Learning Automata with Side-Effects

Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva

University College London

Abstract

Automata learning has been successfully applied in the verification of hardware and software. The size of the automaton model learned is a bottleneck for scalability and hence optimizations that enable learning of compact representations are important. In this paper, we continue the development of a general framework for automata learning based on category theory and develop a class of optimizations and an accompanying correctness proof for learning algorithms. The new algorithm is parametric on a monad, which provides a rich algebraic structure to capture non-determinism and other side-effects. These side-effects are used to learn more compact automaton models and the abstract categorical approach enables us to capture several possible optimizations under the same (p)roof.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases automata, learning, side-effects, monads, algebras

1 Introduction

Automata learning algorithms have been used in various verification tasks, ranging from finding bugs in implementations of network protocols [6] to providing defense mechanisms against botnets [5] and rejuvenating legacy software [13]. Vaandrager has recently written a comprehensive overview of the widespread use of automata learning in verification [14].

A limitation in model-based verification is that, with the increase in complexity of software and hardware, the size of the models can become too large to be handled by tools (not to mention readability by humans). This demands compositional methods and techniques that enable compact representation of behaviors.

In this paper, we exploit the rich structure of languages to add optimizations to learning algorithms in order to obtain compact representations. We will use as playground for our approach the well known $\mathtt{L}^\star$ algorithm of Angluin [1], which learns a minimal deterministic finite automaton (DFA) accepting a regular language. We take an abstract approach, in which category theory is used to devise an optimized learning algorithm and a generic correctness proof that captures a class of examples.

The inspiration for the optimization is quite concrete: it is a well-known fact that non-deterministic finite automata (NFAs) can be much smaller than deterministic ones when accepting a regular language. The subtle point is that given a regular language, there is a canonical deterministic automaton accepting it—the minimal one—but there might be many "minimal" non-deterministic automata accepting the same language. This raises challenges for (learning) algorithms: which non-deterministic automaton should the algorithm learn? In an attempt to overcome this, Bollig et al. [4] developed a version of Angluin's $\mathtt{L}^\star$ algorithm, which they called $\mathtt{NL}^\star$, in which they use a particular class of NFA, namely *Residual Finite State Automata* (RFSAs), which do admit minimal canonical representatives. Though $\mathtt{NL}^\star$ indeed is a first step in incorporating a more compact representation of regular languages, there are several questions that remain to be understood and which we tackle in this paper.

DFAs and NFAs are formally connected by the subset construction. Underpinning this construction is the rich algebraic structure of languages and of the state space of the DFA

obtained by determinizing an NFA. More concretely, the state space of a determinized DFA—which are subsets of the state space of the original NFA—has a join-semilattice structure. Moreover, this structure is preserved in language acceptance: if there are subsets U, V, and $U \cup V$, then the language of the latter is the union of the languages of the first two. Formally, the language map is a join-semilattice map, since languages themselves are just sets of words and have a lattice structure. From a conceptual perspective, Bollig et al. did not explicitly exploit this algebraic structure in their $\mathtt{NL}^\star$ algorithm. And languages are even richer: they have the structure of complete atomic Boolean algebras. This leads us to a series of questions: Can we exploit this and have even more compact representations? What if we slightly change the setting and look at weighted languages over a semiring, which have the structure of a semimodule (or vector space, if the semiring is a field)?

Our key insight is that the algebraic structures mentioned above are in fact algebras for a monad T. In the case of join-semilattices this is the powerset monad, and in the case of vector spaces it is the free vector space monad. These monads can be used to define a notion of T-automaton, with states having the structure of an algebra for the monad T, which generalizes non-determinism as a side-effect. From T-automata we can derive a compact, equivalent version by taking as states a set of *generators* and transferring the algebraic structure of the original state space to the transition structure of the automaton. This process is similar to a reverse construction to determinization.

In this paper, we introduce a general approach to learn automata featuring non-determinism and other side-effects, which are captured by a monad. The framework can be instantiated to obtain specific algorithms, with correctness-by-construction guarantees. We incorporate optimizations arising from the monadic representation, which will lead to more scalable algorithms. We illustrate the versatility of the general approach by applying it to several classes of automata with side-effects.

2 Overview

In this section, we give an overview of our approach. We start by explaining the original $\mathtt{L}^\star$ algorithm. Then we discuss the challenges in adapting the algorithm to learn automata with side-effects. To do this, we consider the particular case of non-deterministic finite automata (NFAs). Finally, we highlight our main contributions.

2.1 $\mathtt{L}^\star$ algorithm

The $\mathtt{L}^\star$ algorithm learns the minimal DFA accepting a language $\mathcal{L} \subseteq A^*$ over a finite alphabet A. The algorithm assumes the existence of an *oracle* that can answer two types of queries:

- **Membership queries**: given a word $w \in A^*$, does it belong to $\mathcal{L}$?
- **Equivalence queries**: given a *hypothesis* DFA $\mathcal{H}$, does it accept $\mathcal{L}$? If this is not the case, the oracle will return a *counterexample* z, which is a word incorrectly accepted or rejected by $\mathcal{H}$.

The algorithm incrementally builds an *observation table*, which is parameterized by two finite sets $S, E \subseteq A^*$. The table is made of two parts: a top part, with rows ranging over S, and a bottom part, with rows ranging over $S \cdot A$. Columns range over E. As an example, and to set notation, consider the table below (over $A = \{a, b\}$).

```
1  S, E ← {ε}
2  repeat
3      while the table is not closed or not consistent
4          if the table is not closed
5              find t ∈ S, a ∈ A such that row_b(t)(a) ≠ row_t(s) for all s ∈ S
6              S ← S ∪ {t}
7          if the table is not consistent
8              find s_1, s_2 ∈ S, a ∈ A, and e ∈ E such that
                   row_t(s_1) = row_t(s_2) and row_b(s_1)(a)(e) ≠ row_b(s_2)(a)(e)
9              E ← E ∪ {ae}
10     Construct the hypothesis H and submit it to the oracle
11     if the oracle replies no, with a counterexample z
12         S ← S ∪ prefixes(z)
13 until the oracle replies yes
14 return H
```

Figure 1 $\mathtt{L}^\star$ algorithm.

		E: ε	a	aa
S	ε	0	0	1
$S \cdot A$	a	0	1	0
	b	0	0	0

$$\mathsf{row_t}\colon S \to 2^E \qquad \mathsf{row_t}(u)(v) = 1 \iff uv \in \mathcal{L}$$
$$\mathsf{row_b}\colon S \to (2^E)^A \qquad \mathsf{row_b}(u)(v)(a) = 1 \iff uav \in \mathcal{L}$$

This table indicates that $\mathcal{L}$ contains at least aa and definitely does not contain the words $\varepsilon, a, b, ba, baa, aaa$. In the following we use the functions $\mathsf{row_t}$ and $\mathsf{row_b}$ to describe the two parts of the table.

A key idea of the algorithm is to construct a hypothesis DFA from the different rows in the table. That is, the state space of the hypothesis is given by the set $H = \{\mathsf{row_t}(s) \mid s \in S\}$. Note that there may be multiple rows having the same contents, but they result in a single state. The initial state is $\mathsf{row_t}(\varepsilon)$, and we use the ε column to determine whether a state is accepting: $\mathsf{row_t}(s)$ is accepting whenever $\mathsf{row_t}(s)(\varepsilon) = 1$. The transition function is defined as $\mathsf{row_t}(s) \xrightarrow{a} \mathsf{row_b}(s)(a)$ (notice that the continuation is drawn from the bottom part of the table). For the hypothesis automaton to be well-defined, the table must satisfy two properties:

- **Closedness** states that each transition actually leads to a state of the hypothesis. That is, the table is closed if for all $t \in S, a \in A$ there is an $s \in S$ such that $\mathsf{row_t}(s) = \mathsf{row_b}(t)(a)$.
- **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 $\mathsf{row_t}(s_1) = \mathsf{row_t}(s_2)$ we have $\mathsf{row_b}(s_1)(a) = \mathsf{row_b}(s_2)(a)$, for all $a \in A$.

The algorithm updates the sets S and E to satisfy these properties, constructs a hypothesis, submits it in an equivalence query and, when given a counterexample, refines the hypothesis. This process continues until the hypothesis is correct. The algorithm is given in Figure 1.

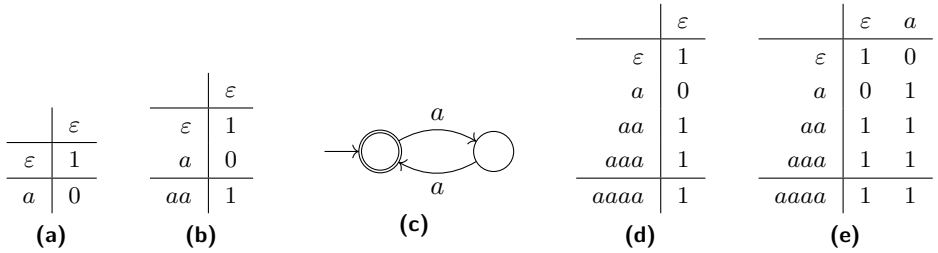
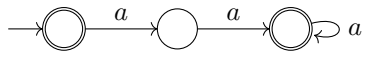


Figure 2 Example run of L* on $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$.

Example Run. We now run the algorithm with the target language $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$. The minimal DFA accepting $\mathcal{L}$ is

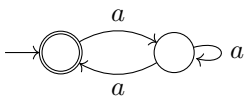
$\mathcal{M} =$  (1)

Initially, $S = E = \{\varepsilon\}$. We build the observation table given in Figure 2a. This table is not closed, because the row with label a, having 0 in the only column, does not appear in the upper part of the table (the only row ε has 1). To fix this, we add the word a to the set S. Now the table (Figure 2b) is closed and consistent, so we construct the hypothesis that is shown in Figure 2c and pose an equivalence query. The oracle replies *no* and informs us that the word aaa should have been accepted. L* handles a counterexample by adding all its prefixes to the set S. We only have to add aa and aaa in this case. The next table (Figure 2d) is closed, but not consistent: the rows for $\varepsilon, aa \in S$ both have value 1, but their extensions a and aaa differ. To fix this, we prepend the continuation a to the column ε on which they differ and add $a \cdot \varepsilon = a$ to E. This distinguishes $\mathsf{row_t}(\varepsilon)$ from $\mathsf{row_t}(aa)$, as seen in the next table in Figure 2e. The table is now closed and consistent, and the new hypothesis automaton is precisely the correct one $\mathcal{M}$.

The key to understanding why the above algorithm produces the minimal DFA is seeing that the hypothesis construction approximates the theoretical construction of the minimal DFA, which is unique up to isomorphism. That is, for $S = E = A^*$ the relation that identifies words of S having the same value in $\mathsf{row_t}$ is precisely Nerode's right congruence.

2.2 Learning non-deterministic automata

As is well known, NFAs can be smaller than the minimal DFA for a given language. For example, the language $\mathcal{L}$ above is accepted by the NFA

$\mathcal{N} =$  (2)

which is smaller than the minimal DFA $\mathcal{M}$. Though in this example, which we chose for simplicity, the state reduction is not massive it is known that in general NFAs can be exponentially smaller than the minimal DFA [10]. This reduction of the state space is enabled by a side-effect — non-determinism, in this case. Another remarkable example is *alternating automata*, where the reduction can even be doubly exponential.

Learning NFAs can lead to a substantial gain in space complexity, but it is challenging. The main difficulty is that NFAs do not have a canonical minimal representative: there may be several non-isomorphic state-minimal NFAs accepting the same language, which poses problems for the development of the learning algorithm. In an attempt to overcome this, Bollig et al. [4] proposed to use a particular class of NFAs, namely RFSAs, which do admit minimal canonical representatives. However, their solution is ad-hoc for NFAs and does not extend to other automata, such as weighted or alternating. In this paper we present a solution that works for any notion of side-effect, specified as a monad.

The crucial observation underlying our approach is that the language semantics of an NFA is defined in terms of its determinization, i.e., the DFA obtained by taking sets of states of the NFA as state space. In other words, this DFA is defined over an algebraic structure induced by the powerset, namely a *join semilattice* (JSL) whose join operation is set union. Being a DFA, this automaton admits a minimal representative, which leads to the key idea for our algorithm: learning NFAs as automata over JSLs.

In order to do so, we use an extended table where rows have a JSL structure, defined as follows. The join of two rows is given by an element-wise or and the bottom element is the row containing only zeroes. More precisely, the new table consists of the two functions

$$\mathsf{row_t} \colon \mathcal{P}(S) \to 2^E \qquad \mathsf{row_b} \colon \mathcal{P}(S) \to (2^E)^A$$

given by $\mathsf{row_t}(U) = \bigvee\{\mathsf{row_t}(s) \mid s \in U\}$ and $\mathsf{row_b}(U)(a) = \bigvee\{\mathsf{row_b}(s)(a) \mid s \in U\}$ where functions on the right-hand side are the original functions. Formally, these functions are JSL homomorphisms. They induce the following definitions of closedness and consistency:

- The table is closed if for all $U \subseteq S, a \in A$ there is $U' \subseteq S$ such that $\mathsf{row_t}(U') = \mathsf{row_b}(U)(a)$.
- The table is consistent if for all $U_1, U_2 \subseteq S$ such that $\mathsf{row_t}(U_1) = \mathsf{row_t}(U_2)$ we have $\mathsf{row_b}(U_1)(a) = \mathsf{row_b}(U_2)(a)$ for any $a \in A$.

We remark that our algorithm does not actually store the whole extended table, which can be quite large. It only needs to store the original table over S, because all other rows in $\mathcal{P}(S)$ are freely generated and can be computed when needed. The only lines in Figure 1 that need to be adjusted are lines 5 and 8, where closedness and consistency are replaced with the new notions give above. Moreover, $\mathcal{H}$ is now built from the extended table.

Optimizations. In this paper we also present two optimizations to our algorithm. For the first one, note that the state space of the hypothesis constructed by the "naive" algorithm bears a full JSL structure, which can be quite large. We show that we can extract a *minimal set of generators* from the table and compute a *succinct hypothesis* that can be represented as an ordinary NFA, without any algebraic structure. For JSLs, this consists in only taking rows that are not the join of other rows, i.e., the *join-irreducibles*. The second optimization is a generalization of the optimized counterexample handling method of Rivest and Schapire [12], originally intended for $\mathrm{L}^\star$ and DFAs. Roughly, it consists in fixing counterexamples by adding a single *suffix* of the counterexample to E, instead of adding all prefixes of the counterexample to S. This can avoid the algorithm posing a large number of membership queries.

Example Revisited. We now run the new algorithm on the language $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$ considered earlier. For the sake of simplifying pictures, we already show the NFA hypothesis obtained by picking join-irreducibles from the table. Starting from $S = E = \{\varepsilon\}$, the observation table, depicted in Figure 2a, is immediately closed and consistent. (It is closed because we have $\mathsf{row_t}(\{a\}) = \mathsf{row_t}(\emptyset)$.) This leads to an NFA hypothesis having a single state that is initial and accepting and has no transitions (Figure 3a). The hypothesis

(a)

	ε
ε	1
a	0
aa	1
aaa	1

(b)

	ε	a
ε	1	0
a	0	1
aa	1	1
aaa	1	1

(c)

Figure 3 Example run of the $\mathtt{L}^\star$ adaptation for NFAs on $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$

is obviously incorrect, and the teacher may supply us with the counterexample aa. Adding the prefixes a and aa to S leads to the table in Figure 3b. The table is again closed, but not consistent: $\mathsf{row_t}(\{a\}) = \mathsf{row_t}(\emptyset)$, but $\mathsf{row_b}(\{a\})(a) = \mathsf{row_t}(\{aa\}) \neq \mathsf{row_t}(\emptyset) = \mathsf{row_b}(\emptyset)(a)$. Thus, we add a to E. The resulting table, shown in Figure 3c, is closed and consistent. We note that the row aa is the union of other rows: $\mathsf{row_t}(\{aa\}) = \mathsf{row_t}(\{\varepsilon, a\})$ (i.e., it is not a join-irreducible), and therefore it can be ignored when building the hypothesis. The new hypothesis has two states, ε and a, and indeed it is the correct one $\mathcal{N}$.

2.3 Contributions

Our main contributions are as follows:

- In Section 4, we develop a general algorithm $\mathtt{L}^\star_T$, which generalizes the NFA one presented in the previous section to an arbitrary *monad* T capturing side-effects. Then we provide a general correctness proof for our algorithm.
- In Section 5, we describe the first optimization and prove its correctness.
- In Section 6 we describe the second optimization. We also show how it can be combined with the one of Section 5, and how it can also lead to a further small optimization, where the consistency check on the table is dropped altogether.
- Finally, in Section 7 we show how $\mathtt{L}^\star_T$ can be applied to several classes of examples.

3 T-automata

In this section we define a notion of T-automaton, a generalization of non-deterministic finite automata parametric in a monad T. We assume familiarity with basic notions of category theory: functors (in the category **Set** of sets and functions) and natural transformations.

Side-effects and different notions of non-determinism can be conveniently captured as a *monad* T. A *monad* $T = (T, \eta, \mu)$ is a triple consisting of an endofunctor T on **Set** and two natural transformations: a *unit* $\eta\colon \mathtt{Id} \Rightarrow T$ and a *multiplication* $\mu\colon T^2 \Rightarrow T$. They satisfy the compatibility laws $\mu \circ \eta_T = \mathsf{id}_T = \mu \circ T\eta$ and $\mu \circ \mu_T = \mu \circ T\mu$.

▶ **Example 1 (Monads).** An example of a monad is the triple $(\mathcal{P}, \{-\}, \bigcup)$, where $\mathcal{P}$ denotes the powerset functor associating a collection of subsets to a set, $\{-\}$ is the function that returns a singleton set, and $\bigcup$ is just union of sets. Another example is the triple $(V(-), e, m)$ where $V(X)$ returns the free semimodule (over a commutative semiring $\mathbb{S}$) over X, namely $\{\varphi \mid \varphi\colon X \to \mathbb{S}, \text{with support of } \varphi \text{ finite}\}$. The support of a function is the set of $x \in X$ such that $\varphi(x) \neq 0$. Then $e\colon X \to V(X)$ returns the characteristic function for each $x \in X$, and $m\colon V(V(X)) \to V(X)$ is defined for $\varphi \in V(V(X))$ and $x \in X$ as $m(\varphi)(x) = \sum_v \varphi(v) \times v(x)$.

Given a monad T, a T-algebra is a pair (X, h) consisting of a carrier set X and a function $h\colon TX \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = \mathsf{id}_X$. A T-homomorphism between two

T-algebras (X, h) and (Y, k) is a function $f\colon X \to Y$ such that $f \circ h = k \circ Tf$. The abstract notion of T-algebra instantiates to expected notions, as illustrated in the following example.

▶ **Example 2 (Algebras for a monad).** Let $T = \mathcal{P}$ be the powerset monad. $\mathcal{P}$-algebras are (complete) join-semilattices, and their homomorphisms are join-preserving functions. If $\mathbb{S}$ is a field, V-algebras are vector spaces and algebra homomorphisms are linear functions.

We will often refer to a T-algebra (X, h) as X if h is understood or if its specific definition is irrelevant. Given a set X, (TX, μ_X) is a T-algebra called the *free T-algebra* on X. One can also build algebras pointwise for some operations. For instance, if Y is a set and (X, x) a T-algebra, then we have a T-algebra (X^Y, f), where $f\colon T(X^Y) \to X^Y$ is given by $f(U)(y) = x \circ T(\mathsf{ev}_y)(U)$ and $\mathsf{ev}_y\colon X^Y \to X$ by $\mathsf{ev}_y(g) = g(y)$. If U and V are T-algebras and $f\colon U \to V$ is a T-algebra homomorphism, then the image $\mathtt{img}(f)$ of f is a T-algebra, with the T-algebra structure inherited from V.

The following proposition connects algebra homomorphisms from the free T-algebra on a set U to an algebra V with functions $U \to V$. We will make use of this later in the section.

▶ **Proposition 3.** *There is a bijective correspondence between T-algebra homomorphisms $f\colon TU \to V$ for a set U and a T-algebra (V, v) and functions $f^\dagger\colon U \to V$: define $f^\dagger = f \circ \eta$ and $g^\sharp = v \circ Tg$ for any function $g\colon U \to V$. Then $g^\sharp$ is a T-algebra homomorphism $TU \to V$ called the* free T-extension of g, *and we have $f^{\dagger\sharp} = f$ and $g^{\sharp\dagger} = g$.*

We now have all the ingredients to define our notion of automaton with side-effects and their language semantics. We fix a monad (T, η, μ) where T preserves finite sets and a T-algebra O that models outputs of automata.

▶ **Definition 4 (T-automaton).** A *T-automaton* is a quadruple $(Q, \delta\colon Q \to Q^A, \mathsf{out}\colon Q \to O, \mathsf{init})$, where the *state space* Q is a T-algebra, the *transition map* δ and *output map* out are T-algebra homomorphisms, and $\mathsf{init} \in Q$ is the *initial state*.

▶ **Example 5.** Recall that $\mathcal{P}$-algebras are join-semilattices, and their homomorphisms are join-preserving functions. In a $\mathcal{P}$-automaton, Q is equipped with a join operation, and Q^A is a join-semilattice over functions $A \to \mathcal{P}(X)$ with pointwise join, namely $(f \vee g)(a) = f(a) \vee g(a)$, for $a \in A$. Since all the automaton maps preserve joins, we have, in particular, $\delta(q_1 \vee q_2)(a) = \delta(q_1)(a) \vee \delta(q_2)(a)$. One can represent an NFA $\mathcal{A}$ over a set of states S as a $\mathcal{P}$-automaton by taking $Q = (\mathcal{P}(S), \cup)$ and $O = 2$, the Boolean join-semilattice. In this case $\mathsf{init} \subseteq S$ is a set of initial states; $\mathsf{out}\colon \mathcal{P}(Q) \to 2$ picks accepting and rejecting sets of states, by stipulating that $X \subseteq S$ is accepting iff it contains at least an accepting state q, i.e. $\mathsf{out}(\{q\}) = 1$; and $\delta\colon \mathcal{P}(S) \to \mathcal{P}(S)^A$ is the extension to sets of the NFA's transition relation (seen as a function $S \to \mathcal{P}(S)^A$). Note that the resulting $\mathcal{P}$-automaton is precisely the determinized version of the NFA.

A *(generalized) language* is a function $\mathcal{L}\colon A^* \to O$. For every T-automaton we have an *observability* and a *reachability* map, telling respectively which language each state recognizes and which state is reached by reading a given word.

▶ **Definition 6 (Observability and reachability maps).** The *observability map* of a T-automaton $\mathcal{A}$ with state space Q is a function $o_\mathcal{A}\colon Q \to O^{A^*}$ inductively defined as follows: $o_\mathcal{A}(q)(\varepsilon) = \mathsf{out}(q)$ and $o_\mathcal{A}(q)(av) = o_\mathcal{A}(\delta(q)(a))(v)$. The *reachability map* of $\mathcal{A}$ is a function $r_\mathcal{A}\colon A^* \to Q$ inductively defined as follows: $r_\mathcal{A}(\varepsilon) = \mathsf{init}$ and $r_\mathcal{A}(ua) = \delta(r_\mathcal{A}(u))(a)$.

The *language accepted by* $\mathcal{A}$ is the function $\mathcal{L}_\mathcal{A}\colon A^* \to O$ given by $o_\mathcal{A}(\mathsf{init}) = \mathsf{out}_\mathcal{A} \circ r_\mathcal{A}$.

▶ **Example 7.** For an NFA $\mathcal{A}$ represented as a $\mathcal{P}$-automaton, as seen in Example 5, $o_{\mathcal{A}}(q)$ is the language of q in the traditional sense. Notice that q, in general, is a set of states: since δ preserves unions, $o_{\mathcal{A}}(q)$ takes the union of languages of singleton states. The set $\mathcal{L}_{\mathcal{A}}$ is the language accepted by the initial states, i.e., the language of the whole NFA. The reachability map $r_{\mathcal{A}}(u)$ returns the set of states reached via all possible paths reading u.

Given a language $\mathcal{L}\colon A^* \to O$, there exists a (unique) *minimal T-automaton* $\mathcal{M}_{\mathcal{L}}$ accepting $\mathcal{L}$. Its existence follows from general facts, see e.g. [15].

▶ **Definition 8 (Minimal T-automaton for $\mathcal{L}$).** Let $t_{\mathcal{L}}\colon A^* \to O^{A^*}$ be the function giving the *residual languages* of $\mathcal{L}$, namely $t_{\mathcal{L}}(u) = \lambda v.\mathcal{L}(uv)$. The minimal T-automaton $\mathcal{M}_{\mathcal{L}}$ accepting $\mathcal{L}$ has state space $M = \mathtt{img}(t^\sharp_{\mathcal{L}})$, initial state $\mathsf{init} = t_{\mathcal{L}}(\varepsilon)$, and T-algebra homomorphisms $\mathsf{out}\colon M \to O$ and $\delta\colon M \to M^A$ given by $\mathsf{out}(t^\sharp_{\mathcal{L}}(U)) = \mathcal{L}(U)$ and $\delta(t^\sharp_{\mathcal{L}}(U))(a)(v) = t^\sharp_{\mathcal{L}}(U)(av)$.

4 A General Algorithm

In this section we introduce our extension of $\mathtt{L}^\star$ to learn automata with side-effects. The algorithm is parametric in the notion of side-effect, represented as a monad T, and is therefore called $\mathtt{L}^\star_T$. We fix a function $\mathcal{L}\colon A^* \to O$, which is the language to be learned.

An observation table consists of a pair of functions:

$$\mathsf{row_t}\colon S \to O^E \qquad \mathsf{row_t}(s)(e) = \mathcal{L}(se)$$
$$\mathsf{row_b}\colon S \to (O^E)^A \qquad \mathsf{row_b}(s)(a)(e) = \mathcal{L}(sae).$$

where $S, E \subseteq A^*$ are finite sets with $\varepsilon \in S \cap E$. For $O = 2$, we recover exactly the $\mathtt{L}^\star$ observation table. The key idea for $\mathtt{L}^\star_T$ is defining closedness and consistency over the free T-extensions of those functions.

▶ **Definition 9 (Closedness and Consistency).** The table is *closed* if for all $U \in T(S)$ and $a \in A$ there exists a $U' \in T(S)$ such that $\mathsf{row}^\sharp_{\mathsf{t}}(U') = \mathsf{row}^\sharp_{\mathsf{b}}(U)(a)$. The table is *consistent* if for all $U_1, U_2 \in T(S)$ such that $\mathsf{row}^\sharp_{\mathsf{t}}(U_1) = \mathsf{row}^\sharp_{\mathsf{t}}(U_2)$ we have $\mathsf{row}^\sharp_{\mathsf{b}}(U_1) = \mathsf{row}^\sharp_{\mathsf{b}}(U_2)$.

For closedness, we do not need to check all elements of $T(S)$, but only those of S, thanks to the following result.

▶ **Lemma 10.** *If for all $t \in S$ and $a \in A$ there is a $U \in T(S)$ such that $\mathsf{row}^\sharp_{\mathsf{t}}(U) = \mathsf{row_b}(t)(a)$, then the table is closed.*

▶ **Example 11.** For NFAs represented as $\mathcal{P}$-automata, the algorithm is precisely the one presented in Section 2.2. Recall that for $T = \mathcal{P}$ and $O = 2$, the Boolean join-semilattice, $\mathsf{row}^\sharp_{\mathsf{t}}$ and $\mathsf{row}^\sharp_{\mathsf{b}}$ describe a table where rows are labeled by finite subsets of S. Then we have, for instance, $\mathsf{row}^\sharp_{\mathsf{t}}(\{s_1, s_2\})(e) = \mathsf{row_t}(s_1)(e) \vee \mathsf{row_t}(s_2)(e)$, i.e., $\mathsf{row}^\sharp_{\mathsf{t}}(\{s_1, s_2\})(e) = 1$ if and only if $\mathcal{L}(s_1 e) = 1$ or $\mathcal{L}(s_2 e) = 1$. Closedness amounts to check whether each row in the bottom part of the table is the join of a set of rows in the top part. Consistency amounts to check whether, for any two sets of rows $U_1, U_2 \subseteq S$ in the top part of the table whose joins are equal, the joins of rows $U_1 \cdot \{a\}$ and $U_2 \cdot \{a\}$ in the bottom part are also equal, for all $a \in A$.

If these properties hold, we can define a hypothesis T-automaton.

▶ **Proposition 12 (Hypothesis).** *Provided the table is closed and consistent, we can build the following* hypothesis *T-automaton $\mathcal{H}$. Its state space is $H = \mathtt{img}(\mathsf{row}^\sharp_{\mathsf{t}})$, $\mathsf{init} = \mathsf{row_t}(\varepsilon)$, and output and transition maps are given by:*

$$\mathsf{out}\colon H \to O \qquad \mathsf{out}(\mathsf{row}^\sharp_{\mathsf{t}}(U)) = \mathsf{row}^\sharp_{\mathsf{t}}(U)(\varepsilon)$$
$$\delta\colon H \to H^A \qquad \delta(\mathsf{row}^\sharp_{\mathsf{t}}(U)) = \mathsf{row}^\sharp_{\mathsf{b}}(U).$$

```
1   S, E ← {ε}
2   repeat
3       while the table is not closed or not consistent
4           if the table is not closed
5               find t ∈ S, a ∈ A such that row_b(t)(a) ≠ row_t^♯(U) for all U ∈ T(S)
6               S ← S ∪ {ta}
7           if the table is not consistent
8               find U_1, U_2 ∈ T(S), a ∈ A, and e ∈ E such that
                    row_t^♯(U_1) = row_t^♯(U_2) and row_b^♯(U_1)(a)(e) ≠ row_b^♯(U_2)(a)(e)
9               E ← E ∪ {ae}
10      Construct the hypothesis H and submit it to the oracle
11      if the oracle replies no, with a counterexample z
12          S ← S ∪ prefixes(z)
13  until the oracle replies yes
14  return H
```

Figure 4 Adaptation of $\mathtt{L}^\star$ for T-automata

We can now give our algorithm $\mathtt{L}^\star_T$. In the same way as for the example in Section 2, we only have to adjust lines 5 and 8 in Figure 1. The resulting algorithm is shown in Figure 4.

Correctness and Termination. As in the original algorithm, termination implies correctness, since at the end the hypothesis is correct by definition of equivalence query.

Let $\mathcal{M}_\mathcal{L}$ be the target automaton, i.e., the minimal T-automaton accepting $\mathcal{L}$. We assume that the state space M of $\mathcal{M}_\mathcal{L}$ is a finite set. In order to show termination, we argue that the state space H of the hypothesis increases while the algorithm loops, and that H cannot be larger than M. In fact, when a closedness defect is resolved (line 6), a row that was not previously found in the image of $\mathsf{row}_\mathsf{t}^\sharp \colon T(S) \to O^E$ is added, so the set H grows larger. When a consistency defect is resolved (line 9), two previously equal rows become distinguished, which also increases the size of H.

As for counterexamples, adding their prefixes to S (line 11) creates a consistency defect, which will be fixed during the next iteration, causing H to increase. This is due to the following result, which says that the counterexample z has a prefix that violates consistency.

▶ **Proposition 13.** *If $z \in A^*$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$ and $\mathsf{prefixes}(z) \subseteq S$, then there are a prefix ua of z, with $u \in A^*$ and $a \in A$, and $U \in T(S)$ such that $\mathsf{row}_\mathsf{t}(u) = \mathsf{row}_\mathsf{t}^\sharp(U)$ and $\mathsf{row}_\mathsf{b}(u)(a) \neq \mathsf{row}_\mathsf{b}^\sharp(U)(a)$.*

Now, note that, by increasing S or E, the hypothesis state space H never decreases in size. Moreover, for $S = A^*$ and $E = A^*$, $\mathsf{row}_\mathsf{t}^\sharp = t_\mathcal{L}^\sharp$, as defined in Definition 21. Therefore, since H and M are defined as the images of those functions, the size of H is bounded by that of M. Since H increases while the algorithm loops, the algorithm must terminate and so is correct.

We note that the RFSA learning algorithm of Bollig et al. does not terminate using this counterexample processing method [4, Appendix F]. This is due to their notion of consistency being weaker than ours while we have shown that progress is guaranteed because a consistency defect, in our sense, is created using this method.

5 Succinct Hypotheses

We now describe the first of two optimizations that are enabled by the use of monads. Our algorithm produces hypotheses that can be quite large, as their state space is the image of $\mathsf{row}_\mathsf{t}^\sharp$, which has the whole set $T(S)$ as its domain. For instance, when $T = \mathcal{P}$, $T(S)$ is exponentially larger than S. We show how we can compute *succinct* hypotheses, whose state space is determined by a subset of S. We start by defining the set of *generators for the table*.

▶ **Definition 14.** A set $S' \subseteq S$ is a *set of generators for the table* whenever for all $s \in S$ there is a $U \in T(S')$ such that $\mathsf{row}_\mathsf{t}(s) = \mathsf{row}_\mathsf{t}^\sharp(U)$.[1]

Intuitively, U is the decomposition of s into a "combination" of generators. When $T = \mathcal{P}$, S' generates the table whenever each row can be obtained as the join of a set of rows labeled by S'. Explicitly: for all $s \in S$ there is $\{s_1, \ldots, s_n\} \subseteq S'$ such that $\mathsf{row}_\mathsf{t}(s) = \mathsf{row}_\mathsf{t}^\sharp(\{s_1, \ldots, s_n\}) = \mathsf{row}_\mathsf{t}(s_1) \vee \cdots \vee \mathsf{row}_\mathsf{t}(s_n)$.

Recall that $\mathcal{H}$, with state space H, is the hypothesis automaton for the table. The existence of generators S' allows us to compute a T-automaton with state space $T(S')$ equivalent to $\mathcal{H}$. We call this succinct hypothesis, although $T(S')$ may be larger than H, because the set S' is the only piece of information needed to compute it. In fact, this T-automaton can be turned into an automaton without algebraic structure, with state space S' and transition function $S' \to T(S')^A$. More formally, we observe that both the transition and output maps are T-algebra homomorphisms with domain $T(S')$. Therefore, by Proposition 3, they correspond to functions from S'. This results in a lower space complexity when storing the hypothesis.

We now show how the succinct hypothesis is computed. Observe that, if generators S' exist, $\mathsf{row}_\mathsf{t}^\sharp$ factors through the restriction of itself to $T(S')$. Denote this latter function $\widehat{\mathsf{row}_\mathsf{t}}^\sharp$. Since we have $T(S') \subseteq T(S)$, the image of $\widehat{\mathsf{row}_\mathsf{t}}^\sharp$ coincides with $\mathtt{img}(\mathsf{row}_\mathsf{t}^\sharp) = H$, and therefore the surjection restricting $\widehat{\mathsf{row}_\mathsf{t}}^\sharp$ to its image has the form $e\colon T(S') \to H$. Any right inverse $i\colon H \to T(S')$ of the function e (that is, $e \circ i = \mathsf{id}_H$, but whereas e is a T-algebra homomorphism, i need not be one) yields a succinct hypothesis as follows.

▶ **Definition 15 Succinct Hypothesis.** The *succinct hypothesis* is the T-automaton $\mathcal{S}$ defined as follows. Its state space is $T(S')$, its initial state is $\mathsf{init} = i(\mathsf{row}_\mathsf{t}(\varepsilon))$, and we have:

$$\begin{aligned} \mathsf{out}\colon T(S') &\to O & \mathsf{out}(U) &= \mathsf{row}_\mathsf{t}^\sharp(U)(\varepsilon) \\ \delta\colon T(S') &\to T(S')^A & \delta(U)(a) &= i(\mathsf{row}_\mathsf{b}^\sharp(U)(a)). \end{aligned}$$

This definition is inspired by that of *scoop* given in [3].

▶ **Proposition 16.** *Any succinct hypothesis of $\mathcal{H}$ accepts the same language as $\mathcal{H}$.*

We now give a simple procedure to compute a *minimal* set of generators, that is, a set S' such that no proper subset is a set of generators.

▶ **Proposition 17.** *The following algorithm returns a minimal set of generators for the table:*

$S' \leftarrow S$
while *there are $s \in S'$ and $U \in T(S' \setminus \{s\})$ s.t. $\mathsf{row}_\mathsf{t}^\sharp(U) = \mathsf{row}_\mathsf{t}(s)$*
  $S' \leftarrow S' \setminus \{s\}$
return S'

[1] Here and hereafter we assume that $T(S') \subseteq T(S)$, and more generally that T preserves inclusion maps. To eliminate this assumption, one could take the inclusion map $f\colon S' \hookrightarrow S$ and write $\mathsf{row}_\mathsf{t}^\sharp(T(f)(U))$ instead of $\mathsf{row}_\mathsf{t}^\sharp(U)$.

▶ **Example 18.** Consider again the powerset monad $T = \mathcal{P}$. We now exemplify two ways of computing succinct hypotheses, which are inspired by how RFSAs are computed in [7]. The basic idea is to start from a deterministic automaton and to remove states that are equivalent to a set of other states. In our setting, this corresponds to the algorithm given in Proposition 17: the minimal S' only contains labels of rows that are not the join of other rows (in case two rows are equal, only one of their labels is kept). In other words, as mentioned in Section 2, S' contains labels of join-irreducible rows.

The function $e\colon \mathcal{P}(S') \to H$ tells us which sets of rows are equivalent to a single state in H. We show two different right inverses $i\colon H \to \mathcal{P}(S')$ for it. The first one,

$$i_1(h) = \{s \in S' \mid \mathsf{row_t}(s) \leq h\},$$

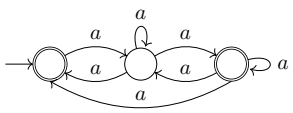
stems from the construction of the *canonical RFSA* of a language. Here we use the order $a \leq b \iff a \vee b = b$ induced by the join-semilattice structure. The resulting construction of a succinct hypothesis was first used by Bollig et al. [4]. This succinct hypothesis has a "maximal" transition function, meaning that no more transitions can be added without changing the language of the automaton.

The second inverse is:

$$\begin{aligned} i_2(h) = \{&s \in S' \mid \mathsf{row_t}(s) \leq h \text{ and} \\ &\text{for all } s' \in S' \text{ s.t. } \mathsf{row_t}(s') \leq h \text{ and } \mathsf{row_t}(s) \leq \mathsf{row_t}(s') \text{ we have } \mathsf{row_t}(s) = \mathsf{row_t}(s')\} \end{aligned}$$

resulting in a more economical transition function, where some redundancies are removed. This corresponds to the *simplified canonical RFSA* of [7].

▶ **Example 19.** Consider again the powerset monad $T = \mathcal{P}$, and recall the table in Figure 3c. When $S' = S$, the right inverse given by i_1 yields the succinct hypothesis shown below.



Note that $i_1(\mathsf{row_t}(aa)) = \{\varepsilon, a, aa\}$. When instead taking i_2, the succinct hypothesis is just the DFA (1) because $i_2(\mathsf{row_t}(aa)) = \{aa\}$. Rather than constructing a succinct hypothesis directly, our algorithm first reduces the set S'. In this case, we note that $\mathsf{row_t}(aa) = \mathsf{row_t^\sharp}(\{\varepsilon, a\})$, so we can remove aa from S'. Now i_1 and i_2 coincide and produce the NFA (2).

▶ Remark. The algorithm in Proposition 17 implicitly assumes an order in which elements of S are checked. Although the algorithm is correct for any such order, different orders may give results that differ in size, as shown in Example B.1.

6 Optimized Counterexample Handling

The second optimization we give generalizes the counterexample processing method due to Rivest and Schapire [12], which improves the worst case complexity of the number of membership queries needed in the learning algorithm. Instead of adding all prefixes of the counterexample to the set S, thereby increasing the number of rows in the table, this method finds a single *suffix* of the counterexample and adds it to E, increasing the number of columns. This suffix is chosen in such a way that it either distinguishes two existing rows or creates a closedness defect, both of which imply that the hypothesis automaton will grow.

The main idea is finding the distinguishing suffix via the hypothesis automaton $\mathcal{H}$. Given a word $u \in A^*$, let q_u be the state in $\mathcal{H}$ reached by reading u: $q_u = r_{\mathcal{H}}(u)$. For each $q \in H$, we pick any $U_q \in T(S)$ that yields q according to the table, i.e., such that $\mathsf{row}_\mathsf{t}^\sharp(U_q) = q$. Then for a counterexample z we have that the residual language w.r.t. U_{q_z} does not "agree" with the residual language w.r.t. z.

The above intuition can be formalized as follows. Let $\mathcal{R}\colon A^* \to O^{A^*}$ be given by $\mathcal{R}(u) = t_\mathcal{L}^\sharp(U_{q_u})$ for all $u \in A^*$, the residual language computation. We have the following technical lemma, saying that a counterexample z distinguishes the residual languages $t_\mathcal{L}(z)$ and $\mathcal{R}(z)$.

▶ **Lemma 20.** *If $z \in A^*$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$, then $t_\mathcal{L}(z)(\varepsilon) \neq \mathcal{R}(z)(\varepsilon)$.*

We assume that $U_{q_\varepsilon} = \eta(\varepsilon)$. Then for a counterexample z, $\mathcal{R}(\varepsilon)(z) = t_\mathcal{L}(\varepsilon)(z) \neq \mathcal{R}(z)(\varepsilon)$. While reading z, the hypothesis automaton passes a sequence of states q_{u_0}, q_{u_1},q_{u_2},$\ldots$,q_{u_n}, where $u_0 = \epsilon$, $u_n = z$, and $u_{i+1} = u_i a$ for some $a \in A$ is a prefix of z. If z were correctly classified by $\mathcal{H}$, all residuals $\mathcal{R}(u_i)$ would classify the remaining suffix v of z, i.e., such that $z = u_i v$, in the same way. However, the previous lemma tells us that, for a counterexample z, this is not case, meaning that for some suffix v we have $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$. In short, this inequality is discovered along a transition in the path to z.

▶ **Corollary 21.** *There are $u, v \in A^*$ and $a \in A$ such that $uav = z$ and $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$.*

To find such a decomposition efficiently, Rivest and Schapire use a binary search algorithm. We conclude with the following result that turns the above property into the elimination of a closedness witness. That is, given a counterexample z and the resulting decomposition uav from the above corollary, we show that, while currently $\mathsf{row}_\mathsf{t}^\sharp(U_{q_{ua}}) = \mathsf{row}_\mathsf{b}^\sharp(U_{q_u})(a)$, after adding v to E we have $\mathsf{row}_\mathsf{t}^\sharp(U_{q_{ua}})(v) \neq \mathsf{row}_\mathsf{b}^\sharp(U_{q_u})(a)(v)$. (To see that the latter follows from the proposition below, note that for all $U \in T(S)$ and $e \in E$, $\mathsf{row}_\mathsf{t}^\sharp(U)(e) = t_\mathcal{L}^\sharp(U)(e)$ and for each $a' \in A$, $\mathsf{row}_\mathsf{b}^\sharp(U)(a')(e) = t_\mathcal{L}^\sharp(U)(a'e)$, by the definition of those maps.) The inequality means that either we have a closedness defect, or there still exists some $U \in T(S)$ such that $\mathsf{row}_\mathsf{t}^\sharp(U) = \mathsf{row}_\mathsf{b}^\sharp(U_{q_u})(a)$. In this case, the rows $\mathsf{row}_\mathsf{t}^\sharp(U)$ and $\mathsf{row}_\mathsf{t}^\sharp(U_{q_{ua}})$ have become distinguished by adding v, which means that the size of H has been increased. We know that a closedness defect leads to an increase in the size of H, so in any case we make progress.

▶ **Proposition 22.** *If $z \in A^*$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$, then there are $u, v \in A^*$ and $a \in A$ such that $\mathsf{row}_\mathsf{t}^\sharp(U_{q_{ua}}) = \mathsf{row}_\mathsf{b}^\sharp(U_{q_u})(a)$ and $t_\mathcal{L}^\sharp(U_{q_{ua}})(v) \neq t_\mathcal{L}^\sharp(U_{q_u})(av)$.*

We now show how to combine this optimized counterexample processing method with the succinct hypothesis optimization from Section 5. Recall that the succinct hypothesis $\mathcal{S}$ is based on a right inverse $i\colon H \to T(S')$ of $e\colon T(S') \to H$. Given $q \in H$, we define $U_q = i(q)$, which by the right inverse property satisfies $\mathsf{row}_\mathsf{t}^\sharp(U_q) = q$. We then have that the function $\mathcal{R}$ can conveniently be computed using the reachability map of the succinct hypothesis.

▶ **Proposition 23.** *For all $u \in A^*$, $\mathcal{R}(u) = t_\mathcal{L}^\sharp(r_\mathcal{S}(u))$.*

Unfortunately, there is one complication. We assumed earlier that $U_{q_\varepsilon} = \eta(\varepsilon)$, or more specifically $\mathcal{R}(\varepsilon)(z) = t_\mathcal{L}(\varepsilon)(z)$. This now may be impossible because we do not even necessarily have $\varepsilon \in S'$. We show next that if this equality does not hold, then there are two rows that we can distinguish by adding z to E. Thus, after testing whether $\mathcal{R}(\varepsilon)(z) = t_\mathcal{L}(\varepsilon)(z)$, we either add z to E (if the test fails) or proceed with the original method.

▶ **Proposition 24.** *If $z \in A^*$ is such that $\mathcal{R}(\varepsilon)(z) \neq t_\mathcal{L}(\varepsilon)(z)$, then $\mathsf{row}_\mathsf{t}^\sharp(\mathsf{init}_\mathcal{S}) = \mathsf{row}_\mathsf{t}(\varepsilon)$ and $t_\mathcal{L}^\sharp(\mathsf{init}_\mathcal{S})(z) \neq t_\mathcal{L}(\varepsilon)(z)$.*

▶ **Example 25.** Recall the succinct hypothesis $\mathcal{S}$ from Figure 3a for the table in Figure 2a. Note that $S' = S$ cannot be further reduced. The hypothesis is based on the right inverse $i\colon H \to \mathcal{P}(S)$ of $e\colon \mathcal{P}(S) \to H$ given by $i(\mathsf{row}_\mathsf{t}(\varepsilon)) = \{\varepsilon\}$ and $i(\mathsf{row}^\sharp_\mathsf{t}(\emptyset)) = \emptyset$. This is in fact the only possible right inverse because e is bijective. For the prefixes of the counterexample aa we have $r_\mathcal{S}(\varepsilon) = \{\varepsilon\}$ and $r_\mathcal{S}(a) = r_\mathcal{S}(aa) = \emptyset$. Note that $t^\sharp_\mathcal{L}(\{\varepsilon\})(aa) = 1$ while $t_\mathcal{L}(\emptyset)(a) = t_\mathcal{L}(\emptyset)(\varepsilon) = 0$. Thus, $\mathcal{R}(\varepsilon)(aa) \neq \mathcal{R}(a)(a)$. Adding a to E would indeed create a closedness defect.

Dropping Consistency. We have shown how the counterexample processing method based around Proposition 22 can be described in terms of the succinct hypothesis $\mathcal{S}$ rather than the actual hypothesis $\mathcal{H}$. We did this by showing that $\mathcal{R}$ can be defined using the succinct hypothesis, which meant the proposition could be reused directly. Interestingly, it is also possible to adapt the proofs of Proposition 22 and the preceding Lemma 20 to only refer to the succinct hypothesis instead of the actual hypothesis. Since the definition of the succinct hypothesis does not rely on the property of consistency to be well-defined, this means we could drop the consistency check from the algorithm altogether. We can still measure progress in terms of the size of the set H, but it will simply not be the state space of an actual hypothesis during intermediate stages. This observation also explains why Bollig et al. [4] are able to use a weaker notion of consistency in their algorithm.

7 Examples

In this section we list several examples that admit a representation as T-automata, and hence can be learnt via an instance of $\mathtt{L}^\star_T$. We remark that since our algorithm operates on finite structures (recall that T preserves finite sets), implementing a "naive" instance of $\mathtt{L}^\star_T$ for each automaton type is straightforward. Due to our general treatment, the optimizations of Sections 5 and 6 apply to all of these instances.

Universal automata. Just like non-deterministic automata, universal automata can be seen as $\mathcal{P}$-automata with a free state space. The difference, however, is that the $\mathcal{P}$-algebra structure on $O = 2$ is dual: it is given by the "and" rather than the "or" operation. Universal automata accept a word when all paths reading that word are accepting.

Partial automata. Consider the *maybe monad* $\mathtt{Maybe}$, given by $\mathtt{Maybe}(X) = 1 + X$, with natural transformations having components $\eta_X\colon X \to 1 + X$ and $\mu_X\colon 1 + 1 + X \to 1 + X$ defined in the obvious way. Partial automata with states X can be represented as $\mathtt{Maybe}$-automata with state space $\mathtt{Maybe}(X) = 1 + X$, where there is an additional *sink state*, and output algebra $O = \mathtt{Maybe}(1) = 1 + 1$, the left value is for rejecting states (including the sink one). The transition map $\delta\colon 1 + X \to (1 + X)^A$ represents an undefined transition as one going to the sink state.

Weighted automata. Recall from Section 3 the *free semimodule monad* V, sending a set X to the free semimodule over a finite semiring $\mathbb{S}$. Weighted automata over a set of states X can be represented as V-automata whose state space is the vector space $V(X)$, the output function $\mathsf{out}\colon V(X) \to \mathbb{S}$ assigns a weight to each state, and the transition map $\delta\colon V(X) \to V(X)^A$ sends each state and each input symbol to a linear combination of states.

Alternating automata. Consider the monad $\mathtt{A}(X) = \mathcal{P}\mathcal{P}(X)$, where the two powersets are interpreted differently: the outer one gives join-semilattices where join is union, the inner

one gives meet-semilattices where meet is intersection. The unit is just $\eta_X(x) = \{\{x\}\}$ and the multiplication is defined using the natural transformation $\chi\colon \mathcal{PP} \Rightarrow \mathcal{PP}$ given in [9]:

$$\chi_X(S) = \{V \subseteq \bigcup S \mid V \cap U \neq \emptyset \quad \text{for each } U \in S\}$$

Intuitively, $\chi_X(S)$ turns a CNF formula into a DNF formula by distributing conjunctions over disjunctions. This can be used to define the multiplication of $\mathtt{A}$ by composition with the multiplications for the two powersets. Alternating automata with states X can be represented as $\mathtt{A}$-automata with state space $\mathcal{PP}(X)$, output map $\mathtt{out}\colon \mathcal{PP}(X) \to \mathcal{PP}(\emptyset) = 2$, and transition map $\delta\colon \mathcal{PP}(X) \to \mathcal{PP}(X)^A$, sending each state to a DNF formula over X.

Monoid automata. Consider the monad $\mathtt{M}(X) = \mathbb{M} \times X$ for a finite monoid $\mathbb{M}$. Its unit $\eta_X\colon X \to \mathbb{M} \times X$ adds the unit e of the monoid, $\eta_X(x) = (e, x)$, and its multiplication $\mu_X\colon \mathbb{M} \times \mathbb{M} \times X$ performs the monoid multiplication, $\mu_X(m_1, m_2, x) = (m_1 m_2, x)$. The algebras for this monad are sets with an $\mathbb{M}$-action. One may take the output object to be the set $\mathbb{M}$ with the monoid multiplication as its action. $\mathtt{M}$-automata with a free state space can be represented as deterministic automata that have an element of $\mathbb{M}$ associated with each transition. The semantics of these is that the encountered $\mathbb{M}$-elements multiply along paths and finally multiply with the output of the last state to produce the actual output.

8 Conclusion

We have presented $\mathtt{L}^\star_T$, a general adaptation of $\mathtt{L}^\star$ to learn an automaton with algebraic structure, as well as a method for finding a succinct equivalent based on its generators. Furthermore, we adapted the optimized counterexample handling method of Rivest and Schapire [12] to this setting and discussed instantiations to non-deterministic, universal, partial, weighted, alternating, and monoid automata. This paper is part of larger effort to develop a categorical automata learning framework (CALF), in which learning algorithms can be understood and developed in a structured way. For more details, including practical applications, see our project website – `http://www.calf-project.org`.

Related Work. An adaptation of $\mathtt{L}^\star$ that produces NFAs was first developed by Bollig et al. [4]. Their algorithm learns a special subclass of NFAs consisting of RFSAs, which were introduced by Denis et al. [7]. Angluin et al. [2] hinted at a more general framework by unifying algorithms for NFAs, universal automata, and alternating automata. The algorithm for weighted automata over a (not necessarily finite) field was introduced in a category theoretical context by Jacobs and Silva [8] and elaborated on by Van Heerdt et al. [17]. The theory of succinct automata used for our hypotheses was developed by Arbib and Manes [3], and a general algorithm for finding a minimal set of generators was given recently in [16].

Future Work. Whereas our general naive algorithm effortlessly instantiates to monads that preserve finite sets, a major challenge lies in investigating monads that do not enjoy this property. Although the algorithm for weighted automata generalizes to an infinite field [8, 17], for an infinite semiring in general we cannot guarantee termination. This is because a finitely generated semimodule may have an infinite chain of strict submodules. We can, however, do it for Noetherian semirings, since submodules of finitely generated ones are still finitely generated. A generalization to proper semirings, recently studied by Milius [11], should also be possible. Moreover, we expect that $\mathtt{L}^\star_T$ can be generalized to l.f.p. categories and finitary monads instead of sets and monads preserving finite sets. We are currently exploiting verification applications of the learning algorithm and we aim at having benchmarks for the various versions presented.

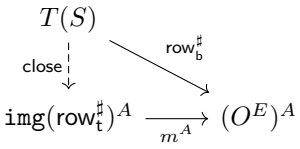
References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75:87–106, 1987.
2. Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *IJCAI*, pages 3308–3314, 2015.
3. Michael A. Arbib and Ernest G. Manes. Fuzzy machines in a category. *Bulletin of the AMS*, 13:169–210, 1975.
4. Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, volume 9, pages 1004–1009, 2009.
5. 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.
6. Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security*, pages 193–206, 2015.
7. François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51:339–368, 2002.
8. Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, volume 8464, pages 384–406, 2014.
9. Bartek Klin and Jurriaan Rot. Coalgebraic trace semantics via forgetful logics. *LMCS*, 12(4), 2016. doi:10.2168/LMCS-12(4:10)2016.
10. Dexter C. Kozen. *Automata and computability*. Springer Science & Business Media, 2012.
11. Stefan Milius. Proper functors. 2017. Submitted, copy obtained in personal communication.
12. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inform. Comput.*, 103:299–347, 1993.
13. 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.
14. Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017. doi:10.1145/2967606.
15. Gerco van Heerdt. An abstract automata learning framework. Master's thesis, Radboud University Nijmegen, 2016.
16. Gerco van Heerdt, Joshua Moerman, Matteo Sammartino, and Alexandra Silva. A (co)algebraic theory of succinct acceptors. 2017. http://www.calf-project.org/publications/succ.pdf.
17. Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. CALF: Categorical Automata Learning Framework, 2017. arXiv:1704.05676.

A Proofs for Section 4 (A General Algorithm)

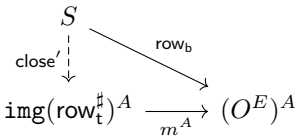
▶ **Lemma 10.** *If for all $t \in S$ and $a \in A$ there is a $U \in T(S)$ such that $\mathsf{row}_\mathsf{t}^\sharp(U) = \mathsf{row}_\mathsf{b}(t)(a)$, then the table is closed.*

Proof. Let $m\colon \mathtt{img}(\mathsf{row}_\mathsf{t}^\sharp) \hookrightarrow O^E$ be the embedding of the image of $\mathsf{row}_\mathsf{t}^\sharp$ into its codomain. According to [vHSS17], the definition of closedness given in Definition 9 amounts to requiring the existence of a T-algebra homomorphism close making the following diagram commute:

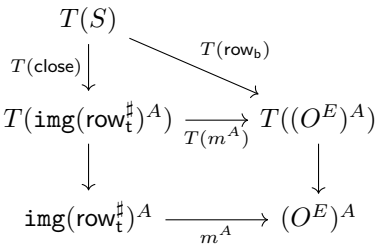


(3)

It is easy to see that the hypothesis of this lemma corresponds to requiring the existence of a function close' making the following diagram in **Set** commute:



This diagram can be made into a diagram of T-algebra homomorphisms as follows:



where the composition of the left and right legs give respectively $\mathsf{close}'^\sharp$ and $\mathsf{row}_\mathsf{b}^\sharp$. This diagram commutes because the top triangle commutes by functoriality of T, and the bottom square commutes by m^A being a T-algebra homomorphism. Therefore we have that (3) commutes for $\mathsf{close} = \mathsf{close}'^\sharp$. ◀

▶ **Proposition 12 (Hypothesis).** *Provided the table is closed and consistent, we can build the following* hypothesis *T-automaton $\mathcal{H}$. Its state space is $H = \mathtt{img}(\mathsf{row}_\mathsf{t}^\sharp)$, $\mathsf{init} = \mathsf{row}_\mathsf{t}(\varepsilon)$, and output and transition maps are given by:*

$$\mathsf{out}\colon H \to O \qquad \mathsf{out}(\mathsf{row}_\mathsf{t}^\sharp(U)) = \mathsf{row}_\mathsf{t}^\sharp(U)(\varepsilon)$$
$$\delta\colon H \to H^A \qquad \delta(\mathsf{row}_\mathsf{t}^\sharp(U)) = \mathsf{row}_\mathsf{b}^\sharp(U).$$

Proof. Follows directly from the abstract treatment given in [vHSS17], instantiated to the category of T-algebras and their homomorphisms. ◀

▶ **Proposition 13.** *If $z \in A^*$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$ and $\mathsf{prefixes}(z) \subseteq S$, then there are a prefix ua of z, with $u \in A^*$ and $a \in A$, and $U \in T(S)$ such that $\mathsf{row}_\mathsf{t}(u) = \mathsf{row}_\mathsf{t}^\sharp(U)$ and $\mathsf{row}_\mathsf{b}(u)(a) \neq \mathsf{row}_\mathsf{b}^\sharp(U)(a)$.*

Proof. Note that

$$\begin{aligned}
\mathsf{row_t}(z)(\varepsilon) &= \mathcal{L}(z) && \text{(definition of } \mathsf{row_t}\text{)}\\
&\neq \mathcal{L}_{\mathcal{H}}(z) && \text{(assumption)}\\
&= \mathsf{out}_{\mathcal{H}}(r_{\mathcal{H}}(z)) && \text{(Definition of } \mathcal{L}_{\mathcal{H}}\text{)}\\
&= r_{\mathcal{H}}(z)(\varepsilon) && \text{(definition of } \mathsf{out}_{\mathcal{H}}\text{),}
\end{aligned}$$

so $\mathsf{row_t}(z) \neq r_{\mathcal{H}}(z)$. Let $p \in A^*$ be the smallest prefix of z satisfying $\mathsf{row_t}(p) \neq r_{\mathcal{H}}(p)$. We have $\mathsf{row_t}(\varepsilon) = \mathsf{init}_{\mathcal{H}} = r_{\mathcal{H}}(\varepsilon)$, so $p \neq \varepsilon$ and therefore $p = ua$ for certain $u \in A^*$ and $a \in A$. Choose any $U \in T(S)$ such that $\mathsf{row}_\mathsf{t}^\sharp(U) = r_{\mathcal{H}}(u)$, which is possible because H is the image of $\mathsf{row}_\mathsf{t}^\sharp$ restricted to the domain $T(S)$. By the minimality property of p we have $\mathsf{row_t}(u) = r_{\mathcal{H}}(u)$, so $\mathsf{row_t}(u) = r_{\mathcal{H}}(u) = \mathsf{row}_\mathsf{t}^\sharp(U)$. Furthermore,

$$\begin{aligned}
\mathsf{row_b}(u)(a) &= \mathsf{row_t}(ua) && \text{(definitions of } \mathsf{row_t} \text{ and } \mathsf{row_b}\text{)}\\
&\neq r_{\mathcal{H}}(ua) && (ua = p \text{ and } \mathsf{row_t}(p) \neq r_{\mathcal{H}}(p))\\
&= \delta_{\mathcal{H}}(r_{\mathcal{H}}(u))(a) && \text{(definition of } r_{\mathcal{H}}\text{)}\\
&= \delta_{\mathcal{H}}(\mathsf{row}_\mathsf{t}^\sharp(U))(a) && (r_{\mathcal{H}}(u) = \mathsf{row}_\mathsf{t}^\sharp(U))\\
&= \mathsf{row}_\mathsf{b}^\sharp(U)(a) && \text{(definition of } \delta_{\mathcal{H}}\text{).}
\end{aligned}$$

◀

B Proofs for Section 5 (Succinct Hypotheses)

▶ **Proposition 16.** *Any succinct hypothesis of $\mathcal{H}$ accepts the same language as $\mathcal{H}$.*

Proof. Assume a right inverse $i\colon H \to T(S')$ of $e\colon T(S') \to H$. We first prove $o_{\mathcal{H}} \circ e = o_{\mathcal{S}}$, by induction on the length of words. For all $U \in T(S')$, we have

$$\begin{aligned}
o_{\mathcal{H}}(e(U))(\varepsilon) &= \mathsf{out}_{\mathcal{H}}(e(U)) && \text{(definition of } o_{\mathcal{H}}\text{)}\\
&= \mathsf{out}_{\mathcal{H}}(\mathsf{row}_\mathsf{t}^\sharp(U)) && \text{(definition of } e\text{)}\\
&= \mathsf{row}_\mathsf{t}^\sharp(U)(\varepsilon) && \text{(definition of } \mathsf{out}_{\mathcal{H}}\text{)}\\
&= \mathsf{out}_{\mathcal{S}}(U) && \text{(definition of } \mathsf{out}_{\mathcal{S}}\text{)}\\
&= o_{\mathcal{S}}(U)(\varepsilon) && \text{(definition of } o_{\mathcal{S}}\text{).}
\end{aligned}$$

Now assume that for a given $v \in A^*$ and all $U \in T(S')$ we have $o_{\mathcal{H}}(e(U))(v) = o_{\mathcal{S}}(U)(v)$. Then, for all $U \in T(S')$ and $a \in A$,

$$\begin{aligned}
o_{\mathcal{H}}(e(U))(av) &= o_{\mathcal{H}}(\delta_{\mathcal{H}}(e(U))(a))(v) && \text{(definition of } o_{\mathcal{H}}\text{)}\\
&= o_{\mathcal{H}}(\delta_{\mathcal{H}}(\mathsf{row}_\mathsf{t}^\sharp(U))(a))(v) && \text{(definition of } e\text{)}\\
&= o_{\mathcal{H}}(\mathsf{row}_\mathsf{b}^\sharp(U)(a))(v) && \text{(definition of } \delta_{\mathcal{H}}\text{)}\\
&= (o_{\mathcal{H}} \circ e \circ i)(\mathsf{row_b}(U)(a))(v) && (e \circ i = \mathsf{id}_H)\\
&= (o_{\mathcal{S}} \circ i)(\mathsf{row_b}(U)(a))(v) && \text{(induction hypothesis)}\\
&= o_{\mathcal{S}}(\delta_{\mathcal{S}}(U)(a))(v) && \text{(definition of } \delta_{\mathcal{S}}\text{)}\\
&= o_{\mathcal{S}}(U)(av) && \text{(definition of } o_{\mathcal{S}}\text{).}
\end{aligned}$$

From this we see that

$$\begin{aligned} o_{\mathcal{S}}(\mathsf{init}_{\mathcal{S}}) &= (o_{\mathcal{S}} \circ i \circ \mathsf{row}_{\mathsf{t}})(\varepsilon) && \text{(definition of } \mathsf{init}_{\mathcal{S}}) \\ &= (o_{\mathcal{H}} \circ e \circ i \circ \mathsf{row}_{\mathsf{t}})(\varepsilon) && (o_{\mathcal{H}} \circ e = o_{\mathcal{S}}) \\ &= (o_{\mathcal{H}} \circ \mathsf{row}_{\mathsf{t}})(\varepsilon) && (e \circ i = \mathsf{id}_H) \\ &= o_{\mathcal{H}}(\mathsf{init}_{\mathcal{H}}) && \text{(definition of } \mathsf{init}_{\mathcal{H}}). \end{aligned}$$ ◀

▶ **Proposition 17.** *The following algorithm returns a minimal set of generators for the table:*

$S' \leftarrow S$
while *there are* $s \in S'$ *and* $U \in T(S' \setminus \{s\})$ *s.t.* $\mathsf{row}_{\mathsf{t}}^\sharp(U) = \mathsf{row}_{\mathsf{t}}(s)$
  $S' \leftarrow S' \setminus \{s\}$
return S'

Proof. Minimality is obvious, as S' not being minimal would make the loop guard true.

We prove that the returned set is a set of generators. For clarity, we denote by $d_{S'} \colon S \to T(S')$ the function associated with a set of generators S'. The main idea is incrementally building $d_{S'}$ while building S'. In the first line, S is a set of generators, with $d_S = \eta_S \colon S \to T(S)$. For the loop, suppose S' is a set of generators. If the loop guard is false, the algorithm returns the set of generators S'. Otherwise, suppose there are there are $s \in S$ and $U \in T(S' \setminus \{s\})$ such that $\mathsf{row}_{\mathsf{t}}^\sharp(U) = \mathsf{row}_{\mathsf{t}}(s)$. Then there is a function

$$f \colon S' \to T(S' \setminus \{s\}) \qquad\qquad f(s') = \begin{cases} U & \text{if } s' = s \\ \eta(s') & \text{if } s' \neq s \end{cases}$$

that satisfies $\mathsf{row}_{\mathsf{t}}(s') = \mathsf{row}_{\mathsf{t}}^\sharp(f(s'))$ for all $s' \in S'$, from which it follows that $\mathsf{row}_{\mathsf{t}}^\sharp(U') = \mathsf{row}_{\mathsf{t}}^\sharp(f^\sharp(U'))$ for all $U' \in T(S')$. Then we can set $d_{S' \setminus \{s\}}$ to $f^\sharp \circ d_{S'} \colon S \to T(S' \setminus \{s\})$ because $\mathsf{row}_{\mathsf{t}}(s') = \mathsf{row}_{\mathsf{t}}^\sharp(d_{S' \setminus \{s\}}(s'))$ for all $s' \in S$. Therefore, $S' \setminus \{s\}$ is a set of generators. ◀

▶ **Example B.1.** Consider the free semimodule monad V for the commutative semiring $\mathbb{Z}_6$ of integers modulo 6, the free output object $\mathbb{Z}_6 \cong V(1)$, and the language $\mathcal{L} \colon \{a\}^* \to \mathbb{Z}_6$ given by $\mathcal{L}(a^n) = (n+1) \bmod 6$. The observation table for $S = \{\varepsilon, a, aa\}$ and $E = \{\varepsilon\}$ is given below.

	ε
ε	1
a	2
aa	3
aaa	4

We initialize $S' = S$. Note that $\mathsf{row}_{\mathsf{t}}(\varepsilon) = \mathsf{row}_{\mathsf{t}}^\sharp(2 \times a + aa)$, so we can remove ε from S'. After doing so, however, neither a nor aa can be removed from S'. If, instead, from $S' = S$ we observe that $\mathsf{row}_{\mathsf{t}}(a) = \mathsf{row}_{\mathsf{t}}^\sharp(2 \times \varepsilon)$ and $\mathsf{row}_{\mathsf{t}}(aa) = \mathsf{row}_{\mathsf{t}}^\sharp(3 \times \varepsilon)$, we may end up with the singleton $S' = \{\varepsilon\}$.

C Proofs for Section 6 (Optimized Counterexample Handling)

▶ **Lemma 20.** *If* $z \in A^*$ *is such that* $\mathcal{L}_{\mathcal{H}}(z) \neq \mathcal{L}(z)$*, then* $t_{\mathcal{L}}(z)(\varepsilon) \neq \mathcal{R}(z)(\varepsilon)$.

Proof. We have

$$\begin{aligned}
t_{\mathcal{L}}(z)(\varepsilon) &= \mathcal{L}(z) && \text{(definition of } t_{\mathcal{L}}\text{)}\\
&\neq \mathcal{L}_{\mathcal{H}}(z) && \text{(assumption)}\\
&= (\mathsf{out}_{\mathcal{H}} \circ r_{\mathcal{H}})(z) && \text{(definition of } \mathcal{L}_{\mathcal{H}}\text{)}\\
&= r_{\mathcal{H}}(z)(\varepsilon) && \text{(definition of } \mathsf{out}_{\mathcal{H}}\text{)}\\
&= q_z(\varepsilon) && \text{(definition of } q_z\text{)}\\
&= \mathsf{row}_{\mathsf{t}}^{\sharp}(U_{q_z})(\varepsilon) && \text{(definition of } U_{q_z}\text{)}\\
&= t_{\mathcal{L}}^{\sharp}(U_{q_z})(\varepsilon) && \text{(definitions of } \mathsf{row}_{\mathsf{t}} \text{ and } t_{\mathcal{L}}\text{)}\\
&= \mathcal{R}(z)(\varepsilon) && \text{(definition of } \mathcal{R}\text{).}
\end{aligned}$$

◀

▶ **Proposition 22.** *If $z \in A^*$ is such that $\mathcal{L}_{\mathcal{H}}(z) \neq \mathcal{L}(z)$, then there are $u, v \in A^*$ and $a \in A$ such that $\mathsf{row}_{\mathsf{t}}^{\sharp}(U_{q_{ua}}) = \mathsf{row}_{\mathsf{b}}^{\sharp}(U_{q_u})(a)$ and $t_{\mathcal{L}}^{\sharp}(U_{q_{ua}})(v) \neq t_{\mathcal{L}}^{\sharp}(U_{q_u})(av)$.*

Proof. By Corollary 21 we have $u, v \in A^*$ and $a \in A$ such that $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$. This directly yields the inequality by the definition of $\mathcal{R}$. Furthermore,

$$\begin{aligned}
\mathsf{row}_{\mathsf{t}}(U_{q_{ua}}) &= q_{ua} && \text{(definition of } U_{q_{ua}}\text{)}\\
&= r_{\mathcal{H}}(ua) && \text{(definition of } q_{ua}\text{)}\\
&= \delta_{\mathcal{H}}(r_{\mathcal{H}}(u))(a) && \text{(definition of } r_{\mathcal{H}}\text{)}\\
&= \delta_{\mathcal{H}}(q_u)(a) && \text{(definition of } q_u\text{)}\\
&= \delta_{\mathcal{H}}(\mathsf{row}_{\mathsf{t}}^{\sharp}(U_{q_u}))(a) && \text{(definition of } U_{q_u}\text{)}\\
&= \mathsf{row}_{\mathsf{b}}^{\sharp}(U_{q_u})(a) && \text{(definition of } \delta_{\mathcal{H}}\text{).}
\end{aligned}$$

◀

▶ **Proposition 23.** *For all $u \in A^*$, $\mathcal{R}(u) = t_{\mathcal{L}}^{\sharp}(r_{\mathcal{S}}(u))$.*

Proof. We have

$$\begin{aligned}
\mathcal{R}(u) &= t_{\mathcal{L}}^{\sharp}(U_{q_u}) && \text{(definition of } h\text{)}\\
&= t_{\mathcal{L}}^{\sharp}(i(q_u)) && \text{(definition of } U_{q_u}\text{)}\\
&= t_{\mathcal{L}}^{\sharp}(i(r_{\mathcal{H}}(u))) && \text{(definition of } q_u\text{)}
\end{aligned}$$

and prove by induction on the length of u that $r_{\mathcal{S}}(u) = i(r_{\mathcal{H}}(u))$. For $u = \varepsilon$,

$$\begin{aligned}
r_{\mathcal{S}}(\varepsilon) &= \mathsf{init}_{\mathcal{S}} && \text{(definition of } r_{\mathcal{S}}\text{)}\\
&= i(\mathsf{row}_{\mathsf{t}}(\varepsilon)) && \text{(definition of } \mathsf{init}_{\mathcal{S}}\text{)}\\
&= i(r_{\mathcal{H}}(\varepsilon)) && \text{(definition of } r_{\mathcal{H}}\text{).}
\end{aligned}$$

Now assume that $u = va$ for certain $v \in A^*$ and $a \in A$, and $r_{\mathcal{S}}(v) = i(r_{\mathcal{H}}(v))$. Then

$$\begin{aligned}
r_{\mathcal{S}}(va) &= \delta_{\mathcal{S}}(r_{\mathcal{S}}(v))(a) && \text{(definition of } r_{\mathcal{S}}\text{)}\\
&= i(\mathsf{row}_{\mathsf{b}}^{\sharp}(r_{\mathcal{S}}(v))(a)) && \text{(definition of } \delta_{\mathcal{S}}\text{)}\\
&= i(\delta_{\mathcal{H}}(\mathsf{row}_{\mathsf{t}}^{\sharp}(r_{\mathcal{S}}(v)))(a)) && \text{(definition of } \delta_{\mathcal{H}}\text{)}\\
&= i(\delta_{\mathcal{H}}(\mathsf{row}_{\mathsf{t}}^{\sharp}(i(r_{\mathcal{H}}(v))))(a)) && \text{(induction hypothesis)}\\
&= i(\delta_{\mathcal{H}}(r_{\mathcal{H}}(v))(a)) && \text{(definition of } i\text{)}\\
&= i(r_{\mathcal{H}}(va)) && \text{(definition of } r_{\mathcal{H}}\text{).}
\end{aligned}$$

◀

▶ **Proposition 24.** *If $z \in A^*$ is such that $\mathcal{R}(\varepsilon)(z) \neq t_{\mathcal{L}}(\varepsilon)(z)$, then $\mathsf{row}_{\mathsf{t}}^\sharp(\mathsf{init}_{\mathcal{S}}) = \mathsf{row}_{\mathsf{t}}(\varepsilon)$ and $t_{\mathcal{L}}^\sharp(\mathsf{init}_{\mathcal{S}})(z) \neq t_{\mathcal{L}}(\varepsilon)(z)$.*

Proof. We have $\mathsf{row}_{\mathsf{t}}^\sharp(\mathsf{init}_{\mathcal{S}}) = \mathsf{row}_{\mathsf{t}}^\sharp(i(\mathsf{row}_{\mathsf{t}}(\varepsilon))) = \mathsf{row}_{\mathsf{t}}(\varepsilon)$ by the definitions of $\mathsf{init}_{\mathcal{S}}$ and i, and

$$\begin{aligned}
t_{\mathcal{L}}^\sharp(i(\mathsf{row}_{\mathsf{t}}(\varepsilon)))(z) &= t_{\mathcal{L}}^\sharp(\mathsf{init}_{\mathcal{S}})(z) && \text{(definition of } \mathsf{init}_{\mathcal{S}}) \\
&= t_{\mathcal{L}}^\sharp(r_{\mathcal{S}}(\varepsilon))(z) && \text{(definition of } r_{\mathcal{S}}) \\
&= \mathcal{R}(\varepsilon)(z) && \text{(Proposition 23)} \\
&\neq t_{\mathcal{L}}(\varepsilon)(z) && \text{(assumption).}
\end{aligned}$$

◀

References for the Appendix

vHSS17 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. CALF: Categorical Automata Learning Framework, 2017.