CALF: Categorical Automata Learning Framework

Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva

University College London

Abstract

Automata learning is a technique that has successfully been applied in verification, with the automaton type varying depending on the application domain. Adaptations of automata learning algorithms for increasingly complex types of automata have to be developed from scratch because there was no abstract theory offering guidelines. This makes it hard to devise such algorithms, and it obscures their correctness proofs. We introduce a simple category-theoretic formalism that provides an appropriately abstract foundation for studying automata learning. Furthermore, our framework establishes formal relations between algorithms for learning, testing, and minimization. We illustrate its generality with two examples: deterministic and weighted automata.

1 Introduction

Automata learning enables the use of model-based verification methods on black-box systems. Learning algorithms have been successfully applied to find bugs in implementations of network protocols [14], to provide defense mechanisms against botnets [12], to rejuvenate legacy software [27], and to learn an automaton describing the errors in a program up to a user-defined abstraction [11]. Many learning algorithms were originally designed to learn a deterministic automaton and later, due to the demands of the verification task at hand, extended to other types of automata. For instance, the popular $\mathtt{L}^\star$ algorithm, devised by Angluin [2], has been adapted for various other types of automata accepting regular languages [8, 3], as well as more expressive automata, including Büchi-style automata [23, 4], register automata [9, 10], and nominal automata [24]. As the complexity of these automata increases, the learning algorithms proposed for them tend to become more obscure. More worryingly, the correctness proofs become involved and harder to verify.

This paper aims to introduce an abstract framework for studying automata learning algorithms in which specific instances, correctness proofs, and optimizations can be derived without much effort. The framework can also be used to study algorithms such as minimization and testing, showing the close connections between the seemingly different problems. These connections also enable the transfer of extensions and optimizations among algorithms.

First steps towards a categorical understanding of active learning appeared in [18]. The abstract definitions provided were based on very concrete data structures used in $\mathtt{L}^\star$, which then restricted potential generalization to study other learning algorithms and capture other data structures used by more efficient variations of the $\mathtt{L}^\star$ algorithm. Moreover, there was no correctness proof, and optimizations and connections to minimization or testing were not explored before in the abstract setting. In the present paper, we develop a rigorous *categorical automata learning framework* (CALF) that overcomes these limitations and can be more widely applied. We start by giving a general overview of the paper and its results.

2 Overview

In this section we provide an overview of CALF. We first establish some global notation, after which we introduce the basic ingredients of active automata learning. Finally, we sketch the structure of CALF and highlight our main results. The reader is assumed to be familiar with elementary category theory and the theory of regular languages.

Notation. The set of words over an alphabet A is written A^*; $A^{\leq n}$ contains all words of length up to $n \in \mathbb{N}$. We denote the empty word by ε and concatenation by $\cdot$ or juxtaposition. We extend concatenation to sets of words $U, V \subseteq A^*$ using $U \cdot V = \{uv \mid u \in U, v \in V\}$. Languages $\mathcal{L} \subseteq A^*$ will often be represented as their characteristic functions $\mathcal{L} \colon A^* \to 2$. Given sets X and Y, we write Y^X for the set of functions $X \to Y$. The set $1 = \{*\}$, where $*$ is an arbitrary symbol, is sometimes used to represent elements $x \in X$ of a set X as functions $x \colon 1 \to X$. We write $|X|$ for the size of a set X.

Active Automata Learning. *Active* automata learning is a widely used technique to learn an automaton model of a system from observations. It is based on direct interaction of the learner with an *oracle* that can answer different types of queries about the system. This is in contrast with *passive* learning, where a fixed set of positive and negative examples is provided and no interaction with the system is possible.

Most active learning algorithms assume the ability to perform *membership queries*, where the oracle is asked whether a certain word belongs to the target language $\mathcal{L}$. The algorithms we focus on use this ability to approximate the construction of the minimal DFA for $\mathcal{L}$. This construction, due to Nerode [26], can be described as follows. The states of the minimal DFA accepting $\mathcal{L}$ over a finite alphabet A are given by the image of the function $t_{\mathcal{L}} \colon A^* \to 2^{A^*}$ defined by $t_{\mathcal{L}}(u)(v) = \mathcal{L}(uv)$, assigning to each word the residual language after reading that word. More precisely, the minimal DFA for $\mathcal{L}$ is given by the following four components:

$$\begin{aligned} M &= \{t_{\mathcal{L}}(u) \mid u \in A^*\} & m_0 &= t_{\mathcal{L}}(\varepsilon) \\ F_M &= \{t_{\mathcal{L}}(u) \mid u \in A^*, \mathcal{L}(u) = 1\} & \delta_M(t_{\mathcal{L}}(u), a) &= t_{\mathcal{L}}(ua), \end{aligned}$$

where M is a finite set of states, $\delta_M \colon M \times A \to M$ is the transition function, $F_M \subseteq M$ is the set of accepting states, and $m_0 \in M$ is the initial state. Note that the transition function is well-defined because of the definition of $t_{\mathcal{L}}$ and M is finite because the language is regular.

Though we know M is finite, computing it as the image of $t_{\mathcal{L}}$ does not terminate, since the domain of $t_{\mathcal{L}}$ is the infinite set A^*. Learning algorithms approximate M by instead constructing two finite sets $S, E \subseteq A^*$, which induce the function $\mathsf{row} \colon S \cup S \cdot A \to 2^E$ defined as $\mathsf{row}(u)(v) = \mathcal{L}(uv)$. The name "row" reflects the usual representation of this function as a table with rows indexed by $S \cup S \cdot A$ and columns indexed by E. This table is called an *observation table* and it induces a *hypothesis* DFA given by

$$\begin{aligned} H &= \{\mathsf{row}(s) \mid s \in S\} & h_0 &= \mathsf{row}(\varepsilon) \\ F_H &= \{\mathsf{row}(s) \mid s \in S, \mathcal{L}(s) = 1\} & \delta_H(\mathsf{row}(s), a) &= \mathsf{row}(sa). \end{aligned}$$

It is often required that ε is an element of both S and E, so that the initial and accepting states are well-defined. For the transition function to be well-defined, the observation table is required to satisfy two properties. These properties were introduced by Gold [15]; we use the names given by Angluin [2].

- **Closedness** states that each transition actually leads to a state of the hypothesis. That is, the table is closed if for all $t \in S \cdot A$ there is an $s \in S$ such that $\mathsf{row}(s) = \mathsf{row}(t)$.
- **Consistency** states that there is no ambiguity in determining the transitions. That is, the table is consistent if for all $s_1, s_2 \in S$ such that $\mathsf{row}(s_1) = \mathsf{row}(s_2)$ we have $\mathsf{row}(s_1 a) = \mathsf{row}(s_2 a)$ for any $a \in A$.

Gold [15] also explained how to update the sets S and E to satisfy these properties. If closedness does not hold, then there exists $sa \in S \cdot A$ such that $\mathsf{row}(s') \neq \mathsf{row}(sa)$ for all $s' \in S$. This is resolved by adding sa to S. If consistency does not hold, then there are $s_1, s_2 \in S$, $a \in A$, and $e \in E$ such that $\mathsf{row}(s_1) = \mathsf{row}(s_2)$, but $\mathsf{row}(s_1 a)(e) \neq \mathsf{row}(s_2 a)(e)$.

In this case we add ae to E to distinguish $\mathsf{row}(s_1)$ from $\mathsf{row}(s_2)$. In both cases, the size of the hypothesis increases. Since the hypothesis cannot be larger than M, the wrapper will eventually be closed and consistent.

The hypothesis is an approximation. Hence, the language it accepts may not be the target language. Gold [15] showed, based on earlier algorithms [16, 5], that for any two chains of languages $S_1 \subseteq S_2 \subseteq \cdots$ and $E_1 \subseteq E_2 \subseteq \cdots$, both in the limit equaling A^*, the following holds. There exists an $i \in \mathbb{N}$ such that all the hypothesis automata derived from S_j, E_j $(j \geq i)$ are isomorphic to M. These hypotheses are built after enforcing closedness and consistency for the sets S_j and E_j. Unfortunately, the convergence point i is not known to the learner.

Termination Conditions—Several Algorithms. To obtain a terminating algorithm, an additional assumption is necessary [25]. Arbib and Zeiger [5] explained the algorithm of Ho [16], which assumes an upper bound n on the size of M. One then simply takes $S, E = A^{\leq n-1}$ and directly obtains a DFA isomorphic to M as the hypothesis.

Angluin [1] showed that, for algorithms with just this extra assumption, the number of membership queries has a lower bound that is exponential in the size of M. She then introduced an algorithm called `ID` that makes a stronger assumption. It assumes a finite set S of words such that each state of M (that does not accept the empty language) is reached by reading a word in S. Initializing $E = \{\varepsilon\}$, the algorithm obtains M up to isomorphism after making the table consistent (for a slightly stronger definition of consistency related to the sink state being kept implicit). This makes `ID` polynomial in the sizes of M, S, and A.

Subsequently, Angluin introduced the algorithm `L*` [2], which makes yet another assumption: it assumes an oracle that can test any DFA for equivalence with the target language. In the case of a discrepancy, the oracle will provide a *counterexample*, which is a word in the symmetric difference of the two languages. `L*` adds the given counterexample and its prefixes to S. One can show that this means that the next hypothesis will classify the counterexample correctly and that therefore the size of the hypothesis must have increased. The algorithm is polynomial in the sizes of M and A and in the length of the longest counterexample.

CALF. The framework we introduce in this paper aims to provide a formal and general account of the concepts introduced above, based on category theory. Although the focus is on automata learning, the general perspective brought by CALF allows us to unveil deep connections between learning and algorithms for minimization and testing.

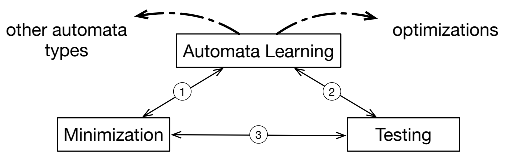

Figure 1 *Overview of CALF.*

The structure of CALF is depicted in Figure 1. After introducing the basic elements of the framework in Section 3, in Section 4 we use the framework to give a new categorical correctness proof, encompassing all the algorithms mentioned above. Then we explore connections with minimization and testing, following the edges of the triangle in Figure 1:

① Section 5 shows how our correctness proof also applies to minimization algorithms. In particular, we connect consistency fixing to the steps in the classical partition refinement algorithm which merge observationally equivalent states. Dually, reachability analysis, which is also needed to get the minimal automaton, is related to fixing closedness defects.

② Section 6 extends the correctness proof to an abstract result about testing and we show how it applies to algorithms such as the W-method [13], which enable testing against a black-box system.

③ From the abstract framework and the observations in ① and ②, we also can see how certain basic sets used in testing algorithms like the W-method can be obtained using generalized minimization algorithms.

This triangular structure is on its own a contribution of the paper, tying together three important techniques that play a role in automata learning and verification. It is our long-term goal to exploit the practical aspects of the framework. For more details, see our project website[1]. One of CALF's strong points is the ability to seamlessly accommodate variants of automata learning algorithms:

- algorithms for *other types of automata*, by varying the category on which automata are defined. This includes weighted automata, which are defined in the category of vector spaces. We present details of this example in Appendix B.
- algorithms with *optimizations*, by varying the main data structure. Section 7 discusses *classification trees* [19], which optimize the observation tables used in classical learning algorithms.

CALF also provides guidelines on how to combine these variants, and on how to obtain corresponding minimization/testing algorithms via the connections described above. This can lead to more efficient algorithms and, because of our abstract approach, correctness proofs can be re-used.

Some proofs have been omitted for space reasons. They can be found in Appendix A.

3 Abstract Learning

In this section, we develop CALF by starting with categorical definitions that capture the basic data structures and definitions used in learning.

3.1 Observation Tables as Approximated Algebras

In the automata learning algorithms described in Section 2, observation tables are used to approximate the minimal automaton M for the target language $\mathcal{L}$. We start by introducing a notion of approximation of an object in a category, called *wrapper*. We work in a fixed category $\mathbf{C}$ unless otherwise indicated.

▶ **Definition 1** (Wrapper)**.** A *wrapper for an object* T is a pair of morphisms $w = (S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$; T is called the *target* of w.

▶ **Example 2.** The algorithms explained in Section 2 work by producing subsequent approximations of M. In each approximation, we have two finite sets of words $S, E \subseteq A^*$, yielding a wrapper $w = (S \xrightarrow{\sigma} M, M \xrightarrow{\pi} 2^E)$. Here σ maps $s \in S$ to the state reached by M after reading s, and π assigns to each state of M the language accepted by that state, but restricted to the words in E. Although M is unknown, the composition of π and σ is given by $(\pi \circ \sigma)(s)(e) = \mathcal{L}(se)$, which can be determined using membership queries. This composition is precisely the component $S \to 2^E$ of the observation table $\mathsf{row}\colon S \cup S \cdot A \to 2^E$.

We claim that the other component $S \cdot A \to 2^E$ of row is an approximated version of the transition function $\delta_M\colon M \times A \to M$, and that it can be obtained via the wrapper w. More generally, we can abstract away from the structure of automata and talk about approximations of algebras for a functor F, i.e., of pairs $(T, f\colon FT \to T)$. Notice that (M, δ_M) is an algebra for $(-) \times A$.

▶ **Definition 3** (Approximation of Algebras Along a Wrapper)**.** Given an F-algebra (T, f) and a wrapper $w = (S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$, the *approximation of f along w* is $\xi_f^w = FS \xrightarrow{F\sigma} FT \xrightarrow{f}$

[1] http://calf-project.org

$T \xrightarrow{\pi} P$. We write ξ^w for $\xi^w_{\mathsf{id}} = \pi \circ \sigma$, and we may leave out superscripts if the relevant wrapper w is clear from the context.

We remark that our theory does not depend on a specific choice of wrappers and approximations along them. Observation tables are just an instance. Other data structures representing approximations of automata can also be captured, as will be shown in Section 7.

Although the algebra f may be unknown—as for instance $f = \delta_M$ is unknown in automata learning algorithms—the approximated version ξ_f can sometimes be recovered via the following simple result, stating that approximations transfer along algebra homomorphisms.

▶ **Proposition 4.** *For all endofunctors F, F-algebras (U, u) and (V, v), F-algebra homomorphisms $h \colon (U, u) \to (V, v)$, and morphisms $\alpha \colon S \to U$ and $\beta \colon V \to P$, $\xi_v^{(h \circ \alpha, \beta)} = \xi_u^{(\alpha, \beta \circ h)}$.*

▶ **Example 5.** Consider again the wrapper in Example 2. There $\sigma = r \circ \alpha$, where $\alpha \colon S \to A^*$ is the inclusion of S into A^* and $r \colon A^* \to M$ is the full reachability map of M. If we equip A^* with the transition structure $c \colon A^* \times A \to A^*$, defined as $c(u, a) = ua$, then r is an algebra homomorphism $(A^*, c) \to (M, \delta)$ for $(-) \times A$. We can therefore apply Proposition 4 and obtain $\xi_\delta^{(r \circ \alpha, \pi)} = \xi_c^{(\alpha, \pi \circ r)}$. It follows that $\xi_\delta \colon S \times A \to 2^E$ is given by $\xi_\delta(s, a)(e) = \mathcal{L}(sae)$, which corresponds to the $S \cdot A \to 2^E$ part of the row function from Section 2.

3.2 Hypotheses as Factorizations

We now formalize the process of deriving a hypothesis automaton from an observation table. Recall from Section 2 that the state space of the hypothesis automaton is precisely the image of the component $S \to 2^E$ of the row function. Image factorizations are abstractly captured by the notion of $(\mathcal{E}, \mathcal{M})$ *factorization system* on a category. We will assume throughout the paper that our category $\mathbf{C}$ has $(\mathcal{E}, \mathcal{M})$ factorizations.

▶ **Definition 6** (Factorization System). A *factorization system* is a pair $(\mathcal{E}, \mathcal{M})$ of sets of morphisms such that

$$\begin{tikzcd} U \arrow[r, two heads, "i"] \arrow[d, "g"'] & V \arrow[d, "h"] \arrow[dl, dashed, "d"'] \\ W \arrow[r, tail, "j"'] & X \end{tikzcd}$$

1. all morphisms f can be decomposed as $f = m \circ e$, with $m \in \mathcal{M}$ and $e \in \mathcal{E}$;
2. for all commutative squares as on the right, where $i \in \mathcal{E}$ and $j \in \mathcal{M}$, there is a unique diagonal d making the triangles commute;
3. both $\mathcal{E}$ and $\mathcal{M}$ are both closed under composition and contain all isomorphisms;
4. every morphism in $\mathcal{E}$ is an epi, and every morphism in $\mathcal{M}$ is a mono.

We will use double-headed arrows ($\twoheadrightarrow$) to indicate morphisms in $\mathcal{E}$ and arrows with a tail ($\rightarrowtail$) to indicate morphisms in $\mathcal{M}$. For instance, in **Set** the pair (surjective functions, injective functions) forms a factorization system, where each function $f \colon X \to Y$ can be decomposed as a surjective map $X \twoheadrightarrow \mathsf{img}(f)$ followed by the inclusion $\mathsf{img}(f) \rightarrowtail Y$.

▶ **Definition 7** (Wrapper Minimization). The *minimization of a wrapper* (σ, π) is the $(\mathcal{E}, \mathcal{M})$-factorization of $\pi \circ \sigma$, depicted on the right. Notice that (e, m) is a wrapper for H.

$$\begin{tikzcd} S \arrow[r, "\sigma"] \arrow[dr, two heads, bend right, "e"'] & T \arrow[r, "\pi"] & P \\ & H \arrow[ur, tail, bend right, "m"'] & \end{tikzcd}$$

In the wrapper of Example 2, H is the state space of the hypothesis automaton defined in Section 2. However, H does not yet have an automaton structure. In the concrete setting, this can be computed from the $S \cdot A \to 2^E$ part of row whenever closedness and consistency are satisfied. We give our abstract characterization of these requirements, which generalize the definitions due to Jacobs and Silva [18].

▶ **Definition 8** (Closedness and Consistency). Let $w = (S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$ be a wrapper and $(S \xrightarrow{e} H, H \xrightarrow{m} P)$ its minimization. Given an endofunctor F and a morphism $f\colon FT \to T$, we say that w is *f-closed* if there exists a morphism $\mathsf{close}^w_f\colon FS \to H$ making the left triangle commute; we say that w is *f-consistent* if there exists a morphism $\mathsf{cons}^w_f\colon FH \to P$ making the right triangle commute.

$$\begin{tikzcd} FS \arrow[r, "Fe"] \arrow[d, dashed, "\mathsf{close}^w_f"'] \arrow[dr, "\xi^w_f"] & FH \arrow[d, dashed, "\mathsf{cons}^w_f"] \\ H \arrow[r, "m"'] & P \end{tikzcd} \tag{1}$$

Closedness and consistency together yield an algebra structure on the hypothesis. This is not just any algebra structure, but in fact one that relates the wrapper to its minimization, as we show next.

▶ **Theorem 9.** *Let $w = (S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$ be a wrapper and $(S \xrightarrow{e} H, H \xrightarrow{m} P)$ its minimization. For each $\mathcal{E}$-preserving endofunctor F and each morphism $f\colon FT \to T$, w is both f-closed and f-consistent if and only if there exists $\theta^w_f\colon FH \to H$ making the diagram on the right commute.*

$$\begin{tikzcd} FT \arrow[d, "f"'] & FS \arrow[l, "F\sigma"'] \arrow[r, two heads, "Fe"] & FH \arrow[d, dashed, "\theta_f"] \\ T \arrow[r, "\pi"] & P & H \arrow[l, tail, "m"'] \end{tikzcd} \tag{2}$$

Proof. Given the morphisms close_f and cons_f, we let θ_f be the unique diagonal provided by the factorization system for the commutative square (1); conversely, given θ_f, we take $\mathsf{close}_f = \theta_f \circ Fe$ and $\mathsf{cons}_f = m \circ \theta_f$. ◀

To understand abstract closedness and consistency properties in **Set**, and to relate them with the concrete definitions given in Section 2, we give the following general result.

▶ **Proposition 10.** *Consider functions f, g, and h as in the diagram below, with f surjective.*

1. *A function $i\colon U \to W$ making the left triangle in the diagram commute exists if and only if for each $u \in U$ there is a $w \in W$ such that $h(w) = g(u)$.*
2. *A function $j\colon V \to X$ making the right triangle commute exists if and only if for all $u_1, u_2 \in U$ such that $f(u_1) = f(u_2)$ we have $g(u_1) = g(u_2)$.*

$$\begin{tikzcd} U \arrow[r, two heads, "f"] \arrow[d, dashed, "i"'] \arrow[dr, "g"] & V \arrow[d, dashed, "j"] \\ W \arrow[r, "h"'] & X \end{tikzcd}$$

For the wrapper in Example 2 and $F = (-) \times A$, δ_M-closedness and δ_M-consistency coincide with closedness and consistency as defined in Section 2. If these are satisfied, then the function θ_{δ_M} is precisely the transition function δ_H defined there. Notice that all endofunctors on **Set** preserve surjections.

Interestingly, closedness and consistency also tell us when initial and accepting states of H can be derived from the observation table. For initial states, we note that the initial state $m_0 \in M$ can be seen as a function $m_0\colon 1 \to M$. As the set 1 can be seen as the constant functor 1 applied to M, this initial state gives rise to another closedness property, which states that there must be an $s \in S$ such that $\xi(s)(e) = \mathcal{L}(e)$ for all $e \in E$. Note that m_0-consistency trivially holds because of the constant functor involved. When m_0-closedness is satisfied, $\mathsf{close}_{m_0}\colon 1 \to H$ is an initial state map. For instance, in $\mathtt{L}^\star$ this property is always satisfied because $\varepsilon \in S$.

For accepting states, one would expect a similar property regarding the set $F_M \subseteq M$, which can be represented by a function $F_M\colon M \to 2$. However, this is a coalgebra (for the constant functor 2) rather than an algebra. Fortunately, a wrapper $(S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$ in $\mathbf{C}$ gives a wrapper (π, σ) in $\mathbf{C^{op}}$, so a morphism $f\colon T \to FT$ in $\mathbf{C}$ yields the approximation $\xi^{(\pi,\sigma)}_f = S \xrightarrow{\sigma} T \xrightarrow{f} FT \xrightarrow{F\pi} FP$. In particular, for $F_M\colon M \to 2$, this leads to a consistency[2]

[2] Technically, this should be called coclosedness, as it is closedness in the category $\mathbf{C^{op}}$. We choose to overload consistency so as not to obscure the terminology.

property stating that for all $s_1, s_2 \in S$ such that $\xi(s_1) = \xi(s_2)$ we must have $\mathcal{L}(s_1) = \mathcal{L}(s_2)$. The $\mathtt{L}^\star$ algorithm ensures this by having $\varepsilon \in E$, using that $\mathcal{L}(s) = \xi(s)(\varepsilon)$ for any $s \in S$.

4 A General Correctness Theorem

In this section we work towards a general correctness theorem. We then show how it applies to the ID algorithm, to the algorithm by Arbib and Zeiger and to $\mathtt{L}^\star$.

The key observation for the correctness theorem is the following. Let $w = (S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$ be a wrapper with minimization $(S \xrightarrow{e} H, H \xrightarrow{m} P)$. If $\sigma \in \mathcal{E}$, then the factorization system gives us a unique diagonal ϕ in the left square of (3), which by (the dual of) Proposition A.1 satisfies $\phi \in \mathcal{E}$. Similarly, if $\pi \in \mathcal{M}$, we have ψ in the right square of (3), with $\psi \in \mathcal{M}$. Composing the two diagrams and using again the diagonal property, one sees that ϕ and ψ must be mutually inverse. We can conclude that H and T are isomorphic. Now, if w is a wrapper produced by a learning algorithm and T is the state space of the target minimal automaton, as in Example 2, our reasoning hints at a correctness criterion: $\sigma \in \mathcal{E}$ and $\pi \in \mathcal{M}$ upon termination ensure that H is (isomorphic to) T. Of course, the criterion will have to guarantee that the automata, not just the state spaces, are isomorphic.

$$\begin{tikzcd}
S \arrow[r, two heads, "\sigma"] \arrow[d, two heads, "e"'] & T \arrow[d, "\pi"] \arrow[dl, dashed, "\phi"'] \\
H \arrow[r, tail, "m"'] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, "\sigma"'] & H \arrow[d, tail, "m"] \arrow[dl, dashed, "\psi"'] \\
T \arrow[r, tail, "\pi"'] & P
\end{tikzcd}
\qquad (3)$$

We first show that the argument above lifts to F-algebras $f\colon FT \to T$, for an arbitrary endofunctor $F\colon \mathbf{C} \to \mathbf{C}$ preserving $\mathcal{E}$.

▶ **Lemma 11.** *For a wrapper $w = (\sigma, \pi)$ and an F-algebra f, if $\sigma \in \mathcal{E}$, then w is f-closed; if $\pi \in \mathcal{M}$, then w is f-consistent.*

Proof. If $\sigma \in \mathcal{E}$, then let ϕ be as in (3) and define $\mathsf{close}_f = \phi \circ f \circ F\sigma$; if $\pi \in \mathcal{M}$, then let ψ be as in (3) and define $\mathsf{cons}_f = \pi \circ f \circ F\psi$. ◀

▶ **Proposition 12.** *For a wrapper $w = (\sigma, \pi)$ and an F-algebra f, if $\sigma \in \mathcal{E}$ and w is f-consistent, then ϕ as given in (3) is an F-algebra homomorphism $(T, f) \to (H, \theta_f)$; if $\pi \in \mathcal{M}$ and w is f-closed, then ψ as given in (3) is an F-algebra homomorphism $(H, \theta_f) \to (T, f)$.*

Proof. Assume that $\sigma \in \mathcal{E}$ and w is f-consistent. The proof for the other part is analogous. Using Lemma 11 we see that θ_f indeed exists. Because $F\sigma$ is epic and m monic, it suffices to show $m \circ \phi \circ f \circ F\sigma = m \circ \theta_f \circ F\phi \circ F\sigma$, which is done on the right. (The definition of θ_f can be found in the proof of Proposition 9.) ◀

$$\begin{tikzcd}
FT \arrow[r, "f"] \arrow[dr, phantom, "①"] & T \arrow[r, "\phi"] \arrow[dr, "\pi"'] \arrow[r, phantom, shift right=3, "②"] & H \arrow[d, tail, "m"] \\
FS \arrow[u, two heads, "F\sigma"] \arrow[rr, "\xi_f"] \arrow[d, "F\sigma"'] \arrow[dr, "Fe"] \arrow[drr, bend left, "\mathsf{close}_f"] & & P \\
FT \arrow[r, "F\phi"'] \arrow[u, phantom, shift left=4, "③"] & FH \arrow[r, "\theta_f"'] \arrow[u, phantom, "④"] & H \arrow[u, tail, "m"'] \arrow[ul, phantom, "⑤"]
\end{tikzcd}$$

① definition of ξ_f
② (3)
③ functoriality, (3)
④ definition of θ_f
⑤ closedness

▶ **Corollary 13.** *If $\sigma \in \mathcal{E}$ and $\pi \in \mathcal{M}$, then ϕ as given in (3) is an F-algebra isomorphism $(T, f) \to (H, \theta_f)$.*

Now we enrich F-algebras with initial and final states, obtaining a notion of *automaton* in a category. Then we give the full correctness theorem for automata. We fix objects I and Y in $\mathbf{C}$, which will serve as initial state selector and output of the automaton, respectively.

▶ **Definition 14** (Automaton). An *automaton in* $\mathbf{C}$ is an object Q of $\mathbf{C}$ equipped with an *initial state map* $\mathsf{init}_Q\colon I \to Q$, an *output map* $\mathsf{out}_Q\colon Q \to Y$, and *dynamics* $\delta_Q\colon FQ \to Q$. An *input system* is an automaton without an output map; an *output system* is an automaton without an initial state map.

$$\begin{tikzcd}
& FQ \arrow[d, "\delta_Q"] & \\
& Q \arrow[dr, "\mathsf{out}_Q"] & \\
I \arrow[ur, "\mathsf{init}_Q"] & & Y
\end{tikzcd}$$

Automata form a category, where morphisms f between automata U and V are F-algebra homomorphisms that commute with initial state and output maps: $f \circ \mathsf{init}_U = \mathsf{init}_V$ and $\mathsf{out}_V \circ f = \mathsf{out}_U$. Composition and identities are as in $\mathbf{C}$. There are analogous categories of input and output systems.

To recover DAs (DFAs that may not be finite) over an alphabet A as automata, we take $I = 1$, $Y = 2$, and $F = (-) \times A$ in the category $\mathbf{C} = \mathbf{Set}$. An input system is then a DA without a classification of states into accepting and rejecting states; an output system is a DA without a designated initial state.

We assume the existence of an initial input system $\mathcal{A}$ and a final output system $\mathcal{Z}$. These give general notions of *reachability* and *observability*.

▶ **Definition 15** (Reachability and Observability). The *reachability map* of an automaton Q is the unique input system homomorphism $r_Q \colon \mathcal{A} \to Q$; its *observability map* is the unique output system homomorphism $o_Q \colon Q \to \mathcal{Z}$. The automaton Q is *reachable* if $r_Q \in \mathcal{E}$; it is *observable* if $o_Q \in \mathcal{M}$. An automaton is *minimal* if it is both reachable and observable.

$$\begin{tikzcd}
F\mathcal{A} \arrow[r, dashed, "Fr_Q"] \arrow[d, "\delta_{\mathcal{A}}"'] & FQ \arrow[r, dashed, "Fo_Q"] \arrow[d, "\delta_Q"] & F\mathcal{Z} \arrow[d, "\delta_{\mathcal{Z}}"] \\
\mathcal{A} \arrow[r, dashed, "r_Q"] & Q \arrow[r, dashed, "o_Q"] \arrow[dr, "\mathsf{out}_Q"'] & \mathcal{Z} \arrow[d, "\mathsf{out}_{\mathcal{Z}}"] \\
I \arrow[u, "\mathsf{init}_{\mathcal{A}}"] \arrow[ur, "\mathsf{init}_Q"'] & & Y
\end{tikzcd}$$

In the DA setting, the set of words A^* forms an initial input system. Its initial state is the empty word $\varepsilon \colon 1 \to A^*$, and its transition function $c \colon A^* \times A \to A^*$ is given by concatenation: $c(u, a) = ua$. This yields the expected definition of the reachability map $r_Q \colon A^* \to Q$ for a DA Q: $r_Q(\varepsilon) = \mathsf{init}_Q(*)$ and $r_Q(ua) = \delta_Q(r_Q(u), a)$. The set of languages 2^{A^*} in this setting forms a final output system. Its accepting states $\varepsilon? \colon 2^{A^*} \to 2$ are those languages that contain the empty word, namely $\varepsilon?(L) = L(\varepsilon)$, for any $L \subseteq A^*$, and its dynamics $\partial \colon 2^{A^*} \times A \to 2^{A^*}$ is given by $\partial(L, a)(v) = L(av)$. The observability map of a DA Q assigns to each state the language it accepts, explicitly: $o_Q(q)(\varepsilon) = \mathsf{out}_Q(q)$ and $o_Q(q)(av) = o_Q(\delta_Q(q, a))(v)$.

In general, we define the *language* of an automaton Q to be $\mathcal{L}_Q = \mathsf{out}_Q \circ r_Q \colon \mathcal{A} \to Y$. For DFAs, this is the usual definition of the accepted language $A^* \to 2$ (alternatively, $\mathcal{L}_Q = o_Q \circ \mathsf{init}_Q \colon 1 \to 2^{A^*}$, which explicitly mentions the initial state). If for automata U and V there exists an automaton homomorphism $U \to V$, one can prove that $\mathcal{L}_U = \mathcal{L}_V$.

We are now ready to give the main result of this section, which extends Proposition 12. Intuitively, it provides conditions for when the hypothesis automaton, used as approximation, is actually the target automaton. Remarkably, unlike Proposition 12, it is enough to require either $\sigma \in \mathcal{E}$ or $\pi \in \mathcal{M}$, thanks to observability and reachability.

▶ **Theorem 16.** *Let $w = (S \xrightarrow{\sigma} Q, Q \xrightarrow{\pi} P)$ be a wrapper for an automaton Q and let H be the hypothesis automaton for w. If at least one the following is true:*

1. *Q is observable and w is an out_Q-consistent and δ_Q-consistent wrapper such that $\sigma \in \mathcal{E}$;*

2. *Q is reachable and w is an init_Q-closed and δ_Q-closed wrapper such that $\pi \in \mathcal{M}$;*

then H and Q are isomorphic automata.

Proof. We only show point 1; the other is analogous. Recall that the initial state map of the automaton is an algebra for the constant functor I while the output map is a coalgebra for the constant functor Y. Hence, we can apply Proposition 12 (or its dual for the coalgebra) to them and to δ_Q to find that $\phi \colon Q \to H$ is an automaton homomorphism. Then $o_Q = o_H \circ \phi$ by finality of $\mathcal{Z}$. Since o_Q is in $\mathcal{M}$, this means that $\phi \in \mathcal{M}$ (see Proposition A.1). Because $m \in \mathcal{M}$ and $\pi = m \circ \phi$ (3), we have $\pi \in \mathcal{M}$. Therefore, we can apply Corollary 13, again three times, and obtain an automaton isomorphism between Q and H. ◀

4.1 Applications

Recall that ID assumes a finite set $S \subseteq A^*$ such that each state of the minimal target DFA M is reached by reading one of the words in S when starting from m_0. In the terminology of the previous section: the algorithm takes a wrapper $(S \xrightarrow{\sigma} M, M \xrightarrow{\pi} 2^E)$ as in Example 2, with E initialized to $\{\varepsilon\}$, makes it δ-consistent, and directly obtains the DFA M as the hypothesis.

The correctness of the algorithm can be explained via Theorem 16(1) as follows. The assumption of ID about S is equivalent to $\sigma \in \mathcal{E}$, because σ is defined to be the reachability map restricted to S. The out_Q-consistency condition is satisfied by initializing $E = \{\varepsilon\}$, and ensuring δ_Q-consistency is precisely what the algorithm does. Therefore, by Theorem 16(1), the final wrapper yields a hypothesis automaton isomorphic to M.

Theorem 16(2) suggests a dual to this algorithm, where we assume a finite set $E \subseteq A^*$ such that every pair of different states of M is distinguished by some word in E. Enforcing closedness will lead to a hypothesis automaton isomorphic to M.

We can also explain the Arbib and Zeiger algorithm. In a minimal DFA M with at most n states, every state is reached via a word of length at most $n-1$; similarly, every pair of states is distinguished by a word of length up to $n-1$. The algorithm of Arbib and Zeiger takes a wrapper $(S \xrightarrow{\sigma} M, M \xrightarrow{\pi} 2^E)$ as in Example 2, with $S, E = A^{\leq n-1}$, and by Corollary 13 (applied once for each of init_M, δ_M, and out_M) immediately obtains M up to isomorphism as the hypothesis. We note that taking this large E is unnecessary, as we could simply apply Theorem 16(1), thus reducing the algorithm to ID.

Finally, we consider the L⋆ algorithm. Let the wrapper $(S \xrightarrow{\alpha} A^* \xrightarrow{r_M} M, M \xrightarrow{\pi} 2^E)$ be as in Example 2, and let H be its hypothesis automaton. We have the following result.

▶ **Proposition 17.** *If for every prefix p of a word $z \in A^*$ there exists an $s \in S$ such that $r_M(s) = r_M(p)$, then $\mathcal{L}_H(z) = \mathcal{L}_M(z)$.*

▶ **Corollary 18.** *If $z \in A^*$ is a counterexample, i.e., $\mathcal{L}_H(z) \neq \mathcal{L}_M(z)$, then adding all prefixes of z to S will increase the size of* $\mathsf{img}(r_M \circ \alpha)$.

Thus, after L⋆ has processed at most $|M|$ counterexamples, the conditions for Theorem 16(1) are satisfied, which means that the next hypothesis will be isomorphic to M.

5 Minimization

In this section we explore the connection ① in Figure 1 between automata learning and minimization algorithms. Recall that minimization algorithms typically have two phases: a reachability analysis phase, which removes unreachable states; and a merging phase, where language-equivalent states are merged. We will rephrase these two phases in the terminology of Section 2, and we will show that Theorem 16 can be used to explain their correctness. We fix a DFA Q throughout the section.

Reachability Analysis Phase. Let R be the reachable part of Q, and S any subset of R. There is a wrapper $w = (S \xrightarrow{\sigma} R, R \xrightarrow{\pi} Q)$, where σ and π are just the inclusions. The set S models the state of an algorithm performing a reachability analysis on Q. Since σ and π are inclusions, the hypothesis for w is the set S itself.

As the inclusion π is an automaton homomorphism, by Proposition 4 we can use the transition function $\delta\colon Q \times A \to Q$ and initial state $q_0\colon 1 \to Q$ to compute

$$\xi^{(\sigma,\pi)}_{\delta\colon R\times A\to R} = \xi^{(\pi\circ\sigma,\mathsf{id})}_{\delta\colon Q\times A\to Q}\colon S \times A \to Q \qquad\qquad \xi^{(\sigma,\pi)}_{r_0\colon 1\to R} = \xi^{(\pi\circ\sigma,\mathsf{id})}_{q_0\colon 1\to Q} = q_0\colon 1 \to Q.$$

The function ξ_δ simply assigns to each state $s \in S$ and each symbol $a \in A$ the state $\delta_Q(s, a)$. The wrapper is therefore δ_R-closed if for all $q \in S$ and $a \in A$ we have $\delta_Q(q, a) \in S$; it is r_0-closed if $q_0 \in S$. The obvious algorithm to ensure these closedness properties, namely initializing $S = \{q_0\}$ and adding $\delta_Q(q, a)$ to S while there are $q \in S$ and $a \in A$ such that $\delta_Q(q, a) \notin S$, is the usual reachability analysis algorithm. Since R is reachable and π injective, Theorem 16(2) confirms that this algorithm finds an automaton isomorphic to R.

Alternatively, one could let S be a subset of A^*, and define σ as the inclusion $S \to A^*$. The wrapper then would be $(S \xrightarrow{\sigma} A^*, A^* \xrightarrow{r} Q)$, where r is the reachability map for Q (see Definition 15). Starting from $S = \{\varepsilon\}$, the algorithm would add to S words reaching states not yet visited. This is closer to how automata learning algorithms fix closedness.

State Merging Phase. Now we are interested in finding the DFA O that is obtained from Q by merging states that accept the same language. Formally, this automaton is obtained by factorizing the observability map o_Q for Q (see Definition 15) as DA homomorphisms $Q \overset{h}{\twoheadrightarrow} O \overset{o}{\rightarrowtail} 2^{A^*}$. Given a finite set $E \subseteq A^*$, these can be made into a wrapper $w = (Q \xrightarrow{h} O, O \xrightarrow{o} 2^{A^*} \xrightarrow{\omega} 2^E)$, where ω restricts a language to the words in E. Consider $\xi^w \colon Q \to 2^E$ (i.e., the composition of the morphisms in w): even though O is not known a priori, this function can be computed by testing for a given state which words in E it accepts. Because h is an automaton homomorphism, $\xi^w_{\delta_O}$ by Proposition 4 equals $\xi^w_{\delta_Q}$, which is just $\xi^w \circ \delta_Q$. Since h is surjective, Theorem 16(1) says that we only have to ensure δ-consistency and out-consistency. One may start with $E = \{\varepsilon\}$ to satisfy the latter. For δ-consistency, for all $q_1, q_2 \in Q$ such that $\xi(q_1) = \xi(q_2)$ we must have $\xi(\delta(q_1, a)) = \xi(\delta(q_2, a))$ for each $a \in A$. This can be ensured in the same fashion as for an observation table.

The algorithm we have just described reminds of Moore's reduction procedure [25]. However, using a table as data structure is less efficient because many cells may be redundant. The same redundancy has been observed in the $\mathtt{L}^\star$ algorithm. In Section 7 we will show how CALF covers optimized versions of these algorithms, but first we discuss yet another application of our framework.

6 Conformance Testing

In this section we consider the connections ② and ③ in Figure 1 between testing and, respectively, automata learning and minimization. More precisely, we consider an instance of *conformance testing* where a known DFA U is tested for equivalence against a black-box DFA V. One application of this problem is the realization of equivalence queries in the $\mathtt{L}^\star$ algorithm, where U is the hypothesis DFA and V is the target.

Testing consists in comparing U and V on a finite set of words, approximating their behavior. Following the idea of using wrappers to generalize approximations, we now capture testing using our abstract learning machinery. This will allows us to explain correctness of testing using Theorem 16. First note that given objects S and P and morphisms $\alpha \colon S \to \mathcal{A}$ and $\omega \colon \mathcal{Z} \to P$, we can associate wrappers to both the known automaton U and the black box V by composing with their reachability and observability maps:

$$w_U = (\sigma_U, \pi_U) \qquad \sigma_U = S \xrightarrow{\alpha} \mathcal{A} \xrightarrow{r_U} U \qquad \pi_U = U \xrightarrow{o_U} \mathcal{Z} \xrightarrow{\omega} P$$
$$w_V = (\sigma_V, \pi_V) \qquad \sigma_V = S \xrightarrow{\alpha} \mathcal{A} \xrightarrow{r_V} V \qquad \pi_V = V \xrightarrow{o_V} \mathcal{Z} \xrightarrow{\omega} P.$$

We now use several approximations in defining the *tests of* U *against* V, covering transition functions, initial states, and final states:

$$\xi^{w_U} = \xi^{w_V} \qquad \xi^{w_U}_{\mathsf{init}_U} = \xi^{w_V}_{\mathsf{init}_V} \qquad \xi^{w_U}_{\delta_U} = \xi^{w_V}_{\delta_V} \qquad \xi^{w_U}_{\mathsf{out}_U} = \xi^{w_V}_{\mathsf{out}_V}. \tag{4}$$

▶ **Example 19.** If U and V are DFAs, we can choose wrappers as in Example 2, namely w_U of the form $(S \xrightarrow{\sigma_U} U, U \xrightarrow{\pi_U} 2^E)$. The approximations along w_U then become $\xi^{w_U}(s)(e) = \mathcal{L}_U(se)$, $\xi^{w_U}_{\mathsf{init}_U}(e) = \mathcal{L}_U(e)$, $\xi^{w_U}_{\delta_U}(s,a)(e) = \mathcal{L}_U(sae)$, and $\xi^{w_U}_{\mathsf{out}_U}(s) = \mathcal{L}_U(s)$, and analogously for w_V. Thus, checking (4) amounts to checking whether the DFAs accept exactly the same words from the set $S \cdot E \cup S \cup S \cdot A \cdot E \cup E$.

Our main result regarding conformance testing captures the properties of the wrappers that need to hold for the above tests to prove that U is equivalent to the black-box V.

▶ **Theorem 20.** *Given the above wrappers, suppose $\sigma_U \in \mathcal{E}$, $\pi_U \in \mathcal{M}$, and either $\sigma_V \in \mathcal{E}$ and V is observable or $\pi_V \in \mathcal{M}$ and V is reachable. Then U is isomorphic to V if and only if all the equalities (4) hold.*

Proof. Assume first that U is isomorphic to V as witnessed by an isomorphism $\phi\colon U \to V$. By initiality, $\phi \circ r_U = r_V$; by finality, $o_V \circ \phi = o_U$. Then $o_U \circ r_U = o_V \circ \phi \circ r_U = o_V \circ r_V$. From this equality the conclusions (4) follow using Proposition 4. Now assume that the equalities (4) hold. From these equations we know that w_V must be init-closed, δ-closed, δ-consistent, and out-consistent, since w_U, satisfying $\sigma_U \in \mathcal{E}$ and $\pi_U \in \mathcal{M}$, has these properties by Lemma 11. Note that (4) also implies that the hypothesis automata of w_U and w_V coincide. Moreover, $\sigma_U \in \mathcal{E}$ and $\pi_U \in \mathcal{M}$ imply that U is minimal because of their definition and Proposition A.1 (and its dual). Using this, in combination with the assumptions for V, we can now apply Theorem 16 and conclude that U and V are isomorphic. ◀

We now comment on the connection between testing and minimization algorithms. Minimization is a natural choice when looking for a set of words approximating U to be tested against V. Formally, recall from Section 5 that we can use reachability and state merging to find sets $S, E \subseteq A^*$. Moreover, reachability analysis gives an $\alpha\colon S \to A^*$ that makes σ_U surjective and state merging gives an $\omega\colon 2^{A^*} \to 2^E$ that makes π_U injective (recall that $\mathcal{A} = A^*$ and $\mathcal{Z} = 2^{A^*}$ in the DFA setting). These, together with reachability and observability maps, give wrappers for U and V. The condition of Theorem 20 on w_V may not hold right away, but these wrappers are convenient starting points for algorithmic techniques, as we now show.

W-method. We now instantiate the above framework to recover the W-method [13]. This algorithm assumes to be given an upper bound n on the number of states of the unknown DFA V. Assume for convenience that U and V are minimal DFAs. We apply our framework as follows: first, we build the wrapper $w_U = (\sigma_U, \pi_U)$. We use the minimization algorithms from Section 5 to find $S \xrightarrow{\alpha} A^*$ and $2^{A^*} \xrightarrow{\omega} 2^E$ with finite $S, E \subseteq A^*$, which yield $\sigma_U = r_U \circ \alpha \in \mathcal{E}$ and $\pi_U = \omega \circ o_U \in \mathcal{M}$. If at this point the equalities (4) do not hold, then we can conclude that U and V accept different languages, and the testing failed. If we assume they hold, then because $\sigma_U \in \mathcal{E}$ and $\pi_U \in \mathcal{M}$ this means that $|\mathsf{img}(\xi^{w_V})| = |\mathsf{img}(\xi^{w_U})| = |U|$. The image of $\xi^{w_V} = \pi_V \circ \sigma_V$ is at least as big as the image of σ_V, so $|\mathsf{img}(\sigma_V)| \geq |U|$. By assumption we know that the size of V is at most n, and hence we update S to $S \cdot A^{\leq(n-|U|)}$, which yields $\sigma_V \in \mathcal{E}$ because $\varepsilon \in S$. Now we have $\sigma_U \in \mathcal{E}$, $\pi_U \in \mathcal{M}$, and $\sigma_V \in \mathcal{E}$. Applying Theorem 20, we can find out whether U and V are isomorphic by testing (4) for the updated wrappers. Instantiating what the equalities in (4) mean (see Example 19), we recover the test sequences generated by the W-method.

7 Optimized Algorithms

When fixing a consistency defect in an observation table, we add a column to distinguish two currently equal rows. Unfortunately, adding a full column implies that a membership query is needed also for each other row. Kearns and Vazirani [19] avoid this redundancy by basing the learning algorithm on a *classification tree* rather than an observation table. We now show that CALF encompasses this optimized algorithm, which will allow us to derive optimizations for other algorithms.

We introduce a notion of *classification tree*, close to the one by Isberner [17] (who calls it *discrimination tree*), and of language classifier induced by the tree.

▶ **Definition 21** (Classification Tree). Given a finite $S \subseteq A^*$, a *classification tree* τ on S is a labeled binary tree with internal nodes labeled by words from A^* and leaves labeled by subsets of S.

The *language classifier* for τ is the function $\omega_\tau \colon 2^{A^*} \to \mathcal{P}S$ that operates as follows. Given a language $L \in 2^{A^*}$, starting from the root of the tree: if the current node is a leaf with label $U \in \mathcal{P}S$, the function returns U; if the current node is an internal node with label $v \in A^*$, we repeat the process from the left subtree if $v \in L$ and from the right subtree otherwise.

In the CALF terminology, each classification tree τ gives rise to a wrapper $(S \xrightarrow{\alpha} A^* \xrightarrow{r_M} M, M \xrightarrow{o} 2^{A^*} \xrightarrow{\omega_\tau} \mathcal{P}S)$, where α is the inclusion and M is the minimal DFA for the language $\mathcal{L}$ that is to be learned. We define a function $\mathsf{sift}_\tau \colon A^* \to \mathcal{P}S$ as the composition $\omega_\tau \circ o_M \circ r_M$. This function *sifts* [19] words $u \in A^*$ through the tree by moving on a node with label $v \in A^*$ to the subtree corresponding to the value of $\mathcal{L}(uv)$. Applying Proposition 4, the approximated initial state map and dynamics of M can be rewritten using this function:

$$\xi_{\mathsf{init}} = \mathsf{sift}_\tau \circ \varepsilon \colon 1 \to \mathcal{P}S \qquad\qquad \xi_\delta = \mathsf{sift}_\tau \circ c \circ (\alpha \times \mathsf{id}_A) \colon S \times A \to \mathcal{P}S.$$

Recall that $\varepsilon \colon 1 \to A^*$ is the empty word and $c \colon A^* \times A \to A^*$ concatenates its arguments. The function $\xi_{\mathsf{out}} \colon S \to 2$ is still the same as for an observation table: $\xi_{\mathsf{out}}(s) = \mathcal{L}(s)$.

From these definitions, we obtain the following notions of closedness and consistency. The wrapper is:

- δ-closed if for each $t \in S \cdot A$ there is an $s \in S$ such that $\mathsf{sift}_\tau(s) = \mathsf{sift}_\tau(t)$;
- δ-consistent if for all $s_1, s_2 \in S$ such that $\mathsf{sift}_\tau(s_1) = \mathsf{sift}_\tau(s_2)$ we have $\mathsf{sift}_\tau(s_1 a) = \mathsf{sift}_\tau(s_2 a)$;
- init-closed if there is an $s \in S$ such that $\mathsf{sift}_\tau(s) = \mathsf{sift}_\tau(\varepsilon)$;
- out-consistent if for all $s_1, s_2 \in S$ such that $\mathsf{sift}_\tau(s_1) = \mathsf{sift}_\tau(s_2)$ we have $\mathcal{L}(s_1) = \mathcal{L}(s_2)$.

Now we can optimize the learning algorithm using Kearns and Vazirani classification trees as follows. Initially, the tree is just a leaf containing all words in S, which may be initialized to $\{\varepsilon\}$. When an out-consistency or δ-consistency defect is found, we have two words $s_1, s_2 \in S$ such that $\xi(s_1) = \xi(s_2) = U$, for some $U \in \mathcal{P}S$. We also have a word $v \in A^*$ such that $\mathcal{L}(s_1 v) \neq \mathcal{L}(s_2 v)$,[3] and we want to use this word to update the tree τ to distinguish $\xi(s_1)$ and $\xi(s_2)$. This is done by replacing the leaf with label U by a node that distinguishes based on the word v. Its left subtree is a leaf containing the words $s \in U$ such that $\mathcal{L}(sv) = 0$ while the $s \in U$ in its right subtree are such that $\mathcal{L}(sv) = 1$. This requires new membership queries, but only one for each word in $S \cup S \cdot A$ that sifts into the leaf with label U; the observation table approach needs queries for all elements of $S \cup S \cdot A$ when a column is added.

The intention of having $\mathcal{P}S$ as the set of labels is that we maintain the trees in such a way that $\xi \colon S \to \mathcal{P}S$ maps each word in S to the unique leaf it is contained in. As a result, the function ξ can be read directly from the tree. This means that, when adding a word to

[3] For an out-inconsistency, $v = \varepsilon$; on a δ-inconsistency, there is an $a \in A$ such that $\mathsf{sift}_\tau(s_1 a) \neq \mathsf{sift}_\tau(s_2 a)$. We take the label u of the lowest common ancestor node of those two leaves and define $v = au$.

S, we need to add it also to the leaf of the tree that it sifts into. Words are added to S when processing a counterexample as in $\mathtt{L}^\star$ and when fixing init-closedness or δ-closedness. These closedness defects occur when a word in $\{\varepsilon\} \cup S \cdot A$ sifts into an empty set leaf. In that case we add the word to S.

The classification tree optimization was originally developed for $\mathtt{L}^\star$, but we note that it can be applied directly to `ID` as well. Because of the abstract nature of Theorem 16, no new correctness proof is necessary.

Transporting Optimizations to Minimization and Testing. Using our correspondence between learning and minimization, the above optimization for learning algorithms immediately inspires an optimization for the state merging phase of Section 5. The main difference is that we sift states of the automaton Q through the tree, rather than words. That is, when sifting a state q at a node with label $v \in A^*$, the subtree we move to corresponds to whether v is accepted starting from q. Thus, instead of taking the labels of leaves from $\mathcal{P}S$, we take them from $\mathcal{P}Q$. The algorithm described above now creates a *splitting tree* [20, 28] for Q.

Interestingly, in this case one does not actually have to represent the trees to perform the algorithm; tracking only the partitioning of Q induced by the tree is enough. This is because the classification of next-states is contained in the classification of states: $\xi_\delta = \xi \circ \delta$ (using Proposition 4). An inconsistency consists in two states being in the same partition, but for some input $a \in A$ the corresponding next states being in different partitions. One then splits the partition of the two states into one partition for each of the partitions obtained after reading a. This corresponds to (repeatedly[4]) updating the tree to split the leaf. This algorithm that does not keep track of the tree is precisely Moore's reduction procedure [25].

The splitting tree algorithm described above can also be plugged into the W-method as it is described in Section 6. Note that the discussion about the correctness of the testing algorithm is not affected by this. The resulting algorithm is closely related to the HSI-method [22].

8 Conclusion

We have presented CALF, a categorical automata learning framework. CALF covers the basic definitions used in automata learning algorithms, and unifies correctness proofs for several of them. We have shown that these proofs extend also to minimization and testing methods. CALF is general enough to capture optimizations for all of these algorithms and provides an abstract umbrella to transfer results between the three areas. We illustrated how an optimization known in learning can be transported to minimization and testing. We have also exploited the categorical nature of the framework by changing the category and deriving algorithms for weighted automata in Appendix B. This example shows the versatility of the framework: weighted automata are naturally presented as coalgebras and CALF can accommodate this perspective as well as the algebraic one (which is used traditionally for DFAs and adopted in the main text).

Related Work. Most of the present paper is based on the first author's Master's thesis [29].

Other frameworks have been developed to study automata learning algorithms, e.g. [6, 17]. However, in both cases the results are much less general than ours and not applicable for example to settings where the automata have additional structure, such as weighted automata (see Appendix B). Isberner [17] hints at the connection between algorithms for minimization

[4] The partition splitting algorithm resolves every inconsistency for a within the partition at once.

and learning. For example, before introducing classification trees for learning, he explains them for minimization. Still, he does not provide a formal link between the two algorithms.

The relation between learning and testing was first explored by Berg et al. [7]. Their discussion, however, is limited to the case where the black-box DFA has at most as many states as the known DFA. This is because their correspondence is a stronger one that relates terminated learning algorithms to test sets of conformance testing algorithms, whereas we have provided algorithmic connections. Our Theorem 20, which allows reasoning about algorithms not making the above assumption, is completely new.

As mentioned in the introduction, a preliminary investigation of generalizing automata learning concepts using category theory was performed by Jacobs and Silva [18]. We were inspired by their work, but we note that they did not attempt to formulate anything like as general our wrappers; definitions are very concrete and dependent on observation tables. As a result, there are no fully abstract proofs and no instantiations to optimizations such as the classification trees discussed in Section 7. Moerman et al [24] designed a learning algorithm for nominal automata and generalized an optimization of $\mathtt{L}^\star$ to the nominal setting, but no general categorical approach was attempted.

Löding et al. [21] recently developed a theory of abstract learning frameworks (ALFs), which are used to study algorithms that use equivalence queries to synthesize objects (which need not be automata) conforming to a given specification. They do not study automata learning algorithms with membership queries, so their work has little overlap with ours.

Future Work. Many directions for future research are left open. For example, an interesting idea inspired by the relation between testing and learning is integrating testing algorithms into $\mathtt{L}^\star$. This could lead to optimizations that are unavailable when those components are kept separate. Further additions to CALF may include an investigation of the minimality of hypotheses and a characterization of the more elementary steps that are used in the algorithms. The only fully abstract correctness proofs at the moment are for the minimization algorithms and ID, where correctness follows from a condition that can be formulated on an abstract level. We would like to have an abstract characterization of progress for the other algorithms, which are of a more iterative nature. Finally, a concrete topic is optimizations for automata with additional structure, such as nondeterministic, weighted, and nominal automata. To the best of our knowledge, no analogue of classification trees exists for learning these classes. If such analogues turn out to exist, then the correspondences discussed in Section 5 and Section 6 would provide optimized learning and testing algorithms as well.

References

1 Dana Angluin. A note on the number of queries needed to identify regular languages. *Inform. Control*, 51:76–87, 1981.

2 Dana Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75:87–106, 1987.

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

4 Dana Angluin and Dana Fisman. Learning regular omega languages. In *ALT*, volume 8776, pages 125–139, 2014.

5 Michael A. Arbib and H. Paul Zeiger. On the relevance of abstract algebra to control theory. *Automatica*, 5:589–606, 1969.

6 José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. 1997.

7 Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In *FASE*, volume 3442, pages 175–189, 2005.

8 Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, volume 9, pages 1004–1009, 2009.

9 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A fresh approach to learning register automata. In *DLT*, volume 7907, pages 118–130, 2013.

10 Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *FAC*, 28(2):233–263, 2016.

11 Martin Chapman, Hana Chockler, Pascal Kesseli, Daniel Kroening, Ofer Strichman, and Michael Tautschnig. Learning the language of error. In *ATVA*, volume 9364, pages 114–130. 2015.

12 Chia Yuan Cho, Domagoj Babić, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *CCS*, pages 426–439. ACM, 2010.

13 Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4:178–187, 1978.

14 Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security*, pages 193–206, 2015.

15 E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

16 Bin-Lun Ho. *On effective construction of realizations from input-output descriptions*. PhD thesis, Stanford University, 1966.

17 Malte Isberner. *Foundations of active automata learning: an algorithmic perspective*. PhD thesis, Technical University of Dortmund, 2015.

18 Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, volume 8464, pages 384–406, 2014.

19 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.

20 David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE T. Comput.*, 43:306–320, 1994.

21 Christof Löding, P Madhusudan, and Daniel Neider. Abstract learning frameworks for synthesis. In *TACAS*, volume 9636, pages 167–185, 2016.

22 Gang Luo, Alexandre Petrenko, and Gregor v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In *Protocol Test Systems*, pages 95–110. 1995.

23 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inform. and Comput.*, 118:316–326, 1995.

24 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. In *POPL*, pages 613–625, 2017.

25 Edward F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

26 Anil Nerode. Linear automaton transformations. *Proceedings of the AMS*, 9:541–544, 1958.

27 Mathijs Schuts, Jozef Hooman, and Frits Vaandrager. Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In *IFM*, volume 9681, pages 311–325, 2016.

28 Rick Smetsers, Joshua Moerman, and David N. Jansen. Minimal separating sequences for all pairs of states. In *LATA*, pages 181–193, 2016.

29 Gerco van Heerdt. An abstract automata learning framework. Master's thesis, Radboud University Nijmegen, 2016.

A Omitted Proofs

▶ **Proposition 4.** *For all endofunctors F, F-algebras (U,u) and (V,v), F-algebra homomorphisms $h\colon (U,u)\to(V,v)$, and morphisms $\alpha\colon S\to U$ and $\beta\colon V\to P$, $\xi_v^{(h\circ\alpha,\beta)}=\xi_u^{(\alpha,\beta\circ h)}$.*

Proof. The F-algebra homomorphism $h\colon (U,u)\to(V,v)$ satisfies $v\circ Fh=h\circ u$ by definition. Therefore,

$$\begin{aligned}
\xi_v^{(h\circ\alpha,\beta)} &= \beta\circ v\circ F(h\circ\alpha) && (\text{definition of } \xi_v^{(h\circ\alpha,\beta)})\\
&= \beta\circ v\circ Fh\circ F\alpha && (\text{functoriality of } F)\\
&= \beta\circ h\circ u\circ F\alpha && (v\circ Fh=h\circ u)\\
&= \xi_u^{(\alpha,\beta\circ h)} && (\text{definition of } \xi_u^{(\alpha,\beta\circ h)}).
\end{aligned}$$
◀

▶ **Proposition 10.** *Consider functions f, g, and h as in the diagram below, with f surjective.*

$$\begin{tikzcd} U \arrow[r, two heads, "f"] \arrow[d, dashed, "i"'] \arrow[dr, "g"] & V \arrow[d, dashed, "j"] \\ W \arrow[r, "h"] & X \end{tikzcd}$$

1. *A function $i\colon U\to W$ making the left triangle in the diagram commute exists if and only if for each $u\in U$ there is a $w\in W$ such that $h(w)=g(u)$.*
2. *A function $j\colon V\to X$ making the right triangle commute exists if and only if for all $u_1,u_2\in U$ such that $f(u_1)=f(u_2)$ we have $g(u_1)=g(u_2)$.*

Proof. **1.** If such an i exists, then for each $u\in U$ we have $h(i(u))=g(u)$. Conversely, define $i(u)$ to be any $w\in W$ such that $h(w)=g(u)$, which exists by assumption. Then $(h\circ i)(u)=h(w)=g(u)$.
2. If such a j exists, then whenever $f(u_1)=f(u_2)$ we have $(j\circ f)(u_1)=(j\circ f)(u_2)$ and therefore $g(u_1)=g(u_2)$. Conversely, define $j(f(u))$ to be $g(u)$, using that f is surjective. We only need to check that this is well-defined; i.e., that whenever $f(u_1)=f(u_2)$ we also have $g(u_1)=g(u_2)$. This is precisely the assumption. ◀

▶ **Proposition 17.** *If for every prefix p of a word $z\in A^*$ there exists an $s\in S$ such that $r_M(s)=r_M(p)$, then $\mathcal{L}_H(z)=\mathcal{L}_M(z)$.*

Proof. Let $(S\xrightarrow{\alpha}A^*\xrightarrow{r}M, M\xrightarrow{\pi}P)$ be the wrapper and $(S\xrightarrow{e}H, H\xrightarrow{m}2^E)$ its minimization. Furthermore, let $z=a_1a_2\ldots a_n$ for n the length of z and each $a_i\in A$. For $0\le i\le n$, define $z_i=a_1a_2\ldots a_n$. Each prefix of z is z_i for some i. Assume that for every i there is an $s_i\in S$ such that $r_M(s_i)=r_M(z_i)$. We will show by induction to n that for all i,

$$\xi(s_i)=(m\circ r_H)(z_i). \tag{5}$$

Note that $z_0=\varepsilon$. We have

$$\begin{aligned}
\xi(s_0) &= (\pi\circ r_M)(s_0) && (\text{definition of } \xi)\\
&= (\pi\circ r_M)(\varepsilon) && (r_M(s_0)=r_M(z_0))\\
&= (\pi\circ \mathsf{init}_M)(*) && (\text{definition of } r_M)\\
&= \xi_{\mathsf{init}}(*) && (\text{definition of } \xi_{\mathsf{init}})\\
&= (m\circ \mathsf{init}_H)(*) && (\text{definition of } \mathsf{init}_H)\\
&= (m\circ r_H)(\varepsilon) && (\text{definition of } r_H).
\end{aligned}$$

Now assume that for a certain $1\le i<n$ we have (5). This implies $e(s_i)=r_H(z_i)$ because

$\xi = m \circ e$ and m is injective. Then

$$\begin{aligned}
\xi(s_{i+1}) &= (\pi \circ r_M)(s_{i+1}) && \text{(definition of } \xi\text{)}\\
&= (\pi \circ r_M)(z_{i+1}) && (r_M(s_{i+1}) = r_M(z_{i+1}))\\
&= (\pi \circ r_M)(z_i a_{i+1}) && (z_{i+1} = z_i a_{i+1})\\
&= (\pi \circ \delta_M)(r_M(z_i), a_{i+1}) && \text{(definition of } r_M\text{)}\\
&= (\pi \circ \delta_M)(r_M(s_i), a_{i+1}) && (r_M(s_i) = r_M(z_i))\\
&= \xi_\delta(s_i, a_{i+1}) && \text{(definition of } \xi_\delta\text{)}\\
&= \mathsf{cons}_\delta(e(s_i), a_{i+1}) && \text{(definition of } \mathsf{cons}_\delta\text{)}\\
&= \mathsf{cons}_\delta(r_H(z_i), a_{i+1}) && \text{(induction hypothesis)}\\
&= (m \circ \delta_H)(r_H(z_i), a_{i+1}) && \text{(definition of } \delta_H\text{)}\\
&= (m \circ r_H)(z_i a_{i+1}) && \text{(definition of } r_H\text{)}\\
&= (m \circ r_H)(z_{i+1}) && (z_{i+1} = z_i a_{i+1}).
\end{aligned}$$

This concludes the proof of (5).

In particular, then, $(m \circ e)(s_n) = \xi(s_n) = (m \circ r_H)(z)$. Because m is injective, we have $e(s_n) = r_H(z)$. Therefore,

$$\begin{aligned}
\mathcal{L}_H(z) &= (\mathsf{out}_H \circ r_H)(z) && \text{(definition of } \mathcal{L}_H\text{)}\\
&= (\mathsf{out}_H \circ e)(s_n) && (e(s_n) = r_H(z))\\
&= \xi_{\mathsf{out}}(s_n) && \text{(definition of } \mathsf{out}_H\text{)}\\
&= (\mathsf{out}_M \circ r_M)(s_n) && \text{(definition of } \xi_{\mathsf{out}}\text{)}\\
&= (\mathsf{out}_M \circ r_M)(z) && (r_M(s_n) = r_M(z))\\
&= \mathcal{L}_M(z) && \text{(definition of } \mathcal{L}_M\text{)}.
\end{aligned}$$

◀

▶ **Proposition A.1.** *Given an $(\mathcal{E}, \mathcal{M})$ factorization system on a category $\mathbf{C}$, if $g \circ f \in \mathcal{M}$, then $f \in \mathcal{M}$.*

Proof. Factorize $f = m \circ e$, with $e \in \mathcal{E}$ and $m \in \mathcal{M}$, and consider the unique diagonal d obtained from the commutative square below.

$$\begin{tikzcd}
U \arrow[r, two heads, "e"] \arrow[dd, "\mathsf{id}"'] & I \arrow[d, "m"] \arrow[ddl, dashed, "d"'] \\
& V \arrow[d, "g"] \\
U \arrow[r, tail, "g \circ f"] & W
\end{tikzcd}$$

We see that $d \circ e = \mathsf{id}$. Then $e \circ d \circ e = e$, and therefore $e \circ d = \mathsf{id}$ because e is epic. Thus, e is an isomorphism. Since $\mathcal{M}$ is closed under composition with isomorphisms, we conclude that $f \in \mathcal{M}$. ◀

B Linear Weighted Automata

So far we have only considered DFAs as examples of automata. In this appendix, we exploit the categorical nature of CALF by changing the base category in order to study linear weighted automata.

$$\begin{tikzcd}
\mathbb{F} \arrow[dr, "\mathsf{init}_Q"'] & & \mathbb{F} \\
& Q \arrow[ur, "\mathsf{out}_Q"'] \arrow[d, "\delta_Q"] & \\
& Q^A &
\end{tikzcd}$$

Let $\mathbf{C}$ be the category $\mathbf{Vect}^{\mathbf{op}}$, the opposite of the category of vector spaces and linear maps over a field $\mathbb{F}$. We need the opposite category because the dynamics of our automata will be coalgebras rather than the algebras that are found in Definition 14. We interpret everything in $\mathbf{Vect}$ rather than $\mathbf{Vect}^{\mathbf{op}}$. The automata we consider are for $I = Y = \mathbb{F}$ and $F = (-)^A$, where the latter for a finite set A as an endofunctor on $\mathbf{Vect}$ is given for a vector space X by the set of functions X^A with a pointwise vector space structure. On linear maps $f\colon W \to X$ we have $f^A\colon W^A \to X^A$ given by $f^A(g)(a) = f(g(a))$. An automaton is now a vector space Q with linear maps as shown in the diagram. These automata are called *linear weighted automata* (LWAs).

Given a set U, let $V(U)$ be the free vector space generated by U, the elements of which are formal sums $\sum_{i\in J} v_i \times u_i$ for finite sets J, $v_i \in \mathbb{F}$, and $u_i \in U$. The operation $V(-)$ is a functor $\mathbf{Set} \to \mathbf{Vect}$ that assigns to a function $f\colon U \to W$ between sets U and W the linear map $V(f)\colon V(U) \to V(W)$ given by $V(f)(u) = f(u)$. In fact, the functor V is left adjoint to the forgetful functor $\mathbf{Vect} \to \mathbf{Set}$ that assigns to each vector space its underlying set. Given a vector space W and a function $f\colon U \to W$, the *linearization* of f is the linear map $\overline{f}\colon V(U) \to W$ given by $\overline{f}\left(\sum_{i\in J} v_i \times u_i\right) = \sum_{i\in J} v_i \times f(u_i)$. Conversely, a linear map with domain $V(U)$ is completely determined by its definition on the elements of U. The adjunction implies that this is a bijective correspondence between functions $U \to W$ and linear maps $V(U) \to W$. Note that $\mathbb{F}$ is isomorphic to $V(1)$, so the initial state map $\mathsf{init}\colon \mathbb{F} \to Q$ of an LWA Q is essentially an element of Q. This element is given by $\mathsf{init}(1)$. Moreover, any linear map is determined by its value on the basis vectors of its domain, which gives us a finite representation for LWAs with a finite-dimensional state space. The automaton in this representation is known as a *weighted automaton*.

The initial input system in this setting is essentially the same as for DAs in $\mathbf{Set}$, but enriched with a free vector space structure.

▶ **Proposition B.1.** *The vector space $V(A^*)$ with*

$$\mathsf{init}\colon \mathbb{F} \to V(A^*) \qquad \mathsf{init}(1) = \varepsilon \qquad \delta\colon V(A^*) \to V(A^*)^A \qquad \delta(u)(a) = ua$$

forms an initial input system.

Proof. Given an input system Q with initial state map $\mathsf{init}_Q\colon V(1) \to Q$ and dynamics $\delta_Q\colon Q \to Q^A$, define a linear map $r\colon V(A^*) \to Q$ by $r(\varepsilon) = \mathsf{init}_Q(1)$ and $r(ua) = \delta_Q(r(u))(a)$. Note that r is an input system homomorphism. Any input system homomorphism $h\colon V(A^*) \to Q$ must satisfy $h(\varepsilon) = \mathsf{init}_Q(1)$ $h(ua) = \delta_Q(h(u))(a)$ and is therefore, by induction on the length of words, equal to r. ◀

Hence, in this setting, reachability is defined in terms of formal sums of words, rather than just single words.

Observability maps are simply as they would be in $\mathbf{Set}$ [18, Lemma 7].

▶ **Proposition B.2.** *The vector space $\mathbb{F}^{A^*}$ with*

$$\mathsf{out}\colon \mathbb{F}^{A^*} \to \mathbb{F} \qquad \mathsf{out}(l) = l(\varepsilon) \qquad \delta\colon \mathbb{F}^{A^*} \to (\mathbb{F}^{A^*})^A \qquad \delta(l)(a)(u) = l(au)$$

forms a final output system.

The observability maps are just as for DAs, but generalized to an arbitrary output set. Thus, $o\colon Q \to \mathbb{F}^{A^*}$ is given by $o(q)(\varepsilon) = \mathsf{out}_Q(q)$ and $o(q)(au) = o(\delta_Q(q)(a))(u)$.

We use the (surjective linear maps, injective linear maps) factorization system in **Vect**. The details are similar to those in **Set**, but now the image of a linear map $f\colon U \to V$ is a subspace of V. To see that the unique diagonals are the same as in **Set**, note the following result.

▶ **Proposition B.3.** *For vector spaces U, W, and X, if $f\colon U \to W$ and $g\colon U \to X$ are linear maps and $h\colon X \to W$ is a function such that $h \circ f = g$ and f is surjective, then h is a linear map.*

Proof. Let J be a finite index set, $\{v_i\}_{i \in J} \subseteq \mathbb{F}$, and $\{w_i\}_{i \in J} \subseteq W$. Because f is surjective, we know that for each $i \in J$ there is a $u_i \in U$ such that $f(u_i) = w_i$. Then

$$\begin{aligned}
h\left(\sum_{i \in J} v_i \times w_i\right) &= h\left(\sum_{i \in J} v_i \times f(u_i)\right) && (f(u_i) = w_i)\\
&= h\left(f\left(\sum_{i \in J} v_i \times u_i\right)\right) && (\text{linearity of } f)\\
&= g\left(\sum_{i \in J} v_i \times u_i\right) && (h \circ f = g)\\
&= \sum_{i \in J} v_i \times g(u_i) && (\text{linearity of } g)\\
&= \sum_{i \in J} v_i \times h(f(u_i)) && (h \circ f = g)\\
&= \sum_{i \in J} v_i \times h(w_i) && (f(u_i) = w_i).
\end{aligned}$$

◀

A language in this setting is a linear map $\mathcal{L}\colon V(A^*) \to \mathbb{F}$. One obtains the minimal LWA for the language $\mathcal{L}$ by starting from the automaton $V(A^*)$ with output map $\mathcal{L}$ and taking the image of its observability map $t\colon V(A^*) \to \mathbb{F}^{A^*}$. Note that if we take the initial state map of $\mathbb{F}^{A^*}$ to be $t \circ \mathsf{init}_{V(A^*)}$, then t is an LWA homomorphism, and its image is an LWA because each category of automata has a factorization system inherited from the base category. It must be the minimal LWA because by initiality and finality the factorization of t must be (r_M, o_M).

The dimension of a vector space U is denoted $\mathsf{dim}(U)$. The kernel of a linear map $f\colon U \to W$ is given by $\mathsf{ker}(f) = \{u \in U \mid f(u) = 0\}$. It is a standard result that if U is of finite dimension, then

$$\mathsf{dim}(U) = \mathsf{dim}(\mathsf{ker}(f)) + \mathsf{dim}(\mathsf{img}(f)). \tag{6}$$

Furthermore, if for two vector spaces U and W we have $U \subseteq W$, then $\mathsf{dim}(U) \leq \mathsf{dim}(W)$. If additionally $U \neq W$, then $\mathsf{dim}(U) < \mathsf{dim}(W)$.

For a linear map $f\colon U \to W$, if U is of finite dimension and $\mathsf{dim}(\mathsf{img}(f)) = \mathsf{dim}(U)$, then $\mathsf{dim}(\mathsf{ker}(f)) = 0$ by (6), implying f is injective. If $\mathsf{dim}(\mathsf{img}(f)) = \mathsf{dim}(W)$, then $\mathsf{img}(f) = W$, making f surjective.

Learning. We elaborate on the learning setting for LWAs as introduced by Jacobs and Silva [18]. In active learning of LWAs, we assume there is a language $\mathcal{L}\colon V(A^*) \to \mathbb{F}$ such that the state space of the minimal LWA M accepting $\mathcal{L}$ is of finite dimension. Given a finite set $E \subseteq A^*$, the function $\omega\colon \mathbb{F}^{A^*} \to \mathbb{F}^E$ given by $\omega(L)(e) = L(e)$ is a linear map. To select elements of $V(A^*)$, we may take a finite subset $S \subseteq A^*$ and apply V to the inclusion $\alpha \colon S \to A^*$ to obtain a linear map $V(\alpha) \colon V(S) \to V(A^*)$ and hence a wrapper

$$(\sigma, \pi) = (V(S) \xrightarrow{V(\alpha)} V(A^*) \xrightarrow{r} M, M \xrightarrow{o} \mathbb{F}^{A^*} \xrightarrow{\omega} \mathbb{F}^E).$$

All approximated linear maps are as expected: $\xi \colon V(S) \to \mathbb{F}^E$ is given by $\xi(s)(e) = \mathcal{L}(se)$, $\xi_\delta \colon V(S) \to (\mathbb{F}^E)^A$ is given by $\xi_\delta(s)(a)(e) = \mathcal{L}(sae)$, $\xi_{\mathsf{out}} \colon V(S) \to \mathbb{F}$ is given by $\xi_{\mathsf{out}}(s) = \mathcal{L}(s)$, and $\xi_{\mathsf{init}} \colon \mathbb{F} \to \mathbb{F}^E$ is given by $\xi_{\mathsf{init}}(1)(e) = \mathcal{L}(e)$. Although these maps can be represented in the same kind of observation table that was used for learning a DFA (except that we have a different output set here), the notions of closedness and consistency are different. This is because we have to deal with the larger domains of the actual linear maps. The characterizations of these properties can still be derived from Proposition 10, as a result of Proposition B.3. It follows directly that δ-closedness holds if and only if for all $l \in V(S)$ and $a \in A$ there is an $l' \in V(S)$ such that $\xi(l') = \xi_\delta(l)(a)$. To simplify this, we have the following result.

▶ **Proposition B.4.** *If J is a finite set and $\{v_i\}_{i \in J} \subseteq \mathbb{F}$, $\{s_i\}_{i \in J} \subseteq S$, and $\{l_i\}_{i \in J} \subseteq V(S)$ are such that $\xi(l_i) = \xi_\delta(s_i)(a)$ for all $i \in J$, then*

$$\xi\left(\sum_{i \in J} v_i \times l_i\right) = \xi_\delta\left(\sum_{i \in J} v_i \times s_i\right)(a).$$

Proof. We have

$$\begin{aligned}
\xi\left(\sum_{i \in J} v_i \times l_i\right) &= \sum_{i \in J} v_i \times \xi(l_i) && \text{(linearity of } \xi\text{)} \\
&= \sum_{i \in J} v_i \times \xi_\delta(s_i)(a) && (\xi(l_i) = \xi_\delta(s_i)(a)) \\
&= \left(\sum_{i \in J} v_i \times \xi_\delta(s_i)\right)(a) && \text{(pointwise vector space structure)} \\
&= \xi_\delta\left(\sum_{i \in J} v_i \times s_i\right)(a) && \text{(linearity of } \xi_\delta\text{).}
\end{aligned}$$ ◀

Thus, δ-closedness really says that for all $sa \in S \cdot A$ there must be an $l \in V(S)$ such that $\xi(l) = \xi_\delta(s)(a)$. As for δ-consistency, this property requires that for all $l_1, l_2 \in V(S)$ such that $\xi(l_1) = \xi(l_2)$ we must have $\xi_\delta(l_1)(a) = \xi_\delta(l_2)(a)$ for all $a \in A$.

Analogously, init-closedness requires there to be an $l \in V(S)$ such that $\xi(l) = \xi_{\mathsf{init}}(1)$ while out-consistency states that for all $l_1, l_2 \in V(S)$ such that $\xi(l_1) = \xi(l_2)$ we must have $\mathcal{L}(l_1) = \mathcal{L}(l_2)$.

Closedness can be determined simply by solving systems of linear equations. We expect that consistency defects can be found using linear programming, but this is currently under investigation.

If a closedness defect is found, we have a word $u \in A^*$ such that $(\omega \circ o_M \circ r_M)(u)$ is not a linear combination of any row indexed by words from S. Because we have subtraction and scalar division, this also means that each of the rows indexed by S is not a linear combination of the other rows and $(\omega \circ o_M \circ r_M)(u)$. Therefore, adding u to S increases the dimension of the hypothesis. The dimension of the hypothesis cannot exceed the dimension of the target M (see Appendix C), which is of finite dimension, so this process must terminate.

If a consistency defect is found, we have $l_1, l_2 \in V(S)$ and $u \in A^*$ such that $\xi(l_1) = \xi(l_2)$, but $(o_M \circ r_M)(l_1)(u) \neq (o_M \circ r_M)(l_2)(u)$. After adding u to E, we have $\xi(l_1) \neq \xi(l_2)$, which means that the kernel of the new ξ is a subset of the kernel of the old ξ. Therefore, the dimension of the kernel must have decreased. We know that this kernel is finite-dimensional (6), so this can only happen finitely many times.

The map $\sigma \colon V(S) \to M$ is surjective just if for each $m \in M$ there is an $l \in V(S)$ such that $r_M(l) = m$. Equivalently, the set $\{r_M(s) \mid s \in S\}$ needs to span M. Note that this means the set needs to include one of the (finite) bases of M. Thus, we have an adaptation of `ID` for LWAs that assumes to be given such a finite set S and enforces out-consistency and δ-consistency to obtain, by Theorem 16(1), a hypothesis isomorphic to M.

As for `L*`, Proposition 17 carries over almost immediately. Due to the similarity, we omit the proof.

▶ **Proposition B.5.** *If for every prefix p of a word $z \in A^*$ there exists an $l \in V(S)$ such that $r_M(l) = r_M(p)$, then $\mathcal{L}_H(z) = \mathcal{L}_M(z)$.*

▶ **Corollary 6.** *If $z \in A^*$ is a counterexample, i.e., $\mathcal{L}_H(z) \neq \mathcal{L}_M(z)$, then adding all prefixes of z to S will increase the dimension of $\mathsf{img}(r_M \circ \alpha)$.*

Once the dimensions of $\mathsf{img}(r_M \circ \alpha)$ and M coincide, $r_M \circ \alpha$ is surjective and we can apply Theorem 16(1). Thus, the number of required counterexamples is bounded by the dimension of M.

Reachability Analysis. Let Q be a finite-dimensional LWA and R its reachable part. Given a finite subset $S \subseteq R$, we have a wrapper $(V(S) \xrightarrow{\overline{\sigma}} R, R \xrightarrow{\pi} Q)$ in **Vect**, where σ and π are the inclusions. This wrapper is init-closed if there is an $l \in V(S)$ such that $\xi(l) = \mathsf{init}_Q(1)$, which can be satisfied by initializing $S = \{\varepsilon\}$. The wrapper is δ-closed if for all $s \in S$ and $a \in A$ there is an $l \in V(S)$ such that $\xi(l) = \delta_Q(\xi(s))(a)$. If for some sa this is not the case, we simply add it to S. As in learning, the dimension of the hypothesis increases by doing so, and therefore the process terminates. Using Theorem 16(2), the final hypothesis is isomorphic to R.

State Merging. Given a finite-dimensional LWA Q, we have a factorization of the observability map o_Q as in Section 5—$Q \overset{h}{\twoheadrightarrow} O \overset{o}{\rightarrowtail} \mathbb{F}^{A^*}$—yielding an observable equivalent LWA O. Fixing a finite set $E \subseteq A^*$, we have a wrapper $(Q \xrightarrow{h} O, O \xrightarrow{o} \mathbb{F}^{A^*} \xrightarrow{\omega} \mathbb{F}^E)$. The linear map $\xi \colon Q \to \mathbb{F}^E$ gives the output of each state on the words in E. The wrapper is out-consistent if for all $q_1, q_2 \in Q$ such that $\xi(q_1) = \xi(q_2)$ we have $\mathsf{out}(q_1) = \mathsf{out}(q_2)$. If the last equation fails, we may add ε to E to distinguish $\xi(q_1)$ and $\xi(q_2)$. The wrapper is δ-consistent if for all $q_1, q_2 \in Q$ and $a \in A$ such that $\xi(q_1) = \xi(q_2)$ we have $\xi(\delta(q_1)(a)) = \xi(\delta(q_2)(a))$. If this last equation fails on some $e \in E$, we may add ae to E to distinguish $\xi(q_1)$ and $\xi(q_2)$. As in learning, this decreases the dimension of the kernel of ξ, which guarantees termination. Using Theorem 16(1), the final hypothesis is isomorphic to O.

Testing. Consider a known finite-dimensional LWA X and an unknown finite-dimensional LWA Z, both of which are minimal. Using the minimization algorithms (but with S for the reachability analysis a subset of A^*), we can find finite $S, E \subseteq A^*$ such that $\varepsilon \in S$ and the wrapper

$$w_X = (\sigma_X, \pi_X) = (V(S) \xrightarrow{V(\alpha)} V(A^*) \xrightarrow{r} X, X \xrightarrow{o} \mathbb{F}^{A^*} \xrightarrow{\omega} \mathbb{F}^E)$$

satisfies $\sigma_X \in \mathcal{E}$ and $\pi_X \in \mathcal{M}$. Define $w_Z = (\sigma_Z, \pi_Z)$ analogously. If at this point the equalities

$$\xi^{w_X} = \xi^{w_Z} \qquad \xi^{w_X}_{\mathsf{init}_X} = \xi^{w_Z}_{\mathsf{init}_Z} \qquad \xi^{w_X}_{\delta_X} = \xi^{w_Z}_{\delta_Z} \qquad \xi^{w_X}_{\mathsf{out}_X} = \xi^{w_Z}_{\mathsf{out}_Z}. \tag{7}$$

given by Theorem 20 do not hold, we can conclude that X and Z accept different languages. Assume these equalities do hold. By Corollary 13 this implies $\mathsf{dim}(\mathsf{img}(\xi^{w_Z})) = \mathsf{dim}(U)$. Assuming a given upperbound n on the dimension of Z, updating S to $S \cdot A^{\leq(n-\mathsf{dim}(X))}$ yields $\sigma_Z \in \mathcal{E}$. Applying Theorem 20, we can find out whether X and Z are isomorphic by testing (7) for the updated wrappers. That is, we have an adaptation of the W-method for LWAs.

C The Hypothesis is Smaller than the Target

Given a wrapper $(S \xrightarrow{\sigma} T, T \xrightarrow{\pi} P)$ in an arbitrary category $\mathbf{C}$ with an $(\mathcal{E}, \mathcal{M})$ factorization system, the hypothesis can be defined using three successive factorizations, as shown below.

$$\begin{tikzcd}
S \arrow[rr, "\sigma"] \arrow[dr, two heads] & & T \arrow[rr, "\pi"] \arrow[dr, two heads] & & P \\
& J \arrow[ur, tail] \arrow[dr, two heads] & & K \arrow[ur, tail] & \\
& & H \arrow[ur, tail] & &
\end{tikzcd}$$

We now explain for the discussed concrete settings why the hypothesis is always “smaller” than the target, for the appropriate notion of size in each category.

In the Category of Sets. We know that if T is a finite set, then J, being a subset of T, is a finite set with $|J| \leq |T|$. Because there is a surjective function $J \to H$, we conclude that H is finite and $|H| \leq |J| \leq |T|$.

In the Category of Vector Spaces. Assume T is of finite dimension. Since J is a subspace of T, we have that J is a finite-dimensional vector space with $\mathsf{dim}(J) \leq \mathsf{dim}(T)$. Note that there exists a surjective linear map $x\colon J \to H$, so H is finite-dimensional and $\mathsf{dim}(\mathsf{img}(x)) = \mathsf{dim}(H)$. It then follows from (6) in Appendix B that $\mathsf{dim}(\mathsf{ker}(x)) = \mathsf{dim}(J) - \mathsf{dim}(H)$, so we must have $\mathsf{dim}(H) \leq \mathsf{dim}(J) \leq \mathsf{dim}(T)$.