a counter-example word $w \in A^*$ such that $(\mathcal{A} \circ \iota)(\triangleright w \triangleleft) \neq \mathcal{L}(\triangleright w \triangleleft)$.

When M is a free monoid, Vilar gives a very natural generalization of Angluin's L* algorithm for learning automata that does exactly this [1, 29]. Here we generalize Vilar's algorithm to arbitrary M by instantiating Algorithm 1 within $\mathbf{Kl}(\mathcal{T}_M)$: we get Algorithm 4. This instantiation is possible because the conditions on M also guarantee

the existence of the minimal biautomata described in Section 2.3. To the best of the author's knowledge, no other algorithm in the literature solves the problem of learning transducers whose output monoid is not free.

Algorithm 4 is very similar to Vilar's algorithm, 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 thus 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 then build 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 $q \in Q$ so that $R(q, e, \cdot)$ is never $\bot^T$ and so that, for any two distinct $q, q' \in S$, $R(q, e, \cdot)$ 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 function 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 $\mathcal{H}(Q, T) = \operatorname{Min} \mathcal{L}$. The three kinds of consistency issues that arise are again explained by the quaternary factorization system of Section 4.4, as the three conditions on line 10 hold when some morphism respectively does not belong to Tot, belongs to Tot but not Inv or belongs to $\mathrm{Tot} \cap \mathrm{Inv}$ but not Inj.

Theorem 5.3. *Algorithm 4 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).*

Algorithm 4 also differs from Vilar's original learning algorithm 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. It 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.

5.2. Minimization

To transform a transducer $\mathcal{A} = (S, (u_0, s_0), t, \odot)$ recognizing $\mathcal{L}$ into $\operatorname{Min} \mathcal{L}$, a first method is of course to use Algorithm 4, implementing $\textsc{Eval}_{\mathcal{L}}$ using $\mathcal{A}$ and $\textsc{Equiv}_{\mathcal{L}}$ by trying

Algorithm 4 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 $\mathcal{H}(Q, T)$ using Λ and R
    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

to compute a morphism of transducers $\operatorname{Reach}\mathcal{A} \twoheadrightarrow \mathcal{H}(Q,T)$. But in practice when M is a free monoid this is not the method that is used: there are algorithms designed specifically for minimizing a transducer, surveyed by Choffrut [9]. These algorithms all decompose the problem into four steps that match exactly those of Definition 4.22.

Computing Reach and Total is straightforward and may be done in linear time using depth-first searches on the underlying graph of $\mathcal{A}$, and this stays true for arbitrary M. The problem of computing Prefix and Obs was first tackled in [15], but the solution relied on the existence of an oracle that computes non-trivial left-divisors of countable (but rational) families, and also leaves $\operatorname{red}\Lambda$ over when repeatedly applied to some Λ. In comparison, our instantiation of Algorithm 2 and its dual requires much weaker tools: only UpToInvLeft and the binary $\wedge$.

The resulting algorithm itself is not new, as it is exactly the one described in [17]. Our main result relative to minimization is instead that this algorithm can be made to work with not just left- and right-cancellative output monoids: left-cancellativity up to invertibles on the left and right-coprime-cancellativity is enough. Similarly, right-noetherianity is enough to ensure termination, although the resulting complexity now also depends on the rank of elements of M. We thus do not describe the implementation in detail, but instead only discuss the insights we get through the big picture of the categorical framework, as well as how easy it is to actually retrieve the concrete algorithms for computing Reach, Total, Prefix and Obs.

Computing Reach**.** Since $\mathbf{Kl}(\mathcal{T}_M)$ has all coproducts, Corollary 3.9 holds and to compute Reach we may directly use Algorithm 3. Moreover all split epimorphisms in $\mathbf{Kl}(\mathcal{T}_M)$ are in $\operatorname{Surj} = \mathcal{E}$, hence we may also use the optimization of Corollary 3.11. The resulting algorithm is the transducer version of Example 3.12, a depth- or breadth-first search depending on the order in which the to-do list is gone through, and Lemma 5.1 tells us that the complexity is linear in the number of states of $\mathcal{A}$.

There is not much to say here: the instantiation goes smoothly.

Computing Total**.** The case of Total is more interesting, as instantiating the algorithm from the categorical framework is more troublesome, yet the result is still trivial: to remove the states that recognize the empty function, it is enough to do a depth- or breadth- first search on the underlying graph of the transducer, starting from the states where the termination function is defined and following transitions backwards. $\mathbf{Kl}(\mathcal{T}_M)$ has all finite products by Proposition 4.12 hence by the dual of Corollary 3.9 we may instantiate this algorithm through the dual of Algorithm 3. But this categorical algorithm tells us to keep track of a $(\operatorname{Surj}\cap\operatorname{Inj}\cap\operatorname{Inv})$-quotient $\mathcal{A}(\mathtt{st}) \twoheadrightarrow_1 \mathcal{A}(\mathtt{st})/T$, that is a partial function that tells us for each state $s \in \mathcal{A}(St)$ whether there is a $t \in T$ such that $\mathcal{A}(t\triangleleft)(s) \neq \bot$. To compute the factorizations incrementally we would then need to factorize the cartesian products of such functions, which, while correct, is quite unnatural.

A more intuitive way to instantiate the algorithm is to notice that $(\operatorname{Surj}\cap\operatorname{Inj}\cap\operatorname{Inv})$-quotients are exactly the split epimorphisms of $\mathbf{Kl}(\mathcal{T}_M)$, and, as such, have right inverses.

Keeping track of the right inverse of the quotient $\mathcal{A}(\mathtt{st}) \twoheadrightarrow_1 \mathcal{A}(\mathtt{st})/T$ would then mean keeping track of its codomain as a subset of $\mathcal{A}(\mathtt{st})$. This is more natural, only now we cannot compute the factorizations incrementally anymore, as we would want to do so by factorizing the coproduct of such embeddings but this is not encompassed by the dual of Lemma 3.7. This obstacle can be overcome by considering the transducer as the transpose of a non-deterministic transducer, i.e. seeing $\mathbf{Kl}(\mathcal{T}_M)^{\mathrm{op}}$ as a subcategory of $\mathbf{Kl}(\mathcal{P}(M^{\mathrm{op}} \times -))$, where $\mathcal{P}$ is the powerset monad. With this point of view, the states that recognize a non-empty function are exactly those that are reachable in the non-deterministic reverse transducer: computing Total can be done by reversing the transducer, computing Reach in the category of non-deterministic M^{op}-transducers and reversing the result back. This categorical algorithm yields precisely the graph traversal we were looking for, with the same catch as that of Example 3.13: it does not tell us that transitions along the same letter and to the same state can be considered separately.

However we choose to view this instantiation, the $(\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv})$-codimension of a set is its cardinal, hence we know the main loop will be run at most once for each state of $\mathcal{A}$.

Computing Prefix**.** The core of the problem is to compute $\mathrm{Prefix}\,\mathcal{A}$ knowing $\mathrm{Total}\,\mathcal{A}$, that is to compute the left-gcds of the functions recognized from each state that is reachable and recognizes a non-empty function. Béal and Carton do it by pulling back letters along the transitions of $\mathcal{A}$ [4], but this relies crucially on M being a free monoid as the letters to pull back must be produced by every transition going out of a given state. This would for instance not be enough to see that the left-gcd of state 3 in Figure 3b is α (when M is the free commutative monoid). Breslauer does it by computing over-approximations and the length of the left-gcds, and then taking the corresponding prefixes [7], but again this algorithm does not generalize to arbitrary M because it relies crucially on the free monoid being graded.

In the general case, $\mathrm{Prefix}\,\mathcal{A}$ may be computed using a fixed-point algorithm, keeping track of a tentative left-gcd $l(s)$ for each state $s \in \mathcal{A}(\mathtt{st})$ and replacing $l(s)$ with $l(s) \wedge \bigwedge_{a \in A}(s \bigcirc a) l(s \cdot a)$ as long as needed [17, Section 4], and this is exactly what we get when we instantiate the dual of Algorithm 3 and the optimization dual to that of Corollary 3.11 for the factorization system $(\mathcal{E}_2, \mathcal{M}_2)$.

An interesting thing to notice here is that since we start from $\mathrm{Total}\,\mathcal{A}$, we may consider that we are computing the $(\mathrm{Surj} \cap \mathrm{Inj}, \mathrm{Inv} \cap \mathrm{Tot})$-factorization of $\mathrm{Total}\,\mathcal{A} \twoheadrightarrow_1 \mathcal{A}^{final}(\mathcal{L})$ in the category of total transducers (those whose final morphism to $\mathcal{A}^{final}$ is in Tot) and total morphisms between them (the morphisms between them are also in Tot, thanks to [8, Lemma 14.9(1)]). Hence we actually compute a $(\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Tot}, \mathrm{Inv} \cap \mathrm{Tot})$-factorization. This is particularly interesting when giving a complexity-bound for the number of times the main loop is run. The $(\mathrm{Surj} \cap \mathrm{Inj})$-codimension of some $e : X \twoheadrightarrow_2 Y$ is indeed $|X| - |Y| + \mathrm{rk}\, e$, whereas the $(\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Tot})$-codimension of some $e : X \underset{2}{\rightarrowtail}\!\underset{1}{\twoheadrightarrow} Y$ is just $\mathrm{rk}\, e$: the prior knowledge that the transducer was total got rid of the cardinality term. Still, [17, Lemma 6] gets a much better complexity result than we do: where we

get a complexity linear in $\operatorname{rk}\mathcal{A}$, they get one linear in $|\mathcal{A}(\mathtt{st})|$. This is because they use an assumption quite different from right-noetherianity (namely, that if v left-divides ω and $\nu\omega$ then it left-divides νv) that allows them to prove some kind of pumping lemma for transducers.

Computing Obs**.** Computing Obs from $\operatorname{Prefix}\mathcal{A}$ is also straightforward when M is the free monoid as it amounts to minimizing a deterministic automaton, which can be done using Hopcroft's algorithm for instance [9]. This is still true when M has no non-trivial invertibles, but for arbitrary M it is not as easy: checking whether two states are equivalent amounts to checking for equality of (left-coprime) subsequential functions up to invertibles.

A general algorithm for computing $\operatorname{Obs}\mathcal{A}$ can be designed by extending Moore's partition refinement algorithm with equality up to invertibles. In a nutshell, this algorithm should maintain a partition of the states into equivalence classes and for every pair of equivalent states an invertible element that witnesses the equivalence, and refine this equivalence as long as needed [17, Section 5]. This is, again, exactly the algorithm we get when we instantiate the dual of Algorithm 3 along with the optimization dual to that of Corollary 3.11 for the factorization system $(\mathcal{E}_3, \mathcal{M}_3)$.

Just as for Prefix we restricted ourselves to the category of total transducers, we can restrict ourselves to the category of onwards transducers (those for which the final morphism to $\mathcal{A}^{final}(\mathcal{L})$ is in $\operatorname{Inv}\cap\operatorname{Tot}$) and $(\operatorname{Inv}\cap\operatorname{Tot})$-morphisms between them. In particular, this means the complexity of computing $\operatorname{Obs}\mathcal{A}$ from $\operatorname{Prefix}\mathcal{A}$ only depends on the number of states of $\operatorname{Total}\mathcal{A}$, and we retrieve the same complexity as that of [17, Lemma 13].

Worklist algorithms everywhere. A final insight we get from applying the minimization algorithm of our categorical framework is that the algorithms for computing Reach, Total, Prefix and Obs all share the same structure: they are worklist algorithms, that is fixed-points of an operator on a graph defined by dataflow-style equations on each vertex.

Since the technical details of the algorithm are quite heavy to lay out, we also provide a modular `OCaml` implementation for right-cancellative output monoids [2] as a proof-of-concept (it is in particular not fully optimized), that includes implementations of the oracles for trace monoids, the group $\mathbb{Z}$ and products of monoids. It uses the `OCamlGraph` library [12] to represent the underlying graph of a transducer, and its `Fixpoint` module to implement each of Reach, Total, Prefix and Obs as worklist algorithms: this module only requires the dataflow-style equations and computes the corresponding fixpoint automatically.

6. Summary and future work

In this work, we extended Colcombet, Petrişan and Stabile's active learning categorical framework with a minimization algorithm, and instantiated it with monoidal transduc-

ers. We gave some simple sufficient conditions on the output monoid for the minimal transducer to exist and be unique up to isomorphism, which in particular extend Gerdjikov's conditions for minimization to be possible [17]. 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. Although we expect them to be useful in concurrency theory and in natural language processing, such examples are thus still to be found. In particular, can monoidal 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 4.3 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 2.12, 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". In: *Information and Computation* 75.2 (Nov. 1, 1987), pp. 87–106. ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90052-6. URL: https://www.sciencedirect.com/science/article/pii/0890540187900526 (visited on 05/04/2023).

[2] Quentin Aristote. *Monoidal Transducers Minimization*. 2022. URL: https://gitlab.math.univ-paris-diderot.fr/aristote/monoidal-transducers-minimization.

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

[4] Marie-Pierre Béal and Olivier Carton. "Computing the Prefix of an Automaton". In: *Informatique Théorique et Applications* 34.6 (2000), p. 503. URL: https://hal.science/hal-00619217 (visited on 05/04/2023).

[5] F. Bergadano and S. Varricchio. “Learning Behaviors of Automata from Multiplicity and Equivalence Queries”. In: *Algorithms and Complexity*. Ed. by M. Bonuccelli, P. Crescenzi, and R. Petreschi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1994, pp. 54–62. ISBN: 978-3-540-48337-3. DOI: 10.1007/3-540-57811-0_6.

[6] Francesco Bergadano and Stefano Varricchio. “Learning Behaviors of Automata from Multiplicity and Equivalence Queries”. In: *SIAM Journal on Computing* 25.6 (Dec. 1996), pp. 1268–1280. ISSN: 0097-5397. DOI: 10.1137/S009753979326091X. URL: https://epubs.siam.org/doi/10.1137/S009753979326091X (visited on 05/04/2023).

[7] Dany Breslauer. “The Suffix Tree of a Tree and Minimizing Sequential Transducers”. In: *Theoretical Computer Science* 191.1 (Jan. 30, 1998), pp. 131–144. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(96)00319-2. URL: https://www.sciencedirect.com/science/article/pii/S0304397596003192 (visited on 05/04/2023).

[8] Gabriele Castellini et al. “Abstract and Concrete Categories”. In: *The American Mathematical Monthly* 98.3 (Mar. 1991), pp. 285–287. ISSN: 0002-9890, 1930-0972. DOI: 10.1080/00029890.1991.11995743. URL: https://www.tandfonline.com/doi/full/10.1080/00029890.1991.11995743 (visited on 03/20/2023).

[9] Christian Choffrut. “Minimizing Subsequential Transducers: A Survey”. In: *Theoretical Computer Science*. Selected Papers in Honor of Jean Berstel 292.1 (Jan. 10, 2003), pp. 131–143. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(01)00219-5. URL: https://www.sciencedirect.com/science/article/pii/S0304397501002195 (visited on 05/04/2023).

[10] Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. *Learning Automata and Transducers: A Categorical Approach*. Oct. 26, 2020. DOI: 10.48550/arXiv.2010.13675. arXiv: 2010.13675 [cs]. URL: http://arxiv.org/abs/2010.13675 (visited on 04/03/2023). preprint.

[11] Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. “Learning Automata and Transducers: A Categorical Approach”. In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Ed. by Christel Baier and Jean Goubault-Larrecq. Vol. 183. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, 15:1–15:17. ISBN: 978-3-95977-175-7. DOI: 10.4230/LIPIcs.CSL.2021.15. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13449 (visited on 05/04/2023).

[12] Sylvain Conchon, Jean-Christophe Filliâtre, and Julien Signoles. “Designing a Generic Graph Library Using ML Functors”. In: *Trends in Functional Programming*. 2008, pp. 124–140. URL: https://www.lri.fr/~filliatr/ftp/publis/ocamlgraph-tfp-8.pdf.

[13] Robert Cori and Dominique Perrin. "Automates et commutations partielles". In: *RAIRO. Informatique théorique* 19.1 (1 1985), pp. 21–32. ISSN: 0399-0540, 2777-3337. DOI: 10.1051/ita/1985190100211. URL: https://www.rairo-ita.org/articles/ita/abs/1985/01/ita1985190100211/ita1985190100211.html (visited on 05/04/2023).

[14] Ulrich Dorsch et al. "Efficient Coalgebraic Partition Refinement". In: *28th International Conference on Concurrency Theory (CONCUR 2017)*. Ed. by Roland Meyer and Uwe Nestmann. Vol. 85. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 32:1–32:16. ISBN: 978-3-95977-048-4. DOI: 10.4230/LIPIcs.CONCUR.2017.32. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7793 (visited on 07/24/2023).

[15] 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. USA: Association for Computational Linguistics, May 27, 2003, pp. 64–71. DOI: 10.3115/1073445.1073454. URL: https://dl.acm.org/doi/10.3115/1073445.1073454 (visited on 05/04/2023).

[16] Charles N. Fischer, Ron K. Cytron, and Richard J. LeBlanc. *Crafting a Compiler*. Crafting a Compiler with C. Boston: Addison-Wesley, 2010. 683 pp. ISBN: 978-0-13-606705-4.

[17] Stefan Gerdjikov. "A General Class of Monoids Supporting Canonisation and Minimisation of (Sub)Sequential Transducers". In: *Language and Automata Theory and Applications*. Ed. by Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 143–155. ISBN: 978-3-319-77313-1. DOI: 10.1007/978-3-319-77313-1_11.

[18] John Hopcroft. "AN n Log n ALGORITHM FOR MINIMIZING STATES IN A FINITE AUTOMATON". In: *Theory of Machines and Computations*. Ed. by Zvi Kohavi and Azaria Paz. Academic Press, Jan. 1, 1971, pp. 189–196. ISBN: 978-0-12-417750-5. DOI: 10.1016/B978-0-12-417750-5.50022-1. URL: https://www.sciencedirect.com/science/article/pii/B9780124177505500221 (visited on 05/16/2023).

[19] Ronald M. Kaplan and Martin Kay. "Regular Models of Phonological Rule Systems". In: *Computational Linguistics* 20.3 (1994), pp. 331–378. URL: https://aclanthology.org/J94-3001 (visited on 05/04/2023).

[20] Kevin Knight and Jonathan May. "Applications of Weighted Automata in Natural Language Processing". In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Monographs in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer, 2009, pp. 571–596. ISBN: 978-3-642-01492-5. DOI: 10.1007/978-3-642-01492-5_14. URL: https://doi.org/10.1007/978-3-642-01492-5_14 (visited on 05/04/2023).

[21] Saunders Mac Lane. *Categories for the Working Mathematician*. Vol. 5. Graduate Texts in Mathematics. New York, NY: Springer, 1978. ISBN: 978-1-4419-3123-8 978-1-4757-4721-8. DOI: 10.1007/978-1-4757-4721-8. URL: http://link.springer.com/10.1007/978-1-4757-4721-8 (visited on 05/05/2023).

[22] Edward F. Moore. "Gedanken-Experiments on Sequential Machines". In: *Automata Studies*. Annals of Mathematics Studies, No. 34. Princeton University Press, Princeton, N.J., 1956, pp. 129–153.

[23] Daniela Petrişan and Thomas Colcombet. "Automata Minimization: A Functorial Approach". In: *Logical Methods in Computer Science* Volume 16, Issue 1 (Mar. 23, 2020). DOI: 10.23638/LMCS-16(1:32)2020. URL: https://lmcs.episciences.org/6213/pdf (visited on 04/12/2023).

[24] Jacques Sakarovitch. "1. The Simplest Possible Machine". In: *Elements of Automata Theory*. Ed. by Thomas Reuben. Cambridge: Cambridge University Press, 2009, pp. 49–216. DOI: 10.1017/CBO9781139195218.005.

[25] M. P. Schützenberger. "On the Definition of a Family of Automata". In: *Information and Control* 4.2 (Sept. 1, 1961), pp. 245–270. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(61)80020-X. URL: https://www.sciencedirect.com/science/article/pii/S001999586180020X (visited on 05/04/2023).

[26] Jenö Szigeti. "On Limits and Colimits in the Kleisli Category". In: *Cahiers de topologie et géométrie différentielle* 24.4 (1983), pp. 381–391. ISSN: 2681-2398. URL: http://www.numdam.org/item/?id=CTGDC_1983__24_4_381_0 (visited on 05/23/2023).

[27] 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. New York, NY, USA: Association for Computing Machinery, July 8, 2020, pp. 900–914. ISBN: 978-1-4503-7104-9. DOI: 10.1145/3373718.3394775. URL: https://dl.acm.org/doi/10.1145/3373718.3394775 (visited on 05/04/2023).

[28] Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. "Learning Automata with Side-Effects". In: *Coalgebraic Methods in Computer Science*. Ed. by Daniela Petrişan and Jurriaan Rot. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 68–89. ISBN: 978-3-030-57201-3. DOI: 10.1007/978-3-030-57201-3_5.

[29] Juan Miguel Vilar. "Query Learning of Subsequential Transducers". In: *Grammatical Interference: Learning Syntax from Sentences*. Ed. by Laurent Miclet and Colin de la Higuera. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 72–83. ISBN: 978-3-540-70678-6. DOI: 10.1007/BFb0033343.

[30] Thorsten Wißmann et al. "A Coalgebraic View on Reachability". In: *Commentationes Mathematicae Universitatis Carolinae* 60.4 (2019), pp. 605–638. ISSN: 0010-2628 (print). URL: https://dml.cz/handle/10338.dmlcz/147977 (visited on 05/05/2023).

A. Proofs for Section 1 (Introduction)

Lemma 1.1. *Let $A = \{a\}$, $\Sigma = \{\alpha, \beta, \gamma\}$, let $\Sigma/\!\sim$ be the monoid Σ^* with the additional assumption that $\alpha\beta = \beta\alpha$ 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 4.9) 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 4 with $M = \Sigma^*$ and $M^\times = \{\epsilon\}$) 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. Proofs for Section 3 (An algorithm for minimizing $(\mathcal{C}, X, Y)$-automata)

Lemma 3.1 ([23, Remark 2.5])**.** *The dual of the $(\mathcal{E}, \mathcal{M})$-factorization $\operatorname{Obs}\mathcal{A}$ in $\mathbf{Auto}(\mathcal{L})$ is the $(\mathcal{M}^{\mathrm{op}}, \mathcal{E}^{\mathrm{op}})$-factorization $\operatorname{Reach}\mathcal{A}^{\mathrm{op}}$ in $\mathbf{Auto}(\mathcal{L}^{\mathrm{op}})$.*

Proof. $\operatorname{Obs}\mathcal{A}$ is given by the $(\mathcal{E}, \mathcal{M})$-factorization $\mathcal{A} \twoheadrightarrow \operatorname{Obs}\mathcal{A} \rightarrowtail \mathcal{A}^{final}$ in $\mathbf{Auto}(\mathcal{L})$. Its dual is thus given by the $(\mathcal{M}^{\mathrm{op}}, \mathcal{E}^{\mathrm{op}})$-factorization $(\mathcal{A}^{final})^{\mathrm{op}} \twoheadrightarrow (\operatorname{Obs}\mathcal{A})^{\mathrm{op}} \rightarrowtail \mathcal{A}^{\mathrm{op}}$ in $\mathbf{Auto}(\mathcal{L})^{\mathrm{op}}$. But $\mathcal{A}^{final}$ is final in $\mathbf{Auto}(\mathcal{L})$ hence its dual is initial in $\mathbf{Auto}(\mathcal{L})^{\mathrm{op}}$. Similarly, $(\mathcal{M}^{\mathrm{op}}, \mathcal{E}^{\mathrm{op}})$ is a factorization system in $\mathbf{Auto}(\mathcal{L})^{\mathrm{op}}$ [8, Proposition 14.3], hence $(\operatorname{Obs}\mathcal{A})^{\mathrm{op}}$ is none other than $\operatorname{Reach}\mathcal{A}^{\mathrm{op}}$. □

Lemma 3.4. *(similar to [10, Lemma 31]) For any $w \in A^*$ and $Q' \subseteq A^*$ such that $Qw = \{qw \mid q \in Q\} \subseteq Q'$, there is a unique morphism $\kappa^{Q'}_{Q\cdot w} \big/ \mathcal{A}(w) : Q/\mathcal{A}(\mathtt{st}) \to Q'/\mathcal{A}(\mathtt{st})$ making the two squares in the diagram below commute. Moreover, if $uv = w$ and $Qw \subseteq Q'v \subseteq Q''$, then $\kappa^{Q''}_{Q'\cdot v} \big/ \mathcal{A}(v) \circ \kappa^{Q'}_{Q\cdot u} \big/ \mathcal{A}(u) = \kappa^{Q''}_{Q\cdot w} \big/ \mathcal{A}(w)$.*

Proof. $\kappa^{Q'}_{Q\cdot w} \big/ \mathcal{A}(w)$ arises immediately by the diagonal fill-in property of $(\mathcal{E}, \mathcal{M})$ applied to $m_{Q'} \circ \left(e_{Q'} \circ \coprod_{q\in Q} \kappa_q\right) = (\mathcal{A}(w) \circ m_Q) \circ e_Q$. Since this diagonal fill-in is unique, by composing these diagrams we get the last equality. □

Theorem 3.6. *If $\mathcal{A}(\mathtt{st})$ is $\mathcal{M}$-noetherian, Algorithm 2 is correct, terminates, and the condition on line 5 is satisfied at most* $\operatorname{length}_{\mathcal{M}}(m_\varnothing : \varnothing/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st}))$ *times.*

Proof. We first focus on termination, then on correctness.

Termination. Every time line 7 is executed, the condition on line 5 must hold. Hence if we write Q_n for the value of Q_{done} after n executions of line 7, then $Q_0 = \varnothing$ and $(Q_n/\mathcal{A}(\mathtt{st}) \rightarrowtail Q_{n+1}/\mathcal{A}(\mathtt{st}))_n$ forms a strict chain of $\mathcal{M}$-subobjects of $\mathcal{A}(\mathtt{st})$ (it is strict because the morphisms it contains are not $\mathcal{E}$-morphisms hence not isomorphisms). But $\mathcal{A}(\mathtt{st})$ is $\mathcal{M}$-noetherian by assumption hence this chain must be finite and line 7 and thus line 6 must be executed a finite number of times, and in particular at most $\operatorname{length}_{\mathcal{M}} m_{Q_0} = \operatorname{length}_{\mathcal{M}} m_\varnothing$ times. But we only add at most $|A|$ elements to the to-do list on line 6, hence the while loop is run a finite number of times, in particular at most $1 + |A| \operatorname{length}_{\mathcal{M}}(m_\varnothing)$ times.

Correctness. Let us first show some preliminary results.

Lemma B.1. *Consider the following diagonal fill-in (where g is an $\mathcal{M}$-morphism):*

$$\begin{array}{ccccc} Y_1 & \xrightarrow{e_y} & Y_1/Y_2 & \xrightarrow{m_y} & Y_2 \\ {\scriptstyle f}\uparrow & & \uparrow{\scriptstyle f/g} & & \uparrow{\scriptstyle g} \\ X_1 & \xrightarrow[e_x]{} & X_1/X_2 & \xrightarrow[m_x]{} & X_2 \end{array}$$

f/g is an $\mathcal{M}$-morphism, and it is an isomorphism if and only if there is an h such that the following diagram commutes:

$$\begin{array}{ccc} Y_1 & \xrightarrow{e_y} & Y_1/Y_2 \\ {\scriptstyle f}\uparrow & \overset{h}{\searrow} & \uparrow{\scriptstyle f/g} \\ X_1 & \xrightarrow[e_x]{} & X_1/X_2 \end{array}$$

Proof. f/g is an $\mathcal{M}$-morphism by [8, Proposition 14.9(1)]. If it is an isomorphism, then $h = (f/g)^{-1} \circ e_y$ immediately makes the diagram above commute.

Conversely, assume such an h exists and $(\mathcal{E}, \mathcal{M})$-factor it as $h = m \circ e : Y_1 \twoheadrightarrow Z \rightarrowtail X_1/X_2$. By unicity of the diagonal fill-in, there is an isomorphism $\chi : Y_1/Y_2 \cong Z$ such

that $e = \chi \circ e_y$ and $f/g \circ m \circ \chi = \mathrm{id}$. We claim that $m \circ \chi$ is the inverse of f/g: we already have that $f/g \circ m \circ \chi = \mathrm{id}$, but also

$$\begin{aligned} g \circ m_x \circ m \circ \chi \circ f/g &= m_y \circ f/g \circ m \circ \chi \circ f/g \\ &= m_y \circ f/g \\ &= g \circ m_x \end{aligned}$$

and

$$\begin{aligned} m \circ \chi \circ f/g \circ e_x &= m \circ \chi \circ e_y \circ f \\ &= h \circ f \\ &= e_x \end{aligned}$$

hence by unicity of the diagonal fill-in (within the square composed of two copies of e_x and $g \circ m_x$), $m \circ \chi \circ f/g = \mathrm{id}$. □

Definition B.2 (prefix-closedness)**.** $Q \subseteq A^*$ is said to be *prefix-closed* if it contains the empty word and if for every $q \in A^*$ and $a \in A$, if $qa \in Q$ then $q \in Q$.

Proposition B.3. *Let $Q \subseteq A^*$ be prefix-closed. If $\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism, then so is $\kappa_Q^{A^*} \Big/ \mathcal{A}(\mathtt{st})$.*

Proof. We apply Lemma B.1 and build a morphism $h : \coprod_{A^*} \mathcal{A}(\mathtt{in}) \to Q/\mathcal{A}(\mathtt{st})$ that makes the corresponding diagram commute. We thus let $h \circ \kappa_\epsilon = e_Q \circ \kappa_\epsilon$ (this is possible because Q is prefix-closed hence contains ϵ) and, by induction, for $w \in A^*$ and $a \in A$ we let $h \circ \kappa_{wa} = \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ \kappa_{Q \cdot a}^{Q \cup QA} \Big/ \mathcal{A}(a) \circ h \circ \kappa_w$.

We then have that $e_Q = h \circ [\kappa_q]_{q \in Q}$. The proof is by induction on Q, which is possible because Q is prefix-closed. We already have $h \circ \kappa_\epsilon = e_Q \circ \kappa_\epsilon$ by definition, and if $h \circ \kappa_w = e_Q \circ \kappa_w$, then for $a \in A$ we have

$$\begin{aligned} h \circ \kappa_{wa} &= \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ \kappa_{Q \cdot a}^{Q \cup QA} \Big/ \mathcal{A}(a) \circ h \circ \kappa_w \\ &= \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ \kappa_{Q \cdot a}^{Q \cup QA} \Big/ \mathcal{A}(a) \circ e_Q \circ \kappa_w \\ &= \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ e_{Q \cup QA} \circ [\kappa_{qa}]_{q \in Q} \circ \kappa_w \\ &= \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} \circ e_{Q \cup QA} \circ [\kappa_q]_{q \in Q} \circ \kappa_{wa} \\ &= e_Q \circ \kappa_{wa} \end{aligned}$$

Moreover, for all $w \in A^*$, we have $\kappa_{Q \cdot w}^{A^*} \Big/ \mathcal{A}(w) = \kappa_{Q \cup QA \cdot w}^{A^*} \Big/ \mathcal{A}(w) \circ \kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})$ hence $\kappa_{Q \cdot w}^{A^*} \Big/ \mathcal{A}(w) \circ \left(\kappa_Q^{Q \cup QA} \Big/ \mathcal{A}(\mathtt{st})\right)^{-1} = \kappa_{Q \cup QA \cdot w}^{A^*} \Big/ \mathcal{A}(w)$ and it follows by induction

(again, thanks to the prefix-closedness of Q) that

$$\begin{aligned}\kappa_Q^{A^*} \Big/ \mathcal{A}(\mathtt{st}) \circ h \circ \kappa_w &= \kappa_{Q\cdot w}^{A^*} \Big/ \mathcal{A}(w) \circ e_Q \circ \kappa_\epsilon \\ &= e_{A^*} \circ [\kappa_{qw}]_{q\in Q} \circ \kappa_\epsilon \\ &= e_{A^*} \circ \kappa_w\end{aligned}$$

hence that $\kappa_Q^{A^*} \Big/ \mathcal{A}(\mathtt{st}) \circ h = e_{A^*}$. □

Lemma B.4 (generalization of [10, Lemma 40])**.** *For every k in some set K, let $P_k \subseteq Q_k \subseteq A^*$ be such that $\kappa_{P_k}^{Q_k} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism. Also assume that $Q_k \cap Q_{k'} = P_k \cap P_{k'}$ for all $k \neq k' \in K$. Then, $\kappa_{\cup_K P_k}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st})$ is also an isomorphism.*

Proof. We apply Lemma B.1 and build $h : \coprod_{\cup_K Q_k} \mathcal{A}(\mathtt{in}) \to \cup_K P_k / \mathcal{A}(\mathtt{st})$. If $p \in P_l$ for some $l \in K$, set $h \circ \kappa_p = e_{\cup_K P_k} \circ \kappa_p$. Otherwise, for $q \in Q_l$ for some (unique) $l \in K$, set $h \circ \kappa_q = \kappa_{P_l}^{\cup_K P_k} \Big/ \mathcal{A}(\mathtt{st}) \circ h_l \circ \kappa_q$ where $h_l : \coprod_{Q_l} \mathcal{A}(\mathtt{in}) \to P_l / \mathcal{A}(\mathtt{st})$ witnesses that $\kappa_{P_l}^{Q_l} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism.

We then have by definition $h \circ [\kappa_w]_{w \in \cup_K P_k} = e_{\cup_K P_k}$ and in particular $\kappa_{\cup_K P_k}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st}) \circ h \circ \kappa_p = e_{\cup_K Q_k} \circ \kappa_p$ when $p \in \bigcup_{k\in K} P_k$. Moreover, for some $l \in K$ and $q \in Q_l$ such that $q \notin \bigcup_{k\in K} P_k$, we have

$$\begin{aligned}\kappa_{\cup_K P_k}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st}) \circ h \circ \kappa_q &= \kappa_{P_l}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st}) \circ h_l \circ \kappa_q \\ &= \kappa_{Q_l}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st}) \circ \kappa_{P_l}^{Q_l} \Big/ \mathcal{A}(\mathtt{st}) \circ h_l \circ \kappa_q \\ &= \kappa_{Q_l}^{\cup_K Q_k} \Big/ \mathcal{A}(\mathtt{st}) \circ e_{Q_l} \circ \kappa_q \\ &= e_{\cup_K Q_k} \circ \kappa_q\end{aligned}$$

hence h has the relevant diagram commute. □

We now have everything we need to show the algorithm is correct.

First note that it is easy to show by induction that all the subsets that go through to-do list are pairwise disjoint, hence why we use disjoint unions in the algorithm. But the fact that these unions are disjoint is not needed for the proof of correctness.

Consider some $q \in Q_{done}$ after the algorithm has stopped, and some $a \in A$. The only way to get added to Q_{done} is through line 7 of the algorithm, hence q must have belonged to some Q. Hence there was an $\tilde{A}$ such that $a \in \tilde{A}$ for which $Q\tilde{A}$ must have been added to the to-do list and thus have been considered in a run of the if statement on line 5. During this run, either $\kappa_{Q_{done}}^{Q_{done} \cup Q\tilde{A}} \Big/ \mathcal{A}(\mathtt{st})$ was already an isomorphism and nothing changed; or it was not, in which case $Q\tilde{A}$ was added to Q_{done} so that $Q_{done} = Q_{done} \cup Q\tilde{A}$ right after, and we would thus still get that $\kappa_{Q_{done}}^{Q_{done} \sqcup Q\tilde{A}} \Big/ \mathcal{A}(\mathtt{st})$ had become an $\mathcal{E}$-morphism

hence an isomorphism (by the dual of [8, Proposition 14.9(1)], because the morphism $\coprod_{Q_{done}} \mathcal{A}(\mathtt{in}) \to \coprod_{Q_{done} \cup Q\tilde{A}} \mathcal{A}(\mathtt{in})$ is an isomorphism).

By Lemma B.4, for any $Q_1 \subseteq Q_2 \subseteq A^*$, if $\kappa_{Q_1}^{Q_1 \cup Q} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism then so is $\kappa_{Q_2}^{Q_2 \cup Q} \Big/ \mathcal{A}(\mathtt{st})$. Hence since Q_{done} may only grow and since we just showed that at some point of the execution of the algorithm the morphism $\kappa_{Q_{done}}^{Q_{done} \cup Q\tilde{A}} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism, it stays so until the algorithm stops.

Hence at the end of the algorithm, for every $q \in Q_{done}$ and $a \in A$ there is some $Q\tilde{A}$ such that $qa \in Q\tilde{A}$ and $\kappa_{Q_{done}}^{Q_{done} \sqcup Q\tilde{A}} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism. Using Lemma B.4 again, we get that $\kappa_{Q_{done}}^{Q_{done} \cup Q_{done}A} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism. But Q_{done} must be prefix-closed: ϵ has gone through Q_{todo} and $qa \in Q\tilde{A}$ only gets added to Q_{todo} if $q \in Q$ has been added to Q_{done} already. Hence Proposition B.3 applies and $\kappa_{Q_{done}}^{A^*} \Big/ \mathcal{A}(\mathtt{st}) : Q_{done}/\mathcal{A}(\mathtt{st}) \cong (\mathrm{Reach}\,\mathcal{A})(\mathtt{st})$ is an isomorphism at the end of the algorithm, and it is easy to check that it forms a morphism of automata between the automaton produced by the algorithm and $\mathrm{Reach}\,\mathcal{A}$. □

Lemma 3.7. *For every $Q, Q' \subseteq A^*$, if the coproduct $Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st})$ exists, $[m_Q, m_{Q'}] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$ $(\mathcal{E}, \mathcal{M})$-factors into*

$$m_{Q \sqcup Q'} \circ \left[\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \,,\, \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \twoheadrightarrow (Q \sqcup Q')/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$$

Proof. Factor $[m_Q, m_{Q'}] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \to \mathcal{A}(\mathtt{st})$ as $e = [z_Q, z_{Q'}] : Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st}) \twoheadrightarrow Z$ and $m : Z \rightarrowtail \mathcal{A}(\mathtt{st})$. Since $e_Q + e_{Q'} : \coprod_Q \mathcal{A}(\mathtt{in}) + \coprod_{Q'} \mathcal{A}(\mathtt{in}) \twoheadrightarrow Q/\mathcal{A}(\mathtt{st}) + Q'/\mathcal{A}(\mathtt{st})$ is an $\mathcal{E}$-morphism (as a coproduct of $\mathcal{E}$-morphisms, by the dual of [8, Proposition 14.15]), $e \circ (e_Q + e_{Q'}) : \coprod_{Q \sqcup Q'} \mathcal{A}(\mathtt{in}) \twoheadrightarrow Z$ and $m : Z \rightarrowtail \mathcal{A}(\mathtt{st})$ form a factorization of $[m_Q \circ e_Q, m_{Q'} \circ e_{Q'}] = [\mathcal{A}(\triangleright q \triangleleft)]_{q \in Q \sqcup Q'} : \coprod_{Q \sqcup Q'} \mathcal{A}(\mathtt{in}) \to \mathcal{A}(\mathtt{st})$ as in Definition 3.3. By unicity of the diagonal fill-in, there is an isomorphism $x : Z \cong (Q \sqcup Q')/\mathcal{A}(\mathtt{st})$ such that $m = m_{Q \sqcup Q'} \circ x$ and $x \circ e \circ (e_Q + e_{Q'}) = e_{Q \sqcup Q'}$.

In particular, $x \circ z_Q \circ e_Q = e_{Q \sqcup Q'} \circ [\kappa_q]_{q \in Q}$ and $m_{Q \sqcup Q'} \circ x \circ z_Q = m_Q$, hence by unicity of the diagonal fill-in $x \circ z_Q = \kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$. The same results holds for Q' hence $[m_Q, m'_Q] = (m \circ x^{-1}) \circ [x \circ z_Q, x \circ z_{Q'}]$ yields the expected factorization. □

Lemma 3.8. *For every $Q \subseteq A^*$, $\mathcal{A}(a) \circ m_Q$ $(\mathcal{E}, \mathcal{M})$-factors into $m_{Qa} \circ \kappa_{Q \cdot a}^{Qa} \Big/ \mathcal{A}(\mathtt{st}) : Q/\mathcal{A}(\mathtt{st}) \twoheadrightarrow Qa/\mathcal{A}(\mathtt{st}) \rightarrowtail \mathcal{A}(\mathtt{st})$.*

Proof. The equality holds by definition of $\kappa_{Q \cdot a}^{Qa} \Big/ \mathcal{A}(\mathtt{st})$ in Lemma 3.4. This morphism is an $\mathcal{E}$-morphism because $[\kappa_{qa}]_{q \in Q}$ is an isomorphism hence the dual of [8, Proposition 14.9(1)] applies. □

Lemma 3.10. *If $\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) : Q/\mathcal{A}(\mathtt{st}) \rightarrowtail (Q \sqcup Q')/\mathcal{A}(\mathtt{st})$ is an isomorphism then*

$$m_{Q'} = m_Q \circ \left(\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right)^{-1} \circ \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$$

Conversely, in the context of Lemma 3.7, assume also that all split epimorphisms are in $\mathcal{E}$ (if $e \circ s = \mathrm{id}$ for some e, s then $e \in \mathcal{E}$). If $m_{Q'}$ factors as $m_Q \circ f$ for some $f : Q/\mathcal{A}(\mathtt{st}) \to (Q \sqcup Q')/\mathcal{A}(\mathtt{st})$ then $\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$ is an isomorphism.

Proof. We immediately have that

$$\begin{aligned} m_{Q'} &= m_{Q \sqcup Q'} \circ \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \\ &= m_Q \circ \left(\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right)^{-1} \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \end{aligned}$$

since $m_Q = m_{Q \sqcup Q'} \circ \kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st})$.

Conversely, assume $m_{Q'} = m_Q \circ f$ for some f. Then $[m_Q, m_{Q'}] = m_Q \circ [\mathrm{id}, f]$. But $[\mathrm{id}, f]$ is a split epimorphism ($[\mathrm{id}, f] \circ \kappa_1 = \mathrm{id}$) hence it is also an $\mathcal{E}$-morphism. We thus get two $(\mathcal{E}, \mathcal{M})$-factorizations for $[m_Q, m_{Q'}]$:

$$m_Q \circ [\mathrm{id}, f] = m_{Q \sqcup Q'} \circ \left[\kappa_Q^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \, , \, \kappa_{Q'}^{Q \sqcup Q'} \Big/ \mathcal{A}(\mathtt{st}) \right]$$

□

C. Proofs for Section 4 (The category of monoidal transducers)

Lemma 4.6. *M is right-noetherian if and only if for any sequences $(u_n)_{n \in \mathbb{N}}$ and $(v_n)_{n \in \mathbb{N}}$ of M such that $v_n = v_{n+1} u_n$ for all $n \in \mathbb{N}$, there is some $n \in \mathbb{N}$ such that for all $i \geq n$, u_i is invertible.*

Proof. The reverse implication is trivial. Now assume M right-noetherian, and consider $(u_n)_{n \in \mathbb{N}}$ and $(v_n)_{n \in \mathbb{N}}$ such that $v_n = v_{n+1} u_n$ for all $n \in \mathbb{N}$. Consider the set $I = \{i_0 < i_1 < \ldots\}$ of all the indices i such that u_i is not invertible and, letting $i_{-1} = -1$, define $u'_n = u_{i_n} \cdots u_{i_{n-1}+1}$ and $v'_n = v_{i_n+1}$ for all $n \in \mathbb{N}$. Then u'_n is never invertible because u_{i_n} is not but $u_{i_n - 1}, \ldots, u_{i_{n-1}+1}$ are (by definition), yet we still have by induction that $v'_n = v'_{n+1} u'_n$. Since M is left-noetherian, I must be finite hence there is some $n \in \mathbb{N}$ such that for all $i \geq n$, m_i is invertible. □

Lemma 4.7. *If M is right-noetherian then all its right- and left-invertibles are invertible.*

Proof. Assume M right-noetherian and consider some x that is right-invertible with x_r its right-inverse. For $n \in \mathbb{N}$, set $u_{2n} = x_r$, $v_{2n} = \epsilon$ and $u_{2n+1} = v_{2n+1} = x$. Then $v_n = v_{n+1} u_n$ for all $n \in \mathbb{N}$ hence by right-noetherianity x is invertible. If x is left-invertible instead, its left-inverse is right-invertible hence invertible and therefore so is x. □

Proposition 4.12. *The following are equivalent:*

(1) $\mathbf{Kl}(\mathcal{T}_M)$ has all countable powers of 1;

(2) there are two functions $\mathrm{lgcd} : (M+1)_*^{\mathbb{N}} \to M$ *and* $\mathrm{red} : (M+1)_*^{\mathbb{N}} \to (M+1)_*^{\mathbb{N}}$ *such that*

a) for all $\Lambda \in (M+1)_*^{\mathbb{N}}$, $\Lambda = \mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda)$*;*

b) for all $\Gamma, \Lambda \in (M+1)_*^{\mathbb{N}}$ *and* $\upsilon, \nu \in M$*, if* $\upsilon\,\mathrm{red}(\Gamma) = \nu\,\mathrm{red}(\Lambda)$ *then* $\upsilon = \nu$ *and* $\mathrm{red}\,\Gamma = \mathrm{red}\,\Lambda$*;*

(3) $\mathbf{Kl}(\mathcal{T}_M)$ *has all countable products.*

Moreover when these hold, since any countable set I *embeds into* $\mathbb{N}$, lgcd *and* red *can be extended to* $(M+1)_*^I$. $\mathrm{lgcd}\,\Lambda$ *is then a left-gcd of* $(\Lambda(i))_{i|\Lambda(i)\neq\perp}$ *and the product of* $(X_i)_{i\in I}$ *for some* $I \subseteq \mathbb{N}$ *is the set of pairs* $(\Lambda, (x_i)_{i\in I})$ *such that* $\Lambda \in \mathrm{red}((M+1)_*^I)$ *and, for all* $i \in I$, $x_i \in X_i$ *if* $\Lambda(i) \neq \perp$ *and* $x_i = \perp$ *otherwise. In particular, the* I*-th power of* 1 *is the set of* irreducible *partial functions* $I \to M+1$*:*

$$\prod_I 1 = \mathrm{Irr}(I, M) = \{\mathrm{red}\,\Lambda \in (M+1)_*^I \mid \Lambda \in (M+1)_*^I\}$$

Proof. (3) ⇒ (1) holds by definition.

Let us now show (1) ⇒ (2). If $\Pi_{\mathbb{N}}1$ exists, then for any $\Lambda \in (M+1)_*^{\mathbb{N}}$ the cone $(\Lambda(n) : 1 \to M+1)_{n\in\mathbb{N}}$ factors through some $h : 1 \to \Pi_{\mathbb{N}}$ given by some $h(*) = (\delta, x_\Lambda)$, as if $h(*) = \perp$ then $\Lambda = \perp^{\mathbb{N}}$. We thus set $\mathrm{lgcd}\,\Lambda = \delta$ and $\mathrm{red}(\Lambda)(n) = \pi_n(x_\Lambda)$ for all $n \in \mathbb{N}$, so that in particular $\mathrm{red}(\Lambda) \neq \perp^{\mathbb{N}}$. Since $\Lambda(n) = \pi_n \circ h$ for all $n \in \mathbb{N}$ we get that $\Lambda = \mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda)$, and if $\Xi = \upsilon\,\mathrm{red}(\Gamma) = \nu\,\mathrm{red}(\Lambda)$, then $(* \mapsto \Xi(n))_{n\in\mathbb{N}}$ factors through both $g(*) = (\upsilon, x_\Gamma)$ and $h(*) = (\nu, x_\Lambda)$: $g = h$ hence $u = v$ and $\mathrm{red}\,\Gamma = \mathrm{red}\,\Lambda$. lgcd and red thus satisfy conditions (2)a and (2)b.

In particular when these conditions are satisfied $\mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda) = \Lambda$ hence $\mathrm{lgcd}\,\Lambda$ left-divides $(\Lambda(i))_{i|\Lambda(i)\neq\perp}$ and if δ left-divides this family then $\Lambda = \delta\Gamma$ for some $\Gamma \neq \perp^{\mathbb{N}}$ hence $\Lambda = (\delta\,\mathrm{lgcd}(\Gamma))\,\mathrm{red}(\Gamma)$ and thus δ left-divides $\mathrm{lgcd}(\Lambda) = \delta\,\mathrm{lgcd}(\Gamma)$.

Finally, let us show (2) ⇒ (3) along with the formula for the product of a countable number of objects. Given $(X_i)_{i\in I}$ indexed by some $I \subseteq \mathbb{N}$, define $\prod_I X_i$ as in the statement of the proposition and the projection $\pi_j : \prod_I X_i \multimap X_j$ by $\pi_j(\Lambda, (x_i)_{i\in I}) = (\Lambda(j), x_j)$ if $\Lambda(j) \neq \perp$ and $\pi_j(\Lambda, (x_i)_{i\in I}) = \perp$ otherwise. Given a cone $(f_i : 1 \multimap X_i)$ and some $i \in I$, write $f_i(*) = (\Lambda(i), x_i)$ when $f_i \neq \perp$ and $\Lambda(i) = x_i = \perp$ otherwise. Define now $h : 1 \multimap \prod_I X_i$ by $h(*) = (\mathrm{lgcd}\,\Lambda, (\mathrm{red}\,\Lambda, (x_i)_{i\in I}))$ if $\Lambda \neq \perp^{\mathbb{N}}$ and $h(*) = \perp$ otherwise. We immediately have that $f_i = \pi_i \circ h$ by condition (2)a, and that h is the only such function by condition (2)b.

□

Lemma 4.13. *Conditions (2)a and (2)b are equivalent to saying that*

(4) a) $\langle \mathrm{lgcd}, \mathrm{red}\rangle$ *is injective;*

b) for all $\Lambda \in (M+1)_*^{\mathbb{N}}$, $\mathrm{lgcd}(\mathrm{red}\,\Lambda) = \epsilon$ *and* $\mathrm{red}(\mathrm{red}\,\Lambda) = \mathrm{red}\,\Lambda$*;*

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

Proof. Assuming conditions (2)a and (2)b, we get that

- (4)a: if $\operatorname{lgcd}\Gamma = \operatorname{lgcd}\Lambda$ and $\operatorname{red}\Gamma = \operatorname{red}\Lambda$ then

$$\begin{aligned}\Gamma &= \operatorname{lgcd}(\Gamma)\operatorname{red}(\Gamma)\\ &= \operatorname{lgcd}(\Lambda)\operatorname{red}(\Lambda)\\ &= \Lambda\end{aligned}$$

- (4)b: $\operatorname{red}\Lambda = \operatorname{lgcd}(\operatorname{red}\Lambda)\operatorname{red}(\operatorname{red}\Lambda)$ hence $\operatorname{lgcd}(\operatorname{red}\Lambda) = \epsilon$ and $\operatorname{red}(\operatorname{red}\Lambda) = \operatorname{red}\Lambda$;
- (4)c: $(\upsilon \operatorname{lgcd}(\Lambda))\operatorname{red}(\Lambda) = \upsilon\Lambda$ hence $\upsilon \operatorname{lgcd}(\Lambda) = \operatorname{lgcd}(\upsilon\Lambda)$ and $\operatorname{red}(u\Lambda) = \operatorname{red}\Lambda$.

And conversely, assuming conditions (4)a, (4)b and (4)c, we get that

- (2)a: by injectivity, $\Lambda = \operatorname{lgcd}(\Lambda)\operatorname{red}(\Lambda)$ since

$$\begin{aligned}\operatorname{lgcd}(\operatorname{lgcd}(\Lambda)\operatorname{red}(\Lambda)) &= \operatorname{lgcd}(\Lambda)\operatorname{lgcd}(\operatorname{red}\Lambda)\\ &= \operatorname{lgcd}\Lambda\end{aligned}\qquad\begin{aligned}\operatorname{red}(\operatorname{lgcd}(\Lambda)\operatorname{red}(\Lambda)) &= \operatorname{red}(\operatorname{red}\Lambda)\\ &= \operatorname{red}\Lambda\end{aligned}$$

- (2)b: if $\upsilon \operatorname{red}(\Gamma) = \nu \operatorname{red}(\Lambda)$ then

$$\begin{aligned}\upsilon &= \upsilon \operatorname{lgcd}(\operatorname{red}\Gamma)\\ &= \operatorname{lgcd}(\upsilon \operatorname{red}(\Gamma))\\ &= \operatorname{lgcd}(\nu \operatorname{red}(\Lambda))\\ &= \nu \operatorname{lgcd}(\operatorname{red}\Lambda)\\ &= \nu\end{aligned}\qquad\begin{aligned}\operatorname{red}\Gamma &= \operatorname{red}(\operatorname{red}\Gamma)\\ &= \operatorname{red}(\upsilon \operatorname{red}(\Gamma))\\ &= \operatorname{red}(\nu \operatorname{red}(\Lambda))\\ &= \operatorname{red}(\operatorname{red}\Lambda)\\ &= \operatorname{red}\Lambda\end{aligned}$$

□

Lemma 4.15. *If right- and left-invertibles of M are all invertibles and if $\mathbf{Kl}(\mathcal{T}_M)$ has all countable powers of 1 then M is both left-cancellative up to invertibles on the left and right-coprime-cancellative, and all non-empty countable families of M have a unique left-gcd up to invertibles on the right.*

Proof. Let us first show left-cancellativity up to invertibles. If $\upsilon\gamma_i = \upsilon\lambda_i$ for some $\upsilon \in M$ and two countable families $(\gamma_i)_{i\in I}, (\lambda_i)_{i\in I}$ of elements of M with $I \subseteq \mathbb{N}_{>0}$, then defining $\Gamma(0) = \Lambda(0) = \epsilon$, $\Gamma(i) = \gamma_i$ and $\Lambda(i) = \lambda_i$ for all $i \in I$ and $\Gamma(j) = \Lambda(j) = \bot$ for all $j \notin I$, we have that $\upsilon\Gamma = \upsilon\Lambda$. Hence $\operatorname{red}\Gamma = \operatorname{red}\Lambda$ and $\upsilon \operatorname{lgcd}(\Gamma) = \upsilon \operatorname{lgcd}(\Lambda)$. But $\operatorname{lgcd}\Gamma$ and $\operatorname{lgcd}\Lambda$ left-divide $\epsilon = \Gamma(0) = \Lambda(0)$ hence they are right-invertible and thus invertible: γ_i and $\lambda_i = \operatorname{lgcd}(\Lambda)\operatorname{lgcd}(\Gamma)^{-1}\gamma_i$ are equal up to an invertible (that does not depend on i) on the left.

Moreover, if $\upsilon\Lambda = \nu\Lambda$ for some Λ such that $(\Lambda(i))_{i|\Lambda(i)\neq\bot}$ is left-coprime, then $\operatorname{lgcd}\Lambda$ is right-invertible (by definition of left-coprimality) and thus since

$$\begin{aligned}\upsilon \operatorname{lgcd}(\Lambda) &= \operatorname{lgcd}(\upsilon\Lambda)\\ &= \operatorname{lgcd}(\nu\Lambda)\\ &= \nu \operatorname{lgcd}(\Lambda)\end{aligned}$$

by left-invertibility of $\operatorname{lgcd}\Lambda$, $\upsilon = \nu$.

Finally, consider δ a left-gcd of some non-empty countable family encoded as $\Lambda \in (M+1)^{\mathbb{N}}_*$. Then there is some υ such that $\delta = \operatorname{lgcd}(\Lambda)\upsilon$ (since $\operatorname{lgcd}(\Lambda)$ left-divides Λ) and some Γ such that $\delta\Gamma = \Lambda$. Hence

$$\begin{aligned}\operatorname{lgcd}\Lambda &= \operatorname{lgcd}(\delta\Gamma)\\ &= \delta\operatorname{lgcd}(\Gamma)\\ &= \operatorname{lgcd}(\Lambda)\upsilon\operatorname{lgcd}(\Gamma)\end{aligned}$$

By left-cancellativity up to invertibles, $\upsilon\operatorname{lgcd}(\Gamma) \in M^\times$ so υ is right-invertible hence invertible : $\delta = \operatorname{lgcd}\Lambda$ up to invertibles on the right. □

Lemma 4.16. *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.*

Proof. Split the set of those $\Lambda \in (M+1)^{\mathbb{N}}_*$ such that $(\Lambda(i))_{i|\Lambda(i)\neq\bot}$ is left-coprime into the equivalence classes given by $\chi\Lambda \sim \Lambda$ for all $\chi \in M^\times$. Then, for each equivalence class C pick a $\operatorname{red} C$ in C (using the axiom of choice) and for all $\upsilon \in M$ define $\operatorname{lgcd}(\upsilon\operatorname{red}(C)) = \upsilon$ and $\operatorname{red}(\upsilon\operatorname{red}(C)) = \operatorname{red} C$ so that in particular $\operatorname{lgcd}(\operatorname{red} C) = \epsilon$ and $\operatorname{red}(\operatorname{red} C) = \operatorname{red} C$.

This is well-defined because if $\upsilon\operatorname{red}(C) = \nu\operatorname{red}(D)$ for $\upsilon, \nu \in M$ and two equivalence classes C, D, then if δ is a left-gcd of $\upsilon\operatorname{red}(C)$ we have that $\delta = \upsilon\upsilon'$ for some υ' (since υ left-divides $\upsilon\operatorname{red}(C)$) and there is some Λ such that $\upsilon\operatorname{red}(C) = \upsilon\upsilon'\Lambda$. But then by left-cancellativity up to invertibles, there is some $\chi \in M^\times$ such that $\upsilon'\Lambda = \chi\operatorname{red}(C)$, hence $\chi^{-1}\upsilon'$ left-divides $\operatorname{red} C$ and as such is right-invertible (it left-divides ϵ, a left-gcd of $\operatorname{red} C$), making υ' right-invertible as well. This shows that υ is a left-gcd of $\upsilon\operatorname{red}(C) = \nu\operatorname{red}(D)$ and we show similarly that this is also true of ν, hence by unicity of the left-gcd there is a $\xi \in M^\times$ such that $\upsilon = \nu\xi$. Therefore by left-cancellativity up to invertibles there is another invertible $\xi' \in M^\times$ such that $\xi\operatorname{red}(C) = \xi'\operatorname{red}(D)$ hence by definition $C = D$ and, by right-coprime-cancellativity, $\upsilon = \nu$.

Moreover this defines $\operatorname{lgcd}\Lambda$ and $\operatorname{red}\Lambda$ for any Λ because if δ is a left-gcd of Λ, then $\Lambda = \delta\Gamma$ for some left-coprime Γ (if δ' left-divides Γ then $\delta\delta'$ left-divides Λ hence δ, therefore δ' is invertible by left-cancellativity up to invertibles) hence $\Lambda = \delta\chi\operatorname{red}(C)$ if $\Gamma \in C$ and $\Gamma = \chi\operatorname{red}(C)$.

We have thus defined two functions lgcd and red that immediately satisfy the conditions (2)a and (2)b of Proposition 4.12. □

Lemma 4.18. *In $\mathbf{Kl}(\mathcal{T}_M)$, $\mathrm{Iso} = \mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot}$, and these four classes are all closed under composition (within themselves).*

Proof. Closure under composition is immediate.

If $f : X \to M \times Y + 1$ is in $\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot}$, it can be restricted to a function $\langle f_1, f_2\rangle : X \to M \times Y$ ($f \in \mathrm{Tot}$). f_2 must be bijective ($f \in \mathrm{Surj} \cap \mathrm{Inj}$) and $f_1(X) \subseteq M^\times$ ($f \in \mathrm{Inv}$). Hence f has an inverse, given by $(f^{-1})(y) = \Big((f_1({f_2}^{-1}(y)))^{-1}, {f_2}^{-1}(y)\Big)$.

Conversely, if $f : X \to M \times Y + 1$ is an isomorphism it has an inverse $f^{-1} : Y \to M \times X + 1$ such that $f \circ f^{-1} = \mathrm{id}_Y$ and $f^{-1} \circ f = \mathrm{id}_X$. For all $x \in X$, $(f^{-1})^\dagger(f(x)) = (\epsilon, x)$ hence $f(x) \neq \bot$: $f \in \mathrm{Tot}$, and similarly for f^{-1}. Writing $f = \langle f_1, f_2 \rangle$ and $f^{-1} = \langle (f^{-1})_1, (f^{-1})_2 \rangle$, we have that f_2 is a bijection with inverse ${f_2}^{-1} = (f^{-1})_2$, hence $f \in \mathrm{Surj} \cap \mathrm{Inj}$. Finally, for all $x \in X$ we have that $f_1(x)(f^{-1})_1(f_2(x)) = \epsilon$ and conversely, hence $f_1(x)$ is invertible and $f \in \mathrm{Inv}$. □

Proposition 4.19. $(\mathcal{E}_1, \mathcal{M}_1) = (\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv}, \mathrm{Tot})$, $(\mathcal{E}_2, \mathcal{M}_2) = (\mathrm{Surj} \cap \mathrm{Inj}, \mathrm{Inv} \cap \mathrm{Tot})$ *and* $(\mathcal{E}_3, M_3) = (\mathrm{Surj}, \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot})$ *are all factorization systems in* $\mathbf{Kl}(\mathcal{T}_M)$.

Proof. By Lemma 4.18 we only need to show for each $i \in \{1, 2, 3\}$ that every $f : X \multimap Y$ may be factored as $m \circ e$ with $e \in \mathcal{E}_i$ and $m \in \mathcal{M}_i$, and that $\mathcal{E}_i$ and $\mathcal{M}_i$ satisfy the diagonal fill-in property of Definition 2.6.

- $(\mathcal{E}_1, \mathcal{M}_1)$. Any $f : X \to M \times Y + 1$ may be factored through $e : X \to M \times f^{-1}(M \times Y) + 1$ (given by $e(x) = (\epsilon, x)$ if $e(x) \neq \bot$ and $e(x) = \bot$ otherwise) and $m : f^{-1}(M \times Y) \to M \times Y + 1$ (given by $m(x) = e(x)$ when $e(x) \neq \bot$).

  Moreover, given a commuting diagram

  $$\begin{array}{ccc} X & \overset{e}{\multimap\!\!\!\!\to} & Y_1 \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle v} \\ Y_2 & \overset{m}{\rightarrowtail} & Z \end{array}$$

  the only possible choice for a $\phi : Y_1 \to M \times Y_2 + 1$ is given by $\phi(y_1) = \bot$ if $v(y_1) = \bot$ and $\phi(y_1) = ({\upsilon_e}^{-1}\upsilon_u, y_2)$ if $e(x) = (\upsilon_e, y_1)$, $u(x) = (\upsilon_u, y_2)$, $v(y_1) = (\upsilon_v, z)$, $m(y_2) = (\upsilon_m, z)$ and $\upsilon_e \upsilon_v = \upsilon_u \upsilon_m$. This definition does not depend on the choice of x because of the bijectiveness property of e, and we immediately have the commuting diagram

  $$\begin{array}{ccc} X & \overset{e}{\multimap\!\!\!\!\to} & Y_1 \\ {\scriptstyle u}\downarrow & \overset{\phi}{\swarrow} & \downarrow{\scriptstyle v} \\ Y_2 & \overset{m}{\rightarrowtail} & Z \end{array}$$

  by definition.

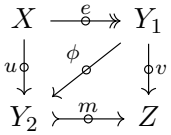
- $(\mathcal{E}_2, \mathcal{M}_2)$. Any $f : X \to M \times Y + 1$ may be factored through $e : X \to M \times f^{-1}(M \times Y) + 1$ (given by $e(x) = (\upsilon, x)$ if $e(x) = (\upsilon, -)$ and $e(x) = \bot$ otherwise) and $m : f^{-1}(M \times Y) \to M \times Y + 1$ (given by $m(x) = (\epsilon, y)$ when $e(x) = (-, y)$).

  Finally, given a commuting diagram

  $$\begin{array}{ccc} X & \overset{e}{\multimap\!\!\!\!\to} & Y_1 \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle v} \\ Y_2 & \overset{m}{\rightarrowtail} & Z \end{array}$$

  the only possible choice for a $\phi : Y_1 \to M \times Y_2 + 1$ is given by $\phi(y_1) = \bot$ if $v(y_1) = \bot$ and $\phi(y_1) = (\upsilon_v {\upsilon_m}^{-1}, y_2)$ if $e(x) = (\upsilon_e, y_1)$, $u(x) = (\upsilon_u, y_2)$, $v(y_1) =$

(υ_v, z), $m(y_2) = (\upsilon_m, z)$ and $\upsilon_e \upsilon_v = \upsilon_u \upsilon_m$. This definition does not depend on the choice of x because of the bijectiveness property of e, and we immediately have the commuting diagram



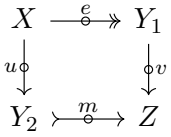
by definition.

- $(\mathcal{E}_3, \mathcal{M}_3)$. Any morphism $f : X \to M \times Y + 1$ factors through $e : X \to M \times Z + 1$ and $m : Z \to M \times Y + 1$ with

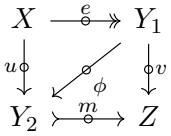
$$Z = \{y \in Y \mid \exists x \in X, f(x) = (-, y)\}$$

$e(x) = f(x)$ and $m(y) = (\epsilon, y)$.

Finally, given a commuting diagram



the only possible choice for a $\phi : Y_1 \to M \times Y_2 + 1$ is given by $\phi(y_1) = \bot$ if $v(y_1) = \bot$ and $\phi(y_1) = (\upsilon_v {\upsilon_m}^{-1}, y_2)$ if $e(x) = (\upsilon_e, y_1)$, $u(x) = (\upsilon_u, y_2)$, $v(y_1) = (\upsilon_v, z)$, $m(y_2) = (\upsilon_m, z)$ and $\upsilon_e \upsilon_v = \upsilon_u \upsilon_m$. This definition does not depend on the choice of x because m is injective on Y, and we immediately have the commuting diagram



by definition.

□

D. Proofs for Section 5 (Algorithms on monoidal transducers)

Lemma 5.1. *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* $\text{length}_{\mathcal{M}}(m : Y \rightarrowtail X) = |X| - |Y|$.

Proof. Let $(x_i : 1 \rightarrowtail X)_{i \in \mathbb{N}}$ be an infinite sequence of distincts elements of an infinite set X. Then the $m_n = [\kappa_i]_{i=0}^n : \coprod_{i=0}^n 1 \rightarrowtail \coprod_{i=0}^{n+1} 1$ provide a counter-example to the $\mathcal{M}$-noetherianity of X as none of them are isomorphisms (they are not surjective) yet $[x_i]_{i=0}^{n+1} \circ m_n = [x_i]_{i=0}^n$ and $[x_i]_{i=0}^n$ is always an $\mathcal{M}$-morphism. Hence infinite sets are never $\mathcal{M}$-noetherian. If X and the sequence $(x_i)_{i \in \mathbb{N}}$ were finite instead, this example would prove that $\text{length}_{\mathcal{M}}(m : Y \rightarrowtail X) \geq |X| - |Y|$.

Conversely, a strict chain of $\mathcal{M}$-subobjects of X is a strict chain of subsets $X_0 \subsetneq X_1 \subsetneq \cdots$ of X. In particular, the cardinality of these subsets is strictly increasing: if X is finite, the chain must be finite as well, and its length at most $|X| - |X_0|$. □

Lemma 5.2. *An object X of $\mathbf{Kl}(\mathcal{T}_M)$ 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* $\operatorname{colength}_{\mathcal{E}}(e : X \multimap Y) = |X| - |Y| + \operatorname{rk} e$ *where* $\operatorname{rk} e = \sum_{e(x)=(v,y)} \operatorname{rk} v$.

Proof. Let $(x_i)_{i\in\mathbb{N}}$ be an infinite sequence of distincts elements of an infinite set X, and set $X_n = \{x_i \mid 0 \le i \le n\}$. Let $e_n : X \multimap X_n$ be defined by $e(x_i) = (\epsilon, x_i)$ for $i \le n$ and $e(x) = \bot$ otherwise, and let $e_n^{n+1} : X_{n+1} \multimap X_n$ be the restriction of e_n to X_{n+1}. None of the e_n^{n+1} are isomorphisms (they are not total) yet $e_n^{n+1} \circ e_{n+1} = e_n$: infinite sets are never $\mathcal{E}$-artinian.

Assume now X is not empty and M is not right-noetherian: there is an element $x_* \in X$ and two sequences $(v_n)_{n\in\mathbb{N}}$ and $(\nu_n)_{n\in\mathbb{N}}$ of elements of M such that for all $n \in \mathbb{N}$, $v_n \notin M^\times$ and $\nu_n = \nu_{n+1} v_n$. Let $e_n : X \multimap 1$ be defined by $e_n(x_*) = (\nu_n, *)$ and $e_n(x) = \bot$ for all other $x \in X$, and let $e_n^{n+1} : 1 \multimap 1$ be defined by $e_n^{n+1}(*) = (v_n, *)$. None of the e_n^{n+1} are isomorphisms (they do not only produce invertible elements of M) yet $e_n^{n+1} \circ e_{n+1} = e_n$: non-empty sets are never $\mathcal{E}$-artinian when M is not right-noetherian.

Conversely, if X is empty then it is immediately $\mathcal{E}$-artinian: there is only one $\mathcal{E}$-morphism out of X, id_X. Suppose now that X is finite and M right-noetherian, and consider a cochain $e_n^{n+1} : X_{n+1} \multimap X_n$ of $\mathcal{E}$-quotients $e_n : X \multimap X_n$. Since X is finite, at most $|X| - |X_0|$ of the $\mathcal{E}$-quotients $X_{n+1} \multimap X_n$ witness a decrease of the cardinality from their domain to their codomain and are not in $\mathrm{Inj} \cap \mathrm{Tot}$. Fix now an $x \in X$ such that $e_0(x) \neq \bot$ and write $e_n(x) = (\nu_n, x_n)$ and $e_n^{n+1}(x_{n+1}) = (v_n, x_n)$ (this is well-defined because $e_n^{n+1} \circ e_{n+1} = e_n$). Then $\nu_n = \nu_{n+1} v_n$ for all $n \in \mathbb{N}$, hence since M is right-noetherian only a finite number of the e_n^{n+1}, at most $\operatorname{rk} \nu_0$, produce a non-invertible element on x_{n+1}. This is true for all $x \in X$, hence a finite number of the e_n^{n+1}, at most $\operatorname{rk} e_0$, are not in Inv, and a finite number of them, at most $|X| - |X_0| + \operatorname{rk} e_0$, are not in Iso: X is $\mathcal{E}$-artinian and $\operatorname{colength}_{\mathcal{E}}(e_0 : X \multimap X_0) \le |X| - |X_0| + \operatorname{rk} e_0$.

Finally, if $\operatorname{rk} e$ is finite, in light of this proof it is now easy to build a strict cochain of $\mathcal{E}$-quotients of X starting with $e : X \multimap Y$ that has length exactly $|X| - |Y| + \operatorname{rk} e$. For each $\nu(x) \in M$ such that $e(x) = (\nu(x), y)$ for some $x \in X$ and $y \in Y$, write indeed $v_1(x), \ldots, v_{\operatorname{rk} \nu(x)}(x)$ for a sequence of non-invertible divisors of ν of maximum length. Each morphism between two consecutive $\mathcal{E}$-quotients in the cochain should either decrease the size of the quotient by 1, or produce exactly one of the $v_i(x)$ on x for exactly one $x \in X$. Similarly, if $\operatorname{rk} e$ is infinite there sequences of divisors of some $\nu(x)$ or arbitrary length and it is then easy to build strict cochains of $\mathcal{E}$-quotients of X of arbitrary lengths. □

Theorem 5.3. *Algorithm 4 is correct and terminates as soon as* $\operatorname{Min} \mathcal{L}$ *has finite state-set and M is right-noetherian. It makes at most* $3\,|\operatorname{Min} \mathcal{L}|_{\mathtt{st}} + \operatorname{rk}(\operatorname{Min} \mathcal{L})$ *updates to Q (lines 8 and 14) and at most* $\operatorname{rk}(\operatorname{Min} \mathcal{L}) + |\operatorname{Min} \mathcal{L}|_{\mathtt{st}}$ *updates to T (line 10).*

Proof. Notice first that Algorithm 4 is indeed the instance of Algorithm 1 in $\mathbf{Kl}(\mathcal{T}_M)$: for all $q \in Q$ and $a \in A \cup \{\epsilon\}$, $\Lambda(q,a)$ is a left-gcd of $L(q,a,\cdot) = (\mathcal{L}(\triangleright qat \triangleleft))_{t\in T} = \Lambda(q,a)R(q,a,\cdot)$, hence $R(q,a,\cdot)$ is left-coprime and there is a $\chi \in M^\times$ such that $\Lambda(q,a)\chi^{-1} = \mathrm{lgcd}(L(q,a,\cdot))$ and $\chi R(q,a,\cdot) = \mathrm{red}(L(q,a,\cdot))$. Hence $Q/T = \{\mathrm{red}(L(q,e,\cdot)) \mid q \in Q, L(q,e,\cdot) \neq \bot^T\}$ is the quotient of the set $\{R(q,e,\cdot) \mid q \in Q, R(q,e,\cdot) \neq \bot^T\}$ by equality up to invertibles on the left. It follows that $Q/T \rightarrowtail (Q \cup \{qa\})/T$ is an $\mathcal{E}$-morphism if and only if it is surjective, that is if and only if the condition on line 7 is not satisfied, and $Q/(T \cup \{at\}) \twoheadrightarrow Q/T$ is an $\mathcal{M}$-morphism if and only if it is total, produces only invertibles elements and is injective, that is if and only if it respectively does not satisfy any of the three conditions on line 9: it is easy to see that it is total if and only if the first condition is not satisfied; it is total but does not only produce invertible elements if and only if there is some $q \in Q$ and $at \in AT$ such that $L(q,a,t)$ is not left-divisible by $\Lambda(q,e)$ which is equivalent to the second condition holding; and finally it is easy to see that it is total and only produces invertible elements but is not injective if and only if the third condition holds.

The correction and termination is then given by Theorem 2.16, thanks to Lemmas 5.1 and 5.2. These two lemmas also provide the complexity bound of the algorithm, as Theorem 2.16 is proven in [10] by showing that each addition to T contributes to a morphism in a strict chain of $\mathcal{M}$-subobjects of $(\mathrm{Min}\,\mathcal{L})(\mathtt{st})$ starting with $\{\epsilon\}/A^* \rightarrowtail A^*/A^* = (\mathrm{Min}\,\mathcal{L})(\mathtt{st})$ [10, Lemma 33], and each addition to Q contributes to a morphism in a chain of $\mathcal{E}$-quotients of $(\mathrm{Min}\,\mathcal{L})(\mathtt{st})$ ending with $(\mathrm{Min}\,\mathcal{L})(\mathtt{st}) = A^*/A^* \twoheadrightarrow A^*/\{\epsilon\}$ [10, Lemma 33] and whose isomorphisms may only be contributed by the addition of a counter-example outputted by $\textsc{Equiv}_{\mathcal{L}}$ and are immediately followed by a non-isomorphism in the chain for T or the cochain for Q [10, Lemma 36]. □