Learning Automata with Side-Effects

Gerco Van Heerdt, Matteo Sammartino, Alexandra Silva

Learning Automata with Side-Effects*

Gerco van Heerdt[1] (✉), Matteo Sammartino[1,2], and Alexandra Silva[1]

[1] University College London, United Kingdom
{gerco.heerdt,alexandra.silva}@ucl.ac.uk
[2] Royal Holloway, University of London, United Kingdom
matteo.sammartino@rhul.ac.uk

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. This paper exploits monads, both as a mathematical structure and a programming construct, to design and prove correct a wide class of such optimizations. Monads enable the development of a new learning algorithm and correctness proofs, building upon a general framework for automata learning based on category theory. The new algorithm is parametric on a monad, which provides a rich algebraic structure to capture non-determinism and other side-effects. We show that this allows us to uniformly capture existing algorithms, develop new ones, and add optimizations.

Keywords: automata · learning · side-effects · monads · algebras.

1 Introduction

The increasing complexity of software and hardware systems calls for new scalable methods to design, verify, and continuously improve systems. Black-box inference methods aim at building models of running systems by observing their response to certain queries. This reverse engineering process is very amenable for automation and allows for fine-tuning the precision of the model depending on the properties of interest, which is important for scalability.

One of the most successful instances of black-box inference is automata learning, which has been used in various verification tasks, ranging from finding bugs in implementations of network protocols [15] to rejuvenating legacy software [29]. Vaandrager [30] has written a comprehensive overview of the widespread use of automata learning in verification.

A limitation in automata learning is that the models of real systems can become too large to be handled by tools. This demands compositional methods and techniques that enable compact representation of behaviors.

* This work was partially supported by the ERC Starting Grant ProFoundNet (grant code 679127), a Leverhulme Prize (PLP–2016–129), and the EPSRC Standard Grant CLeVer (EP/S028641/1).

In this paper, we show how monads can be used 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 [2], which learns a minimal deterministic finite automaton (DFA) accepting a regular language by interacting with a *teacher*, i.e., an oracle that can reply to specific queries about the target language. Monads allow us to take an abstract approach, in which category theory is used to devise an optimized learning algorithm and a generic correctness proof for a broad class of compact models.

The inspiration for this work is quite concrete: it is a well-known fact that non-deterministic finite automata (NFAs) can be much smaller than deterministic ones for 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 a challenge for learning algorithms: which non-deterministic automaton should the algorithm learn? To overcome this, Bollig et al. [11] developed a version of Angluin's $\mathtt{L}^\star$ algorithm, which they called $\mathtt{NL}^\star$, in which they use a particular class of NFAs, *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 addressed. We tackle them 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. The state space of a determinized DFA—consisting of 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 and V, then the language of $U \cup V$ is the union of the languages of the first two. Formally, the function that assigns to each state its language is a join-semilattice map, since languages themselves are just sets of words and have a lattice structure. And languages are even richer: they have the structure of complete atomic Boolean algebras. This leads to several questions: Can we exploit this structure 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)?

The latter question is strongly motivated by the widespread use of weighted languages and corresponding *weighted finite automata* (WFAs) in verification, from the formal verification of quantitative properties [13,17,25], to probabilistic model-checking [5], to the verification of on-line algorithms [1].

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 general perspective enables us to generalize $\mathtt{L}^\star$ to a new algorithm $\mathtt{L}^\star_T$, which learns compact automata featuring non-determinism and other side-effects captured by a monad. Moreover, $\mathtt{L}^\star_T$ incorporates further optimizations arising from the monadic representation, which lead to more scalable algorithms.

We start by giving an overview of our approach, which also states our main contributions in greater detail and ends with a road map of the rest of the paper.

2 Overview and Contributions

In this section, we explain the original $\mathtt{L}^\star$ algorithm and discuss the challenges in adapting the algorithm to learn automata with side-effects, illustrating them through a concrete example—NFAs. We then highlight our main contributions.

$\mathtt{L}^\star$ algorithm. This algorithm learns the minimal DFA accepting a language $\mathcal{L} \subseteq A^*$ over a finite alphabet A. It assumes the existence of a *minimally adequate teacher*, which is an oracle that can answer two types of queries: 1. *Membership queries*: given a word $w \in A^*$, does w belong to $\mathcal{L}$? and 2. *Equivalence queries*: given a *hypothesis* DFA $\mathcal{H}$, does $\mathcal{H}$ accept $\mathcal{L}$? If not, the teacher returns a *counterexample*, i.e., a word incorrectly classified by $\mathcal{H}$. The algorithm incrementally builds an *observation table* made of two parts: a top part, with rows ranging over a finite set $S \subseteq A^*$; and a bottom part, with rows ranging over $S \cdot A$ ($\cdot$ is pointwise concatenation). Columns range over a finite $E \subseteq A^*$. For each $u \in S \cup S \cdot A$ and $v \in E$, the corresponding cell in the table contains 1 if and only if $uv \in \mathcal{L}$. Intuitively, each row u contains enough information to fully identify the Myhill–Nerode equivalence class of u with respect to an approximation of the target language—rows with the same content are considered members of the same equivalence class. Cells are filled in using membership queries.

As an example, and to set notation, consider the table below over $A = \{a, b\}$. It shows that $\mathcal{L}$ contains the word aa and does not contain the words ε (the empty word), a, b, ba, aaa, and baa.

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

$\mathsf{row_t} : S \to 2^E$ $\qquad \mathsf{row_t}(u)(v) = 1 \iff uv \in \mathcal{L}$

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

We use functions $\mathsf{row_t}$ and $\mathsf{row_b}$ to describe the top and bottom parts of the table, respectively. Notice that S and $S \cdot A$ may intersect. For conciseness, when tables are depicted, elements in the intersection are only shown in the top part.

A key idea of the algorithm is to construct a hypothesis DFA from the different rows in the table. The construction is the same as that of the minimal DFA from the Myhill–Nerode equivalence, and exploits the correspondence between table rows and Myhill–Nerode equivalence classes. The state space of the hypothesis

$S, E \leftarrow \{\varepsilon\}$
repeat
  while the table is not closed or not consistent
    if the table is not closed
      find $t \in S, a \in A$ such that $\mathsf{row_b}(t)(a) \neq \mathsf{row_t}(s)$ for all $s \in S$
      $S \leftarrow S \cup \{ta\}$
    if the table is not consistent
      find $s_1, s_2 \in S$, $a \in A$, and $e \in E$ such that
        $\mathsf{row_t}(s_1) = \mathsf{row_t}(s_2)$ and $\mathsf{row_b}(s_1)(a)(e) \neq \mathsf{row_b}(s_2)(a)(e)$
      $E \leftarrow E \cup \{ae\}$
  Construct the hypothesis $\mathcal{H}$ and submit it to the teacher
  if the teacher replies *no*, with a counterexample z
    $S \leftarrow S \cup \mathsf{prefixes}(z)$
until the teacher replies *yes*
return $\mathcal{H}$

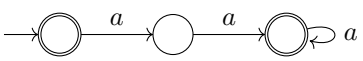
Fig. 1: $\mathtt{L}^\star$ algorithm.

DFA is given by the set $H = \{\mathsf{row_t}(s) \mid s \in S\}$. Note that there may be multiple rows with the same content, but they result in a single state, as they all belong to the same Myhill–Nerode equivalence class. 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, ε must be in S and E, and 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 $t \in S$ and $a \in A$ there is $s \in S$ 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 $\mathsf{row_b}(s_1) = \mathsf{row_b}(s_2)$.

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 shown in Fig. 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 Fig. 2a. This table is not closed, because the row with label a, having 0 in the only column,

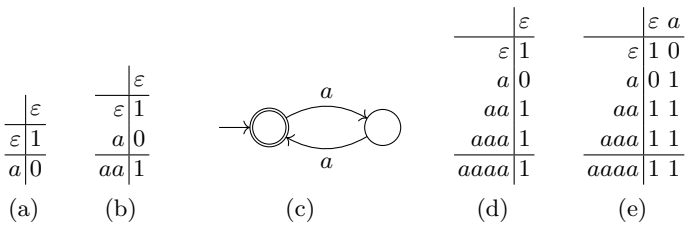
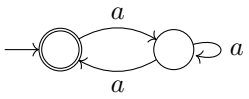


Fig. 2: Example run of $\mathtt{L}^\star$ on $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$.

does not appear in the top 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 (Fig. 2b) is closed and consistent, so we construct the hypothesis that is shown in Fig. 2c and pose an equivalence query. The teacher replies *no* and informs us that the word aaa should have been accepted. $\mathtt{L}^\star$ 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 (Fig. 2d) is closed, but not consistent: the rows ε and aa 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 Fig. 2e. The table is now closed and consistent, and the new hypothesis automaton is precisely the correct one $\mathcal{M}$.

As mentioned, 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 the Myhill–Nerode's right congruence.

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

$$\mathcal{N} = \quad \text{  } \tag{2}$$

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 [24]. This reduction of the state space is enabled by a side-effect—non-determinism, in this case.

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. To overcome this, Bollig et al. [11] proposed to use a particular class of NFAs, namely RFSAs, which do admit minimal canonical representatives.

However, their ad-hoc solution for NFAs does not extend to other automata, such as weighted or alternating. In this paper we present a solution that works for any 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 (complete) *join semilattice* (JSL) whose join operation is set union. This automaton model does admit minimal representatives, 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

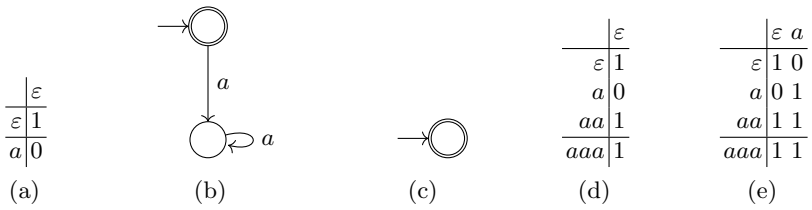
$$\mathsf{row}_\mathsf{t}^\sharp \colon \mathcal{P}(S) \to 2^E \qquad \mathsf{row}_\mathsf{b}^\sharp \colon \mathcal{P}(S) \to (2^E)^A$$

given by $\mathsf{row}_\mathsf{t}^\sharp(U) = \bigvee\{\mathsf{row}_\mathsf{t}(s) \mid s \in U\}$ and $\mathsf{row}_\mathsf{b}^\sharp(U)(a) = \bigvee\{\mathsf{row}_\mathsf{b}(s)(a) \mid s \in U\}$. Formally, these functions are JSL homomorphisms, and they induce the following general definitions:

- The table is *closed* if for all $U \subseteq S, a \in A$ there is $U' \subseteq S$ such that $\mathsf{row}_\mathsf{t}^\sharp(U') = \mathsf{row}_\mathsf{b}^\sharp(U)(a)$.
- The table is *consistent* if for all $U_1, U_2 \subseteq S$ s.t. $\mathsf{row}_\mathsf{t}^\sharp(U_1) = \mathsf{row}_\mathsf{t}^\sharp(U_2)$ we have $\mathsf{row}_\mathsf{b}^\sharp(U_1) = \mathsf{row}_\mathsf{b}^\sharp(U_2)$.

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 as needed, with no additional membership queries. The only lines in Fig. 1 that need to be adjusted are lines 5 and 8, where closedness and consistency are replaced with the new notions given 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 algorithm can be very large since it encodes the entire algebraic structure. We show that we can extract a *minimal set of generators* from the table and compute a *succinct hypothesis* in the form of an automaton with side-effects, 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*. By applying this optimization to this specific case, we essentially recover the learning algorithm of Bollig et al. [11]. The second optimization is a generalization of the optimized counterexample handling method of Rivest and Schapire [28], originally intended for $\mathtt{L}^\star$ and DFAs. It consists in processing 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.



(a) (b) (c) (d) (e)

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

Example Revisited. We now run the new algorithm on the language $\mathcal{L} = \{w \in \{a\}^* \mid |w| \neq 1\}$ considered earlier. Starting from $S = E = \{\varepsilon\}$, the observation table (Fig. 3a) is immediately closed and consistent. (It is closed because we have $\mathsf{row}^\sharp_\mathsf{t}(\{a\}) = \mathsf{row}^\sharp_\mathsf{t}(\emptyset)$.) This gives the JSL hypothesis shown in Fig. 3b, which leads to an NFA hypothesis having a single state that is initial, accepting, and has no transitions (Fig. 3c). The hypothesis is incorrect, and the teacher may supply us with counterexample aa. Adding prefixes a and aa to S leads to the table in Fig. 3d. The table is again closed, but not consistent: $\mathsf{row}^\sharp_\mathsf{t}(\{a\}) = \mathsf{row}^\sharp_\mathsf{t}(\emptyset)$, but $\mathsf{row}^\sharp_\mathsf{b}(\{a\})(a) = \mathsf{row}^\sharp_\mathsf{t}(\{aa\}) \neq \mathsf{row}^\sharp_\mathsf{t}(\emptyset) = \mathsf{row}^\sharp_\mathsf{b}(\emptyset)(a)$. Thus, we add a to E. The resulting table (Fig. 3e) is closed and consistent. We note that row aa is the union of other rows: $\mathsf{row}^\sharp_\mathsf{t}(\{aa\}) = \mathsf{row}^\sharp_\mathsf{t}(\{\varepsilon, a\})$ (i.e., it is not a join-irreducible), and therefore can be ignored when building the succinct hypothesis. This hypothesis has two states, ε and a, and indeed it is the correct one $\mathcal{N}$.

Contributions and road map of the paper. After some preliminary notions in Section 3, we present the main contributions:

- In Section 4, we develop a general algorithm $\mathrm{L}^\star_T$, which generalizes the NFA one presented in Section 2 to an arbitrary *monad* T capturing side-effects, and 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 lead to a further small optimization, where the consistency check on the table is dropped.
- Finally, in Section 7 we show how $\mathrm{L}^\star_T$ can be applied to several automata models, highlighting further case-specific optimizations when available.

3 Preliminaries

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 can be conveniently captured as a *monad*. 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* $\mu\colon T^2 \Rightarrow T$, which 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 singleton operation, and $\bigcup$ is just union of sets. Another example is the triple $(V(-), e, m)$, where $V(X)$ is the free semimodule (over a semiring $\mathbb{S}$) over X, namely $\{\varphi \mid \varphi\colon X \to \mathbb{S} \text{ having finite support}\}$. The support of a function $\varphi\colon X \to \mathbb{S}$ is the set of $x \in X$ such that $\varphi(x) \neq 0$. Then $e\colon X \to V(X)$ is 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 $m(\varphi)(x) = \sum_{\psi \in V(X)} \varphi(\psi) \times \psi(x)$.

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). The $\mathcal{P}$-algebras are the (complete) join-semilattices, and their homomorphisms are join-preserving functions. If $\mathbb{S}$ is a field, V-algebras are vector spaces, and their homomorphisms are linear maps.

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 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 $f(W)(y) = (x \circ T(\mathsf{ev}_y))(W)$ 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. *Given a set U and a T-algebra (V, v), there is a bijective correspondence between T-algebra homomorphisms $TU \to V$ and functions $U \to V$: for a T-algebra homomorphism $f\colon TU \to V$, define $f^\dagger = f \circ \eta\colon U \to V$; for a function $g\colon U \to V$, define $g^\sharp = v \circ Tg\colon TU \to V$. Then $g^\sharp$ is a T-algebra homomorphism 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) with T preserving finite sets, as well as a T-algebra O that models outputs of automata.

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

Example 5. DFAs are $\mathtt{Id}$-automata when $O = 2 = \{0, 1\}$ is used to distinguish accepting from rejecting states. For the more general case of O being any set, DFAs generalize into *Moore automata*.

Example 6. Recall that $\mathcal{P}$-algebras are JSLs, 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 with pointwise join: $(f \vee g)(a) = f(a) \vee g(a)$ for $a \in A$. Since 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 over a set of states S as a $\mathcal{P}$-automaton by taking $Q = (\mathcal{P}(S), \bigcup)$ and $O = 2$, the Boolean join-semilattice with the *or* operation as its join. Let $\mathsf{init} \subseteq S$ be the set of initial states and $\mathsf{out}\colon \mathcal{P}(Q) \to 2$ and $\delta\colon \mathcal{P}(S) \to \mathcal{P}(S)^A$ the respective extensions (Proposition 3) of the NFA's output and transition functions. The resulting $\mathcal{P}$-automaton is precisely the determinized version of the NFA.

More generally, an automaton with side-effects given by a monad T always represents a T-automaton with a free state space.

Proposition 7. *A T-automaton of the form $((TX, \mu_X), \delta, \mathsf{out}, \mathsf{init})$, for any set X, is completely defined by the set X with the element $\mathsf{init} \in TX$ and functions*

$$\delta^\dagger\colon X \to (TX)^A \qquad\qquad \mathsf{out}^\dagger\colon X \to O.$$

We call such a T-automaton a *succinct* automaton, which we sometimes identify with the representation $(X, \delta^\dagger, \mathsf{out}^\dagger, \mathsf{init})$. These automata are closely related to the ones studied in [18].

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 state is reached by reading a given word and which language each state recognizes.

Definition 8 (Reachability/Observability Maps). *The* reachability map *of a T-automaton $\mathcal{A}$ is a function $r_\mathcal{A}\colon A^* \to Q$ inductively defined as: $r_\mathcal{A}(\varepsilon) = \mathsf{init}$ and $r_\mathcal{A}(ua) = \delta(r_\mathcal{A}(u))(a)$. The* observability map *of $\mathcal{A}$ is a function $o_\mathcal{A}\colon Q \to O^{A^*}$ given by: $o_\mathcal{A}(q)(\varepsilon) = \mathsf{out}(q)$ and $o_\mathcal{A}(q)(av) = o_\mathcal{A}(\delta(q)(a))(v)$.*

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

Example 9. For an NFA $\mathcal{A}$ represented as a $\mathcal{P}$-automaton, as seen in Example 6, $o_\mathcal{A}(q)$ is the language of q in the traditional sense. Note that q, in general, is a set of states: $o_\mathcal{A}(q)$ 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 NFA. The reachability map $r_\mathcal{A}(u)$ returns the set of states reached via all paths reading u.

Given a language $\mathcal{L}\colon A^* \to O$, there exists a (unique) *minimal T-automaton* $\mathcal{M}_\mathcal{L}$ accepting $\mathcal{L}$, which is minimal in the number of states. Its existence follows from general facts. See for example [19].

Definition 10 (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 $\delta(t^\sharp_\mathcal{L}(U))(a)(v) = t^\sharp_\mathcal{L}(U)(av)$.*

In the following, we will also make use of the *minimal Moore automaton* accepting $\mathcal{L}$. Although this always exists—by instantiating Definition 10 with $T = \mathsf{Id}$—it need not be finite. The following property says that finiteness of Moore automata and of T-automata accepting the same language are related.

Proposition 11. *The minimal Moore automaton accepting $\mathcal{L}$ is finite if and only if the minimal T-automaton accepting $\mathcal{L}$ is finite.*

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 the monad T, and is therefore called $\mathtt{L}^\star_T$. We fix a language $\mathcal{L}\colon A^* \to O$ that is to be learned, and we assume that there is a finite T-automaton accepting $\mathcal{L}$. This assumption generalizes the requirement of $\mathtt{L}^\star$ that $\mathcal{L}$ is regular (i.e., accepted by a specific class of T-automata, see Example 5).

An observation table consists of a pair of functions

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

given by $\mathsf{row_t}(s)(e) = \mathcal{L}(se)$ and $\mathsf{row_b}(s)(a)(e) = \mathcal{L}(sae)$, 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 12 (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 $T(S) \times A$ against elements of $T(S)$, but only those of $S \times A$, thanks to the following result.

Lemma 13. *If for all $s \in S$ and $a \in A$ there is $U \in T(S)$ such that $\mathsf{row}^\sharp_\mathsf{t}(U) = \mathsf{row_b}(s)(a)$, then the table is closed.*

Example 14. For NFAs represented as $\mathcal{P}$-automata, the properties are as presented in Section 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 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 all 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 $a \in A$.

1 $S, E \leftarrow \{\varepsilon\}$
2 **repeat**
3 **while** the table is not closed or not consistent
4 **if** the table is not closed
5 find $s \in S$, $a \in A$ such that $\mathsf{row_b}(s)(a) \neq \mathsf{row_t^\sharp}(U)$ for all $U \in T(S)$
6 $S \leftarrow S \cup \{sa\}$
7 **if** the table is not consistent
8 find $U_1, U_2 \in T(S)$, $a \in A$, and $e \in E$ such that
 $\mathsf{row_t^\sharp}(U_1) = \mathsf{row_t^\sharp}(U_2)$ and $\mathsf{row_b^\sharp}(U_1)(a)(e) \neq \mathsf{row_b^\sharp}(U_2)(a)(e)$
9 $E \leftarrow E \cup \{ae\}$
10 Construct the hypothesis $\mathcal{H}$ and submit it to the teacher
11 **if** the teacher replies *no*, with a counterexample z
12 $S \leftarrow S \cup \mathsf{prefixes}(z)$
13 **until** the teacher replies *yes*
14 **return** $\mathcal{H}$

Fig. 4: Adaptation of $\mathtt{L}^\star$ for T-automata.

If closedness and consistency hold, we can define a *hypothesis* T-automaton $\mathcal{H}$, with state space $H = \mathtt{img}(\mathsf{row_t^\sharp})$, $\mathsf{init} = \mathsf{row_t}(\varepsilon)$, and output and transitions

$$\mathsf{out}\colon H \to O \qquad \mathsf{out}(\mathsf{row_t^\sharp}(U)) = \mathsf{row_t^\sharp}(U)(\varepsilon)$$
$$\delta\colon H \to H^A \qquad \delta(\mathsf{row_t^\sharp}(U)) = \mathsf{row_b^\sharp}(U).$$

The correctness of this definition follows from the abstract treatment of [21], instantiated to the category of T-algebras and their homomorphisms.

We can now give algorithm $\mathtt{L}^\star_T$. Similarly to the example in Section 2, we only have to adjust lines 5 and 8 in Fig. 1. The resulting algorithm is shown in Fig. 4.

Correctness. Correctness for $\mathtt{L}^\star_T$ amounts to proving that, for any target language $\mathcal{L}$, the algorithm terminates returning the minimal T-automaton $\mathcal{M}_\mathcal{L}$ accepting $\mathcal{L}$. As in the original $\mathtt{L}^\star$ algorithm, we only need to prove that the algorithm terminates, that is, that only finitely many hypotheses are produced. Correctness follows from termination, since line 13 causes the algorithm to terminate only if the hypothesis automaton coincides with $\mathcal{M}_\mathcal{L}$.

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, the state space of $\mathcal{M}_\mathcal{L}$. In fact, when a closedness defect is resolved (line 6), a row that was not previously found in the image of $\mathsf{row_t^\sharp}\colon T(S) \to O^E$ 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. Note that the hypothesis $\mathcal{H}$ in the statement below is the hypothesis obtained before adding the prefixes of z to S.

Proposition 15. *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 $E = A^*$, $\mathsf{row}_{\mathsf{t}}^{\sharp} = t_{\mathcal{L}}^{\sharp}$. Therefore, since H and M are defined as the images of $\mathsf{row}_{\mathsf{t}}^{\sharp}$ and $t_{\mathcal{L}}^{\sharp}$, respectively, the size of H is bounded by that of M. Since H increases while the algorithm loops, the algorithm must terminate and is thus correct.

Note that the learning algorithm of Bollig et al. does not terminate using this counterexample processing method [10, Appendix F]. This is due to their notion of consistency being weaker than ours: we have shown that progress is guaranteed because a consistency defect, in our sense, is created using this method.

Query complexity. The complexity of automata learning algorithms is usually measured in terms of the number of both membership and equivalence queries asked, as it is common to assume that computations within the algorithm are insignificant compared to evaluating the system under analysis in applications. The cost of answering queries themselves is not considered, as it depends on the implementation of the teacher, which the algorithm abstracts from.

The table is a T-algebra homomorphism, so membership queries for rows labeled in S are enough to determine all other rows. We measure the query complexities in terms of the number of states n of the minimal Moore automaton, the number of states t of the minimal T-automaton, the size k of the alphabet, and the length m of the longest counterexample. Note that t cannot be smaller than n, but it can be much bigger. For example, when $T = \mathcal{P}$, t may be in $\mathcal{O}(2^n)$.[3]

The maximum number of closedness defects fixed by the algorithm is n, as a closedness defect for the setting with algebraic structure is also a closedness defect for the setting without that structure. The maximum number of consistency defects fixed by the algorithm is t, as fixing a consistency defect distinguishes two rows that were previously identified. Since counterexamples lead to consistency defects, this also means that the algorithm will not pose more than t equivalence queries. A word is added to S when fixing a closedness defect, and $\mathcal{O}(m)$ words are added to S when processing a counterexample. The number of rows that we need to fill using queries is therefore in $\mathcal{O}(tmk)$. The number of columns added to the table is given by the number of times a consistency defect is fixed and thus in $\mathcal{O}(t)$. Altogether, the number of membership queries is in $\mathcal{O}(t^2mk)$.

[3] Take the language $\{a^p\}$, for some $p \in \mathbb{N}$ and a singleton alphabet $\{a\}$. Its residual languages are $\emptyset$ and $\{a^i\}$ for all $0 \leq i \leq p$, thus the minimal DFA accepting the language has $p + 2$ states. However, the residual languages w.r.t. sets of words are all the subsets of $\{\varepsilon, a, aa, \ldots, a^p\}$—hence, the minimal T-automaton has 2^{p+1} states.

5 Succinct Hypotheses

We now describe the first of two optimizations, which is 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 given by a subset of S. We start by defining sets of *generators for the table*.

Definition 16. *A set $S' \subseteq S$ is a* set of generators for the table *whenever for all $s \in S$ there is $U \in T(S')$ such that $\mathsf{row}_\mathsf{t}(s) = \mathsf{row}_\mathsf{t}^\sharp(U)$.*[4]

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 the *succinct hypothesis*, although $T(S')$ may be larger than H. Proposition 7 tells us that the succinct hypothesis can be represented as an automaton with side-effects in T that has S' as its state space. 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 17 (Succinct Hypothesis). *The* succinct hypothesis *is the T-automaton $\mathcal{S} = (T(S'), \delta, \mathsf{out}, \mathsf{init})$ given by $\mathsf{init} = i(\mathsf{row}_\mathsf{t}(\varepsilon))$ and*

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

This definition is inspired by that of a *scoop*, due to Arbib and Manes [4].

Proposition 18. *Any succinct hypothesis of $\mathcal{H}$ accepts the language of $\mathcal{H}$.*

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. This generalizes a procedure defined by Angluin et al. [3] for non-deterministic, universal, and alternating automata.

[4] 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 $\mathsf{row}_\mathsf{t}^\sharp(U)$.

Proposition 19. *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'

To determine whether U as in the above algorithm exists, one can always naively enumerate all possibilities, using that T preserves finite sets. This is what we call the basic algorithm. For specific algebraic structures, one may find more efficient methods, as we show in the following example.

Example 20. Consider the powerset monad $T = \mathcal{P}$. We now exemplify two ways of computing succinct hypotheses, which are inspired by canonical RFSAs [16]. The basic idea is to start from a deterministic automaton and to remove states that are equivalent to a set of other states. The algorithm given in Proposition 19 computes a minimal S' that 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. To concretize the algorithm efficiently, we use a method introduced by Bollig et al. [11], which essentially exploits the natural order on the JSL of table rows. In contrast to the basic exponential algorithm, this results in a polynomial one.[5] Bollig et al. determine whether a row is a join of other rows by comparing the row just to the join of rows below it. Like them, we make use of this also to compute right inverses of e, for which we will formalize the order.

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 right inverses $H \to \mathcal{P}(S')$ for it. The first one,

$$i_1(h) = \{s \in S' \mid \mathsf{row}_\mathsf{t}(s) \leq h\},$$

stems from the construction of the *canonical RFSA* of a language [16]. Here we use the order $a \leq b \iff a \vee b = b$ induced by the JSL structure. The resulting construction of a succinct hypothesis was first used by Bollig et al. [11]. This succinct hypothesis has a "maximal" transition function, meaning that no more transitions can be added without changing the language of the automaton.

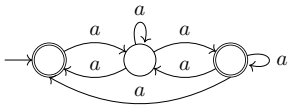
The second inverse is

$$i_2(h) = \{s \in S' \mid \mathsf{row}_\mathsf{t}(s) \leq h \text{ and for all } s' \in S' \text{ s.t. } \mathsf{row}_\mathsf{t}(s) \leq \mathsf{row}_\mathsf{t}(s') \leq h \text{ we have } \mathsf{row}_\mathsf{t}(s) = \mathsf{row}_\mathsf{t}(s')\},$$

resulting in a more economical transition function, where some redundancies are removed. This corresponds to the *simplified canonical RFSA* [16].

Example 21. Consider $T = \mathcal{P}$, and recall the table in Fig. 3e. When $S' = S$, the right inverse given by i_1 yields the succinct hypothesis shown below.

[5] When we refer to computational complexities, as opposed to query complexities, they are in terms of the sizes of S, E, and A.



Note that $i_1(\mathsf{row}_\mathsf{t}(aa)) = \{\varepsilon, a, aa\}$. Taking i_2 instead, the succinct hypothesis is just the DFA (1) because $i_2(\mathsf{row}_\mathsf{t}(aa)) = \{aa\}$. Rather than constructing a succinct hypothesis directly, our algorithm first reduces the set S'. In this case, we have $\mathsf{row}_\mathsf{t}(aa) = \mathsf{row}^\sharp_\mathsf{t}(\{\varepsilon, a\})$, so we remove aa from S'. Now i_1 and i_2 coincide and produce the NFA (2). Minimizing the set S' in this setting essentially comes down to determining what Bollig et al. [11] call the *prime* rows of the table.

Remark 22. The algorithm in Proposition 19 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.

6 Optimized Counterexample Handling

The second optimization we give generalizes the counterexample processing method due to Rivest and Schapire [28], which improves the worst case complexity of the number of membership queries needed in $\mathtt{L}^\star$. Maler and Pnueli [26] proposed to add all suffixes of the counterexample to the set E instead of adding all prefixes to the set S. This eliminates the need for consistency checks in the deterministic setting. The method by Rivest and Schapire finds a *single* suffix of the counterexample and adds it to E. 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 $u \in A^*$, let q_u be the state in $\mathcal{H}$ reached by reading u, i.e., $q_u = r_\mathcal{H}(u)$. For each $q \in H$, we pick any $U_q \in T(S)$ that yields q according to the table, i.e., such that $\mathsf{row}^\sharp_\mathsf{t}(U_q) = q$. 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 $\mathcal{R}(u) = t^\sharp_\mathcal{L}(U_{q_u})$ 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 23. *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 $U_{q_\varepsilon} = \eta(\varepsilon)$. For a counterexample z, we then have $\mathcal{R}(\varepsilon)(z) = t_\mathcal{L}(\varepsilon)(z) \neq \mathcal{R}(z)(\varepsilon)$. 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 24. *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 $uav = z$ 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 $a' \in A$, $\mathsf{row}_{\mathsf{b}}^\sharp(U)(a')(e) = t_{\mathcal{L}}^\sharp(U)(a'e)$.) 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 increased. A closedness defect also increases the size of H, so in any case we make progress.

Proposition 25. *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 $e\colon T(S') \to H$. Choosing such an i is equivalent to choosing U_q for each $q \in H$. We then redefine $\mathcal{R}$ using the reachability map of the succinct hypothesis. Specifically, $\mathcal{R}(u) = t_{\mathcal{L}}^\sharp(r_{\mathcal{S}}(u))$ for all $u \in A^*$.

Unfortunately, there is one complication. We assumed earlier that $U_{q_\varepsilon} = \eta(\varepsilon)$, or more specifically $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$. 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 $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, we either add z to E (if the test fails) or proceed with the original method.

Proposition 26. *If $z \in A^*$ is such that $\mathcal{R}(\varepsilon)(z) \neq \mathcal{L}(z)$, then $\mathsf{row}_{\mathsf{t}}^\sharp(\mathsf{init}_{\mathcal{S}}) = \mathsf{row}_{\mathsf{t}}(\varepsilon)$ and $t_{\mathcal{L}}^\sharp(\mathsf{init}_{\mathcal{S}})(z) \neq t_{\mathcal{L}}(\varepsilon)(z)$.*

To see that the original method still works, we prove the analogue of Proposition 25 for the new definition of $\mathcal{R}$.

Proposition 27. *If $z \in A^*$ is such that $\mathcal{L}_{\mathcal{S}}(z) \neq \mathcal{L}(z)$ and $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, then there are $u, v \in A^*$ and $a \in A$ such that $\mathsf{row}_{\mathsf{t}}^\sharp(r_{\mathcal{S}}^\dagger(ua)) = \mathsf{row}_{\mathsf{b}}^\sharp(r_{\mathcal{S}}^\dagger(u))(a)$ and $t_{\mathcal{L}}^\sharp(r_{\mathcal{S}}^\dagger(ua))(v) \neq t_{\mathcal{L}}^\sharp(r_{\mathcal{S}}^\dagger(u))(av)$.*

Example 28. Recall the succinct hypothesis $\mathcal{S}$ from Fig. 3c for the table in Fig. 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 $i(\mathsf{row}_{\mathsf{t}}^\sharp(\emptyset)) = \emptyset$. This is 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 $t_{\mathcal{L}}^\sharp(\{\varepsilon\})(aa) = 1$ 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.

Query complexity. Again, we measure the membership and equivalence query complexities in terms of the number of states n of the minimal Moore automaton, the number of states t of the minimal T-automaton, the size k of the alphabet, and the length m of the longest counterexample.

A counterexample now gives an additional column instead of a set of rows, and we have seen that this leads to either a closedness defect or to two rows being distinguished. Thus, the number of equivalence queries is still at most t, and the number of columns is still in $\mathcal{O}(t)$. However, the number of rows that we need to fill using membership queries is now in $\mathcal{O}(nk)$. This means that a total of $\mathcal{O}(tnk)$ membership queries is needed to fill the table.

Apart from filling the table, we also need queries to analyze counterexamples. The binary search algorithm mentioned after Corollary 24 requires for each counterexample $\mathcal{O}(\log m)$ computations of $\mathcal{R}(x)(y)$ for varying words x and y. Let r be the maximum number of queries required for a single such computation. Note that for $u, v \in A^*$, and letting $\alpha\colon TO \to O$ be the algebra structure on O, we have $\mathcal{R}(u)(v) = \alpha(T(\mathsf{ev}_v \circ t_{\mathcal{L}})(U_{q_u}))$ for the original definition of $\mathcal{R}$ and

$$\mathcal{R}(u)(v) = \alpha(T(\mathsf{ev}_v \circ t_{\mathcal{L}})(r_{\mathcal{S}}^{\dagger}(u)))$$

in the succinct hypothesis case. Since the restricted map $T(\mathsf{ev}_v \circ t_{\mathcal{L}})\colon TS \to TO$ is completely determined by $\mathsf{ev}_v \circ t_{\mathcal{L}}\colon S \to O$, r is at most $|S|$, which is bounded by n in this optimized algorithm. For some examples (see for instance the writer automata in Section 7), we even have $r = 1$. The overall membership query complexity is $\mathcal{O}(tnk + tr \log m)$.

Dropping Consistency. We described the counterexample processing method based around Proposition 25 in terms of the succinct hypothesis $\mathcal{S}$ rather than the actual hypothesis $\mathcal{H}$ by showing that $\mathcal{R}$ can be defined using $\mathcal{S}$. 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 not be the state space of an actual hypothesis during intermediate stages. This observation also explains why Bollig et al. [11] are able to use a weaker notion of consistency in their algorithm. Interestingly, they exploit the canonicity of their choice of succinct hypotheses to arrive at a polynomial membership query complexity that does not involve the factor t.

7 Examples

In this section we list several examples that can be seen as T-automata and hence learned via an instance of $\mathtt{L}^\star_T$. We remark that, since our algorithm operates on finite structures (recall that T preserves finite sets), for each automaton type one can obtain a basic, correct-by-construction instance of $\mathtt{L}^\star_T$ for free, by plugging the concrete definition of the monad into the abstract algorithm. However, we note that this is not how $\mathtt{L}^\star_T$ is intended to be used in a real-world context; it should be

seen as an abstract specification of the operations each concrete implementation needs to perform, or, in other words, as a template for real implementations.

For each instance below, we discuss whether certain operations admit a more efficient implementation than the basic one, based on the specific algebraic structure induced by the monad. Due to our general treatment, the optimizations of Sections 5 and 6 apply to all of these instances.

Non-deterministic automata. As discussed before, non-deterministic automata are $\mathcal{P}$-automata with a free state space, provided that $O = 2$ is equipped with the "or" operation as its $\mathcal{P}$-algebra structure. We also mentioned that, as Bollig et al. [11] showed, there is a polynomial time algorithm to check whether a given row is the join of other rows. This gives an efficient method for handling closedness straight away. Moreover, as shown in Example 20, it allows for an efficient construction of the succinct hypothesis. Unfortunately, checking for consistency defects seems to require a number of computations exponential in the number of rows. However, as explained at the end of Section 6, we can in fact drop consistency altogether.

Universal automata. Just like non-deterministic automata, universal automata can be seen as $\mathcal{P}$-automata with a free state space. The difference 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. One can dualize the optimized specific algorithms for the case of non-deterministic automata. This is precisely what Angluin et al. [3] have done.

Partial automata. Consider the *maybe monad* $\mathtt{Maybe}(X) = 1 + X$, 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 standard 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 $O = \mathtt{Maybe}(1) = 1 + 1$. Here 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. The algorithm $\mathtt{L}^\star_{\mathtt{Maybe}}$ is mostly like $\mathtt{L}^\star$, except that implicitly the table has an additional row with zeroes in every column. Since the monad only adds a single element to each set, there is no need to optimize the basic algorithm for this specific case.

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 semimodule $V(X)$, the output function $\mathtt{out} \colon V(X) \to \mathbb{S}$ 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. The obvious semimodule structure on $\mathbb{S}$ extends to a pointwise structure on the potential rows of the table. The basic algorithm loops over all linear combinations of rows to check closedness and over all pairs of combinations of rows to check consistency, making them extremely expensive operations. If $\mathbb{S}$ is a field, a row can be decomposed into a linear

combination of other rows in polynomial time using standard techniques from linear algebra. As a result, there are efficient procedures for checking closedness and constructing succinct hypotheses. It was shown by Van Heerdt et al. [21] that consistency in this setting is equivalent to closedness of the transpose of the table. This trick is due to Bergadano and Varricchio [7], who first studied learning of weighted automata.

Alternating automata. We use the characterization of alternating automata due to Bertrand and Rot [9]. Recall that, given a partially ordered set $(P, \leq)$, an *upset* is a subset U of P such that, if $x \in U$ and $x \leq y$, then $y \in U$. Given $Q \subseteq P$, we write $\uparrow Q$ for the *upward closure* of Q, that is the smallest upset of P containing Q. We consider the monad $\mathtt{A}$ that maps a set X to the set of all upsets of $\mathcal{P}(X)$. Its unit is given by $\eta_X(x) = \uparrow\{\{x\}\}$ and its multiplication by

$$\mu_X(U) = \{V \subseteq X \mid \exists_{W \in U}\, \forall_{Y \in W}\, \exists_{Z \in Y}\, Z \subseteq V\}.$$

Algebras for the monad $\mathtt{A}$ are *completely distributive lattices* [27]. The sets of sets in $\mathtt{A}(X)$ can be seen as DNF formulae over elements of X, where the outer powerset is disjunctive and the inner one is conjunctive. Accordingly, we define an algebra structure $\beta\colon \mathtt{A}(2) \to 2$ on the output set 2 by letting $\beta(U) = 1$ if $\{1\} \in U$, 0 otherwise. Alternating automata with states X can be represented as $\mathtt{A}$-automata with state space $\mathtt{A}(X)$, output map $\mathtt{out}\colon \mathtt{A}(X) \to 2$, and transition map $\delta\colon \mathtt{A}(X) \to \mathtt{A}(X)^A$, sending each state to a DNF formula over X. The only difference with the usual definition of alternating automata is that $\mathtt{A}(X)$ is not the full set $\mathcal{PP}(X)$, which is not a monad [23]. However, for each formula in $\mathcal{PP}(X)$ there is an equivalent one in $\mathtt{A}(X)$.

An adaptation of $\mathtt{L}^\star$ for alternating automata was introduced by Angluin et al. [3] and further investigated by Berndt et al. [8]. The former found that given a row $r \in 2^E$ and a set of rows $X \subseteq 2^E$, r is equal to a DNF combination of rows from X (where logical operators are applied component-wise) if and only if it is equal to the combination defined by $Y = \{\{x \in X \mid x(e) = 1\} \mid e \in E \wedge r(e) = 1\}$. We can reuse this idea to efficiently find closedness defects and to construct the hypothesis. Even though the monad $\mathtt{A}$ formally requires the use of DNF formulae representing upsets, in the actual implementation we can use smaller formulae, e.g., Y above instead of its upward closure. In fact, it is easy to check that DNF combinations of rows are invariant under upward closure. Similar as before, we do not know of an efficient way to ensure consistency, but we could drop it.

Writer automata. The examples considered so far involve existing classes of automata. To further demonstrate the generality of our approach, we introduce a new (as far as we know) type of automaton, which we call *writer automaton*.

The *writer monad* $\mathtt{Writer}(X) = \mathbb{M} \times X$ for a finite monoid $\mathbb{M}$ has a unit $\eta_X\colon X \to \mathbb{M} \times X$ given by adding the unit e of the monoid, $\eta_X(x) = (e, x)$, and a multiplication $\mu_X\colon \mathbb{M} \times \mathbb{M} \times X \to \mathbb{M} \times X$ given by performing the monoid multiplication, $\mu_X(m_1, m_2, x) = (m_1 m_2, x)$. In Haskell, the writer monad is used for such tasks as collecting successive log messages, where the monoid is given by the set of sets or lists of possible messages and the multiplication adds a message.

The algebras for this monad are sets Q equipped with an $\mathbb{M}$-action. One may take the output object to be the set $\mathbb{M}$ with the monoid multiplication as its action. `Writer`-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 is as expected: $\mathbb{M}$-elements multiply along paths and finally multiply with the output of the last state to produce the actual output.

The basic learning algorithm has polynomial time complexity. To determine whether a given row is a combination of rows in the table, i.e., whether it is given by a monoid value applied to one of the rows in the table, one simply tries all of these values. This allows us to check for closedness, to minimize the generators, and to construct the succinct hypothesis, in polynominal time. Consistency involves comparing all ways of applying monoid values to rows and, for each comparison, at most $|A|$ further comparisons between one-letter extensions. The total number of comparisons is clearly polynomial in $|\mathbb{M}|$, $|S|$, and $|A|$.

8 Conclusion

We have presented $\mathtt{L}^\star_T$, a general adaptation of $\mathtt{L}^\star$ that uses monads 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 [28] to this setting and discussed instantiations to non-deterministic, universal, partial, weighted, alternating, and writer automata.

Related Work. This paper builds on and extends the theoretical toolkit of Van Heerdt et al. [21,19], who are developing a categorical automata learning framework (CALF) in which learning algorithms can be understood and developed in a structured way.

An adaptation of $\mathtt{L}^\star$ that produces NFAs was first developed by Bollig et al. [11]. Their algorithm learns a special subclass of NFAs consisting of RFSAs, which were introduced by Denis et al. [16]. Angluin et al. [3] unified algorithms for NFAs, universal automata, and alternating automata, the latter of which was further improved by Berndt et al. [8]. We are able to provide a more general framework, which encompasses and goes beyond those classes of automata. Moreover, we study optimized counterexample handling, which [3,11,8] do not consider.

The algorithm for weighted automata over an arbitrary field was studied in a category theoretical context by Jacobs and Silva [22] and elaborated on by Van Heerdt et al. [21]. The algorithm itself was introduced by Bergadano and Varricchio [7]. The theory of succinct automata used for our hypotheses is based on the work of Arbib and Manes [4], revamped to more recent category theory.

Future Work. Whereas our general algorithm effortlessly instantiates to monads that preserve finite sets, a major challenge lies in investigating monads that do not enjoy this property. The algorithm for weighted automata generalizes to an infinite field [7,22,21] and even a principal ideal domain [20]. However, for an

infinite semiring in general we cannot guarantee termination, which is because a finitely generated semimodule may have an infinite chain of strict submodules [20]. Intuitively, this means that while fixing closedness defects increases the size of the hypothesis state space semimodule, an infinite number of steps may be needed to resolve all closedness defects. In future work we would like to characterize more precisely for which semirings we can learn, and ideally formulate this characterization on the monad level.

As a result of the correspondence between learning and conformance testing [6,21], it should be possible to include in our framework the W-method [14], which is often used in case studies deploying $\mathsf{L}^\star$ (e.g. [12,15]). We defer a thorough investigation of conformance testing to future work.

References

1. Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.
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. Michael A. Arbib and Ernest G. Manes. Fuzzy machines in a category. *Bulletin of the AMS*, 13:169–210, 1975.
5. Christel Baier, Marcus Größer, and Frank Ciesinski. Model checking linear-time properties of probabilistic systems. In *Handbook of Weighted automata*, 2009.
6. 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.
7. Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25:1268–1280, 1996.
8. Sebastian Berndt, Maciej Liśkiewicz, Matthias Lutter, and Rüdiger Reischuk. Learning residual alternating automata. In *AAAI*, pages 1749–1755, 2017.
9. Meven Bertrand and Jurriaan Rot. Coalgebraic determinization of alternating automata. *arXiv preprint arXiv:1804.02546*, 2018.
10. Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of nfa (research report lsv-08-28). Technical report, ENS Cachan, 2008.
11. Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, volume 9, pages 1004–1009, 2009.
12. Georg Chalupar, Stefan Peherstorfer, Erik Poll, and Joeri de Ruiter. Automated reverse engineering using Lego®. In *WOOT*, 2014.
13. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In *CSL*, pages 385–400, 2008.
14. Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4:178–187, 1978.
15. Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security*, pages 193–206, 2015.
16. François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51:339–368, 2002.

17. Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In *ICALP*, pages 513–525, 2005.
18. Sergey Goncharov, Stefan Milius, and Alexandra Silva. Towards a coalgebraic chomsky hierarchy. In *TCS*, pages 265–280. Springer, 2014.
19. Gerco van Heerdt. An abstract automata learning framework. Master's thesis, Radboud University Nijmegen, 2016.
20. Gerco van Heerdt, Clemens Kupke, Jurriaan Rot, and Alexandra Silva. Learning weighted automata over principal ideal domains. *arXiv preprint arXiv:1911.04404*, 2019.
21. Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. CALF: categorical automata learning framework. In *CSL*, pages 29:1–29:24, 2017.
22. Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, volume 8464, pages 384–406, 2014.
23. Bartek Klin and Julian Salamanca. Iterated covariant powerset is not a monad. *Electronic Notes in Theoretical Computer Science*, 341:261–276, 2018.
24. Dexter C. Kozen. *Automata and computability*. Springer Science & Business Media, 2012.
25. Denis Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.
26. Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inform. and Comput.*, 118:316–326, 1995.
27. George Markowsky. Free completely distributive lattices. *Proceedings of the American Mathematical Society*, 74(2):227–228, 1979.
28. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inform. Comput.*, 103:299–347, 1993.
29. 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.
30. Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.