Applications of a categorical framework for minimization and active learning of transition systems

Quentin Aristote, ENS Paris, PSL University
supervised by Daniela Petrişan, IRIF, Université Paris-Cité

August 20th, 2022

1 Introduction

Context. In a 1987 foundational work, Angluin showed how the minimal automaton recognizing a regular language $\mathcal{L}$ could be computed in polynomial time using the L* learning algorithm [1]. This algorithm learns the automaton by querying two oracles: a *membership* oracle, which decides whether a given word belongs to $\mathcal{L}$, and an *equivalence* oracle, which decides whether the language recognized by a hypothesis automaton is $\mathcal{L}$, and if not returns a counter-example word on which the automaton and the language disagree.

The L* algorithm has proven widely useful in many areas of computer science, and has been extended to many other families of automata, for instance weighted automata (automata whose state-spaces are vector spaces and transitions are linear maps) [2] or transducers (automata whose transitions can also produce output words) [3].

In 2021, Colcombet and Petrişan thus introduced a categorical framework for automata minimization and learning which neatly encapsulates these previous examples in a unifying way [4, 5]. It is not the first such framework [e.g. 6, 7] , but it is arguably simpler and less restrictive then the previous ones.

The research problem. One limit of this categorical framework is that it does not give rise to new results, as all the examples come from already existing generalizations of the L* algorithm. A natural question is then whether it can be instanciated to do so, that is whether it applies to families of transition systems to which the L* algorithm has not been generalized yet, and whether the framework makes this easy. In particular, a positive answer would likely encourage the use of category theory in the automata theory community.

Our first such instanciation of this framework is for transducers whose outputs may now belong to any arbitrary monoid, the original object of study of this internship. Their minimization has been studied before [8], but the algorithm is quite restrictive and nothing is said of learning. The second instanciation is for weighted automata with weights in arbitrary rings and not just fields, which we originally studied in the hope of encompassing weighted transducers. There is a learning algorithm for principal domains [9], but it does not produce a minimal automaton, and no notion of minimality has been defined that gives a unique (up to isomorphism) minimal automaton. The third and final instan-

ciation is for automata whose transition graphs are quasi-ordered: they can of course be minimized and learnt using standard automata algorithms, but there is no guarantee this preserves the quasi-order on the graph. We studied this third instance because we hoped it would relate automata learning and the Valk-Jantzen-Goubault-Larrecq lemma, which is used to learn upwards-closed sets appearing in the theory of well-structured transition systems [10].

Contributions. In this work we successfully apply the categorical framework of Colcombet and Petrişan to the three instances described above. To this means we also extend the framework itself, providing new categorical algorithms for minimization and improper learning.

For transducers with output in arbitrary monoids, we give necessary and sufficient conditions on the output monoid for the framework to apply, and describe the minimality notion and the learning algorithm that ensue. We also describe a minimization algorithm which happens to have categorical roots but whose instanciation is non-trivial, and we thus implement it in `OCaml` as a proof-of-concept.

For weighted automata, we exhibit a whole family of minimality notions that are equivalent when the weights are over a field but not in general. We show that the corresponding minimal automata are well-behaved when the weights are in Dedekind domains (a generalization of principal domains), and that they can be learnt with L*-like algorithms when they recognize a rational language.

For quasi-ordered automata, we show that the minimal automaton is equipped with a topological ordering and that it can be learnt by only considering quasi-ordered automata. We then show how the equivalence oracle in the learning algorithm can be implemented using a more powerful membership oracle, effectively relating automata learning and the Valk-Jantzen-Goubault-Larrecq lemma.

Advantages of our method. Our proofs are particularly interesting because they are clean: the minimality notions and the correctness of the algorithms are already handled by the categorical framework, so we focus on the intricate part which is showing that the specific categories corresponding to the different families of automata have a certain structure. In particular, this makes it easier to find the weakest possible conditions under which our arguments hold.

Future work. As these examples were all quite theoretical, a first question is whether they actually apply to practical problems. For instance, we have hope that transducers with output in arbitrary monoids may be used in natural language processing or concurrency theory.

On the theoretical side, some of our arguments worked specifically because the category used in the corresponding example had a special well-behaved structure. It would thus be interesting to understand which categorical properties actually give rise to these structures and to determine the most general context in which these arguments work. For example, to reason on quasi-ordered automata, we generalized Colcombet and Petrişan's framework to categories enriched with partial orders (where the morphisms are ordered and the functors are monotone), but we believe it can actually be generalized to arbitrary enrichment categories, and that this would for example also instanciate the minimization and learning of nominal automata [11].

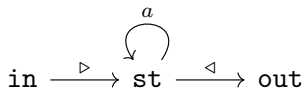
Contents

2 Preliminaries

In this section we recall (and extend) the definitions and results of Colcombet and Petrişan [4, 5], that we will instanciate in three different contexts. We assume basic knowledge of category theory [12], but we also focus on the example of deterministic complete automata and mention the examples of transducers and weighted automata that will be detailed and generalized in Sections 3 and 4 respectively.

2.1 Automata and languages as functors

Fix an input alphabet A. In this framework, 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 A. It 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. 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 $\mathcal{A}(\mathtt{out}) = Y$. For example, 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 $\mathcal{A}(a) : S \to S$, 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, if $\mathbb{K}$ is a field, we may see $\mathbb{K}$-weighted automata as functors from $\mathcal{I}$ to the category of $\mathbb{K}$-vector-spaces $\mathbb{K}\mathbf{Vec}$, and if B is an output alphabet, we may see deterministic transducers as functors from $\mathcal{I}$ to the Kleisli category $\mathbf{Kl}(\mathcal{T}_{B^*})$ of free algebras for the monad $\mathcal{T}_{B^*}X = B^* \times X + 1$.



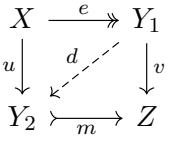
In the same way, we define a *language* to be a functor $\mathcal{L} : \mathcal{O} \to \mathcal{C}$, where $\mathcal{O}$ is the full subcategory of $\mathcal{I}$ on $\mathtt{in}$ and $\mathtt{out}$. In other words, a language is the data of two objects $X = \mathcal{L}(\mathtt{in})$ and $Y = \mathcal{L}(\mathtt{out})$ in $\mathcal{C}$, 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 $\mathcal{A}$. For example, 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.

Given a category $\mathcal{C}$ and a language $\mathcal{L} : \mathcal{O} \to \mathcal{C}$, we then 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).

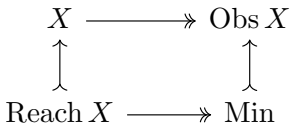
2.2 Factorization systems and the minimal automaton

Consider now an arbitrary category $\mathcal{C}$ (our leading example being $\mathbf{Auto}_{\mathcal{L}}$ for some $\mathcal{L}$). A *factorization system* $(\mathcal{E}, \mathcal{M})$ on $\mathcal{C}$ is the data of two classes of morphisms $\mathcal{E}$ and $\mathcal{M}$ such that **(i)** $\mathcal{E} \cap \mathcal{M}$ is exactly the class of isomorphisms of $\mathcal{C}$ and $\mathcal{E}$ and $\mathcal{M}$ are both stable under composition;



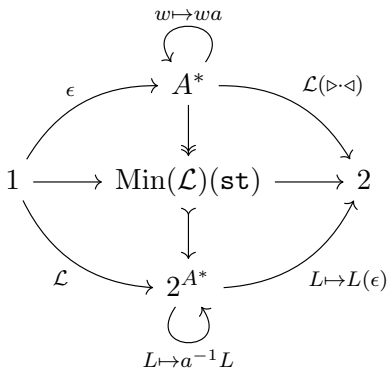
(ii) for every commutative square such that $v \circ e = m \circ u$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$, there is a unique d such that $u = d \circ e$ and $v = m \circ d$; **(iii)** for every morphism $f : X \to Y$ in $\mathcal{C}$, there is a Z in $\mathcal{C}$ (the $(\mathcal{E}, \mathcal{M})$-*factorization* of f) and morphisms $e : X \twoheadrightarrow Z$ in $\mathcal{E}$ and $m : Z \rightarrowtail Y$ (note how we use $\twoheadrightarrow$ for morphisms in $\mathcal{E}$ and $\rightarrowtail$ for arrows in $\mathcal{M}$) in $\mathcal{M}$ such that $f = m \circ e$ (and this choice is unique up to isomorphism by condition (ii)). In **Set** or in $\mathbb{K}$**Vec**, a classic factorization system is given by $\mathcal{E}$-morphisms being the surjective maps and $\mathcal{M}$-morphisms being the injective maps (the factorization of a morphism being its image), and for $\mathcal{L} : \mathcal{O} \to \mathcal{C}$ any factorization system on $\mathcal{C}$ may be lifted to $\mathbf{Auto}_{\mathcal{L}}$ by having the $\mathcal{E}$-morphisms be those morphisms of automata $f : \mathcal{A}(\mathtt{st}) \to \mathcal{A}'(\mathtt{st})$ such that $f \in \mathcal{E}$ (in $\mathcal{C}$), and similarly for $\mathcal{M}$.

When a category $\mathcal{C}$ is equipped with a factorization $(\mathcal{E}, \mathcal{M})$, we may then define its minimal object as soon as it has an initial object I (for every object X of $\mathcal{C}$, there is exactly one morphism $I \to X$) and a final object F (for every object X of $\mathcal{C}$, there is exactly one morphism $X \to F$). Indeed, there is then a unique morphism $I \to F$: the minimal object is the object that $(\mathcal{E}, \mathcal{M})$-factors this morphism. It is minimal in the sense that it $(\mathcal{E}, \mathcal{M})$-*divides* any other object: if we write $\operatorname{Reach} X$ for the $(\mathcal{E}, \mathcal{M})$-factorization of the morphism $I \to X$ and $\operatorname{Obs} X$ for the $(\mathcal{E}, \mathcal{M})$-factorization of $X \to F$, we get that $\operatorname{Min} \cong \operatorname{Reach}(\operatorname{Obs} X) \cong \operatorname{Obs}(\operatorname{Reach} X)$, as well as the (non-necessarily commuting) diagram on the right. In **Set**, the domain of an injection has smaller cardinal than its codomain, and conversely for a surjection, so $(\mathcal{E}, \mathcal{M})$-division implies smaller cardinality.



To be able to define the minimal automaton recognizing a language, the question is then whether this is possible in $\mathbf{Auto}_{\mathcal{L}}$, that is whether this category has an initial and a final object. For $(\mathbf{Set}, 1, 2)$-automata, the initial automaton recognizing a language $\mathcal{L}$ has state-set $\mathcal{A}^{init}(\mathtt{st}) = A^*$, initial state ϵ, accepting states those w that are in the language, and transition functions $\delta_a(w) = wa$. Similarly, the final automaton has state-set $\mathcal{A}^{final}(\mathtt{st}) = 2^{A^*}$,

initial state $\mathcal{L}$ itself, accepting states the languages that contain ϵ, and transition function $\delta_a(L) = a^{-1}L = \{w \mid aw \in L\}$. The unique morphism between these two automata sends a word (a state of $\mathcal{A}^{init}$) to its residual language (a state of $\mathcal{A}^{final}$), and the minimal automaton is thus the factorization of this morphism: its states are the Myhill-Nerode equivalence classes for $\mathcal{L}$. This can be summed up by the commuting diagram below, which, when generalized to arbitrary output categories, leads to Theorem 2.1.



Theorem 2.1. *Given a language $\mathcal{L} : \mathcal{I} \to \mathcal{C}$, if $\mathcal{C}$ has $|A^*|$-copowers of $\mathcal{L}(\mathtt{in})$ and $|A^*|$-powers of $\mathcal{L}(\mathtt{out})$ then $\mathbf{Auto}_{\mathcal{L}}$ has both initial and final objects $\mathcal{A}^{init}(\mathcal{L})$ and $\mathcal{A}^{final}(\mathcal{L})$, and hence a minimal object* $\mathrm{Min}\,\mathcal{L}$.

Similarly, the minimal $(\mathbb{K}\mathbf{Vec}, \mathbb{K}, \mathbb{K})$-automaton is the minimal weighted automaton [13], and the minimal $(\mathbf{Kl}(T_{B^*}), 1, 1)$-automaton is the minimal transducer [14].

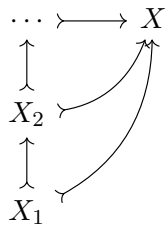
While Petrişan and Colcombet do not give a categorical generalization of the fixpoint algorithm for computing Reach or Obs, their proof methods can be adapted to show that Algorithm 1 is such an algorithm, where $\mathcal{A}$ is a fixed $\mathcal{C}$-automaton and $\mathcal{J}_Q$ is defined to be the $(\mathcal{E}, \mathcal{M})$-factorization of the morphism $\coprod_{q \in Q} \mathcal{A}(\triangleright q) : \coprod_Q \mathcal{L}(\mathtt{in}) \to \mathcal{A}(\mathtt{st})$ for a set $Q \subseteq A^*$: in $\mathbf{Set}$, $\mathcal{J}_Q$ is the subset of states of $\mathcal{A}(\mathtt{st})$ that are reachable by following words in Q. There is a canonical morphism $\mathcal{J}_Q \rightarrowtail \mathcal{J}_{Q \cup QA}$ and the automaton structure on $\mathcal{A}$ can be restricted to $\mathcal{J}_Q$ when this morphism is an isomorphism.

Algorithm 1 A categorical algorithm for computing Reach

Input: a $\mathcal{C}$-automaton $\mathcal{A}$
Output: Reach $\mathcal{A}$

$Q_{todo} = \{\epsilon\}$
$Q_{done} = \varnothing$
while there is a $q \in Q_{todo}$ **do**
  move q from Q_{todo} to Q_{done}
  for $a \in A$ such that $\mathcal{J}_{Q_{done} \sqcup Q_{todo}} \rightarrowtail \mathcal{J}_{Q_{done} \sqcup Q_{todo} \sqcup \{qa\}}$ is not an $\mathcal{E}$-morphism **do**
    add qa to Q_{todo}
  end for
end while
return the automaton with state-space $\mathcal{J}_{Q_{done}}$

Define an object X in $\mathcal{C}$ to be $\mathcal{M}$-noetherian when every strict chain of $\mathcal{M}$-subobjects of X (as on the right) is finite, and call the (possibly infinite) supremum of the lengths of such chains, $\dim_{\mathcal{M}} X$, the *$\mathcal{M}$-dimension* of X. Dually, define X to be $\mathcal{E}$-artinian when every strict cochain of $\mathcal{E}$-quotients of X is finite, and call the supremum of the lengths of such cochains, $\mathrm{codim}_{\mathcal{E}} X$, the *$\mathcal{E}$-codimension* of X. In $\mathbf{Set}$ for example these two quantities are both the cardinal of X.



Proposition 2.2. *Algorithm 1 is correct. It also terminates when $\mathcal{A}(\mathtt{st})$ is $\mathcal{M}$-noetherian, iterat-*

ing the `while` *loop (line 3) at most* $\dim_{\mathcal{M}} \mathcal{A}(\mathtt{st})$ *times.*

The dual algorithm computes Obs and has its complexity bounded by $\operatorname{codim}_{\mathcal{E}} \mathcal{A}$ (see Algorithm 4 in Appendix A.1). These two algorithms instanciate for example the depth-first search computing reachable states of a $(\mathbf{Set}, 1, 2)$-automaton and the partition refinement that computes their Myhill-Nerode equivalence classes.

2.3 Learning

In this section, we fix a language $\mathcal{L} : \mathcal{I} \to \mathcal{C}$ 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 all the countable copowers of $\mathcal{L}(\mathtt{in})$ and all the countable powers of $\mathcal{L}(\mathtt{out})$ so that Theorem 2.1 applies. Our goal is to compute $\operatorname{Min}\mathcal{L}$ with the help of two oracles: $\textsc{Eval}_{\mathcal{L}}$ computes $\mathcal{L}(\triangleright w \triangleleft)$ given a $w \in A^*$, while $\textsc{Equiv}_{\mathcal{L}}$ decides whether a $\mathcal{C}$-automaton $\mathcal{A}$ recognizes $\mathcal{L}$, and, if not, computes 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, this problem is solved using Angluin's L* algorithm [1], which maintains sets Q and T of prefixes and suffixes. Using $\textsc{Eval}_{\mathcal{L}}$, it incrementally builds a table $\mathcal{L}_{Q,T} : Q \times (A \cup \{\epsilon\}) \times T \to 2$ that represents partial knowledge of $\mathcal{L}$ until $\mathcal{L}_{Q,T}$ can be merged into a (minimal) automaton. This automaton is then submitted to $\textsc{Equiv}_{\mathcal{L}}$: if it is accepted it must be $\operatorname{Min}\mathcal{L}$, otherwise the counter-example is added to Q and the algorithm loops over.

The FunL* algorithm (Algorithm 2) generalizes this to arbitrary $\mathbf{Auto}_{\mathcal{L}}$. The partial table $\mathcal{L}_{Q,T}$ (the restriction of $\mathcal{L}$ to words in $Q(A \cup \{\epsilon\})T$) is represented as a (Q,T)-biautomaton, where $Q, T \subseteq A^*$ are respectively *prefix-closed* (if $wa \in Q$ then $w \in Q$) and *suffix-closed* (if $aw \in T$ then $w \in T$): formally, a (Q,T)-*biautomaton* is, like an automaton, a functor $\mathcal{A} : \mathcal{I}_{Q,T} \to \mathcal{C}$, except $\mathcal{I}_{Q,T}$ 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 we also require the diagram on the right to commute. In $\mathbf{Set}$, a biautomaton is thus given by two state-sets $\mathcal{A}(\mathtt{st}_1)$ and $\mathcal{A}(\mathtt{st}_2)$, and each prefix $q \in Q$ leads to a state in $\mathcal{A}(\mathtt{st}_1)$ while, for every suffix $t \in T$, each state in $\mathcal{A}(\mathtt{st}_2)$ is known to accept or reject t. There are also transition functions from $\mathcal{A}(\mathtt{st}_1)$ to $\mathcal{A}(\mathtt{st}_2)$ that are consistent with the prefixes and suffixes.

$$\begin{array}{ccc} \mathtt{st}_2 & \xrightarrow{at\triangleleft} & \mathtt{out} \\ {\scriptstyle\epsilon}\uparrow & & \uparrow{\scriptstyle t\triangleleft} \\ \mathtt{st}_1 & \xrightarrow{a} & \mathtt{st}_2 \\ {\scriptstyle\triangleright q}\uparrow & & \uparrow{\scriptstyle\epsilon} \\ \mathtt{in} & \xrightarrow[\triangleright qa]{} & \mathtt{st}_1 \end{array}$$

Theorem 2.1 can be adapted for biautomata, and the initial and final biautomata recognizing $\mathcal{L}_{Q,T}$ are then made of finite coproducts of $\mathcal{L}(\mathtt{in})$ and finite products of $\mathcal{L}(\mathtt{out})$. Writing $\mathcal{J}_{Q,T}$ for the $(\mathcal{E}, \mathcal{M})$-factorization of $\coprod_{q \in Q} \prod_{t \in T} \mathcal{L}(\triangleright q t \triangleleft)$, the corresponding minimal biautomaton then has state-spaces $(\operatorname{Min}\mathcal{L}_{Q,T})(\mathtt{st}_1) = \mathcal{J}_{Q \cup QA, T}$ and $(\operatorname{Min}\mathcal{L}_{Q,T})(\mathtt{st}_2) = \mathcal{J}_{Q, T \cup AT}$: in $\mathbf{Set}$, these are respectively the Myhill-Nerode equivalence classes of Q and $Q \cup QA$ with respect to suffixes in $T \cup TA$ and T. The minimal biautomaton may thus be fully computed using $\textsc{Eval}_{\mathcal{L}}$, and it can be merged into a *hypothesis automaton* $\mathcal{H}_{Q,T}\mathcal{L}$ precisely when $\epsilon^{min}_{Q,T} = (\operatorname{Min}\mathcal{L}_{Q,T})(\epsilon)$ is an isomorphism.

This algorithm instanciates Angluin's L* algorithm, but also its extensions to weighted automata [2] and to transducers [3]: in particular, the condition that $\epsilon^{min}_{Q,T}$ is an isomorphism instanciates exactly the conditions of *closedness* ($\epsilon^{min}_{Q,T} \in \mathcal{E}$) and *consistency* ($\epsilon^{min}_{Q,T} \in \mathcal{M}$) that are

Algorithm 2 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 $\mathcal{J}_{Q,T} \rightarrowtail \mathcal{J}_{Q\cup\{qa\},T}$ is not an $\mathcal{E}$-morphism; add it to Q
    else if $\epsilon^{min}_{Q,T}$ is not an $\mathcal{M}$-morphism **then**
      find $at \in AT$ such that $\mathcal{J}_{Q,T\cup\{at\}} \twoheadrightarrow \mathcal{J}_{Q,T}$ is not an $\mathcal{M}$-morphism; add it to T
    end if
  end while
  merge $\text{Min}\,\mathcal{L}_{Q,T}$ into the hypothesis automaton $\mathcal{H}_{Q,T}\mathcal{L}$
  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

Algorithm 3 A variant of the FunL*-algorithm

Input: $\text{Eval}_{\mathcal{L}}$ and $\text{Equiv}_{\mathcal{L}}$
Output: an automaton recognizing $\mathcal{L}$

$Q = T = \{\epsilon\}$
loop
  while there is a $qa \in QA$ such that $\mathcal{J}_{Q,T} \rightarrowtail \mathcal{J}_{Q\cup\{qa\},T}$ is not an $\mathcal{E}$-morphism **do**
    add qa to Q
  end while
  build an automaton $\mathcal{H}$ with state-space $\coprod_Q \mathcal{L}(\texttt{in})$
  if $\text{Equiv}_{\mathcal{L}}(\mathcal{H})$ outputs some counter-example w **then**
    add w and its suffixes to T
  else
    return $\mathcal{H}$
  end if
end loop

required to merge the table into a hypothesis automaton in these settings.

$\mathcal{M}$-noetherianity and $\mathcal{E}$-artinianity conditions are also required for the algorithm to terminate. Extending Colcombet and Petrişan's results, we also use the $\mathcal{M}$-dimension and $\mathcal{E}$-codimension to give a bound on the complexity of the algorithm.

Theorem 2.3. *Algorithm 2 is correct. It also terminates when* $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *is* $\mathcal{M}$*-noetherian and* $\mathcal{E}$*-artinian, making at most* $\dim_{\mathcal{M}}(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *updates to* Q *(lines 5 and 12) (including calls to* $\textsc{Equiv}_{\mathcal{L}}$*) and* $\operatorname{codim}_{\mathcal{E}}(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *updates to* T *(line 7).*

There is a variant of L* algorithms that only checks for closedness but not consistency, at the cost of producing an automaton that is not minimal [9]. This variant was not studied by Colcombet and Petrişan but it also has a categorical generalization given by Algorithm 3, with the condition that $\mathcal{E}$-morphisms $\coprod_Q \mathcal{L}(\mathtt{in}) \twoheadrightarrow \mathcal{J}_{Q,T}$ have right-inverses. When this holds, a hypothesis automaton with state-space $\coprod_Q \mathcal{L}(\mathtt{in})$ can be built as soon as $\mathcal{J}_{Q,T} \rightarrowtail \mathcal{J}_{Q\cup QA,T}$ is an $\mathcal{E}$-morphism, but this also requires that these right-inverses be computable.

Proposition 2.4. *Algorithm 3 is correct. It also terminates when* $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *is* $\mathcal{M}$*-noetherian and* $\mathcal{E}$*-artinian, making at most* $\dim_{\mathcal{M}}(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *updates to* Q *(line 4) and* $\operatorname{codim}_{\mathcal{E}}(\operatorname{Min}\mathcal{L})(\mathtt{st})$ *updates to* T *(including calls to* $\textsc{Equiv}_{\mathcal{L}}$*).*

3 Transducers with output in arbitrary monoids

Our first new instanciation of the framework of Section 2 is for monoidal transducers. Here we call a *transducer* with *input alphabet* A and *output alphabet* B a (non-necessarily complete) deterministic automaton with input alphabet A, but such that the transitions also produce words in B^*. An example of such a transducer with $A = \{a, b\}$ and $B = \{\alpha, \beta\}$ is given in Figure 1: it recognizes a language such that, for instance, $\mathcal{L}(\triangleright aba\triangleleft) = \alpha^2\beta\alpha$ but $\mathcal{L}(\triangleright abb\triangleleft) = \bot$ (it is undefined).

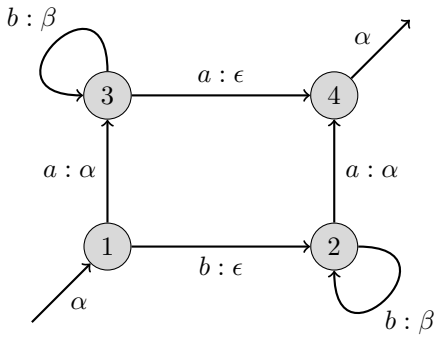


Figure 1: A (monoidal) transducer $\mathcal{A}$

Transducers can be minimized [14] and the minimal transducer recognizing a language can be learned using an L*-like algorithm [3]. These results are actually instances of the categorical framework of Section 2 [4, 5], but the arguments used to show it applies are very specific to the corresponding output category and rely on the freeness of B^* as a monoid, hence the question of whether they generalize.

In this section, we thus extend them to *monoidal transducers*, that is transducers whose outputs may now be considered as elements of an arbitrary monoid. For instance, the transducer of Figure 1 is minimal as a transducer with output in B^*, but it is not when considered to output elements in $B^{\circledast}$ (where $\alpha\beta = \beta\alpha$).

Formally, let M be an *output monoid* with unit ϵ[1]. We write its elements with greek letters to differentiate them from words in A^*, and as such we write e for the empty word in A^*. We define an *M-transducer* as a functor $\mathcal{A} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ such that $\mathcal{A}(\mathtt{in}) = \mathcal{A}(\mathtt{out}) = 1$, where $\mathbf{Kl}(\mathcal{T}_M)$ is the Kleisli category for the monad $\mathcal{T}_M X = M \times X + 1$. Recall that this category has sets for objects and, for arrows $f : X \nrightarrow Y$, functions $f : X \to M \times Y + 1$ or equivalently functions $f : M \times X + 1 \to M \times Y + 1$ verifying $f(\bot) = \bot$, $f(\upsilon, x) = (\upsilon\nu, y)$ if $f(\epsilon, x) = (\nu, y)$ and $f(\upsilon, x) = \bot$ otherwise. In particular, we write $X + 1 = X \sqcup \{\bot\}$, the identity on a set X is the function $\mathrm{id}_X(x) = (\epsilon, x)$, and the composition of two arrows $f : X \nrightarrow Y$ and $g : Y \nrightarrow Z$ is given by the function $(g \circ f)(x) = (\upsilon\nu, z)$ if $f(x) = (\upsilon, y)$ and $g(y) = (\nu, z)$, and $(g \circ f)(x) = \bot$ otherwise.

An M-transducer $\mathcal{A}$ thus has a *state-set* $S = \mathcal{A}(\mathtt{st})$, partial *transition functions* $- \odot a = \mathcal{A}(a) : S \to M \times S + 1$ (given by the pair of $- \cdot a : S \to S + 1$ and $- \bigcirc a : S \to M + 1$), possibly undefined *initialization value* and *state* $(\upsilon_0, s_0) = \mathcal{A}(\triangleright)(*) \in S + 1$ and a partial *termination function* $t = \mathcal{A}(\triangleleft) : S \to M + 1$. The *language* recognized by a transducer is a functor $\mathcal{L} : \mathcal{I} \to \mathbf{Kl}(\mathcal{T}_M)$ with $\mathcal{L}(\mathtt{in}) = \mathcal{L}(\mathtt{out}) = 1$, that is a function $w \mapsto \mathcal{L}(\triangleright w \triangleleft)(*) : A^* \to M + 1$. It is computed for a word w by multiplying together the outputs produced along the transitions for w in the transducer: $\mathcal{L}(\triangleright a_1 \dots a_n \triangleleft)(*) = t((\upsilon_0, s_0) \odot a_1 \odot \cdots \odot a_n))$.

3.1 Minimal transducers

To apply the framework of Section 2, we need three ingredients in $\mathbf{Kl}(\mathcal{T}_M)$: countable copowers of $\mathcal{L}(\mathtt{in}) = 1$, countable powers of $\mathcal{L}(\mathtt{out}) = 1$, and a factorization system. Since **Set** has all coproducts, $\mathbf{Kl}(\mathcal{T}_M)$ does as well as it is a Kleisli category. The *initial transducer* recognizing a language $\mathcal{L}$ thus exists and can be computed using Theorem 2.1: it has state-set A^*, transition functions $w \odot a = (\epsilon, wa)$, initialization value $\upsilon_0 = \epsilon$, initial state $s_0 = e$, and termination function $t(w) = \mathcal{L}(\triangleright w \triangleleft)(*)$.

The final transducer. The existence of countable powers of 1, and thus of the final transducer, is not as trivial: in general a Kleisli category need not have products. For $M = B^*$, the existence of this power relied on the specific structure of $\mathbf{Kl}(\mathcal{T}_{B^*})$. We generalize this argument to arbitrary M, but it relies on the fact that arrows $1 \nrightarrow \coprod 1$ factors through one of the coproduct inclusions $\kappa : 1 \nrightarrow \coprod 1$, so there is little hope that it generalizes to other monads.

Let us thus define partial functions $\Lambda : \mathbb{N} \to M + 1$. We write $\bot^{\mathbb{N}}$ for the nowhere defined function $n \mapsto \bot$ and $(M+1)^{\mathbb{N}}_* = (M+1)^{\mathbb{N}} - \{\bot^{\mathbb{N}}\}$ for the set of partial functions that are not nowhere defined. We extend the product $\otimes : M^2 \to M$ of M as a left-action of M on $(M+1)^{\mathbb{N}}_*$ by setting $(\upsilon \otimes \Lambda)(n) = \upsilon \otimes \Lambda(n)$ for $n \in \mathbb{N}$ such that $\Lambda(n) \neq \bot$ and $(\upsilon \otimes \Lambda)(n) = \bot$ otherwise. Lemma 3.1 then translates the universal property of the product in this setting.

Lemma 3.1. *1 has all countable powers in $\mathbf{Kl}(\mathcal{T}_M)$ if and only if there are two functions* $\mathrm{lgcd} : (M+1)^{\mathbb{N}}_* \to M$ *and* $\mathrm{red} : (M+1)^{\mathbb{N}}_* \to (M+1)^{\mathbb{N}}_*$ *such that for all* $\Lambda \in (M+1)^{\mathbb{N}}_*$, $\Lambda = \mathrm{lgcd}(\Lambda)\,\mathrm{red}(\Lambda)$*; and for all* $\Gamma, \Lambda \in (M+1)^{\mathbb{N}}_*$ *and* $\upsilon, \nu \in M$*, if* $\upsilon\Gamma = \nu\Lambda$ *then* $\upsilon = \nu$ *and* $\Gamma = \Lambda$.

The power $\prod_I 1$ *is then the set* $\mathrm{Irr}(I, M) = \{\mathrm{red}\,\Lambda \mid \Lambda \in (M+1)^I_*\}$.

[1] a set equipped with a binary associative operation $\otimes$ such that $\epsilon \otimes \upsilon = \upsilon \otimes \epsilon = \upsilon$

When these conditions are verified, Theorem 2.1 gives us the *final transducer* recognizing a language $\mathcal{L}$. It has state-set $\mathcal{A}^{final}(\mathtt{st}) = \mathrm{Irr}(A^*, M)$, transition functions $\Gamma \mapsto (\mathrm{lgcd}(a^{-1}\Gamma), \mathrm{red}(a^{-1}\Gamma))$ (where $(a^{-1}\Gamma)(w) = \Gamma(aw)$), initialization value $\upsilon_0 = \mathrm{lgcd}\,\mathcal{L}$, initial state $s_0 = \mathrm{red}\,\mathcal{L}$, and termination function $t(\Gamma) = \Gamma(e)$.

Note that our conditions are very similar to those sufficient for a minimal monoidal transducer to exist [8]. While ours are a bit stronger, they allow for a better-behaved notion of minimality (verifying a universal property), and in particular for active learning.

Recall moreover that if $\chi\chi' = \epsilon$, we say that χ is *right-invertible* and χ' is *left-invertible*. An element is *invertible* when it is both left- and right-invertible, and the set of invertibles is written $M^\times$. If $\upsilon\Lambda = \Gamma$, we say υ *left-divides* Γ: a *left-gcd* is a left-divisor that is left-divided by all other left-divisors, and Λ is *left-coprime* if it has only invertible left-divisors. If for all υ $\upsilon\Lambda = \upsilon\Gamma$ implies $\Lambda = \Gamma$, Λ and Γ, we say that M is *left-cancellative* (up to invertibles on the left when it only implies $\Lambda = \chi\Gamma$ for some $\chi \in M^\times$). Similarly, if $\upsilon\Lambda = \nu\Lambda$ implies $\upsilon = \nu$ for all left-coprime Λ, we say that M is *right-coprime-cancellative*. These natural properties of monoids can then be used to rewrite the conditions of Lemma 3.1, showing for example that they hold for trace monoids, that is monoids obtained from free monoids by having certain letters commute.

Proposition 3.2. 1 *has all countable powers in* $\mathbf{Kl}(\mathcal{T}_M)$ *as soon as* 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. These conditions are also necessary as soon as left- and right-invertibles of* M *are all invertibles.*

Factorization systems. Finally, even for $M = B^*$, the factorization system consisting of surjective and injective functions is not enough because the corresponding minimal transducer need not have a minimal number of states. Instead, we let Surj, Inj, Tot and Inv be sets of morphisms defined as follows. For $f : X \to M \times Y + 1$, write $f(x) = (f_1(x), f_2(x))$ when $f(x) \neq \bot$, and have $f \in \mathrm{Surj}$ whenever f_2 is surjective on Y, $f \in \mathrm{Inj}$ whenever f_2 is injective when corestricted to Y, $f \in \mathrm{Tot}$ when $f(x) \neq \bot$ for all x, and $f \in \mathrm{Inv}$ when $f_1(x)$ is always invertible (so $f_1(x) = \epsilon$ when $M = B^*$ in particular).

Lemma 3.3. $(\mathcal{E}_1, \mathcal{M}_1) = (\mathrm{Surj} \cap \mathrm{Inj} \cap \mathrm{Inv}, \mathrm{Tot})$, $(\mathcal{E}_2, \mathcal{M}_2) = (\mathrm{Surj} \cap \mathrm{Inj}, \mathrm{Tot} \cap \mathrm{Inv})$ *and* $(\mathcal{E}_3, \mathcal{M}_3) = (\mathrm{Surj}, \mathrm{Inj} \cap \mathrm{Inv} \cap \mathrm{Tot})$ *form a* tetranary factorization system*, that is they are all factorization systems and* $\mathcal{E}_1 \subseteq \mathcal{E}_2 \subseteq \mathcal{E}_3$*. In particular, a morphism* f *in* $\mathbf{Kl}(\mathcal{T}_M)$ *factors uniquely (up to isomorphism) as* $f = m \circ f_2 \circ f_1 \circ e$ *with* $e \in \mathcal{E}_1$*,* $f_1 \in \mathcal{M}_1 \cap \mathcal{E}_2$*,* $f_2 \in \mathcal{M}_2 \cap \mathcal{E}_3$ *and* $m \in \mathcal{M}_3$*.*

The factorization system that we choose to define the minimal transducer (for which we compute Reach and Obs) is $(\mathcal{E}_3, \mathcal{M}_3)$, hence we also write it $(\mathcal{E}, \mathcal{M})$; $\mathrm{Min}\,\mathcal{L}$ is then characterized by that it has the minimal possible number of states, and all its outputs pulled (as much as possible) towards the initial state. But we also get that if $\mathcal{A}$ recognizes $\mathcal{L}$, $\mathrm{Min}\,\mathcal{L} = \mathrm{Obs}(\mathrm{Prefix}(\mathrm{Total}(\mathrm{Reach}\,\mathcal{A})))$, where $\mathrm{Total}\,\mathcal{A}$ is the $(\mathcal{E}_1, \mathcal{M}_1)$-factorization of $\mathcal{A} \to \mathcal{A}^{final}(\mathcal{L})$, and $\mathrm{Prefix}\,\mathcal{A}$ is its $(\mathcal{E}_2, \mathcal{M}_2)$-factorization. In practice, Reach just removes the unreachable states, Total removes those that recognize $\bot^{A^*}$, Prefix pulls the outputs towards the initial state,

and Obs merges equivalent states together. For instance, starting with the transducer $\mathcal{A}$ of Figure 1 seen as a transducer over the free commutative monoid on $\{\alpha, \beta\}$, $\operatorname{Reach}\mathcal{A} = \mathcal{A}$ (all states are reachable), $\operatorname{Total}\mathcal{A} = \mathcal{A}$ (all states produce an output when following some word), $\operatorname{Prefix}\mathcal{A}$ is the transducer of Figure 2 (the output letters α and β commute), and $\operatorname{Obs}(\operatorname{Prefix}\mathcal{A}) = \operatorname{Min}\mathcal{L}$ is the transducer of Figure 3.

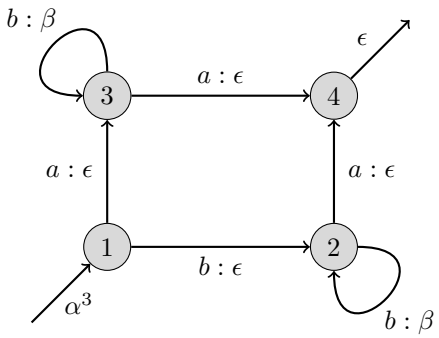


Figure 2: $\operatorname{Prefix}\mathcal{A}$

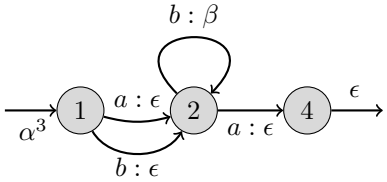


Figure 3: $\operatorname{Obs}(\operatorname{Prefix}\mathcal{A})$

3.2 Algorithms

Let $\mathcal{L}$ be a rational language, that is a language recognized by a transducer with finite state-set $\mathcal{A}$. To effectively compute $\operatorname{Min}\mathcal{L}$, we need additional asusmptions on M.

First and foremost, M has to be right-noetherian, meaning that any chain of left-divisors of an element $v \in M$ must be finite: we write $\operatorname{rk} v$ for the greatest possible length of a chain of left-divisors of $v \in M$. In particular, this is equivalent to having $(\operatorname{Min}\mathcal{L})(\mathtt{st})$ be $\mathcal{E}$-artinian and $\mathcal{M}$-noetherian, hence to having Theorem 2.3 apply. But M being right-noetherian implies that its left- and right-invertibles are invertibles, hence Proposition 3.2 applies as well, and $\operatorname{Min}\mathcal{L}$ does exist. Second, we need a few basic operations in M to be effective, 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 $v \wedge \bot = \bot \wedge v = \bot$); LeftDivide that takes as input $\delta, v \in M$ and computes a $\nu \in M$ such that $v = \delta\nu$ (with $\nu = \bot$ when $v = \bot$) or fails; and UpToInvLeft that takes as input two families $\Lambda, \Gamma \in M^I$ indexed by a finite set I and outputs a $\chi \in M^\times$ such that $\Lambda = \chi\Gamma$ if such a χ exists, and $\bot$ otherwise.

Let us define notations to describe the complexity of the algorithms. We write $|\mathcal{A}|_{\mathtt{st}}$ for the number of states of the transducer $\mathcal{A}$, and $|\mathcal{A}|_{\to}$ for its number of transitions. If $\operatorname{rk}\Lambda = \min\{\operatorname{rk}(\Lambda(i)) \mid \Lambda(i) \neq \bot\}$ for $\Lambda \in (M+1)^I_*$, we also write $\operatorname{rk}\mathcal{A} = \sum_{s \in (\operatorname{Total}\mathcal{A})(\mathtt{st})} \operatorname{rk}\mathcal{L}_s$, where $\mathcal{L}_s$ is the language recognized from state s in $\mathcal{A}$.

Learning. Since Theorem 2.3 applies, we almost immediately get an L*-like algorithm computing $\operatorname{Min}\mathcal{L}$ by instanciating Algorithm 2 in $\mathbf{Kl}(\mathcal{T}_M)$ (see Algorithm 6 in Appendix A.2). We do not go into details here because the resulting algorithm is very similar to Vilar's one for

non-monoidal transducers (where $M = B^*$) [3]: the main differences are that the longest common prefix in B^* is replaced by the left-gcd, and that, in some places, testing for equality is replaced by testing for equality up to invertibles on the left (with UpToInvLeft). Complexity-wise, the algorithm makes at most $|\operatorname{Min}\mathcal{L}|_{\mathtt{st}}$ updates to Q (and thus of calls to $\textsc{Equiv}_{\mathcal{L}}$), and $|\operatorname{Min}\mathcal{L}|_{\mathtt{st}} + \operatorname{rk}(\operatorname{Min}\mathcal{L})$ updates to T.

Note that, conversely, Algorithm 3 does not apply because here $\mathcal{E}$-morphisms need not have right-inverses.

Minimization There is also an algorithm to directly transform $\mathcal{A}$ into $\operatorname{Min}\mathcal{L}$. As hinted by the categorical structure in Section 3.1, it consists in applying sucessively the operators Reach, Total, Prefix and Obs, which may themselves be computed by instanciating Algorithm 1 and its dual. These four steps are the one that are commonly followed when $M = B^*$ [14]. Computing Reach and Total is straightforward and can be done in linear time using depth-first searches on the underlying graph, and this stays true for arbitrary M. The problem of computing Prefix and Obs in general is tackled in [8], but the corresponding solution relies on the existence of an oracle that computes non-trivial left-divisors of countable (but rational) families. We thus provide another algorithm, which only requires UpToInvLeft and the binary $\wedge$.

The core of the problem is to compute $\operatorname{Prefix}\mathcal{A}$ from $\operatorname{Total}(\operatorname{Reach}\mathcal{A})$, that is to compute the left-gcds of the languages recognized by each state. Béal and Carton do it by pulling back letters along the transitions of $\mathcal{A}$ [15], 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 2 in Figure 1 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 [16], but again this algorithm does not generalize to arbitrary M because it relies crucially on the free monoid being graded. In the general case, $\operatorname{Prefix}\mathcal{A}$ may instead be computed using a fixed-point algorithm (see Algorithm 7 in Appendix A.2). The idea is to notice that $\operatorname{lgcd}\mathcal{L}_s = t(s) \wedge \bigwedge_{a\in A}(s \circ a)\operatorname{lgcd}\mathcal{L}_{s\cdot a}$, and, applying the Kleene fixed-point theorem to an appropriate lattice, we then get an algorithm that terminates in at most $O((|A|_{\mathtt{st}}+|A|_{\to})\operatorname{rk}\mathcal{A})$ calls to the various oracles.

Computing Obs is traditionally done using partition refinement. The generalization to arbitrary M is not trivial as equality up to invertibles has to be taken into account to decide whether two states are equivalent, and it can be done quite intuitively but tediously, in particular in $O(|A|^2_{\mathtt{st}}\,|\mathcal{A}|_{\to})$ calls to the various oracles when M is left-cancellative.

As the overall algorithm introduces new ideas, it was implemented in `OCaml` to demonstrate feasability [17]. The use of the `Fixpoint` module from the `OCamlGraph` library [18], while not optimal, actually makes it quite easy to implement each of the four operators.

3.3 Summary and future work

We studied transducers which may have their outputs in arbitrary monoids, and gave conditions on the output monoid for the framework of Section 2 to apply. The resulting notion of minimal transducer can thus be computed either using active learning in the style of Angluin, or by minimizing another transducer following steps

hinted by the categorical structure. This last algorithm was implemented as a proof-of-concept, and this whole section will be the subject of an article that is available as a draft at the time of writing [17].

This was mainly a theoretical development, and finding examples of where it applies is an interesting question. In particular, it holds for trace monoids so we hope to find applications in concurrency theory. Transducers are also used for natural language processing so we hope monoidal transducers can prove useful as well.

On the theoretical side, an interesting question is whether the categorical framework can be extended to encompass monoids for which notions of minimality can be defined but do not lead to a unique minimal transducer. Finally, the quaternary factorization system on $\mathbf{Kl}(\mathcal{T}_M)$ may be seen as arising from the factorization system on **Set**, and a natural question is thus what conditions are required on a monad for factorization systems on its Kleisli category to arise in the same way.

4 Weighted automata over integral domains

Our second instanciation of the framework of Section 2 is for weighted automata with weights over integral domains. Recall that if R is a *semiring*[2], a *finite R-weighted automaton* $\mathcal{A}$ over the alphabet A is given by a non-deterministic finite automaton where each transition is given a weight in R and each state is given both an input weight and an output weight in R. The weight of a path is then the product of the input weight of the first state, the weights of the transitions in the path and the output weight of the final state, and the language recognized by an automaton is defined as $\mathcal{L}(\triangleright w \triangleleft)$ being the sum of the weights of the paths corresponding to w.

When R is a *field*[3], a finite automaton can be minimized effectively [13] and the corresponding minimal automaton can be learned using an L*-like algorithm [2]. These two results are moreover instances of the framework of Section 2 [4, 5].

When R is an *integral domain*[4], an R-weighted automaton can be considered as a K-weighted automaton where K is R's field of fractions[5] and can thus be minimized and learned as such, but the corresponding minimal automaton may have weights that are not in R. But if R is also *principal*[6], this minimal K-weighted automaton can be transformed in an R-weighted automaton with the same (minimal) number of states [19] and an automaton recognizing a rational language can always be learned with an *improper* $L*$-like algorithm [9]. But there are limits: the learned automaton may not be minimal, and as shown by Example 4.1, transforming a K-weighted minimal automaton into an R-weighted automaton does not lead to a notion of minimality for which the R-weighted minimal automaton is unique (even up to isomorphism).

Example 4.1. Let $R = \mathbb{Z}$ (its field of fraction is $K = \mathbb{Q}$) and $\mathcal{L}$ be the $\mathbb{Z}$-language on the alphabet $A = \{a\}$ given by $\mathcal{L}(\triangleright a^{2n} \triangleleft) = 1$ and

[2] a set equipped with two binary operations $+$ and $\times$ such that $(R, +)$ is a commutative monoid, $(R, \times)$ is a monoid, $\times$ distributes over $+$ and 0, the additive unit, is absorbing for $\times$

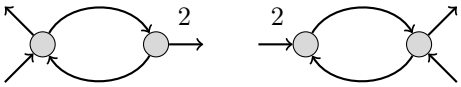
[3] $\times$ is commutative, $(R, +)$ is a group i.e. every element has an inverse for $+$, and $(R_*, \times)$, with $R_* = R - \{0\}$, is a group as well

[4] $(R, +)$ is a group, $\times$ is commutative and the scalar multiplications by non-zero $r \in R_*$ are injective

[5] its formal completion into a field

[6] each element has a unique prime factorization

$\mathcal{L}(\triangleright a^{2n+1} \triangleleft) = 2$. $\mathcal{L}$ is recognized by both automata of Figure 4 (where we only draw transitions with non-zero weights and the default weight is the multiplicative unit 1).



(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$

Figure 4: Two non-isomorphic $\mathbb{Z}$-weighted automata with minimal number of states

$\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic as $\mathbb{Q}$-weighted automata and are the minimal such automata recognizing $\mathcal{L}$. Yet, they are not isomorphic as $\mathbb{Z}$-automata: there is only a morphism $\mathcal{A}_1 \to \mathcal{A}_2$.

Hence the need to study minimality when R is not a field.

4.1 Notions of minimality

Let R be an integral domain. To model R-weighted automata in our framework, we use functors $\mathcal{I} \to R\mathbf{Mod}_{Free}$ whose output category is that of *free R-modules*: its objects are *R-modules*[7] that have a basis[8]. In particular, if $\mathcal{A}$ has state-set S, we set $\mathcal{A}(\mathtt{st})$ to be $\coprod_S R$, the free R-module with basis S: for Figures 4a and 4b, the state-space is the module $\mathbb{Z}^2$. When R is a field this is enough to apply to framework of Section 2 as every module is a free module: every vector space has a basis. But this does not work in the general case because a direct product of free modules need not be free: for instance, $\mathbb{Z}^{A^*}$ is not a free $\mathbb{Z}$-module.

[7] commutative groups with with a scalar multiplication by elements of R

[8] there is a subset $B \subset M$ such that every element in M can be written uniquely as a finite R-linear combination of elements of B

Hence we consider instead the output category $R\mathbf{Mod}_{TF}$ of *torsion-free R-modules*[9]. This category has all products and coproducts hence the initial and final automata recognizing a language exist by Theorem 2.1, but the corresponding notion of automata, that we call *R-modular automata*, and in particular the minimal R-modular automata recognizing *rational languages*[10], need not be weighted automata in the usual sense: we will study conditions for this to be true in Section 4.2.

To define a notion of minimality, the only ingredient left is thus a factorization system. There is actually a whole lattice of factorization systems on $R\mathbf{Mod}_{TF}$. Recall indeed that, for a *multiplicative subset*[11] S of R_* and for two modules $N \subseteq M$, the S-saturation of N in M is $\mathrm{sat}^S_M N = \{m \in M \mid \exists s \in S, sm \in N\}$.

Lemma 4.2. *Let S and S' be multiplicative subsets of R_*. Define $(\mathcal{E}_S, \mathcal{M}_S)$ by $e : X \to Y \in \mathcal{E}_S$ when $\mathrm{sat}^S_Y(\mathrm{im}\, e) = Y$; and $m : X \to Y \in \mathcal{M}_S$ when m is injective and $\mathrm{sat}^S_Y(\mathrm{im}\, m) = \mathrm{im}\, m$. Then $(\mathcal{E}_S, \mathcal{M}_S)$ is a factorization system on $R\mathbf{Mod}_{TF}$, and $\mathcal{E}_S \subseteq \mathcal{E}_{S'}$ if and only if S divides S', that is if for all $s \in S$ there is a $r \in R_*$ such that $sr \in S'$.*

In particular, if $S = \{1\}$ then $(\mathcal{E}_S, \mathcal{M}_S)$ is the usual factorization system given by surjections and injections (regular epimorphisms and monomorphisms), and when $S = R_*$, the $\mathcal{E}_S$-morphisms are exactly the epimorphisms in $R\mathbf{Mod}_{TF}$, and the $\mathcal{M}_S$-morphisms are exactly the regular monomorphisms (the categorical analogue of embeddings) in $R\mathbf{Mod}_{TF}$: we

[9] modules for which the scalar multiplications by non-zero elements are injective

[10] languages recognized by a finite R-weighted automaton

[11] it contains 1 and is stable under multiplication

respectively write these two factorization systems $(\mathcal{E}_{reg}, \mathcal{M})$ and $(\mathcal{E}, \mathcal{M}_{reg})$. We also get that if S contains only invertible elements then $(\mathcal{E}_S, \mathcal{M}_S) = (\mathcal{E}_{reg}, \mathcal{M})$, so when R is a field the entire lattice of factorization systems collapses to $(\mathcal{E}_{reg}, \mathcal{M})$.

We may relate the R-minimal and K-minimal automata recognizing a rational R-valued language using the tensor product in $R\mathbf{Mod}_{TF}$. Indeed, if S is a multiplicative subset of R_*, the functor $M \mapsto S^{-1}M = S^{-1}R \otimes_R M$, sending a R-module to the $S^{-1}R$-module of fractions with numerators in M and denominators in S, maps a R-weighted automaton to the corresponding $S^{-1}R$-weighted one (weights in R are also weights in $S^{-1}R$).

Proposition 4.3. *Let S and S' be two multiplicative subsets of R_*. Then the functor $S^{-1}R \otimes_R -$ preserves $(\mathcal{E}_{S'}, \mathcal{M}_{S'})$-factorizations. Moreover, if S' divides S, then this functor sends the $(\mathcal{E}_{S'}, \mathcal{M}_{S'})$-minimal R-modular automata to the $(\mathcal{E}_{reg}, \mathcal{M})$-minimal $S^{-1}R$-modular ones. Finally, R-valued rational languages are also rational when seen as $S^{-1}R$-valued languages.*

The minimal R-modular automata defined by these factorization systems are therefore all isomorphic and minimal when seen as K-weighted automata. In particular, they all have the same number of states (their state-spaces all have the same generic rank). Moreover, a morphism f in $R\mathbf{Mod}_{TF}$ is in $\mathcal{M}$ if and only if $R^{-1}f$ is injective, and it is in $\mathcal{E}$ if and only if $R^{-1}f$ is surjective, hence $(\mathcal{E}_{reg}, \mathcal{M})$ and $(\mathcal{E}, \mathcal{M}_{reg})$ define the widest range of factorization systems whose corresponding minimal R-modular automata are also minimal as K-weighted automata.

4.2 The case of Dedekind and principal domains

Let S be a multiplicative subset of R_*. For the $(\mathcal{E}_S, \mathcal{M}_S)$-minimal automata to be usable in practice, we need them to be representable with finite memory. A first result is that if R is noetherian[12], then the minimal automaton recognizing a rational language has finitely generated state-space: it is the quotient of a finite R-weighted automaton. But this is still not very useful in practice, so we will now require stronger conditions on R, namely that it be *Dedekind*[13] or even principal.

Dedekind domains are interesting because their finitely generated torsion-free modules are projective (they are direct summands of free modules), and this implies in turn that they are isomorphic to some $R^n \oplus I$, where $n \in N$ and I is a fractional ideal, that is a finitely generated sub-R-module of K: in particular, there is an other fractional ideal I^{-1} such that $II^{-1} = R$ and $I \oplus I^{-1} \cong R^2$ [20]. R-modular automata with finitely generated projective state-space are thus almost R-weighted, in the sense that they can be represented by a R-weighted automaton with a single special state s_I such that transitions going into s_I may be weighted with elements of I, and transitions going out of s_I may be weighted with elements of I^{-1}. Principal domains, in turn, are interesting because they are Dedekind but have all their fractional ideals isomorphic to R, hence their finitely generated torsion-free modules are free: R-modular automata with finitely-generated state-space are R-weighted.

Proposition 4.4. *If R is Dedekind, the*

[12] all the submodules of R are finitely generated

[13] every element can be factorized using a finite number of primes

$(\mathcal{E}_S, \mathcal{M}_S)$-minimal R-modular (bi)automata recognizing a rational languages have finitely generated projective state-space. If R is also principal, they all are finite R-weighted automata.

For R principal, we immediately get that the minimal automaton is R-weighted. For R Dedekind, if $(\operatorname{Min}\mathcal{L})(\mathtt{st}) = R^n \oplus I$ then an R-weighted automaton with $n+2$ states (so only one additional state) may always be built by extending $\operatorname{Min}\mathcal{L}$ along $R^n \oplus I \hookrightarrow R^n \oplus I \oplus I^{-1} \cong R^{n+2}$, as depicted by Example 4.5. However, there may also be a (smaller) R-weighted automaton with $n+1$ states recognizing $\mathcal{L}$.

Example 4.5. Let R be the Dedekind domain $\mathbb{Z}[\sqrt{-5}]$ (it is not principal because $2 \times 3 = 6 = (1-\sqrt{-5})(1+\sqrt{-5})$). Consider the automaton $\mathcal{A}$ over $A = \{a, b\}$ with state-space R^4 depicted in Figure 5.

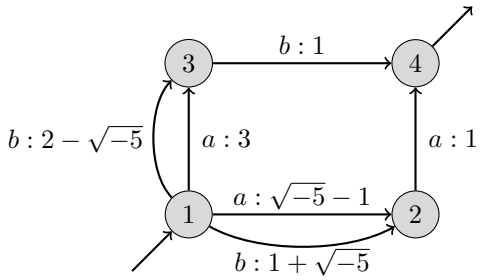


Figure 5: A $\mathbb{Z}[\sqrt{-5}]$-weighted automaton

It recognizes the language given by $\mathcal{L}(\triangleright aa\triangleleft) = \sqrt{-5} - 1$, $\mathcal{L}(\triangleright ab\triangleleft) = 3$, $\mathcal{L}(\triangleright ba\triangleleft) = 1 + \sqrt{-5}$, $\mathcal{L}(\triangleright bb\triangleleft) = 2 - \sqrt{-5}$ and $\mathcal{L}(\triangleright w\triangleleft) = 0$ for every other w. It has the minimum number of states among R-weighted automata recognizing $\mathcal{L}$, yet it cannot be $(\mathcal{E}_S, \mathcal{M}_S)$-minimal for some S because the saturated submodule of R^4 of reachable configurations is $R \oplus I \oplus R$, where I is the fractional ideal $\{((\sqrt{-5}-1)\lambda + (1+\sqrt{-5})\mu, 3\lambda + (2-\sqrt{-5})\mu) \mid \lambda, \mu \in R\} \cong (2-\sqrt{-5}, 3)$ (the isomorphism is given by $i \leftrightarrow (i(\sqrt{-5}-1)/3, i)$).

Hence $\operatorname{Reach}_{\mathcal{E}_{reg},\mathcal{M}}\mathcal{A}$ has state-space $R \oplus I \oplus R$ and is given by Figure 6. It is $(\mathcal{E}_{reg}, \mathcal{M})$-minimal, and adding an extra state yields back an automaton isomorphic to the one of Figure 5.

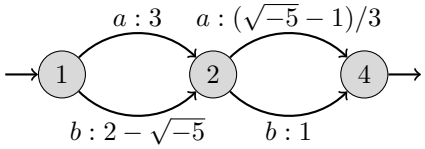


Figure 6: A minimal projective $\mathbb{Z}[\sqrt{-5}]$-modular automaton

Assuming the computability of basic operations over R amounting to computing factorizations (note that the saturation of a finitely generated module is computable [21] and morphisms between finitely generated projective modules have an effective matrix representation [22]), the minimal automata can be computed by instancianting Algorithms 1 and 2 (in the category of projective modules), but unfortunately the categorical framework does not give a complete bound on the complexity of the resulting algorithms because a noetherian domain can have arbitrary long strict (finite) chains of submodules, for instance $\mathbb{Z}$ and the $2^n\mathbb{Z}$. Algorithm 3 applies as well for $(\mathcal{E}_{reg}, \mathcal{M})$ (because surjections between finitely generated projective modules have right inverses), and it answers the open question asked by van Heerdt et al. of finding a categorical framework for their algorithm implementing improper learning of weighted automata [9]: in particular, their assumption that linear systems are solvable is encompassed by the computability of right inverses of $\mathcal{E}_{reg}$-morphisms.

Proposition 4.6. *If R is Dedekind, finitely generated torsion-free modules are $\mathcal{M}_S$-noetherian*

and $\mathcal{E}_S$-artinian in the category of projective modules. Moreover, $\operatorname{codim}_{\mathcal{E}_{reg}} M = \dim_{\mathcal{M}_{reg}} M$ *is the rank of* M*.*

Note that this relies crucially on the fact that finitely generated torsion R-modules are artinian, which is specific to Dedekind domains.

4.3 Summary and future work

We studied automata weighted over integral domains. We showed that, allowing for the state-spaces to be any torsion-free modules, there is actually a whole lattice of notions of minimality, and that the corresponding minimal automata are all isomorphic when the weights are considered in the domain's field of fractions. When the domain is principal, these minimal automata are weighted automata in the classic sense and they can be computed or learned using the algorithms of Section 2. The same results hold when the domain is Dedekind, except an additional state may be needed to represent the minimal automaton as a weighted automaton.

Again, this was mainly a theoretical development, and concrete applications of these results are yet to be found, in particular when the weights are taken over a Dedekind domain.

On the theoretical side, a first question is whether a bound on the complexity of the algorithms could be found by making the categorical framework more precise, as its current version only gives a partial bound. Another question is whether these results can be generalized, in particular to non-commutative Dedekind rings and to semi-rings enjoying similar properties, but also to monoid rings for which weighted automata are really weighted monoidal transducers.

5 Quasi-ordered automata

Our third and final instanciation of the framework of Section 2 is to *quasi-ordered automata*, that is automata whose underlying graph is quasi-ordered. Before delving into their definition, let us recall the theory of well quasi-orders [23] that motivate their use.

A *well quasi-order* (or *WQO*) $(X, \leq)$ is a set X equipped with a *quasi-order*[14] such that every *upwards-closed* subset[15] $U \subseteq X$ has a finite *basis*: there are elements $u_1, \ldots, u_n$ such that $U = \uparrow u_1 \cup \cdots \cup \uparrow u_n$, where $\uparrow u_i$ is the upwards-closed subsets of elements that are greater than u_i. WQOs are useful because when the set of configurations of a transition system is a WQO, the termination and reachability problems for this transition system can be proven to be decidable (with additional conditions).

The main example of WQO we will be using is the *subword ordering* $(A^*, \leq_*)$ of words on a quasi-ordered alphabet $(A, \leq)$: we write $w \leq_* w'$ if and only if w is a *subword* of w', that is $w = a_1 \ldots a_n$ and $w' = w'_0 a'_1 w'_1 a'_2 \ldots w'_{n-1} a'_n w'_n$ with $w'_i \in A^*$ and $a_i \leq a'_i$. When $\leq$ is a WQO, Higman's lemma states that this makes A^* a WQO as well [23]. It follows that we can give a WQO structure to any monoid that is a quotient of A^*: if $f : A^* \to M$ is surjective, we set $m \sqsubseteq_M m'$ if $m = f(w)$ and $m' = f(w')$ with $w \leq_* w'$, and we take $(M, \leq_M)$ to be the WQO given by the *transitive closure* of $\sqsubseteq_M$ ($m \leq_M m'$ if and only if $m \sqsubseteq m_1 \sqsubseteq \ldots \sqsubseteq m_n \sqsubseteq m'$ for some $m_1, \ldots, m_n \in M$). Note that these orderings are compatible with the monoid structure in that ϵ is a minimum element[16] and the monoid product

[14] a reflexive and transitive relation

[15] if $u \in U$ and $u \leq v \in X$ then $v \in U$ as well

[16] it is smaller than every other element

is *monotone* in both variables [17].

5.1 Upwards-closed sets as functors

Upwards-closed subsets for the subword ordering on A^* are subsets of A^*, so they can be represented as $(\mathbf{Set}, 1, 2)$-languages, that is functors $\mathcal{L} : \mathcal{O} \to \mathbf{Set}$ such that $\mathcal{L}(\mathtt{in}) = 1$ and $\mathcal{L}(\mathtt{out}) = 2 = \{\bot, \top\}$: we take $w \in \mathcal{L}$ if and only if $\mathcal{L}(\triangleright w \triangleleft) = \top$. But not every language can be seen as an upwards-closed subset: we also need to ask that if $w \leq_* w'$ then $\mathcal{L}(\triangleright w \triangleleft) \leq \mathcal{L}(\triangleright w' \triangleleft)$ for the *pointwise ordering*[18] on functions $1 \to 2$, where we set $\bot < \top$ in 2.

We thus restrict ourselves to the category **Ord** whose objects are quasi-orders and whose morphisms are monotone maps. **Ord** is an **Ord**-enriched category, in that the sets of morphisms $\mathbf{Ord}(X, Y)$ between two objects X and Y are themselves objects in **Ord**: they are quasi-ordered (with the pointwise ordering), and the composition of morphisms is monotone. $\mathcal{O}$ can also be seen as an **Ord**-enriched category, because $\mathcal{O}(\mathtt{in}, \mathtt{out}) = A^*$ can be equipped with the subword ordering, and an $(\mathbf{Ord}, 1, 2)$-language $\mathcal{L}$ corresponds to an upwards-closed set exactly when the functor $\mathcal{L}$ is **Ord**-enriched, that is when it is a monotone map of morphisms.

The corresponding automata notion is thus defined as **Ord**-enriched functors $\mathcal{A} : \mathcal{I} \to \mathbf{Ord}$ such that $\mathcal{A}(\mathtt{in}) = 1$ and $\mathcal{A}(\mathtt{out}) = 2$, where $\mathcal{I}$ is **Ord**-enriched by ordering $\mathcal{I}(\mathtt{st}, \mathtt{st}) = A^*$ with the subword ordering. In practice, these correspond to quasi-ordered (complete deterministic) automata: their state-set are quasi-ordered and the transition and output functions are monotone: if $s \leq s' \in \mathcal{A}(\mathtt{st})$ then $\mathcal{A}(a)(s) \leq \mathcal{A}(a)(s')$ for all $a \in A$, and if s is accepting then so is s'. The **Ord**-enriched structure also means that if $w \leq_* w'$ then $\mathcal{A}(w)(s) \leq \mathcal{A}(w')(s)$ for all $s \in \mathcal{A}(\mathtt{st})$, so in particular the initial state is a minimum element among reachable states, and transitions can only go up with respect to the ordering of states: for instance, transitions from accepting states can only go into other accepting states.

While the framework of Section 2 technically does not apply to **Ord**-enriched categories and functors, it can easily be extended, at least for our specific purposes. The factorization system on **Ord** is given by surjective monotone maps and injective monotone order-reflecting maps[19]. Theorem 2.1 does not apply anymore; instead, the initial automaton recognizing a language has state-space A^*, but ordered with the subword ordering (and not with equality). Similarly, the final automaton has state-space $\mathbf{Ord}(A^*, 2)$, the set of monotone maps from A^* to 2. The minimal automaton is then the factorization of the map that sends a $u \in A^*$ to $v \mapsto \mathcal{L}(\triangleright uv \triangleleft)$ seen as a monotone function from A^* to 2: it has states the Myhill-Nerode equivalence classes of $\mathcal{L}$, ordered by their corresponding residual languages. Note the quasi-order on $\mathbf{Ord}(A^*, 2)$ is antisymmetric and thus an order, hence so is the quasi-order on the state-space of the minimal automaton: it is also *acyclic*, meaning its underlying graph has no cycle of two or more transitions. The acyclic paths from the initial state to the (necessarily unique) accepting state in the minimal automaton thus give a minimal finite basis of $\mathcal{L}$.

Example 5.1. The quasi-ordered automaton of Figure 7 (with alphabet $(\{a, b\}, =)$) has its states ordered from left to right. It is minimal, and a minimal finite basis for the language it recognizes

[17] if $m_1 \leq_M m_1'$ and $m_2 \leq_M m_2'$ then $m_1 m_2 \leq_M m_1' m_2'$

[18] $f \sqsubseteq g : X \to Y$ if and only if $f(x) \leq_Y f(x)$ for all $x \in X$

[19] $f(x) \leq f(y)$ if and only if $x \leq y$

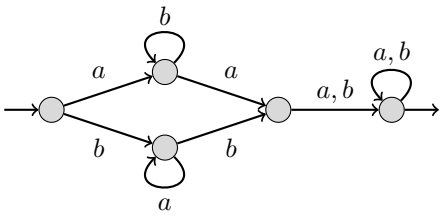


Figure 7: A quasi-ordered automaton

is $\{aaa, aab, bba, bbb\}$.

5.2 Automata learning and the VJGL lemma

Given a WQO $(X, \leq)$, define an *ideal* of X to be a subset I of X that is non-empty, *directed*[20] and *downwards-closed* [21], and let $\widehat{X}$ be the *sobrification* of X, that is its set of ideals ordered with inclusion. Note that the map $X \to \widehat{X}$ that assigns an $x \in X$ to $\downarrow x$, the ideal of elements smaller than x, is injective, monotone and order-reflecting.

Another useful property of WQOs is the Valk-Jantzen-Goubault-Larrecq (VJGL) lemma [10]. It states that if a WQO X has the effective complement property, i.e. if there is a computable function that takes some $x_1, \ldots, x_n \in X$ and outputs some $I_1, \ldots, I_m \in \widehat{X}$ such that $X - (\uparrow x_1 \cup \cdots \cup \uparrow x_n) = I_1 \cup \cdots \cup I_m$, then a finite basis of any upwards-closed subset $U \subseteq X$ may be learned using a finite number of queries to an oracle $\textsc{Eval}_{\widehat{U}}$ that decides whether any $I \in \widehat{X}$ is such that $I \cap U = \varnothing$. In a nutshell, the algorithm enumerates the elements x of X, checks whether they belong to U using $\textsc{Eval}_{\widehat{U}}(\downarrow x)$, and if so adds them to the potential basis and checks whether this basis generates U: if B is the current basis and $X - \uparrow B = I_1 \cup \cdots \cup I_m$ (by the effective complement property), it is enough to check whether $I_i \cap U = \varnothing$ for all i by using $\textsc{Eval}_{\widehat{U}}(I_i)$.

This resembles automata learning greatly, in that the algorithm first builds a hypothesis basis by querying which elements are in U, and then checks whether this basis generates U. This is more-or-less what happens when applying Algorithm 2 to learn **Ord**-enriched $(\mathbf{Ord}, 1, 2)$-languages (see Algorithm 8 in Appendix A.3). The `while` loop first builds a hypothesis automaton which represents a hypothesis basis for $\mathcal{L}$, using queries to $\textsc{Eval}_{\mathcal{L}}$, which can itself be implemented using $\textsc{Eval}_{\widehat{\mathcal{L}}}$. It tries to do so in a clever way, in that the bound on the number of calls to $\textsc{Equiv}_{\mathcal{L}}$ is the number of states of $\operatorname{Min}\mathcal{L}$, while the algorithm in the VGJL lemma checks whether the basis B is correct at least as many times as there are words in B (if the enumeration of X is well-chosen). Note that finding the basis in such a clever way may be costly, as the bound on the number of updates to T is quadratic in the number of states of $\operatorname{Min}\mathcal{L}$. The hypothesis automaton $\mathcal{H}_{Q,T}\mathcal{L}$ is still an **Ord**-enriched functor, hence an ordered acyclic automaton: the $\textsc{Equiv}_{\mathcal{L}}$ oracle can thus be implemented by first checking that every word in the corresponding basis is indeed in $\mathcal{L}$, and then checking not recognized by $\mathcal{H}_{Q,T}\mathcal{L}$ is not in $\mathcal{L}$ neither. This last step requires implementing the effective complement procedure of the subword ordering on the automaton representation: for each acyclic path $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$ in the underlying graph such that s_0 is the initial state and s_n is the accepting state, if A_i is the set of letters whose transition from s_i loop back to s_i then one should check that $I \cap \mathcal{L} = \varnothing$, where $I = A_0^* \downarrow(a_1) A_1^* \cdots \downarrow(a_{n-1}) A_{n-1}^*$. But finding an actual counter-example word to return when this is not the case still requires enumerating words in I in a brute-force way.

[20] $\forall i_1, i_2 \in I, \exists j \in I, i_1 \leq j \wedge i_2 \leq j$

[21] its complement is upwards-closed

A perhaps more clever solution is thus to consider $\mathcal{L}$ as the intersection $\widehat{\mathcal{L}} \cap A^*$, where $\widehat{\mathcal{L}}$ is the upwards-closed subset $\{I \in \widehat{A^*} \mid I \cap \mathcal{L} \neq \varnothing\}$ of $\widehat{A^*}$ (recall that A^* embeds in $\widehat{A^*}$ through $w \to {\downarrow}w$). $\widehat{A^*}$ inherits a monoid structure from that of A^*, and it is generated by $\Sigma = \widehat{A} \cup \{E^* \mid E \in \mathcal{P}_f(\widehat{A})\} \subseteq \widehat{A^*}$ [10]. It also happens that the subword ordering on this monoid (where Σ is ordered with inclusion) is enough to capture the ordering on A^* when seen as a subset of $\widehat{A^*}$. From all of this one may deduce that a basis for $\mathcal{L}$ may be learned by learning a directed acyclic automaton with input alphabet Σ and recognizing $\widehat{\mathcal{L}}$. This can in turn be done as described above, and the $\textsc{Equiv}_{\mathcal{L}}$ oracle does not require any enumeration to find a counter-example as the ideal $A_0^* {\downarrow}(a_1) A_1^* \cdots {\downarrow}(a_{n-1}) A_{n-1}^*$ itself is the counter-example (this uses the fact that $\widehat{\widehat{A^*}} = \widehat{A^*}$, because $\widehat{X}$ is the ideal completion of X). This comes at the cost of Σ being exponentially bigger than A, hence the `while` loop requiring an exponential overhead. Still, this allows for precisely quantifying the complexity of learning a basis for $\mathcal{L}$ using $\textsc{Eval}_{\widehat{\mathcal{L}}}$: when A is finite, the complexity is exponential in the size of A.

Another advantage of this viewpoint is that it immediately gives new families of WQOs for which the VGJL lemma applies, namely the subword orderings on monoids M arising as quotients free monoids A^* (assuming M verifies basic effectivity properties): their upwards-closed subsets can be seen as upwards-closed subsets of A^* that are also closed under equivalence with respect to M. For instance, this shows that the VJGL lemma applies to subword orderings on free commutative monoids (this was already known as multiset orderings, but the automata learning viewpoint is new), or more generally to subword orderings on any trace monoid (which, to the best of our knowledge, was not known previously). This should be seen mainly as a decidability result, as the automata representation of monoid languages may be very inefficient.

5.3 Summary and future work

We extended the framework of Section 2 to work with **Ord**-enriched functor $\mathcal{I} \to \mathbf{Ord}$. In this setting, the $\textsc{Equiv}_{\mathcal{L}}$ oracle can be implemented using a stronger version of the $\textsc{Eval}_{\mathcal{L}}$ oracle. This shows how automata learning and the VGJL lemma, used to learn a basis of an upwards-closed subset of a WQO, are closely related. The automata learning viewpoint works only for subword orderings, but its implementation is more clever than the simple brute-force enumeration of the VGJL lemma. It also gives a universal proof that subword orderings on monoids verifying certain effectivity conditions, for instance trace monoids, verify the VGJL lemma, a result which was not known previously.

There is further work to be done in two orthogonal directions. First, the VGJL lemma applies not only to subword orderings, but also to other families of WQOs, especially subtree orderings (on sets of trees generated by a signature). There is ongoing work to adapt the results of Section 2 to minimizing and learning tree automata, and the corresponding **Ord**-enriched language (upwards-closed sets for subtree orderings) should give an implementation of the VGJL lemma for subtree orderings. Second, we showed here by hand that the framework of Section 2 extends to **Ord**-enriched categories, but we hope that this is actually an instance of a more general result, namely that it extends to any enriched category. In practice this would then also instanciate the minimization and learning of nominal automata [11], for example.

References

[1] D. Angluin, "Learning regular sets from queries and counterexamples", *Information and Computation*, vol. 75, no. 2, pp. 87–106, Nov. 1987, ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90052-6.

[2] F. Bergadano and S. Varricchio, "Learning behaviors of automata from multiplicity and equivalence queries", in *Algorithms and Complexity*, M. Bonuccelli, P. Crescenzi, and R. Petreschi, Eds., ser. 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.

[3] J. M. Vilar, "Query learning of subsequential transducers", in *Grammatical Interference: Learning Syntax from Sentences*, L. Miclet and C. de la Higuera, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1996, pp. 72–83, ISBN: 978-3-540-70678-6. DOI: 10.1007/BFb0033343.

[4] D. Petrişan and T. Colcombet, "Automata Minimization: A Functorial Approach", *Logical Methods in Computer Science*, vol. Volume 16, Issue 1, Mar. 2020. DOI: 10.23638/LMCS-16(1:32)2020.

[5] T. Colcombet, D. Petrişan, and R. Stabile, "Learning Automata and Transducers: A Categorical Approach", in *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, C. Baier and J. Goubault-Larrecq, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 183, 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.

[6] H. Urbat and L. Schröder, "Automata Learning: An Algebraic Approach", in *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS '20, New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 900–914, ISBN: 978-1-4503-7104-9. DOI: 10.1145/3373718.3394775.

[7] G. van Heerdt, M. Sammartino, and A. Silva, "CALF: Categorical Automata Learning Framework", in *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, V. Goranko and M. Dam, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 82, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 29:1–29:24, ISBN: 978-3-95977-045-3. DOI: 10.4230/LIPIcs.CSL.2017.29.

[8] J. 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 - NAACL '03*, vol. 1, Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 64–71. DOI: 10.3115/1073445.1073454.

[9] G. van Heerdt, C. Kupke, J. Rot, and A. Silva, "Learning Weighted Automata over Principal Ideal Domains", in *Foundations of Software Science and Computation Structures*, J. Goubault-Larrecq and B. König, Eds., vol. 12077, Cham: Springer International Publishing, 2020, pp. 602–621, ISBN: 978-3-030-45230-8. DOI: 10.1007/978-3-030-45231-5_31.

[10] J. Goubault-Larrecq, "On a generalization of a result by Valk and Jantzen", Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-09-09, May 2009.

[11] J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szynwelski, "Learning nominal automata", *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 613–625, Jan. 2017, ISSN: 0362-1340. DOI: 10.1145/3093333.3009879.

[12] S. Mac Lane, *Categories for the Working Mathematician* (Graduate Texts in Mathematics), Second. New York: Springer-Verlag, 1978, ISBN: 978-0-387-98403-2. DOI: 10.1007/978-1-4757-4721-8.

[13] M. P. Schützenberger, "On the definition of a family of automata", *Information and Control*, vol. 4, no. 2, pp. 245–270, Sep. 1961, ISSN: 0019-9958. DOI: 10.1016/S0019-9958(61)80020-X.

[14] C. Choffrut, "Minimizing subsequential transducers: A survey", *Theoretical Computer Science*, Selected Papers in Honor of Jean Berstel, vol. 292, no. 1, pp. 131–143, Jan. 2003, ISSN: 0304-3975. DOI: 10.1016/S0304-3975(01)00219-5.

[15] M.-P. Béal and O. Carton, "Computing the prefix of an automaton", *Informatique Théorique et Applications*, vol. 34, no. 6, p. 503, 2000.

[16] D. Breslauer, "The suffix tree of a tree and minimizing sequential transducers", *Theoretical Computer Science*, vol. 191, no. 1, pp. 131–144, Jan. 1998, ISSN: 0304-3975. DOI: 10.1016/S0304-3975(96)00319-2.

[17] Q. Aristote, *Monoidal transducers minimization*, https://gitlab.math.univ-paris-diderot.fr/aristote/monoidal-transducers-minimization, 2022.

[18] S. Conchon, J.-C. Filliâtre, and J. Signoles, "Designing a generic graph library using ML functors", in *Trends in Functional Programming*, 2008, pp. 124–140.

[19] A. Salomaa and M. Soittola, "Automata-Theoretic Aspects of Formal Power Series", in *Texts and Monographs in Computer Science*, 1978. DOI: 10.1007/978-1-4612-6264-0.

[20] N. Bourbaki, *Commutative Algebra*. Springer Berlin, Heidelberg, 1999, ISBN: 978-3-540-64239-8.

[21] I. Yengui and F. B. Amor, "Saturation of finitely-generated submodules of free modules over Prüfer domains", *Armenian Journal of Mathematics*, vol. 13, no. 1, Mar. 2021, ISSN: 1829-1163.

[22] H. Cohen, "Hermite and Smith normal form algorithms over Dedekind domains", *Mathematics of Computation*, vol. 65, no. 216, pp. 1681–1699, 1996, ISSN: 0025-5718, 1088-6842. DOI: 10.1090/S0025-5718-96-00766-1.

[23] S. Demri, A. Finkel, J. Goubault-Larrecq, S. Schmitz, and P. Schnoebelen, "Well-Quasi-Orders for Algorithms", 2018.

A Additional algorithms

A.1 Categorical algorithms

Algorithm 4 The dual of Algorithm 1, computing Obs. $\mathcal{J}_T$ is now the $(\mathcal{E},\mathcal{M})$-factorization of $\mathcal{A} \to \prod_T \mathcal{L}(\texttt{out})$. When $\mathcal{A}(\texttt{st})$ is $\mathcal{E}$-artinian, the `while` loop is iterated at most $\operatorname{codim}_{\mathcal{E}} \mathcal{A}(\texttt{st})$ times.

Input: a $\mathcal{C}$-automaton $\mathcal{A}$
Output: $\operatorname{Obs}\mathcal{A}$

$T_{todo} = \{\epsilon\}$
$T_{done} = \varnothing$
while there is a $t \in T_{todo}$ **do**
  move t from T_{todo} to T_{done}
  for $a \in A$ such that $\mathcal{J}_{T_{done} \sqcup T_{todo} \sqcup \{at\}} \rightarrowtail \mathcal{J}_{T_{done} \sqcup T_{todo}}$ is not an $\mathcal{E}$-morphism **do**
    add at to T_{todo}
  end for
end while
return the automaton with state-space $\mathcal{J}_{T_{done}}$

Algorithm 5 A second variant of the FunL*-algorithm, dual to Algorithm 3. The morphisms $\mathcal{J}_{Q,T} \rightarrowtail \prod_T \mathcal{L}(\texttt{out})$ are now required to have (computable) left-inverses. When $(\operatorname{Min}\mathcal{L})(\texttt{st})$ is both $\mathcal{M}$-noetherian and $\mathcal{E}$-artinian, it makes at most $\dim_{\mathcal{M}}(\operatorname{Min}\mathcal{L})(\texttt{st})$ updates to Q (including calls to $\textsc{Equiv}_{\mathcal{L}}$) and $\operatorname{codim}_{\mathcal{E}}(\operatorname{Min}\mathcal{L})(\texttt{st})$ updates to T.

Input: $\textsc{Eval}_{\mathcal{L}}$ and $\textsc{Equiv}_{\mathcal{L}}$
Output: an automaton recognizing $\mathcal{L}$

$Q = T = \{\epsilon\}$
loop
  while there is a $at \in AT$ such that $\mathcal{J}_{Q,T \cup AT} \twoheadrightarrow \mathcal{J}_{Q,T}$ is not an $\mathcal{M}$-morphism **do**
    add at to T
  end while
  build an automaton $\mathcal{H}$ with state-space $\prod_T \mathcal{L}(\texttt{out})$
  if $\textsc{Equiv}_{\mathcal{L}}(\mathcal{H})$ outputs some counter-example w **then**
    add w and its prefixes to Q
  else
    return $\mathcal{H}$
  end if
end loop

A.2 Transducer algorithms

Algorithm 6 The FunL*-algorithm applied to monoidal transducers. The table $\mathcal{L}_{Q,T}$ representing partial knowledge of $\mathcal{L}$ over $Q(A\cup\{\epsilon\})T$ is now given by $\mathcal{L}(\triangleright qat\triangleleft) = \Lambda(q,a)R(q,a,t)$ with R right-coprime so that $\Lambda(q,a)$ is a left-gcd of $(\mathcal{L}(\triangleright qat\triangleleft))_{t\in T}$.

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

1: $Q = T = \{e\}$
2: **for** $a \in A\cup\{e\}$ **do**
3: $\quad\Lambda(e,a) = \textsc{Eval}_{\mathcal{L}}(a)$
4: $\quad R(e,a,e) = \epsilon$
5: **end for**
6: **loop**
7: $\quad$**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**
8: $\quad\quad$add qa to Q and update Λ and R using $\textsc{Eval}_{\mathcal{L}}$
9: $\quad$**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 $\textsc{LeftDivide}(\Lambda(q,e), \Lambda(q,a)R(q,a,t)) \neq \chi\textsc{LeftDivide}(\Lambda(q',e), \Lambda(q',a))R(q',a,t))$

$\quad$**then**
10: $\quad\quad$add at to T and update Λ and R using $\textsc{Eval}_{\mathcal{L}}$
11: $\quad$**else**
12: $\quad\quad$build $\mathcal{H}(Q,T)$ using Λ and R
13: $\quad\quad$**if** $\textsc{Equiv}_{\mathcal{L}}(\mathcal{H}_{Q,T}\mathcal{L})$ outputs some counter-example w **then**
14: $\quad\quad\quad$add w and its prefixes to Q
15: $\quad\quad$**else**
16: $\quad\quad\quad$**return** $\mathcal{H}(Q,T)$
17: $\quad\quad$**end if**
18: $\quad$**end if**
19: $\quad$update Λ and R using $\textsc{Eval}_{\mathcal{L}}$
20: **end loop**

Algorithm 7 A fixed-point algorithm for computing $\operatorname{Prefix}\mathcal{A}$. It is obtained by applying Algorithm 4 to monoidal transducers and further optimizing the resulting algorithm.

Input: an M-transducer $\operatorname{Total}\mathcal{A} = (S, (u_0, s_0), t, \odot)$
Output: $\operatorname{Prefix}\mathcal{A}$

$T_{todo} = \varnothing$
$\Lambda = s \mapsto \bot$
for $s \in S$ **do**
  if $t(s) \neq \bot$ **then**
    $\Lambda(s) = t(s)$
    add s to T_{todo}
  end if
end for
 # compute the left-gcds of the languages recognized by each state
while there is some $s \in T_{todo}$ **do**
  for each transition $s' \odot a = (v, s)$ **do**
    remove s from T_{todo}
    if $\Lambda(s') \wedge v\Lambda(s)$ is not equal to $\Lambda(s')$ up to invertibles on the left **then**
      $\Lambda(s') = \Lambda(s') \wedge v\Lambda(s)$
      add s' to T_{todo}
    end if
  end for
end while
 # use Λ *to compute* $\operatorname{Prefix}\mathcal{A}$
$u'_0 = u_0\Lambda(s_0)$
for $s \in S$ **do**
  $t'(s) = \textsc{LeftDivide}(\Lambda(s), t(s))$
  for $a \in A$ **do**
    $s \bigcirc' a = \textsc{LeftDivide}(\Lambda(s), (s \bigcirc a)\Lambda(s \cdot a))$
  end for
end for
return $(S, (u'_0, s_0), t', \odot')$

A.3 Algorithms on quasi-ordered automata

Algorithm 8 The FunL*-algorithm for learning an upwards-closed language

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

$Q = T = \{\epsilon\}$
for $a \in \{\epsilon\} \cup A$ **do**
  $L(\epsilon, a, \epsilon) = \textsc{Eval}_{\mathcal{L}}(a)$
end for
loop
  if there is a $ua \in QA$ such that for all $u' \in Q$, $L(u, a, \cdot) \neq L(u', \epsilon, \cdot)$ **then**
    add ua to Q and update L
  else if there is a $at \in AT$ such that $L(u, \epsilon, \cdot) = L(u', \epsilon, \cdot)$ but $L(u, a, t) \neq L(u', a, t)$ for some $u, u' \in Q$ **then**
    add at to T and update L
  else if there is a $at \in AT$ such that $L(u, \epsilon, \cdot) \sqsubseteq L(u', \epsilon, \cdot)$ but $L(u, a, t) = \top$ and $L(u', a, t) = \bot$ for some $u, u' \in Q$ **then**
    add at to T and update L
  else
    build $\mathcal{H}_{Q,T}\mathcal{L}$ using L
    if $\textsc{Equiv}_{\mathcal{L}}(\mathcal{H}_{Q,T}\mathcal{L})$ outputs some counter-example $w \in A^*$ **then**
      add w and its prefixes to Q
    else
      return $\mathcal{H}_{Q,T}\mathcal{L}$
    end if
  end if
end loop