How to Handle Long Counterexamples: Heuristics, Optimizations and Asymptotic Lower Bounds for L$^{\#}$

Master Thesis in Computing Science
Radboud University Nijmegen

Author:
Loes Kruger
s1001459

First supervisor/assessor:
Prof. dr. F.W. Vaandrager

Second assessor:
Dr. J.C. Rot

Submitted on:
02/02/2023

Abstract. We present an adaptation of the $L^{\#}$ algorithm that avoids using long witnesses when possible and if they need to be used, we apply the long witnesses in a more informed manner. Additionally, we add two new rules that exploit special cases in the observation tree. The new algorithm is called $L^{\#}_Z$ and has the same asymptotic query and symbol complexities as the best active learning algorithms. Experiments with the prototype implementations of $L^{\#}$ and $L^{\#}_Z$ show that $L^{\#}_Z$ is competitive with $L^{\#}$ and thus also with other active learning algorithms. In scenarios where long witnesses frequently occur, $L^{\#}_Z$ uses 20.5% fewer symbols during learning compared to $L^{\#}$. Moreover, we attempt to get closer to answering a major open question in the field of automata learning: 'Is the length of the longest counterexample a necessary term in the lower bound in the query complexity of automata learning algorithms?'. We present a general query lower bound proof for Mealy machine learning algorithms with a construction that only allows counterexamples that are bounded by the number of states in the Mealy machine. Additionally, we prove the query and symbol lower bound for the $L^{\#}$algorithm. The length of the counterexamples can be arbitrarily long in this construction.

1 Introduction

Automata learning is an automated technique that is used to infer a black-box software or hardware system [20]. Active automata learning algorithms construct the automata by directly interacting with the system. These algorithms often use the Minimally Adequate Teacher (MAT) framework as designed by Angluin [1]. In this framework, the learner can ask questions to the teacher about the automata. Two types of questions are allowed:

Membership queries The learner asks whether the target automaton accepts a certain word. The goal is to improve the current hypothesis automaton.

Equivalence queries The learner asks if the current automaton is equivalent to the target automaton. If the automata are not equivalent, the teacher provides a counterexample.

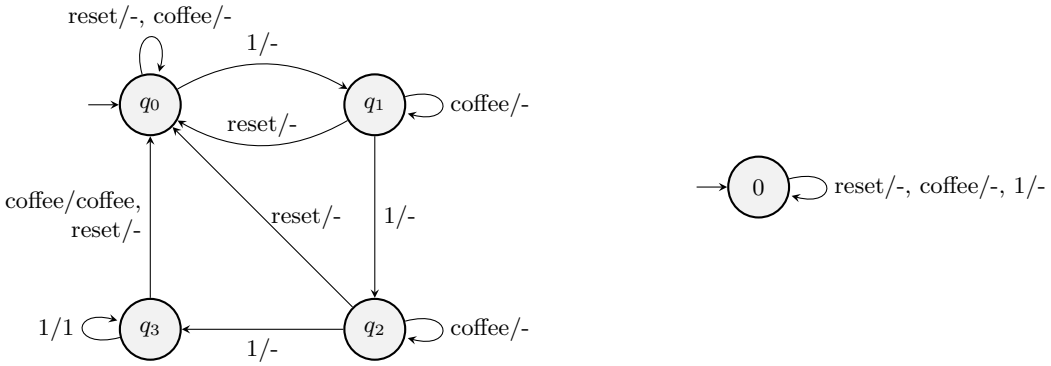
Many of the algorithms in this framework use an observation table to keep track of the found states and posed membership queries [1, 8, 14, 17]. Besides observation tables, there is also a set of algorithms that use discrimination trees [7, 10, 12]. The $L^{\#}$ algorithm [21], takes a different approach and works directly on the observation tree. Experiments suggest that a prototype implementation of $L^{\#}$ is competitive with the implementations of the other active learning algorithms. The best active learning algorithms designed in the last thirty years have the same worst-case query complexity

$$\mathcal{O}(kn^2 + n \log m)$$

where k is the number of input symbols, n is the number of states, and m is the length of the longest counterexample. For practical learning scenarios, we often look at the symbol complexity because the total time needed to learn the automaton is usually proportional to the number of input symbols in the queries asked of the teacher. The $L^{\#}$ algorithm has the following input symbol complexity:

$$\mathcal{O}(kmn^2 + nm \log m)$$

It is easy to see that the query and symbol complexity must depend on the parameters k and n. However, it is an open problem to determine whether the parameter m is necessary [3]. If the teacher is nice, then they can always return a counterexample of length at most n because, for any pair of distinct states, there is a sequence of length at most n which shows that these states are distinct [15]. In practice, execution logs are often used as a starting point for learning and for finding counterexamples for hypothesis models. Therefore, counterexamples provided by the teacher in a practical scenario are rarely bounded by n. We will show why this is problematic with the example automaton in Figure 1a.



(a) *Example target system* ***(b)*** *Initial hypothesis*

Fig. 1: *Running example of a coffee machine and the initial observation tree and hypothesis*

The target automaton in Figure 1a represents a hypothetical coffee machine that accepts 1 euro coins, provides coffee for 3 euros, and can be reset to set the euro counter back to zero. This example is loosely based on the running example in [10]. If we apply the $L^{\#}$ algorithm to learn this automaton, the first equivalence query would ask whether the initial hypothesis displayed in Figure 1b is correct. Because the hypothesis is incorrect, we would receive a counterexample from the teacher. In the ideal case, the teacher provides a minimal counterexample, such as *1 1 1 1*. However, the teacher can also provide the following counterexample.

reset coffee reset reset coffee reset 1 coffee coffee 1 coffee 1 1 reset 1 1 coffee

The $L^{\#}$ algorithm reduces this counterexample to the following sequence.

1 coffee coffee 1 coffee 1 1

The reduced counterexample still contains redundant information, the *coffee*'s. Strategic $L^{\#}$, a variation of $L^{\#}$ that avoids using equivalence queries when possible, is forced to use this reduced counterexample to learn transitions before it can ask another equivalence query or further process the counterexample. This specific counterexample is relatively short, but note that there is no limit to the number of *coffee* inputs that can be injected in this counterexample. It is perfectly possible that the teacher can return counterexamples that are several thousands of input symbols long.

Non-minimal counterexamples are not a problem specific to $L^\#$. In fact, similar to $L^\#$, most active learning algorithms use binary search to shorten the long counterexample as much as possible and then use the reduced counterexample to extend their data structures [10, 17]. The question 'Is it possible to make a polynomial active learning algorithm where the number of queries asked does not depend on the length of the longest counterexample returned by the teacher?` has been an open question for 25 years [3]. We can answer this question in two ways:

1. To answer *yes*, an algorithm can be constructed where the upper bound complexity does not contain the parameter m.
2. To answer *no*, a general lower bound for learning algorithms needs to be proven that contains the parameter m and considers the possibility that the teacher returns long counterexamples.

Constructing an algorithm with a better worst-case complexity is a non-trivial task, as all algorithms designed in the last thirty years have similar worst-case complexities and all use the parameter m. We present a variation on the $L^\#$ algorithm, called $L_Z^\#$[1], which has four additional rules that aim to use fewer long output queries or exploit special cases in the observation tree. The implementations of $L^\#$ and $L_Z^\#$ are tested on benchmarks of [16]. This shows that $L_Z^\#$ outperforms $L^\#$ in scenarios where long witnesses frequently occur and the algorithms are comparable otherwise. However, $L_Z^\#$ has the same worst-case complexity as the best active learning algorithms. This gives reason to believe that the answer to the question above is 'no' and a general lower bound needs to be proven. To prove a general lower bound, it is customary to first prove lower bounds for specific algorithms. By pinpointing differences between the lower bounds for specific algorithms, it is possible to derive that a specific problem cannot be solved efficiently. In that case, we can construct a general lower bound proof. In this thesis, we take the first steps toward proving a general lower bound. We take inspiration from the general lower bound proof for learning DFA by Balcázar et al. [3] and the general lower bound proof for learning DFA with only equivalence queries by Angluin [2]. We will first prove a general lower bound for learning Mealy machines, then that construction is adapted to prove a lower bound query and symbol complexity for $L^\#$ and $L_Z^\#$.

Related work. For an overview of active learning algorithms, we refer the reader to the paper by Vaandrager et al. [21]. Additionally, the thesis of Isberner [9] provides an overview of the data structures, counterexample handling methods, and complexities used in active learning algorithms. Many of the discussed algorithms decompose the counterexample into a prefix, a relevant part, and a suffix. Discrimination-tree based algorithms use binary or exponential search to find the decomposition [9]. In discrimination-tree based algorithms, the parameter m only occurs in the counterexample handling part of the query complexity. Interestingly, the lower bound proof by Balcázar et al. [3] does not consider the queries needed to process counterexamples. They present a lower bound proof for acyclic DFAs that shows the asymptotic optimality of the Observation Pack algorithm. Isberner [9] reports the same lower bound as Balcázar et al. for the TTT algorithm. The construction used by Angluin [2] which shows that there is no polynomial algorithm that uses only equivalence queries to learn a DFA, is also acyclic. Because all constructions use acyclic DFAs and thus only consider short counterexamples, there are no query lower bounds that include the parameter m.

Approach. We first discuss some preliminaries about Mealy machines, the $L^\#$ algorithm, and separating sequences (Section 2). Then, we present the $L_Z^\#$ algorithm (Section 3) and benchmark the prototype algorithm (Section 4). Additionally, we construct a general lower bound proof for Mealy machine learning and lower bound proofs specific to the $L^\#$(Section 5). The proofs related to the $L_Z^\#$ algorithm can be found in Appendix A. The benchmark results and additional information on the three experiments can be found in Appendices B, C and D.

[1] Pronounced L-zarp.

2 Preliminaries

The following definitions about Mealy machines and the $L^\#$ algorithm are a subset of the preliminaries from the original $L^\#$ paper [21]. Only the most important definitions are reiterated, for further information, we refer the reader to the $L^\#$ paper. Besides preliminaries from the $L^\#$ paper, we also provide a brief summary of the workings of the $L^\#$ algorithm. Lastly, we provide some preliminaries about separating sequences.

2.1 Partial Mealy Machines, Observation Trees and Apartness

The $L^\#$ algorithm learns a hidden and complete Mealy machine. The primary data structure that $L^\#$ operates on is a *partial* Mealy machine. We first fix notation for partial maps.

We write $f\colon X \rightharpoonup Y$ to denote that f is a partial function from X to Y and write $f(x)\!\downarrow$ to indicate that f is defined on input x, $\exists y \in Y\colon f(x) = y$. We write $f(x)\!\uparrow$ if f is undefined for x. We often identify a partial function $f\colon X \rightharpoonup Y$ with the set $\{(x, y) \in X \times Y \mid f(x) = y\}$. The composition of partial maps $f\colon X \rightharpoonup Y$ and $g\colon Y \rightharpoonup Z$ is denoted by $g \circ f\colon X \rightharpoonup Z$, and $(g \circ f)(x)\!\downarrow$ holds if and only if $f(x)\!\downarrow$ and $g(f(x))\!\downarrow$. The partial order on $X \rightharpoonup Y$ is defined by $f \sqsubseteq g$ for $f, g\colon X \rightharpoonup Y$ if for all $x \in X$, $f(x)\!\downarrow$ implies $g(x)\!\downarrow$ and $f(x) = g(x)$.

Definition 2.1. *A* Mealy machine *is a tuple $\mathcal{M} = (Q, I, O, q_0, \delta, \lambda)$, where*
- *Q is a finite set of* states *and $q_0 \in Q$,*
- *I is a finite set of* inputs
- *O is a finite set of* outputs
- *$q_0 \in Q$ is the* initial state
- *$\langle \lambda, \delta \rangle\colon Q \times I \rightharpoonup O \times Q$ is a partial map whose components are an* output function *$\lambda\colon Q \times I \rightharpoonup O$ and a* transition function *$\delta\colon Q \times I \rightharpoonup Q$ (hence, $\delta(q, i)\!\downarrow \Leftrightarrow \lambda(q, i)\!\downarrow$, for $q \in Q$ and $i \in I$).*

We use superscript $\mathcal{M}$ to explain to which Mealy machine we refer, e.g. $Q^{\mathcal{M}}$, $q_0^{\mathcal{M}}$, $\delta^{\mathcal{M}}$ and $\lambda^{\mathcal{M}}$. We write $q \xrightarrow{i/o} q'$, for $q, q' \in Q$, $i \in I$, $o \in O$ to denote $\lambda(q, i) = o$ and $\delta(q, i) = q'$. We say that $\mathcal{M}$ is complete *if δ is total, i.e., $\delta(q, i)$ is defined for all states q and inputs i. We generalize the transition and output functions to input words of length $n \in \mathbb{N}$ by composing $\langle \lambda, \delta \rangle$ n times with itself: we define maps $\langle \lambda_n, \delta_n \rangle\colon Q \times I^n \to O^n \times Q$ by $\langle \lambda_0, \delta_0 \rangle = \mathsf{id}_Q$ and*

$$\langle \lambda_{n+1}, \delta_{n+1} \rangle\colon\ Q \times I^{n+1} \xrightarrow{\langle \lambda_n, \delta_n \rangle \times \mathsf{id}_I} O^n \times Q \times I \xrightarrow{\mathsf{id}_{O^n} \times \langle \lambda, \delta \rangle} O^{n+1} \times Q$$

Whenever it is clear from the context, we use λ and δ also for words.

Definition 2.2. *States q, q' in possibly different Mealy machines are* equivalent*, written $q \approx q'$, if $\lambda(q, \sigma) = \lambda(q', \sigma)$ for every $\sigma \in I^*$. Mealy machines $\mathcal{M}$ and $\mathcal{N}$ are* equivalent *if their respective initial states are equivalent: $q_0^{\mathcal{M}} \approx q_0^{\mathcal{N}}$.*

Definition 2.3 ((Observation) Tree). *A Mealy machine $\mathcal{T}$ is a* tree *if for each $q \in Q^{\mathcal{T}}$ there is a unique sequence $\sigma \in I^*$ s.t. $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) = q$. We write $\mathsf{access}(q)$ for the sequence of inputs leading to q. A tree $\mathcal{T}$ is an* observation tree *for a Mealy machine $\mathcal{M}$ if there is a functional simulation $f\colon \mathcal{T} \to \mathcal{M}$. A function simulation $f\colon \mathcal{T} \to \mathcal{M}$ is a map $f\colon Q^{\mathcal{T}} \to Q^{\mathcal{M}}$ with*

$$f(q_0^{\mathcal{T}}) = q_0^{\mathcal{M}} \qquad \text{and} \qquad q \xrightarrow{i/o} q' \text{ implies } f(q) \xrightarrow{i/o} f(q').$$

Intuitively, an observation tree is a Mealy machine that represents all inputs and outputs we have observed so far in learning. With the functional simulation, we can construct a Mealy machine based on the observation tree, see Figure 2. The $L^\#$ algorithm performs output and equivalence queries to build an observation tree for the unknown Mealy machine $\mathcal{M}$. The learner does not know the functional simulation but using the notion of apartness, the learner can infer that certain states in the observation tree cannot map to the same state of $\mathcal{M}$.

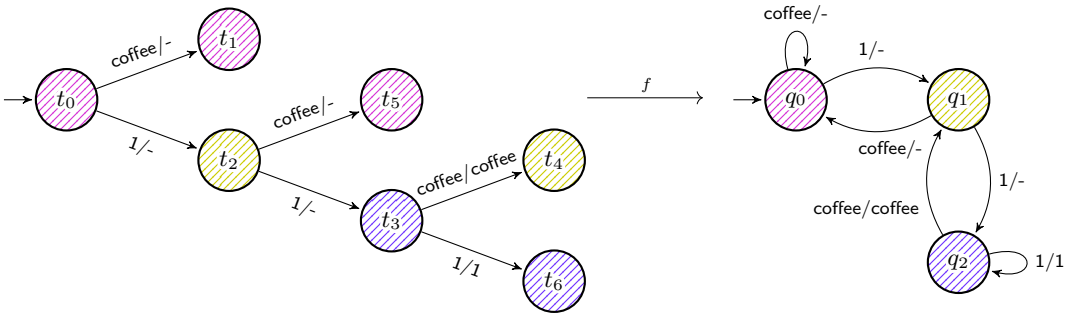


Fig. 2: *An observation tree (left) for a Mealy machine (right)*

Definition 2.4. *For a Mealy machine $\mathcal{M}$, we say that states $q, p \in Q^{\mathcal{M}}$ are* apart *(written $q \# p$) if there is some $\sigma \in I^*$ such that $\delta(q,\sigma)\downarrow$, $\delta(p,\sigma)\downarrow$, and $\lambda(q,\sigma) \neq \lambda(p,\sigma)$. We say that σ is the* witness *of $q \# p$ and write $\sigma \vdash q \# p$.*

Note that the apartness relation $\# \subseteq Q \times Q$ is irreflexive and symmetric. For the observation tree of Figure 2, we can derive the following apartness pairs and corresponding witnesses:

$$\mathit{coffee} \vdash t_0 \# t_3, \quad 1 \vdash t_2 \# t_3, \quad 1 \cdot \mathit{coffee} \vdash t_0 \# t_2$$

Whenever states are apart in the observation tree $\mathcal{T}$, the learner knows that these are distinct states in the hidden Mealy machine $\mathcal{M}$ (see Lemma 2.7 in [21]). Moreover, if $\sigma \vdash r \# r'$ and there is a sequence of transitions σ starting in state q, then q must be apart from r or r'. This notion is called weak co-transitivity and is often used in the $L^\#$ algorithm and the $L^\#_Z$ algorithm discussed in Section 3. For completeness, we copy the Lemma of weak co-transitivity from [21].

Lemma 2.5 (Weak co-transitivity). *In every Mealy machine $\mathcal{M}$,*

$$\sigma \vdash r \# r' \ \wedge \ \delta(q,\sigma)\downarrow \implies r \# q \ \vee \ r' \# q \qquad \text{for all } r, r', q \in Q^{\mathcal{M}}, \sigma \in I^*.$$

2.2 The $L^\#$ Algorithm

The task solved by active automata learning algorithms such as $L^\#$ is to find a strategy for the learner in the following game:

Definition 2.6. *In the learning game between a learner and a teacher, the teacher has a complete Mealy machine $\mathcal{M}$ and answers the following queries from the learner:*

OutputQuery(σ)**:** *For $\sigma \in I^*$, the teacher replies with the corresponding output sequence $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \in O^*$.*

EquivalenceQuery$(\mathcal{H})$**:** *For a complete Mealy machine $\mathcal{H}$, the teacher replies* `yes` *if $\mathcal{H} \approx \mathcal{M}$ or* `no`*, providing a counterexample $\sigma \in I^*$ such that $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \neq \lambda^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$.*

The $L^\#$ algorithm stores the answers in an observation tree $\mathcal{T} = (Q, q_0, \delta, \lambda)$. No other information is available about the unknown complete Mealy machine $\mathcal{M}$. The observation tree is structured into three equivalence classes:

1. The states $S \subseteq Q^{\mathcal{T}}$, each of these states represents a distinct state in the teacher's hidden Mealy machine. We call S the *basis*. Initially, $S := \{q_0^{\mathcal{T}}\}$, and throughout the execution of $L^{\#}$, S forms a subtree of $\mathcal{T}$. All states in S are pairwise apart: $\forall p, q \in S, p \neq q : p \# q$.
2. the *frontier* $F \subseteq Q^{\mathcal{T}}$, one of the frontier states will be the next state that is added to S. Throughout the execution, F is the set of immediate non-basis successors of basis states: $F := \{q' \in Q \setminus S \mid \exists q \in S, i \in I : q' = \delta(q, i)\}$.
3. the remaining states $Q \setminus (S \cup F)$.

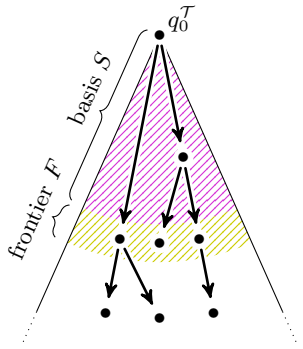


Fig. 3: *$\mathcal{T}$, S, and F*

Initially, $\mathcal{T}$ consists of only an initial state $q_0^{\mathcal{T}}$ with no transitions. With every output query or counterexample provided by the teacher, the inputs and the corresponding responses are added to the observation tree. The learner can use this information to build a hypothesis $\mathcal{H}$. A hypothesis is defined as follows.

Definition 2.7. *Let $\mathcal{T}$ be an observation tree with basis S and frontier F. A hypothesis $\mathcal{H}$ is a complete Mealy machine such that*

- $Q^{\mathcal{H}} = S$.
- $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q)) = q$ *for all* $q \in S$.
- $q \xrightarrow{i/o'} p'$ *in* $\mathcal{H}$ *and* $q \xrightarrow{i/o} p$ *in* $\mathcal{T}$ *imply* $o = o'$ *and* $\neg(p \# p')$ *in* $\mathcal{T}$ *for all* $q \in S$.

This definition indicates that the transitions in the hypothesis are based on transitions between basis states and transitions from basis to frontier states. To determine the destination of transitions from basis to frontier states, we determine for every frontier state a basis state, for which the learner conjectures that they are equivalent in the target Mealy machine. Additionally, we say that a hypothesis $\mathcal{H}$ is consistent with an observation tree $\mathcal{T}$ if all observations so far are consistent with the hypothesis $\mathcal{H}$.

Definition 2.8. *A hypothesis $\mathcal{H}$ is* consistent *with the observation tree $\mathcal{T}$ if there exists a functional simulation $f : \mathcal{T} \to \mathcal{H}$.*

The learner can check whether the hypothesis is consistent without posing queries, the algorithm for this will be explained in more detail below. Moreover, we introduce the idea of an input sequence that leads to a conflict.

Definition 2.9. *For a hypothesis $\mathcal{H}$ and an observation tree $\mathcal{T}$, an input sequence $\sigma \in I^*$* ***leads to conflict*** *if $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) \# \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$ (in $\mathcal{T}$).*

Finally, we define several concepts related to the observation tree.

Definition 2.10. *In an observation tree $\mathcal{T}$, the* ***candidate set*** *of some state $q \in Q^{\mathcal{T}}$ is defined as follows.*

$$C(q) = \{p \in S \mid \neg(p \# q)\}$$

A state $q \in F$ is ***isolated*** *if it is apart from all states in S or $C(q) = \emptyset$. A state $q \in F$ is* ***identified*** *if it is apart from all states in S except one, or when $C(q)$ is a singleton. The basis S is* ***complete*** *if each state in S has a transition for each input $i \in I$.*

The $L^{\#}$ algorithm poses output queries and equivalence queries to extend the basis. When the basis is big enough, the hidden Mealy machine of the teacher is found. The algorithm uses four rules that can be applied non-deterministically until none of them can be applied anymore:

(R1) If F contains an isolated state, then we have discovered a new state not yet present in S, hence we move it from F to S.

(R2) When a state $q \in S$ has no outgoing i-transition, for some $i \in I$, the output query for $\mathsf{access}(q)\ i$ extends the frontier by adding the generated successor state.

(R3) When $q \in F$ is a state in the frontier that is not yet identified, then q is not apart from at least two states $r, r' \in S$. In that case, the algorithm picks a witness $\sigma \in I^*$ for $r \# r'$. After the OutputQuery($\mathsf{access}(q)\,\sigma$), the observation tree is extended, and thus q will be apart from at least r or r' by weak co-transitivity.

(R4) When the frontier has no isolated states and the basis is complete, a hypothesis $\mathcal{H}$ is built with BuildHypothesis based on the basis and frontier. Using the function CheckConsistency we check whether $\mathcal{H}$ is consistent with the observation tree. If $\mathcal{H}$ is not consistent, we get a conflict σ. Otherwise, we ask an equivalence query for $\mathcal{H}$. If the hypothesis is correct, $L^\#$ terminates, and otherwise, we obtain a counterexample ρ. This counterexample is handled by ProcCounterEx and as a result, $\mathcal{H}$ is not a hypothesis for the updated $\mathcal{T}$ anymore.

In rule (R4), several sub-routines are used. We will briefly explain how these sub-routines work:

BuildHypothesis We build a hypothesis based on the information in the observation tree.

CheckConsistency The hypothesis is not always consistent with the observation tree. Using a breadth-first search of the Cartesian product of $\mathcal{T}$ and $\mathcal{H}$, we check whether a functional simulation exists. If no functional simulation exists, this sub-routine returns a sequence $\sigma \in I^*$ leading to a conflict. Otherwise, it returns a boolean indicating that $\mathcal{H}$ is consistent with $\mathcal{T}$.

ProcCounterEx The counterexample decomposes into two words $\sigma\eta$, where σ leads to the conflict and η witnesses it. The input sequence σ indicates that one of the frontier states was merged with a basis state in $\mathcal{H}$ while they are distinct states in the target automaton, causing a wrong transition in $\mathcal{H}$. Since σ can be very long, ProcCounterEx(σ) tries to reduce the length of σ using binary search. Let $q = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$ and $r = \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$. If $r \in S \cup F$, then r must be apart from all other states in S and $\mathcal{H}$ is not a hypothesis anymore. Otherwise, let $\sigma_1\sigma_2 := \sigma$ such that running σ_1 in $\mathcal{T}$ ends halfway between the frontier and r. If σ_1 already leads to conflict, we can avoid using σ_2 in recursive calls. Otherwise, we can avoid using σ_1 in recursive calls. After ProcCounterEx, $\mathcal{H}$ is not a hypothesis for the updated $\mathcal{T}$ anymore.

In practice, equivalence queries are very expensive and we would like to avoid posing equivalence queries as long as possible. In the case of $L^\#$, this means only applying rule (R4) when no other rules are applicable. This variant is called strategic $L^\#$. Another variant of $L^\#$ uses adaptive distinguishing sequences (ADS) to reduce the number of output queries needed to identify frontier states. In this thesis, we will focus on strategic $L^\#$.

2.3 Separating Sequences

As mentioned in the introduction, for any pair of distinct states in a Mealy machine with n states, a sequence of length at most n exists that shows that the two states are distinct. Formally, these sequences are called separating sequences [19] and are often used in the field of conformance testing [4, 13]. For completeness, we copy the definition of a separating sequence from [19].

Definition 2.11. *A separating sequence for states q and q' is a sequence $\sigma \in I^*$ such that $\lambda(q, \sigma) \neq \lambda(q', \sigma)$.*

A separating sequence is similar to a witness. The main difference between the two sequences is that the length of a separating sequence is always less than the number of states in the Mealy machine, this is not the case for a witness. Additionally, separating sequences are always generated using a partition refinement algorithm while witnesses are found by posing queries to the teacher. In the new variation of $L^\#$, separating sequences are used to ensure that the length of the witness between any two basis states is shorter than $|\mathcal{H}|$, where $|\mathcal{H}|$ is the number of states in the current hypothesis.

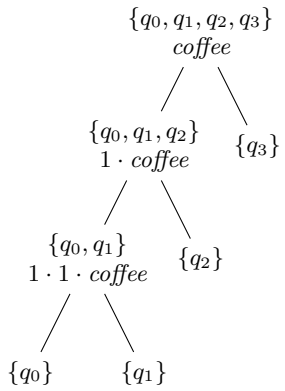


Fig. 4: *The splitting tree for Figure 1a generated by Algorithm 1 described in [19]*

Many algorithms exist to find the separating sequence for a specific Mealy machine. For example, the algorithms by Moore [15], Hopcroft [6] and the algorithm by Smetsers et al. [19]. All these algorithms are based on the concept of partition refinement. A partition P of some set S is a set of pairwise disjoint and non-empty subsets of S. The union of the partition is exactly S. The term *block* is used to denote an element in P. A partition P' is a refinement of P, if P and P' are both partitions of S and every block in P' is a subset in a block of P. In this setting, the goal of partition refinement algorithms is to construct the finest partition, such that all equivalent states belong to the same block.

Partition refinement algorithms that construct separating sequences build splitting trees where the root node contains the set of all states. That set is then repeatedly refined until every leaf is a set of equivalent states. Let B be a block and i an input symbol. There are two ways to refine B:

1. We can split B w.r.t the output $\lambda(B, i)$. We create a new block for each subset of B that gives the same output for input i. The separating sequence that distinguishes the blocks resulting from the split is i.
2. We can split B w.r.t the state after i. Each state after i defines a new block. Let σ be the separating sequence that distinguishes the states after i. The separating sequence that distinguishes the blocks resulting from the split is $i \cdot \sigma$.

An algorithm that implements these rules can be found in Algorithm 1 from [19]. In this algorithm, we always try to split w.r.t. output. Only if that is not possible, splits w.r.t. state after i are considered. Algorithm 1 terminates and is proven to be correct [19]. We present an example run of Algorithm 1 on the Mealy machine in Figure 1a to make the concept of building separating sequences clear.

Example. The splitting tree displayed in Figure 4 is constructed by Algorithm 1 as follows. First, the root node is initialized as $\{q_0, q_1, q_2, q_3\}$. This block can be split w.r.t. the output after $coffee$ because $\lambda(q_3, \textit{coffee}) = 1$ while $\lambda(q, coffee) = 0$ for $q \in \{q_0, q_1, q_2\}$. This leads to new blocks $\{q_3\}$ and $\{q_0, q_1, q_2\}$. No more splits w.r.t. output can be performed to split $\{q_0, q_1, q_2\}$ because the output is $\delta(q, i) = -$ for all $i \in I$ and $q \in \{q_0, q_1, q_2\}$. It is possible to split $\{q_0, q_1, q_2\}$ w.r.t. the state after 1 because q_3 is already split from q_1, q_2 and $\delta(q_0, 1) = q_1$, $\delta(q_1, 1) = q_2$ and $\delta(q_2, 1) = q_3$. This leads to the new blocks $\{q_2\}$ and $\{q_0, q_1\}$. The block $\{q_0, q_1\}$ can be split w.r.t. the state after 1, similar to the previous step. After this step, the algorithm terminates because each state belongs to a distinct block. The separating sequences are displayed directly under the nodes in Figure 4.

3 Heuristics and Optimizations for $L^\#$

In this section, we discuss a variation on the $L^\#$ algorithm called $L^\#_Z$. This algorithm avoids using long witnesses as long as possible. When long witnesses can no longer be avoided, $L^\#_Z$ first identifies the immediate successors of a basis state which has a long witness with some other basis state. In some sense, these new rules resolve a part of the non-determinism present in rule (R3) of the $L^\#$ algorithm. Additionally, two new rules are introduced that always lead to a new basis state or a short witness, but can only be applied in very specific scenarios. Moreover, similar to the TTT-algorithm [10], $L^\#_Z$ guarantees that all witnesses are short before a new equivalence query is posed by using separating sequences.

First, we introduce new definitions and split the basis into two equivalence classes W and Z (Section 3.1). Next, we provide some motivation and lemmas related to the new rules (Section 3.2). In Section 3.3, the main loop of $L^\#_Z$ is presented and proven correct. Finally, a complexity analysis of $L^\#_Z$ is presented in Section 3.4.

3.1 Definitions and Observation Tree Sets

In this section, we define when a witness is short or long. Based on this definition, we split the basis into two equivalence classes W and Z.

Definition 3.1. *Suppose S is the basis for some observation tree $\mathcal{T}$. Then we call a witness $\sigma \in I^+$ **short** if $|\sigma| \leq |S|$ and **long** otherwise.*

Alternative definitions of short and long witnesses are possible. For example, $\sigma \in I^+$ is short if $|\sigma| \leq 5 \cdot |S|$ would also be a valid definition for the purpose of this thesis. The exact border between short and long witnesses is irrelevant. It is only important that the case distinction between short and long witnesses can be made. Because there are two different types of witnesses, we introduce new notation for apartness with those two types of witnesses.

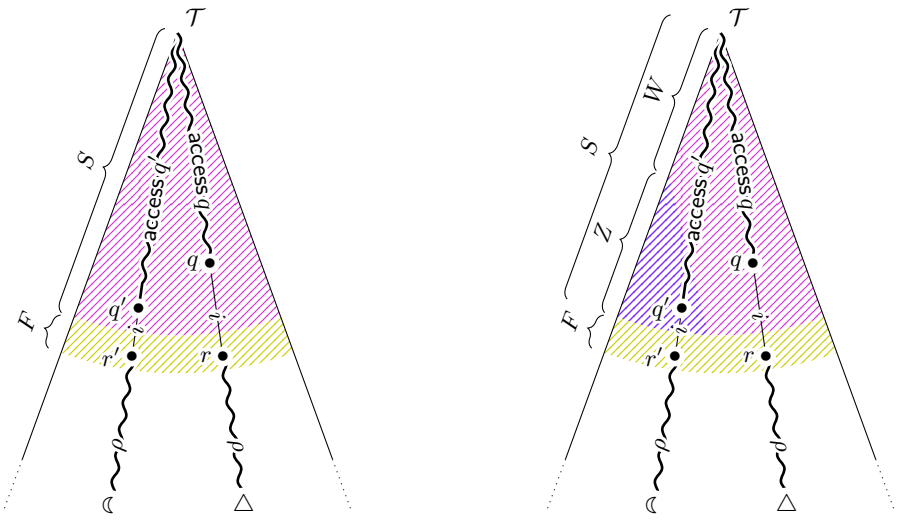
Definition 3.2. *For a Mealy machine $\mathcal{M}$, we say that states $q, p \in Q^{\mathcal{M}}$ are **s-apart** (written $q \#_s p$) if there is some $\sigma \in I^*$ such that σ is short and $\sigma \vdash q \# p$.*

Definition 3.3. *For a Mealy machine $\mathcal{M}$, we say that states $q, p \in Q^{\mathcal{M}}$ are **l-apart** (written $q \#_l p$) if $q \# p$ and $\neg(q \#_s p)$.*

In the original strategic $L^\#$ algorithm, the observation tree is split up into three equivalence classes S, F, and the remaining states $Q \setminus (S \cup F)$, see Section 2.2. We propose a refinement of the equivalence classes mentioned above. This modification splits the original basis S into two equivalence classes, W and Z, based on the length of the shortest witness.

- the *s-basis* $W \subseteq Q^{\mathcal{T}}$. All states in W are pairwise apart with a short witness, $\forall p, q \in W, p \neq q : p \#_s q$.
- the *l-basis* $Z \subseteq Q^{\mathcal{T}}$. All states in Z are apart from all states in $W \cup Z$, $\forall p, q \in W \cup Z, p \neq q : p \# q$. Moreover, for all $p \in Z$ there is at least one state $q \in W$ such that $\sigma \vdash p \#_l q$.

For every pair of inequivalent states in a Mealy machine a short witness exists [15]. Therefore, if the hypothesis is correct, then short witnesses should be available between each pair of basis states. Intuitively, W contains states for which these short witnesses are already available and Z contains states for which we still have to find at least one short witnesses. Figure 5 shows the difference between the equivalence classes in $L^\#$ and $L^\#_Z$.



(a) *Schematic depiction of the original observation tree equivalence classes*

(b) *Schematic depiction of the new observation tree equivalence classes*

Fig. 5: *Schematic depiction of the original observation tree (left) and the new observation tree (right)*

3.2 Motivation for New Rules

Consider the target automata in Figure 1. As stated in the introduction, strategic $L^{\#}$ poses an equivalence query with the initial hypothesis displayed in Figure 1b. Suppose the teacher returns the following counterexample.

1 coffee 1 coffee coffee 1 1

Then the ProcCounterEx sub-routine processes the counterexample. In general, this always leads to a new witness between a frontier and basis state. However, if all the frontier states are identified before the equivalence query, the procedure finds a new state. In this example, all frontier states were identified and ProcCounterEx finds the new state t_3. Suppose that after ProcCounterEx, rules (R1) and (R2) are applied, then this leads to the observation tree displayed in Figure 6. Before we can pose a new equivalence query, all frontier states have to be identified according to strategic $L^{\#}$. Currently, the frontier states t_1, t_{21} and t_{22} are not identified yet because each of these frontier states is not apart from t_0 and t_3. To further identify the frontier states, we can perform output queries with the witness *coffee 1 coffee coffee 1 1* $\vdash t_0 \# t_3$. The order in which the frontier states are identified is not specified in strategic $L^{\#}$, therefore a possible order is t_1, t_{21}, t_{22}. Only the output query performed to identify frontier state t_{22} provides new information, it shows that

OutputQuery(*1 1 coffee 1 coffee coffee 1 1*) = - - - - *coffee* - - -

With this new observation, we find that state t_{22} is apart from states t_0 and t_3 with witness *coffee 1 coffee*. This also leads to a shorter witness between states t_0 and t_3, *1 coffee 1 coffee*. However, we had to use long witnesses on all frontier states in observation tree $\mathcal{T}_1$ (Figure 6) before we found a shorter witness and a new state while we already knew that a shorter witness or an additional basis should exist. This holds because if the hidden Mealy machine has two states, then a witness of length 1 must exist. If the Mealy machine has more than two states, then a state is missing in the current basis. In general, if we have some witness σ with $|\sigma| > |S|$, then the following holds.

- A shorter witness exists.
- A state is missing in the basis

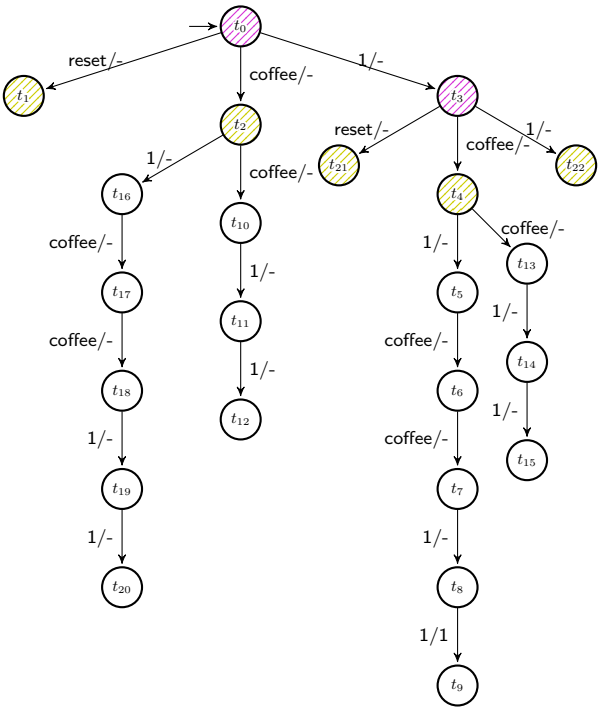


Fig. 6: *Observation tree* $\mathcal{T}_1$

The new rule (R3_{long}) tries to find a shorter witness or a new basis state when we have a long witness between two basis states. Specifically, rule (R3_{long}) further identifies some frontier state $r \in F$ with OUTPUTQUERY(access(r) ρ) if

- $\sigma \vdash q \# q'$ for some $\sigma \in I^+$ and $q \in S$ and $q' \in Z$.
- $\delta(q', i) = r$ for some $i \in I$.
- i occurs in the witness σ.
- $\neg(r \# u), \neg(r \# u')$, $\rho \vdash u \# u'$ for some $\rho \in I^+$ and $u, u' \in S$

This rule often removes redundant symbols in the long witness and, therefore, finds a shorter witness. In this example, the *coffee* symbols are redundant in *coffee 1 coffee coffee 1 1*. If we use rule (R3_{long}) after rules (R1) and (R2) instead of (R3), we find the shorter witness *1 coffee 1 coffee* which already removed two of the redundant *coffee* symbols. Additionally, we find new state t_{22} without first identifying t_{21} and t_1 with long witnesses. This is a difference of only two queries with long witnesses. However, in models with many input symbols, rule (R3_{long}) can lead to significantly fewer output queries with long witnesses.

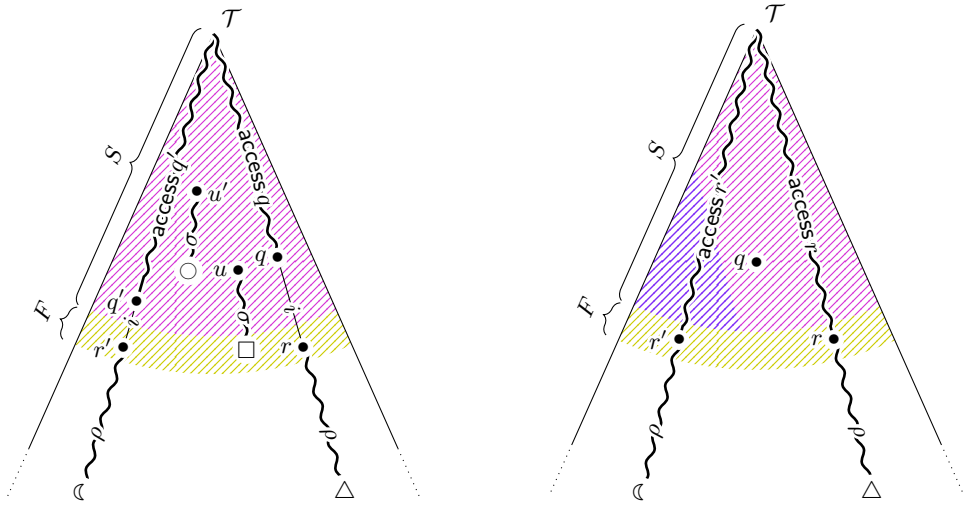
Moreover, strategic $L^\#$ does not prioritize output queries with short witnesses over output queries with long witnesses. However, output queries with short witnesses usually require fewer input symbols. Therefore, we introduce the new rule (R3_{short}). This rule only uses short witnesses to identify frontier states. If we use rule (R3_{short}) before any other rule that uses long witnesses, we ensure that output queries with long witnesses are only performed when absolutely necessary.

Additionally, we introduce two new rules that exploit special cases in the observation tree. Rule (R5) exploits the scenario in Figure 7a with additional conditions $C(r) = u, C(r') = u'$ and $|\sigma| < |S|$. In this special case, we have a long witness between q and q' and frontier states r and r' are on the path of the long witness. These frontier states each have only one candidate state left and there is a short witness between the candidate states. We always want to perform an output query with the short witness from states r and r' because this is guaranteed to give us useful information. After the output query, we always find a new basis state or a short witness between q and q', as stated in Lemma 3.4.

Lemma 3.4. *Let $q, q' \in S$ with $i\rho \vdash q \#_l q'$ for some $i \in I, \rho \in I^+$. Moreover, let $r, r' \in S \cup F$ and $\delta^{\mathcal{T}}(q, i) = r, \delta^{\mathcal{T}}(q', i) = r'$. Moreover, let $C(r) = \{u\}$ and $C(r') = \{u'\}$ and $\sigma \vdash u \#_s u'$ for some $\sigma \in I^+$ with $|\sigma| < |S|$. If we perform an output query from r and r' with σ, then we always find a new basis state or $i\sigma \vdash q \#_s q'$.*

Rule (R6) exploits the scenario in Figure 7b with additional condition $C(r) = \{q\} = C(r')$. In this special case, we have a witness that shows that frontier states r and r' are inequivalent. However, they map to the same state in the basis q. This implicitly means that the current hypothesis is inconsistent. If we perform an output query from state q with σ, we will always find that q is no longer a candidate state for either r or r'. This is useful information because this always gives us a new basis state, which follows from Lemma 3.5.

Lemma 3.5. *Let $r, r' \in F$ with $\sigma \in r \# r'$ for some $\sigma \in I^+$. Moreover, let $C(r) = \{q\} = C(r')$ with $q \in S$. If we perform an output query from q with σ, then r or r' is apart from all states in the basis after the output query.*



(a) *Schematic depiction of the observation tree when rule (R5) can be applied. Additionally, we have conditions $C(r) = u,\ C(r') = u'$ and $|\sigma| < |S|$*

(b) *Schematic depiction of the observation tree when rule (R6) can be applied. Additionally, we have condition $C(r) = \{q\} = C(r')$*

Fig. 7: *Schematic depiction of the observation trees when rules (R5) and (R6) can be applied*

The rules (R5) and (R6) always find a short witness or a new state. The rules to identify frontier states are not guaranteed to find a short witness or a new state, only a new apartness pair. Therefore, we prioritize using rules (R5) and (R6) over rules to identify frontier states with long witnesses.

3.3 Main Loop of $L_Z^\#$

Similar to the main loop of strategic $L^\#$, the main loop of $L_Z^\#$ uses the guarded command notation as introduced by Dijkstra [5]. The guarded command notation indicates that the rules described below can be applied non-deterministically until none of them are applicable anymore. The pseudocode of the $L_Z^\#$ algorithm is listed in Algorithm 1.

(R1$_W$) If there exists a state $r \in Z$ that has a short witness with all states in W, we can move state r from Z to W. After this rule, W still only contains states that are pairwise apart with a short witness.

Algorithm 1 Overall $L_Z^\#$ algorithm

procedure LZARP
  do $r \#_s q$, for all $q \in W$ and for some $r \in Z \rightarrow$ ▷ Rule ($R1_W$)
    $W \leftarrow W \cup \{r\}$
    $Z \leftarrow Z \setminus \{r\}$
  q isolated, for some $q \in F \rightarrow$ ▷ Rule ($R1_Z$)
    $Z \leftarrow Z \cup \{q\}$
  $\delta^{\mathcal{T}}(q, i)\uparrow$, for some $q \in S, i \in I \rightarrow$ ▷ Rule (R2)
    OUTPUTQUERY(access(q) i)
  $\neg(r \# q')$, $\neg(r \# q)$, $\sigma \vdash q' \#_s q$, for some $\sigma \in I^+$, $r \in F$, and $q',\ q \in S \rightarrow$ ▷ Rule ($R3_{short}$)
    OUTPUTQUERY(access(r) σ)
  $\sigma \vdash q \#_l q'$ for $\sigma \in I^+$ and $q \in S,\ q' \in Z$,
    $r = \delta^{\mathcal{T}}(q', i)$ with σ contains i, for some $r \in F$ and $i \in I$
    $\neg(r \# u), \neg(r \# u'), \rho \vdash u \# u'$ for some $\rho \in I^+$ and $u, u' \in S \rightarrow$ ▷ Rule ($R3_{long}$)
    OUTPUTQUERY(access(r) ρ)
  $\neg(r \# q')$, $\neg(r \# q')$, $\sigma \vdash q' \# q$ for some $r \in F$, $q, q' \in S$ and $\sigma \in I^+ \rightarrow$ ▷ Rule ($R3_{orig}$)
    OUTPUTQUERY(access(r) σ)
  F has no isolated states, basis S is complete $\rightarrow$ ▷ Rule (R4)
    $\mathcal{H} \leftarrow$ BUILDHYPOTHESIS
    $(b, \sigma) \leftarrow$ CHECKCONSISTENCY($\mathcal{H}$)
    if $b =$ yes **then**
      if there exists $q', q \in S$ with $q' \#_l q$ **then**
        COMPUTESEPSEQ($\mathcal{H}$)
      else
        $(b, \rho) \leftarrow$ EQUIVALENCEQUERY($\mathcal{H}$)
        if $b =$ yes **then: return** $\mathcal{H}$
        else: $\sigma \leftarrow$ shortest prefix of ρ such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$ (in $\mathcal{T}$)
        PROCESSCOUNTEREXAMPLE($\mathcal{H}, \sigma$)
      end if
    else
      PROCESSCOUNTEREXAMPLE($\mathcal{H}, \sigma$)
    end if
  $i\rho \vdash q \#_l q'$ for some $i \in I,\ \rho \in I^+$ and $q, q' \in S$,
    $\delta^{\mathcal{T}}(q, i) = r, \delta^{\mathcal{T}}(q', i) = r'$ for $r, r' \in S \cup F$,
    $C(r) = \{u\}, C(r') = \{u'\}$, $\sigma \vdash u \#_s u'$ with $|\sigma| < |S|$
    for some $\sigma \in I^+$ and $u, u' \in S \rightarrow$ ▷ Rule (R5)
    OUTPUTQUERY(access(r) σ)
    OUTPUTQUERY(access(r') σ)
  $\sigma \vdash r \# r'$ and $C(r) = \{q\} = C(r')$ for some $\sigma \in I^+$, $q \in S$, and $r, r' \in F \rightarrow$ ▷ Rule (R6)
    OUTPUTQUERY(access(q) σ)
  end do
end procedure

($R1_Z$) If F contains an isolated state, then this means that we have discovered a new state not yet present in S. Because it is uncertain whether this new state has a short witness with all states in W, we move it from F to Z.

(R2) When a state $q \in S$ has no outgoing i-transition, for some $i \in I$, the output query for access(q) i extends the frontier by adding the generated successor state.

($R3_{short}$) When $r \in F$ is a state in the frontier that is not yet identified, then there are at least two states $q', q \in S$ that are not apart from r. If there exists a witness $\sigma \in I^+$ with $\sigma \vdash q \#_s q'$, then we can use this short witness to extend the observation tree. After the OUTPUTQUERY(access(r) σ), r will be apart from at least q' or q by weak co-transitivity.

(R3$_{long}$) Suppose we have $q \in S,\ q' \in Z$ and $\delta^{\mathcal{T}}(q', i) = r$ and i is in a long witness between q' and q. If $r \in F$ is an unidentified frontier state, then there are at least two states in $u', u \in S$ that are not apart from r. If there exists a witness $\rho \in I^+$ with $\rho \vdash u \# u'$, then we can use this witness to extend the observation tree. After the OutputQuery($\mathsf{access}(r)\,\rho$), r will be apart from at least u' or u by weak co-transitivity.

(R3$_{orig}$) When $r \in F$ is a state in the frontier that is not yet identified, then there are at least two states in $q', q \in S$ that are not apart from r. If there exists a witness $\sigma \in I^+$ with $\sigma \vdash q' \# q$, then we can use this witness to extend the observation tree. After the OutputQuery($\mathsf{access}(r)\,\sigma$), r will be apart from at least q' or q by weak co-transitivity. Note that it does not matter whether the witness is short or long for this rule.

(R4) When F has no isolated states and S is complete, we apply rule (R4) which uses the sub-routines BuildHypothesis and CheckConsistency to check whether the model is consistent. If the model is inconsistent, then we process the counterexample that witnesses the conflict. If the model is consistent, then we first check whether all pairs of states have a short witness. If there is a pair of states that has no short witness, then we use the new sub-routine ComputeSepSeq which ensures that all pairs of states have short witnesses after the sub-routine if no new states are discovered. If all pairs of states in the basis have a short witness, then we pose an equivalence query and execute the same steps as in the $L^\#$ algorithm. The sub-routines BuildHypothesis. CheckConsistency and ProcessEquivalenceQuery are exactly the same as in the $L^\#$ paper [21]. Because no changes are made that intervene with the correctness and termination of these sub-routines, the proofs are not reiterated in this thesis.

(R5) Suppose $q, q' \in S$ with $i\rho \vdash q \#_l q'$ for some $i \in I, \rho \in I^+$. Moreover, let $r, r' \in S \cup F$ and $\delta^{\mathcal{T}}(q, i) = r, \delta^{\mathcal{T}}(q', i) = r'$. Let $C(r) = \{u\}$ and $C(r') = \{u'\}$ and $\sigma \vdash u \#_s u'$ for some $\sigma \in I^+$ with $|\sigma| < |S|$. Then we can perform an output query from r and r' with σ and this guarantees that we find a short witness or a new basis state (Lemma 3.4).

(R6) When $r, r' \in F$ map to the same basis state q but we have a witness $\sigma \in I^+$ with $\sigma \vdash r \# r'$, then we can perform an output query from q with σ. This guarantees that r or r' is apart from all other states in the basis (Lemma 3.5).

Before the correctness and termination of the overall $L_Z^\#$ algorithm are discussed, we explain the sub-routine ComputeSepSeq and prove that it terminates and is correct. The sub-routine ComputeSepSeq computes separating sequences for the generated hypothesis using Algorithm 1 from [19] as described in Section 2.3. To compute separating sequences in the hypothesis, it must hold that all pairs of states in the hypothesis are inequivalent, i.e. the hypothesis has to be minimal. This constraint is immediately satisfied because we only enter sub-routine ComputeSepSeq if the hypothesis is consistent and consistency implies that the hypothesis is minimal, see Lemma 3.6.

Lemma 3.6. *If a hypothesis $\mathcal{H}$ is consistent, then $\mathcal{H}$ is also minimal.*

However, because hypothesis $\mathcal{H}$ is not final and it is possible that there are undiscovered states. Therefore, we have to test whether the found separating sequence indeed distinguishes the pair of states. This can be done with an OutputQuery from both states with the separating sequence. This will either lead to a short witness or a new state, see Lemma 3.7. If we find a new state, we terminate the sub-routine ComputeSepSeq. The sub-routine ComputeSepSeq is listed in Algorithm 2, PartitionRefinement refers to Algorithm 1 from [19].

Lemma 3.7. *Algorithm 2 terminates and is correct, that is, for each pair of states $q, q' \in S$, there is a witness σ that shows that these two states are apart and $|\sigma| \leq |S|$ or after termination the hypothesis $\mathcal{H}$ is no longer a hypothesis for $\mathcal{T}$.*

With all the sub-routines proven correct and guaranteed to terminate, we can prove that the overall $L_Z^\#$ algorithm terminates and is correct. Correctness of $L_Z^\#$ amounts to proving termination because the algorithm only terminates when the learner has found the correct model. To prove termination,

Algorithm 2 Uses partition refinement to identify possible separating short witnesses.

```
procedure ComputeSepSeq(H)
    SplittingTree ← PartitionRefinement(H)
    for (q', q) with q' #_ι q for some q', q ∈ S do
        sepseq ← sequence that belongs to the node in SplittingTree that splits q' and q
        OutputQuery(access(q') sepseq)
        OutputQuery(access(q) sepseq)
        if C(q') = ∅ ∨ C(q) = ∅ then
            break
        end if
    end for
end procedure
```

we use a similar approach as is taken in the original $L^\#$ paper [21]; we show that with each rule $W, Z, F, \#_s$ (restricted to $S \times S$) or $\#$ (restricted to $S \times F$) grows. Because the sets are all bounded by the hidden Mealy machine, the algorithm must terminate. We define the norm $\mathcal{N}(\mathcal{T})$ by

$$N_W(\mathcal{T}) + N_Z(\mathcal{T}) + |N_\downarrow(\mathcal{T})| + |N_{\#_s - S\times S}(\mathcal{T})| + |N_{\# - S\times F}(\mathcal{T})| \tag{1}$$

where the abbreviations are defined as follows:

$$N_W(\mathcal{T}) = 2|S|\cdot(|W|+1)$$
$$N_Z(\mathcal{T}) = |S|\cdot(|Z|+1)$$
$$N_\downarrow(\mathcal{T}) = \{(q,i) \in S \times I \mid \delta(q,i)\downarrow\}$$
$$N_{\#_s - S\times S}(\mathcal{T}) = \{(q,q') \in S \times S \mid q \#_s q'\}$$
$$N_{\# - S\times F}(\mathcal{T}) = \{(q,q') \in S \times F \mid q \# q'\}$$

Theorem 3.8. *Every rule application in $L^\#_Z$ increases the norm in 1.*

3.4 Complexity Analysis

Before we discuss the query and symbol complexity of $L^\#_Z$, we introduce **strategic** $L^\#_Z$. Similar to strategic $L^\#$, strategic $L^\#_Z$ postpones asking equivalence queries as long as possible. However, strategic $L^\#_Z$ also prioritizes posing output queries with short witnesses.

Definition 3.9. ***Strategic*** *$L^\#_Z$ is a special case of Algorithm 2 where we prioritize applying certain rules over applying other rules.*

$$(R1_W),\ (R1_Z),\ (R2),\ (R3_{short}),\ (R5) > (R6) > (R3_{long}) > (R3_{orig}) > (R4)$$

In practice, we apply the first five rules in the order ($\mathrm{R1}_W$), ($\mathrm{R1}_Z$), (R2), ($\mathrm{R3}_{short}$), (R5). It is allowed to apply, for example, ($\mathrm{R3}_{short}$) when ($\mathrm{R1}_W$) can be applied as well. We obtain the following query complexity for the strategic $L^\#_Z$ algorithm.

Theorem 3.10. *Strategic $L^\#_Z$ learns the correct Mealy machine within $\mathcal{O}(kn^2 + n\log m)$ output queries and at most $n-1$ equivalence queries.*

The query complexity of $L^\#_Z$ equals the query complexity for $L^\#$ and the other best known query complexity for active learning algorithms, like the TTT algorithm [10] and observation pack [7]. As mentioned in the paper by Vaandrager et al. [21], the total number of input symbols is another useful metric for comparing learning algorithms. The symbol complexity of $L^\#_Z$ is the same as the symbol complexity of $L^\#$.

Theorem 3.11. *If $n \in \mathcal{O}(m)$, then strategic $L^\#_Z$ learns the correct Mealy machine with $\mathcal{O}(kmn^2 + nm\log m)$ input symbols.*

4 Experimental Evaluation

In the previous section, we have introduced the $L_Z^\#$ algorithm. In this section, we describe several experiments to show that $L_Z^\#$ outperforms $L^\#$ if long witnesses frequently occur and is comparable to $L^\#$ otherwise. All source code and data is available online.[2]

4.1 Experiment 1

In Experiment 1, we show that $L_Z^\#$ is competitive with $L^\#$ using the benchmarks also used in the $L^\#$ paper [21].

Implementing equivalence queries. Equivalence queries make use of conformance testing with the Wp-method [13] [3]. The Wp-method builds counterexamples in three parts: an access sequence to a certain state, a random input sequence, and a state identifier. We check for 5 extra states, and the expected number of infix symbols is set to 2. With the expected number of infix symbols, an input sequence of expected length 2 is generated using a geometric distribution. The random input sequence between the access sequence and the state identifier contains a subsequence of length 5, from the extra states, and the subsequence generated with the expected number of infix symbols.

Data-set and metrics. We use the benchmarks from [16] which are also used in the experimental evaluation of the $L^\#$ algorithm [21]. The largest model has 66 states and 13 input symbols. We record the number of output queries and input symbols used during learning and testing. We also record the number of equivalence queries required to learn each model. The number of input symbols is the length of the sequence σ in OutputQuery(σ) plus one reset symbol. The reset symbol shows the number of output queries because a reset symbol is used to reset the system under test (SUT) to its initial state. Additionally, we record the number of symbols used during queries originating from rules/sub-routines (R3$_{short}$), (R5), (R6), (R3$_{long}$), (R3$_{orig}$), ComputeSepSeq and ProcCounterEx. Moreover, we compute the difference in performance per model as follows.

$$diff_m = \frac{\#symbols\ used\ in\ L_Z^\#\ for\ model\ m - \#symbols\ used\ in\ L^\#\ for\ model\ m}{\#symbols\ used\ in\ L^\#\ for\ model\ m} \cdot 100\% \quad (2)$$

Then we compute the total difference in performance as follows:

$$total\text{-}diff = \frac{\sum_{m \in models} diff_m}{\#models} \quad (3)$$

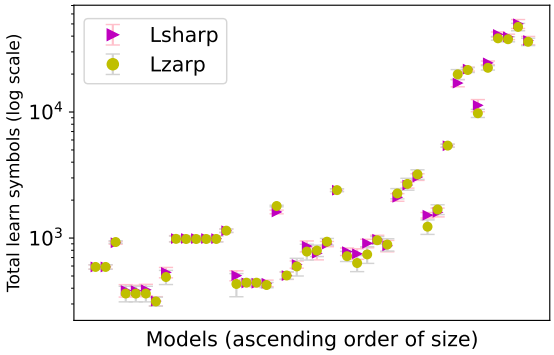
here *symbols used in* can indicate total learn + test symbols, total learn symbols or total learn symbols for queries with long witnesses.

Experiment set-up. All experiments were run on a Ryzen 5 4500U processor with 16GB of memory, running Windows. In each experiment, a complete model of the SUT is learned. Each experiment is repeated 50 times to account for the effects of randomization.

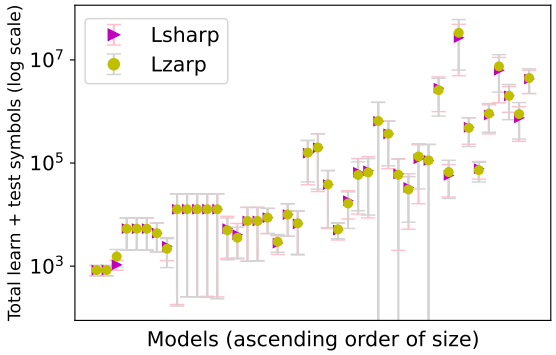
Results and discussion. Figure 8a shows the total number of input and reset symbols sent by the learning algorithms via output queries, this accounts for both the size and the number of output queries. Figure 8b shows the total size of data sent during learning and testing. This includes the input symbols sent to the SUT by the teacher to find counterexamples. Figure 8c shows the number of input symbols in output queries with long witnesses originating from the rules (R6), (R3$_{long}$) and (R3$_{orig}$). The y-axis is log-scaled in the plots in the top row. The models, sorted in increasing number of states, are displayed on the x-axis. The bars indicate the standard deviation.

[2] `https://gitlab.science.ru.nl/sws/lsharp/-/tree/loeslongwitnesses`

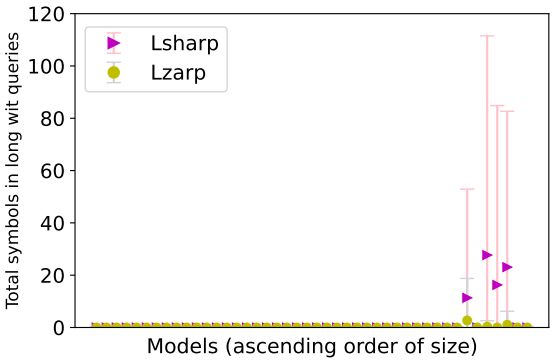
[3] A different testing method was chosen due to incompatibilities between the Hybrid-ADS implementation requirements and the operating system of the author of this thesis



(a) Symbols used during learning phase



(b) Symbols used both learning and testing



(c) Symbols used during long witness queries

Fig. 8: *Performance plots of the $L^\#$ and $L_Z^\#$ on benchmark models from [16] with the Wp-method, lower is better*

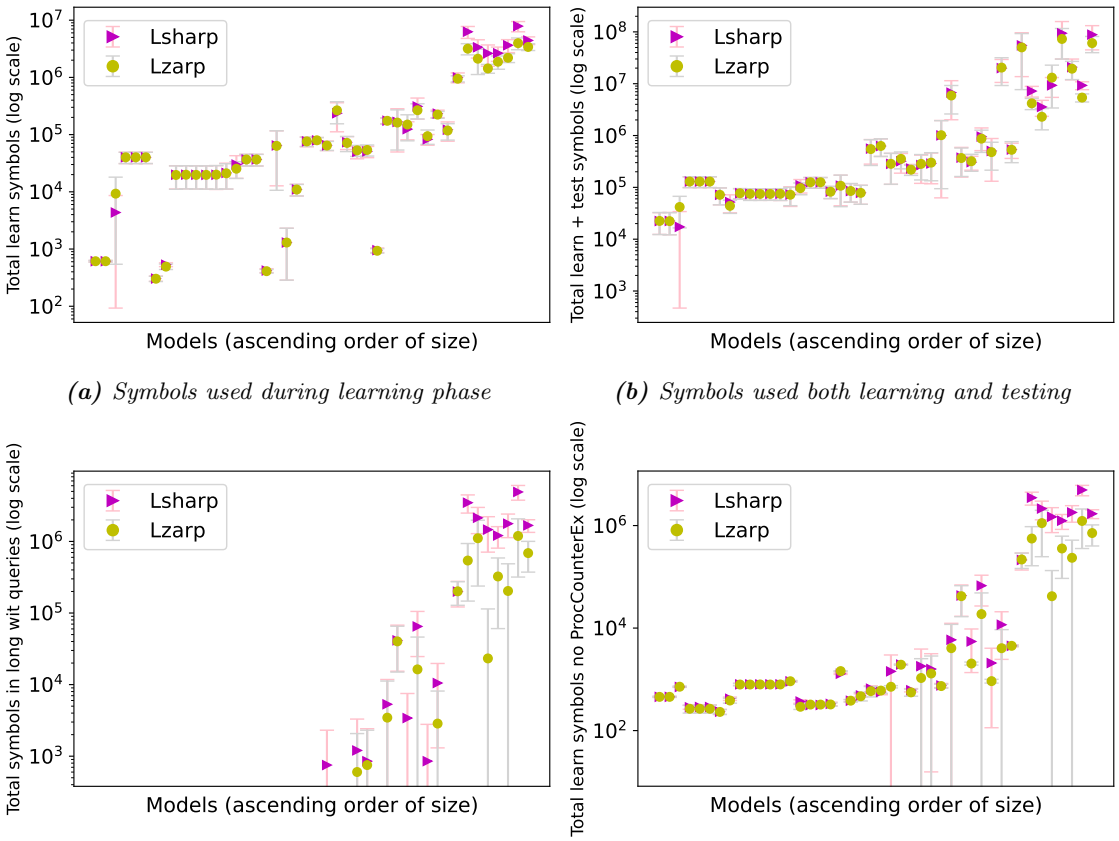
From Figure 8a, we can observe that $L_Z^\#$ is competitive with $L^\#$. The $L_Z^\#$ algorithm seems to outperform $L^\#$ on certain models, however, $L_Z^\#$ only uses 2.3% fewer symbols during learning. Moreover, we can observe that the error bars are small, thus the measurements are stable. From Figure 8b, we can observe that $L_Z^\#$ and $L^\#$ are also competitive w.r.t. the number of input symbols used during testing and learning. Here, $L_Z^\#$ uses 1.6% more symbols. From Figure 8c, we can observe that output queries with long witnesses only occur in four models. In these four models, $L_Z^\#$ uses fewer symbols during output queries with long witnesses compared to $L^\#$.

The complete benchmark results with more detailed information can be found in Appendix B. The results are displayed in tables and the smallest number (best result) per column and the uses of the new rules are highlighted. In these tables, we can see that rules (R5) and (R6) are used a few times in the larger models.

4.2 Experiment 2

In Experiment 1, we showed that $L_Z^\#$ is competitive with $L^\#$. However, queries with long witnesses rarely occurred in the experiment. Therefore, we could argue that Experiment 1 does not show the full benefit of using $L_Z^\#$ over $L^\#$. To test the benefits accurately, we perform another experiment where long witnesses occur more often.

Implementing equivalence queries. Similar to the previous experiment, the equivalence queries make use of conformance testing with the Wp-method. The number of extra states to check for is



(a) Symbols used during learning phase

(b) Symbols used both learning and testing

(c) Symbols used learning with long witnesses

(d) Symbols used learning, no ProcCounterEx

Fig. 9: *Performance plots of the $L^{\#}$ and $L^{\#}_Z$ on benchmark models from [16] with the Wp-method with long counterexamples, lower is better*

set to 10, and the expected number of infix symbols is also set to 10. This increases the size of the counterexample and the chance of multiple states being hidden in the counterexample. Additionally, we add 1000 self-loop symbols, computed in the hypothesis, between every two input symbols in the found counterexample. This blows up the length of the counterexamples, and therefore, the length of the witnesses. If the blown-up counterexample is not a valid counterexample, then the teacher returns the original counterexample. This method frequently leads to witnesses with several thousands of input symbols.

Data-set, metrics and experiment set-up. We use the same data-set and metrics as in the previous counterexample. The experiments are repeated 20 times instead of 50 because the run time for models increases when the counterexample length increases.

Results and discussion. Figure 9a shows the total number of input and reset symbols sent by the learning algorithms via output queries, this accounts for both the size and the number of output queries. Figure 9b shows the total size of data sent during learning and testing. This includes the input symbols sent to the SUT by the teacher to find counterexamples. Figure 9c shows the number of input symbols in output queries with long witnesses originating from the rules (R6), ($R3_{long}$) and ($R3_{orig}$). Figure 9d shows the total number of input symbols used during learning without input symbols used in output queries originating from the sub-routine ProcCounterEx. In all plots, the y-axis is log-scaled. The models, sorted in increasing number of states, are displayed on the x-axis. The bars indicate the standard deviation. Additionally, Figures 14a, 14b and 14c in Appendix C.2 show the total symbols used in queries originating from rule (R5), rule (R6) or

sub-routine ComputeSepSeq.

From Figure 9a, we can observe that the total number of symbols used by $L_Z^\#$ and $L^\#$ during learning is similar for the small models. However, for the larger models, $L_Z^\#$ often performs better than $L^\#$. This result is also visible in Figure 9b. The error bars are small, indicating stable results. However, $L_Z^\#$ only uses 3.8% fewer learning symbols and 0.9% fewer learning and testing symbols.

The low increase in performance can be explained by the fact that some models never use long witnesses, as can be seen in 9c. This happens because, for example, all pairs of states have a witness of length 1 which indicates that after exploring the frontier we always have a witness of length 1. Because both algorithms explore the frontier completely before identifying frontier states, long witnesses are never used. The benchmarks contain 23 models where no long witnesses will be used, the list of these models can be found in Table 9 in Appendix C.3.

If we only take into account the 14 models where both $L_Z^\#$ and $L^\#$ have used at least one long witness, we find that $L_Z^\#$ uses 20.5% less symbols during learning and 11.3% less symbols during learning and testing. Additionally, $L_Z^\#$ uses 55.3% less symbols compared to $L^\#$ when we only look at the symbols used during queries with long witnesses. By comparing Figures 9a and 9d, we can derive that most symbols used during learning come from the subroutine ProcCounterEx, this is expected with the described counterexample generation method. Without the symbols from ProcCounterEx, the difference between $L_Z^\#$ and $L^\#$ is better visible (see Figure 9d). Together, this shows that $L_Z^\#$ outperforms $L^\#$ in scenarios where long witnesses frequently occur.

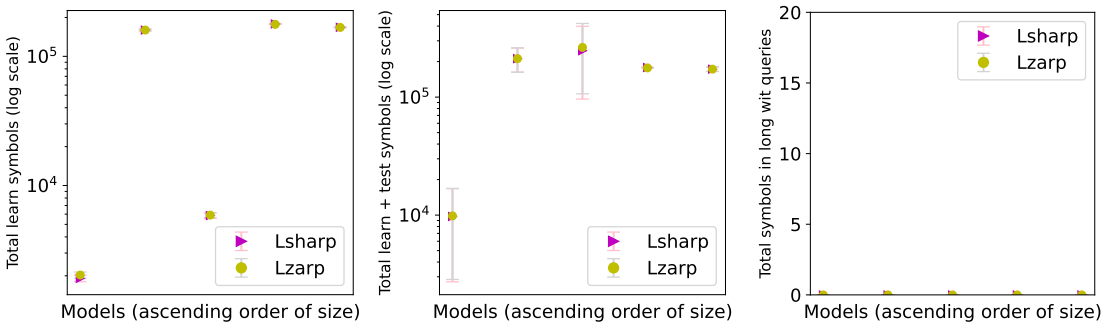
Additionally, the figures in Appendix C.2 show that rule (R5), rule (R6) and ComputeSepSeq are used several times in the larger models. It is hard to say which new rule has the most influence on the number of symbols used. More experiments should be performed to determine the benefit of each rule.

The complete benchmark results with more detailed information can be found in Appendix C.1. The results are displayed in tables and the smallest number (best result) per column and the uses of the new rules are highlighted.

4.3 Experiment 3

In Experiment 1 and Experiment 2, we showed the difference in performance between $L_Z^\#$ and $L^\#$. However, one could argue that these experiments are not realistic. In the paper by Yang et al. [22], a combination of active and passive learning is used to learn several components in the ASML TWINSCAN lithography machines. This passive approach tries to find counterexamples in execution logs instead of posing queries to the teacher. Yang et al. found that the combination reduces the learning time and learns behaviors that are missed when a strictly active learning approach is used. One could argue that this combination of active and passive learning applied to components of real-world systems represents a more realistic scenario than the scenario of the previous two experiments. To test whether $L_Z^\#$ outperforms $L^\#$ in this more realistic scenario, we perform a final experiment where we learn a small subset of the ASML TWINSCAN lithography machines components using execution logs to generate counterexamples when possible.

Implementing equivalence queries. A new equivalence query is designed that returns counterexamples found in execution logs if such a counterexample exists. If no such counterexample exists, we use the Wp-method to generate a counterexample. When the Wp-method is used, we check for 3 extra states and the number of infix symbols is set to 2. Here we use a lower number of extra states because the models in this experiment have a sink state which would often be reached with an expected random sequence of length 7.



(a) *Symbols used during learning phase* ***(b)*** *Symbols used during both learning and testing* ***(c)*** *Symbols used during long witness queries*

Fig. 10: *Performance plots of the $L^{\#}$ and $L_Z^{\#}$ on ASML benchmark models with the Wp-method with logs, lower is better*

Data-set and metrics. We use five models from the ASML TWINSCAN lithography machines components. These models have been made for the RERS challenge 2019 in collaboration with the Dutch company ASML [11]. The models are available online[4] and have been selected for this experiment based on running time. The largest model has 43 states and 102 input symbols. Moreover, each model has a corresponding execution log. We record the number of output queries and input symbols used during learning and testing. Additionally, we record the number of input symbols used in equivalence queries. Counterexamples found by analyzing the execution log do not lead to an increase in test inputs. Finally, we record the number of symbols used during queries originating from rules/sub-routines (R3_{short}), (R5), (R6), (R3_{long}), (R3_{orig}), ComputeSepSeq and ProcCounterEx.

Experiment set-up. The same experiment set-up is used as in Experiment 1.

Results and discussion. Figure 10c shows the total number of input and reset symbols sent by the learning algorithms via output queries. This accounts for both the size and the number of output queries. Figure 10b shows the total size of data sent during learning and testing. This includes the input symbols sent to the SUT by the teacher to find counterexamples. Figure 8c shows the number of input symbols in output queries with long witnesses originating from the rules (R6), (R3_{long}) and (R3_{orig}). The y-axis is log-scaled in plots 10a and 10b. The models, sorted in increasing number of states, are displayed on the x-axis. The bars indicate the standard deviation.

From Figures 10a and 10b, we can observe that the total number of symbols used by $L_Z^{\#}$ and $L^{\#}$ during learning is often close together. The $L_Z^{\#}$ algorithm uses 1.2% more symbols during learning and 1.4% more symbols during learning and testing. This result is not significant, therefore, we conclude that $L_Z^{\#}$ and $L^{\#}$ are comparable for this set of models. Moreover, from Figure 10c, we can observe that no long witnesses are used. This result is unexpected because execution logs often contain long sequences with a lot of information. It is possible that long witnesses do occur in runs with other realistic sets of models, but not with this specific set of models.

The complete benchmark results with more detailed information can be found in Appendix D. The results are displayed in tables and the smallest number (best result) per column and the uses of the new rules are highlighted. In these tables, we can see that rules (R5) and (R6) are used a few times in the larger models.

[4] `https://automata.cs.ru.nl/BenchmarkASMLRERS2019/Description`

5 Lower Bound Proofs

In this section, we discuss several lower bounds related to active automata learning. First, we prove a general lower bound for Mealy machine learning algorithms with an acyclic construction (except for the sink state). Then, we show a method that can be used by the teacher to make arbitrarily long counterexamples. Next, we combine the original construction with the method for generating long counterexamples to prove lower bounds specific to strategic $L^{\#}$.

5.1 A Lower Bound Query Complexity for Mealy Machine Learning Algorithms

The work from Balcázar et al. [3] shows that there exists a lower bound query complexity for all DFA learning algorithms. This proof uses the notation $\#equiv(A, n, m, k)$ to indicate the number of equivalence queries posed by some algorithm A in the worst case over any teacher and for any DFA with the specified parameters. The parameters n, m and k indicate the number of states, the length of the longest counterexample and the number of input symbols. Moreover, the definition of $\#memb(A, n, m, k)$ is analogous to that of $\#equiv(A, n, m, k)$. If parameters are clear from the context or irrelevant, then they are omitted from the notation. They prove the following statement for DFAs where $k \geq 2$ holds and o indicates an upper bound that cannot be tight

$$\#equiv(A, n, k) \in o(kn) \quad \text{implies} \quad \#memb(A, n, k) \in \Omega(kn^2)$$

However, many of the newer learning algorithms are designed to work on Mealy machines [10, 18, 21]. In this subsection, we adapt the construction proposed by Balcázar et al. [3] to work for Mealy machines. First, we introduce the notation $\#outp(A, n, m, k)$ to indicate the Mealy machine variant of $\#memb(A, n, m, k)$ where we count output queries rather than membership queries. Note that there is some repetition from the original construction such that this proof can be understood without reading the original paper by Balcázar et al. [3].

Theorem 5.1. *There is a constant $c_0 > 0$ such that for every learning algorithm and every $k \geq 2$,*

$$\#outp(A, n, k) \geq c_0^2 \cdot (k-1) \cdot n^2 - c_0 \cdot n \cdot \#equiv(A, n, k)$$

Proof. Similar to the original construction, we define $l = (n+1)/4$ for a given n. For simplicity, we assume that l is a power of 2. Next, we define a family of Mealy machines $\mathcal{M}_{n,k,f}$ with $n, k \in \mathbb{N}, f : Q \times I \rightharpoonup Q$. Generally machine $\mathcal{M} = (Q, I, O, q_\varepsilon, \delta, \lambda)$ in this family, the following holds.

$$|Q| = n, \qquad |I| = k, \qquad O = \{0, 1\}$$

We require $k \geq 2$ such that we can fix $a, b \in I$. Let $Q = \{q_w, p_w \mid w \in \{a, b\}^*, |w| \leq log\ l\} \cup \{q_{sink}\}$. We construct each Mealy machine in this family such that the output function returns 1 as the last output symbol for all words of the form $wawa$ with $w \in \{a, b\}^*$ and $|w| = log\ l$. Each Mealy machine constructed in this way consists of a selector tree with a branch for every w (the q-states), a transition with input symbol a, a checking tree (the p-states), and a transition with symbol a that leads to output 1. To connect the selector and checking tree, we define $\delta(q_w, a) = p_w$ for each w of length $log\ l$. Moreover, for each q_w with $|w| = log\ l$ and each letter $c \in I - \{a\}$, if $f(q_w, c) = p_v$, then we define $\delta(q_w, c) = p_v$. This indicates that the output function returns 1 as the last output symbol for the word $wcva$. There are $(k-1) \cdot l^2$ such potential transitions, of which up to $(k-1)l$ can actually exist in a Mealy machine if the output for each transition is equal, which is true in this construction. For all other words, the last output symbol is 0. From this output function, it can be derived that there is exactly one state with a transition $a/1$, state p_ε. Moreover, all words u that do not form a prefix of a word $wawa$ or $wcva$ with $\lambda(q_\varepsilon, wawa) = 0^{2|w|+1}1$ or $\lambda(q_\varepsilon, wcva) = 0^{2|w|+1}1$, lead to the sink state, q_{sink}. We can now define δ and λ for each Mealy machine $\mathcal{M} = (Q, I, O, q_\varepsilon, \delta, \lambda)$ in the family $\mathcal{M}_{n,k,f}$.

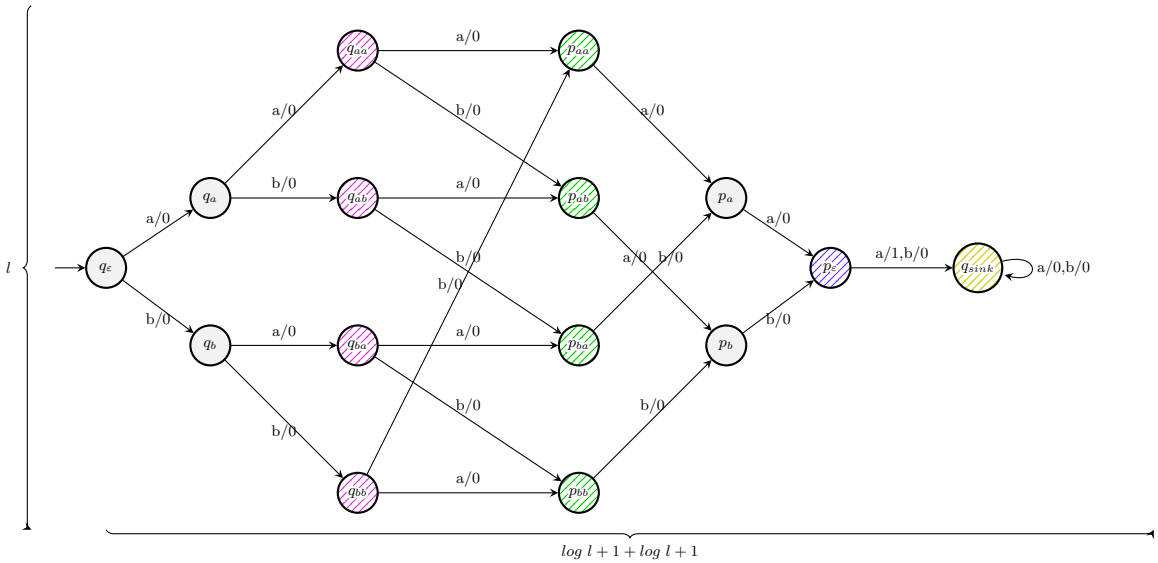


Fig. 11: *Example of a possible construction for the Mealy machine query lower bound with $k = 2$, $n = 15$. Let $|w| = logl$, then Pink states indicate q_w states, green states indicate p_w states, blue indicates p_ε and yellow indicates q_{sink}. All transitions that are not drawn lead to the q_{sink} with output 0.*

$\delta(q_w, a) = q_{wa}$	if $w \in \{a,b\}^*$ with $\|w\| < \log l$
$\delta(q_w, b) = q_{wb}$	if $w \in \{a,b\}^*$ with $\|w\| < \log l$
$\delta(p_{aw}, a) = p_w$	if $w \in \{a,b\}^*$ with $\|w\| < \log l$
$\delta(p_{bw}, b) = p_w$	if $w \in \{a,b\}^*$ with $\|w\| < \log l$
$\delta(q_w, a) = p_w$	if $w \in \{a,b\}^*$ with $\|w\| = \log l$
$\delta(q_w, c) = p_v$	if $f(q_w, c) = p_v$ and $c \in I - \{a\}, w \in \{a,b\}^*$ with $\|w\| = \log l$
$\delta(p_\varepsilon, a) = q_{sink}$	-
$\delta(q, j) = q_{sink}$	for $q \in Q, j \in I$ if not specified otherwise
$\lambda(p_\varepsilon, a) = 1$	-
$\lambda(q, j) = 0$	for all $q \in Q, j \in I$ if $\neg(q = q_\varepsilon \wedge j = a)$

From this, we can derive the following output transitions.

$\lambda(q_\varepsilon, wawa) = 0^{2\|w\|+1}1$	if $w \in \{a,b\}^*, \|w\| = log\ l$
$\lambda(q_\varepsilon, wcva) = 0^{2\|w\|+1}1$	if $w, v \in \{a,b\}^*, c \in I - \{a\}, f(q_w, c) = p_v, \|w\| = \|v\| = log\ l$
$\lambda(q_\varepsilon, u) = 0^{\|u\|}$	otherwise

Each Mealy machine in the family $\mathcal{M}_{n,k,f}$ has exactly $(2l-1) + (2l-1) + 1 = 4l - 1 = n$ states because all states q_w and p_w must be distinct. An example of the Mealy machine $\mathcal{M}_{15,2,f}$ can be found in Figure 11 where f is defined as follows

$$f(q_{aa}, b) = p_{ab}, \qquad f(q_{ab}, b) = p_{ba}, \qquad f(q_{ba}, b) = p_{bb}, \qquad f(q_{bb}, b) = p_{aa}$$

The learning task for the Mealy machine is the same as for the DFA. Informally, A must determine, for every q_w and c, the p_v such that $\delta(q_w, c) = p_v$, if any. We show that even if the learner knows the structure of the target Mealy machine, this task requires many output and equivalence queries.

Consider an adversarial teacher that answers A's queries as follows. It maintains a set S of strings that have been returned as counterexamples. Initially, $S = \emptyset$ and, at every moment, the answers given by the adversary are consistent with the target Mealy machine $\mathcal{M}$.

1. Any output query x: return the output returned by $\lambda^{\mathcal{M}}(q_\varepsilon, x)$.
2. An equivalence query with $\mathcal{H}$ where $\lambda^{\mathcal{H}}(q_\varepsilon, wcva) = 0^{|wcva|}$ while $\lambda^{\mathcal{M}}(q_\varepsilon, wcva) = 0^{|wcv|}1$ for some word $wcva \in S$ or $wcva$ has the form $wawa$: return *(no, wcva)*.
3. An equivalence query with $\mathcal{H}$ where $\lambda^{\mathcal{H}}(q_\varepsilon, wcva) = 0^{|wcv|}1$ while $\lambda^{\mathcal{M}}(q_\varepsilon, wcva) = 0^{|wcva|}$ for some word $wcva \notin S$: return *(no, wcva)*.
4. An equivalence query accepting exactly all words of the form $wawa$ and words in S; only here the adversary is forced to reveal a new transition: select some $wcva$ with $\lambda^{\mathcal{M}}(q_\varepsilon, wcva) = 0^{|wcv|}1$ such that no string $wcv'a$ is in S and $wcva$ never appeared before in the dialog; add $wcva$ to S and return *(no,wcva)* as a counterexample. (If no such $wcva$ exists, the adversary admits that A has succeeded and returns *yes*).

The same argument for the lower bound complexity is made as in the original paper. In case 2, we do not learn any new transition because if the word is of the form $wcva$, then $wcva \in S$ so we already had that information. In case 3, we learn that one possible transition $\delta(q_w, c) = p_v$ is not present in the target automaton. In case 4, l transitions are revealed, namely, $\delta(q_w, b) = p_v$ so $\delta(q_w, b) \neq p'_v$, for all $v' \neq v$.

If the learner is only allowed to ask k equivalence queries, it can know only $k \cdot l$ transitions. However, to learn the Mealy machine, the presence or absence of all possible $wcva$ transitions needs to be known. There are $(k-1)l^2$ possible $wcva$ transitions. Therefore, if the learner asks at most k equivalence queries, then there must be at least $(k-1)l^2 - kl$ output queries. Rewriting with $l = \frac{n+1}{4}$, gives

$$\#outp(A, n, k) \geq (k-1) \cdot \left(\frac{n+1}{4}\right)^2 - \frac{n+1}{4} \cdot \#equiv(A, n, k)$$

thus, the theorem holds with c_0 about $1/4$. □

5.2 Generating Long Counterexamples

The above-mentioned construction generates acyclic Mealy machines except for the sink state. In these Mealy machines, the adversary teacher can only return counterexamples of which the length of the relevant part is bounded by the number of states in the Mealy machine. In practice, Mealy machines often contain loops that allow for counterexamples of unbounded length. Current algorithms use $\mathcal{O}(n \log m)$ output queries to process counterexamples. It is an open problem to determine whether the parameter m is necessary or not [3]. The original construction by Balcázar et al. does not consider the queries needed to process counterexamples. In this section, we show a simple construction that allows for the generation of counterexamples of unbounded length. Specifically, we show that at least i output queries of length $m/2$ are needed, where i is some natural number that is fixed during the learning process and can be increased to make longer counterexamples.

Suppose we have the target Mealy machine displayed in Figure 12. The initial hypothesis constructed by $L^\#$ has the following transition and output function.

$$\delta(q_0, a) = q_0, \quad \delta(q_0, x) = q_0, \qquad \lambda(q_0, a) = 0, \quad \lambda(q_0, x) = 0$$

When the equivalence query with this hypothesis is asked of the teacher, the teacher can return the unique minimal counterexample $aaaaaa$. However, the teacher can also return the following counterexample where n indicates the number of states, which is 7 in this example.

$$aaxaxaxax^{n \cdot 2^i}a$$

For an explanation about the strategic $L^\#$ mechanics and notation, we refer the reader back to the strategic $L^\#$ algorithm explanation in section 2.2. Counterexample processing splits the

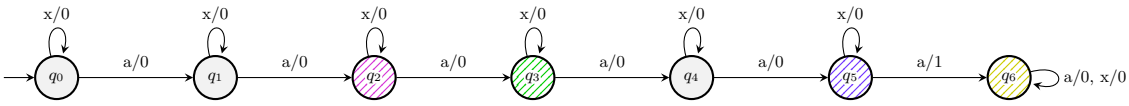


Fig. 12: *Example of a Mealy machine that allows the generation of very long counterexamples. Some states are colored to indicate that they are related to certain states in the constructions of the lower bound proofs.*

counterexample σ into σ_1 and σ_2. Then we ask the teacher OUTPUTQUERY($\mathsf{access}(q') \cdot \sigma_2 \cdot \eta$). If the output query does not lead to the same output sequence as a subsequence of the counterexample, then counterexample processing is called again with the σ_1. For this specific example, if $|\sigma_1| \geq 7$, then at most one input symbol a from before the $x^{7 \cdot 2^i}$ part is in σ_2. These uninformative output queries always have the form $ax...xa$ or $x...xa$. There is no access part because q_0 is the only state and has no access sequence. We now show that $|\sigma_1| \geq n$ occurs at least i times, thus these uninformative output queries must occur at least i times. Initially,

$$|\sigma| = |aa| + 2|aaa| + n \cdot 2^i$$

The following statement shows that at least i output queries are needed for each counterexample.

$$\frac{|aa| + 2|aaa| + n \cdot 2^i}{2^i} = \frac{|aa| + 2|aaa|}{2^i} + \frac{n \cdot 2^i}{2^i} = \frac{|aa| + 2|aaa|}{2^i} + n \geq n$$

Additionally, we know that at least $i \cdot m/2$ input symbols are needed to process this counterexample. The first query has a length of at least $(m-1)/2$ because each frontier state can be accessed by exactly one input symbol. The next $i-1$ queries use the output query in the first query as a witness. Therefore, each of the i queries uses at least $(m-1)/2$ input symbols. However, if $i \geq 2$, the next query has at least

$$m - \lfloor \frac{\frac{m+1}{2} + 1}{2} \rfloor \geq \frac{3m+1}{4}$$

queries. This indicates that the first two queries have $\frac{m-1}{2} + \frac{3m+1}{4} = \frac{5m-1}{4} \geq \frac{2m}{2}$ queries together. All output queries after this use the witness of length $\frac{3m+1}{4}$. Therefore, if $i \geq 2$, strategic $L^\#$ needs at least $i \cdot m/2$ input symbols to process the counterexample.

5.3 A Lower Bound Query Complexity for $L^\#$

In this section, we combine the construction for Mealy machines with the method for generating long counterexamples to prove the following theorem.

Theorem 5.2. *There is a constant $c_0 > 0$ such that for every $k \geq 6$,*

$$\#outp(strategic\ L^\#, n, k, m) \geq c_0 \cdot (k-2) \cdot n^2 + log(\frac{m}{n} - 1) \cdot \#equiv(strategic\ L^\#, n, k, m)$$

Proof. Let $l = (n+1)/4$ for a given n. Moreover, we define some natural number $i \geq 2$ that remains the same during the whole learning process. Similar to the general lower bound proof for Mealy machines, we define a family of Mealy machine $\mathcal{M}_{n,k}$, note that we do not require the parameter f for this construction. For each Mealy machine $\mathcal{M} = (Q, I, O, q_\varepsilon, \delta, \lambda)$ in this family, the following holds.

$$|Q| = n, \qquad |I| = k, \qquad O = \{0, 1\}$$

We require $k \geq 6$ and $k\ mod\ 2 = 0$ such that we can fix two subsets I_{loop} and I_{min} with

$$I_{loop} \cup I_{min} = I, \quad I_{loop} \cap I_{min} = \emptyset, \quad |I_{loop}| = |I_{min}|$$

We fix letters $a, b, c \in I_{min}$. Let $Q = \{q_w, p_w \mid w \in \{b,c\}^*, |w| \leq log\ l\} \cup \{q_{sink}\}$. We construct each Mealy machine in this family such that the output function returns 1 as the last output symbol for all words of the form $wawa$ with $w \in \{b,c\}^*$ and $|w| = log\ l$. Each Mealy machine constructed in this way consists of a selector tree with a branch for every w (the q-states), a transition with input symbol a, a checking tree (the p-states), and a transition with symbol a that leads to output 1. To connect the selector and checking tree, we define $\delta(q_w, a) = p_w$ for each w of length $log\ l$. Moreover, we add a self-loop with all input symbols from I_{loop} to each state except the initial state (for fast access to the sink state). This construction does not consider transitions $\delta(q_w, d) = p_v$ with $w, v \in \{b,c\}^*, d \in I_{min} - \{a\}$ with $|w| = |v| = log\ l$. Let $g_{min} : I^* \to I^*_{min}$ indicate a function that removes all $x \in I_{loop}$ from the input word. For all words u, if $g_{min}(u)$ is not a prefix of a word $wawa$, then the last output symbol is 0. From this output function, it follows that there is exactly one state with a transition $a/1$, state p_ε. We can now formally define δ and λ for each Mealy machine $\mathcal{M} = (Q, I, O, q_\varepsilon, \delta, \lambda)$ in the family $\mathcal{M}_{n,k}$ as follows.

$\delta(q_w, b) = q_{wb}$	if $w \in \{b,c\}^*$ with $\|w\| < \log l$
$\delta(q_w, c) = q_{wc}$	if $w \in \{b,c\}^*$ with $\|w\| < \log l$
$\delta(p_{bw}, b) = p_w$	if $w \in \{b,c\}^*$ with $\|w\| < \log l$
$\delta(p_{cw}, c) = p_w$	if $w \in \{b,c\}^*$ with $\|w\| < \log l$
$\delta(q_w, a) = p_w$	if $w \in \{b,c\}^*$ with $\|w\| = \log l$
$\delta(p_\varepsilon, a) = q_{sink}$	-
$\delta(q, j) = q$	for all $j \in I_{loop}$ and $q \neq q_\varepsilon$
$\delta(q, j) = q_{sink}$	for $q \in Q, j \in I$ if not specified otherwise
$\lambda(p_\varepsilon, a) = 1$	-
$\lambda(q, j) = 0$	for all $q \in Q, j \in I$ if $\neg(q = q_\varepsilon \wedge j = a)$

From this, we can derive the following output transitions.

$\lambda(q_\varepsilon, u) = 0^{\|u\|-1}1$	if $g_{min}(u) = wawa$ with $\|w\| = log\ l, w \in \{b,c\}^*$
$\lambda(q_\varepsilon, u) = 0^{\|u\|}$	otherwise

Each Mealy machine in the family $M_{n,k}$ has exactly $(2l-1) + (2l-1) + 1 = 4l - 1 = n$ states because all states q_w and p_w must be distinct. The new construction changes the task of the learner. Previously, we showed that even if the learner knows the general structure of the target Mealy machine, many output queries are needed to figure out the transitions between q_w and p_v. We exploit the fact that $L^\#$ has no knowledge about the structure of the target Mealy machine. Because the sink state does not have a witness that immediately indicates that we are in the sink state, all other states need to be ruled out when identifying a frontier state that maps to the sink state. A schematic depiction of the construction with $k = 6$ and $n = 15$ can be found in Figure 13.

Let $f_i : I^+ \to I^+$ be a conversion function that takes a minimal counterexample and injects many redundant letters in I_{loop}. This function is a generalized version of the method of generating long counterexamples described in Section 5.2. Let $a_{loop} \in I_{loop}$ and let the minimal counterexample be of the form $wawa$, then we can specify the function $f_i : I^+ \to I^+$ as follows:

1. Initialize a new string with only the letters in $w \cdot a_{loop}$
2. Inject a_{loop} between every two letters in aw.
3. Construct a string of length $n \cdot 2^i$ containing only letters in I_{loop}. The last $n \cdot 2^i - n \cdot 2^{i-1}$ must form a unique string for each counterexample returned by the teacher.
4. Append a.

Because i is fixed during the learning process, all counterexamples in one learning process have the same length. To create a unique string for l counterexamples with $k/2$ letters, we only need space for $log\ l$ letters. We have space for $n \cdot 2^i - n \cdot 2^{i-1} = n \cdot 2^{i-1}$ letters. Because $i \geq 2$, we always have space for at least $2n$ letters and $2n \geq log\ l$. Therefore, we can always create a unique string.

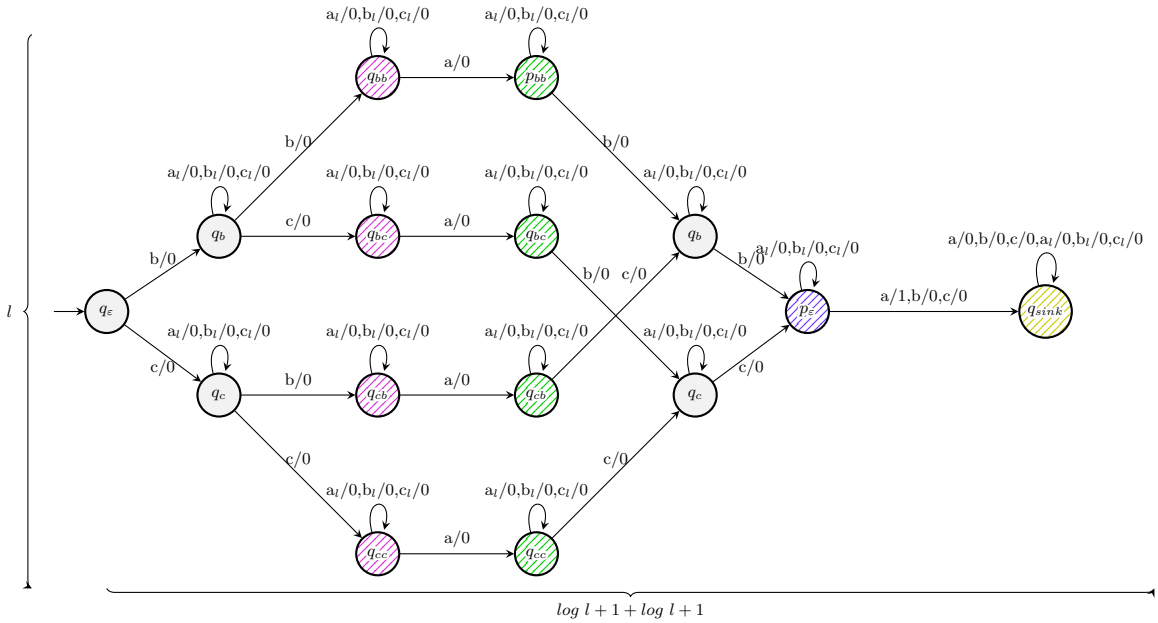


Fig. 13: *Example of a possible construction for the Mealy machine query lower bound with $k = 6$, $n = 15$. Let $|w| = logl$, then Pink states indicate q_w states, green states indicate p_w states, blue indicates p_ε and yellow indicates q_{sink}. All transitions that are not drawn lead to the q_{sink} with output 0.*

One simple method to create this unique string is to pair each input symbol in I_{min} with an input symbol in I_{loop}. Let $h : I^*_{min} \to I^*_{loop}$ be the function that transforms each symbol in the input word into the paired I_{loop} input symbol. Because w is unique for each required counterexample, $h(w)$ is also unique. However, it is possible that $|h(w)| \leq n \cdot 2^i - n \cdot 2^{i-1}$. To fix this, we append the input a_{loop} behind $h(w)$ until the required length is reached.

An adversarial teacher can answer $L^\#$'s queries as follows:

1. Any output query x: return the output of $\lambda^{\mathcal{M}}(q_\varepsilon, x)$.
2. An equivalence query with $\mathcal{H}$ where $\lambda^{\mathcal{H}}(q_\varepsilon, wawa) = 0^{|wawa|}$ with $|w| = log\ l$ and $w \in \{b, c\}^*$. The teacher returns *(no,wawa)*.
3. An equivalence query with $\mathcal{H}$ where $\mathcal{H} \approx \mathcal{M}$. The teacher returns *yes*

With these answers, strategic $L^\#$ asks exactly l equivalence queries, one for every word of the form $wawa$. This holds because all states are reached with counterexamples originating of the form $wawa$ except the q_{sink}. The sink state can be found by posing output queries with all witnesses to any frontier state that maps to the q_{sink} in the hypothesis. None of the witnesses lead to an output sequence that contains the output symbol 1 which indicates that q_{sink} is apart from all other states.

To determine the lower bound query complexity, we first consider the output queries needed to process each counterexample. Contrary to the output queries discussed in Section 5.2, the output queries for this construction have the following form:

$$\mathsf{access}(q') \cdot \text{self-loops} \cdot a$$

However, it still holds that the output queries always have a different output than the counterexample, thus a recursive call is needed. These uninformative output queries must occur at least i times. Initially,

$$|\sigma| = |w| + 2|aw| + n \cdot 2^i$$

Moreover, $|\mathsf{access}(q')| < n$, but to prove that the number of output queries is at least i we take $|\mathsf{access}(q')| = 0$. When $|\mathsf{access}(q')| = 0$, σ_2 contains informative letters faster. The following statement shows that at least i output queries are needed for each counterexample:

$$\frac{|w|+2|aw|+n\cdot 2^i}{2^i} = \frac{|w|+2|aw|}{2^i} + \frac{n\cdot 2^i}{2^i} = \frac{|w|+2|aw|}{2^i} + n \geq n$$

Each counterexample returned by the teacher has at most length $n + n \cdot 2^i$. We can rewrite the statement $m \leq n + n \cdot 2^i$ to get the result $i > log(\frac{m}{n} - 1)$. The following number of output queries are needed to process one counterexamples:

$$log(\frac{m}{n} - 1)$$

Let q'_w and p'_w indicate q_w and p_w states with $w \in \{b, c\}^*, |w| = log\ l$. The counterexample processing output queries contain no information about the transitions from $\delta(q'_w, d) = q_{sink}$ where $d \in I_{min} - \{a\}$ because all output queries during counterexample processing have the form $\mathsf{access}(q')\cdot$self-loops$\cdot a$. The sequence $\mathsf{access}(q')$ can reach at most the state q'_w. Because no symbols d can be contained in the second part of the counterexample, we never learn the destinations of the transitions $\delta(q'_w, d)$. We only receive observations of the form $f_i(wawa)$ from the teacher. This also reveals no information about the transitions $\delta(q'_w, d) = q_{sink}$. Therefore, we know that there are no paths with observations that start in the frontier state $\delta(q'_w, d)$ directly after finding a new state on the path $wawa$ which also passes q'_w. Because $\delta(q'_w, d) = q_{sink}$, we have to eliminate at least all p'_w states. Each p'_w has a unique witness and there are l such states, so at least $l = (n+1)/4 \geq n/4$ output queries are needed per state found after an equivalence query. The following number of output queries are needed to identify all $\delta(q_w, d)$:

$$\frac{k-2}{2} \cdot \frac{n}{4} \cdot \frac{n}{4} = \frac{1}{32} \cdot (k-2) \cdot n^2$$

Because $L^\#$ never uses an equivalence query before all frontier states are identified, we know that the output queries to identify $\delta(q'_w, d)$ and to process the counterexamples do not overlap. Therefore, at least

$$\frac{1}{32} \cdot (k-2) \cdot n^2 + log(\frac{m}{n} - 1) \cdot \#equiv(strategic\ L^\#, n, k, m)$$

output queries are needed to learn the target automaton. The statement of the theorem follows with $c_0 = 1/32$. □

5.4 A Lower Bound Symbol Complexity for $L^\#$

The result found above provides new information about the $L^\#$ algorithm. However, as noted by Vaandrager et al. [21], the total time needed to learn a Mealy machine is proportional to the number of input symbols asked by the learner. Therefore, we would also like to prove a lower bound for the symbol complexity. We define $\#inp\_symb(A, n, k, m)$ to indicate the number of input symbols asked by some algorithm A in the worst case over any teacher.

Theorem 5.3. *There is a constant $c_0 > 0$ such that for every $k \geq 6$,*

$$\#inp\_symb(strategic\ L^\#, n, k, m) \geq c_0 \cdot \Big(\frac{1}{32} \cdot (k-2) \cdot (m-1) \cdot n^2 + m \cdot log(\frac{m}{n} - 1) \cdot \#equiv(strategic\ L^\#, n, k, m)\Big)$$

Proof. We use the same construction as above. In the counterexample processing output queries, the first query has a length of at least $(m-2(log\ l+1))/2$ because the $|\rho|$ can be at most $2(log\ l+1)$. This can be derived from the following statement:

$$m - \lfloor \frac{2(log\ l+1)+m}{2} \rfloor \geq \frac{2m - 2(log\ l+1) - m}{2} = \frac{m - 2(log\ l+1)}{2}$$

The next $i-1$ queries use the output query in the first query as a witness. Therefore, each of the i queries uses at least $(m-2(log\ l+1))/2$ input symbols. However, if $i \geq 2$, the next query has at least:

$$\begin{aligned} m - \lfloor \frac{\frac{m+2(log\ l+1)}{2} + 2(log\ l+1)}{2} \rfloor &\geq \frac{3m - 6(log\ l+1)}{4} \\ &\geq \frac{2m + 4n + 2(log\ l+1) - 6(log\ l+1)}{4} \\ &\geq \frac{m + 2(log\ l+1)}{2} \end{aligned}$$

In the proof above, we use the fact that $i \geq 2$ and therefore $m \geq 4n + 2(log\ l+1)$. The additional $2(log\ l+1)$ in the second query compensates for the $2(log\ l+1)$ missing in the first query and all other $i-2$ queries use at least $\frac{m+2(log\ l+1)}{2}$ input symbols. Therefore, if $i \geq 2$, the number of input symbols needed for each counterexample is at least $i \cdot m/2$.

For the identification output queries, we need to dive deeper into the strategic $L^{\#}$ mechanics. Specifically, we look at what happens directly after strategic $L^{\#}$ discovers a state q_w. Let q'_w and p'_w indicate q_w and p_w states with $w \in \{b,c\}^*, |w| = log\ l$. Whenever a state q'_w is found, we know that p'_w has not been found. To find state p'_w, we need a witness u such that $g_{min}(u) = wa$. However, if we have found state q'_w then that must have happened with a counterexample u' with $g_{min}(u') = awa$. Specifically, counterexample u' contains a_{loop} between the first a and w in ce. Therefore, state p'_w is not present in the observation tree at any point in counterexample processing. Additionally, p'_w cannot be identified as a new state using rule (R3). This is because there exists no witness u'' with $g_{min}(u'') = wa$ and only witness of that form executed from p'_w lead to the output symbol 1. This also implies that the witnesses for states q'_w have at least length $n \cdot 2^i$. Moreover, directly after finding q'_w, no paths with subsequence $f_i(dva)$ with $d \in I_{min} - \{a\}$ and $v \in \{b,c\}^*$ exist that start in q_w because the counterexample can only have subsequence $f_i(awa)$. Any path starting in q'_w with first symbol d has a length that is different from length $f_i(dva)$. This indicates that all candidate states p'_v still have to be ruled out from the candidate set of $\delta(q'_w, d)$.

Because no new states can be added to the basis after adding q'_w and before rule (R4), strategic $L^{\#}$ is forced to apply rules (R2) and (R3) until the basis is complete. Specifically, rule (R3) forces strategic $L^{\#}$ to use the witness $f_i(awa)$ on at least $\frac{(k-2)}{2}$ frontier states to find out whether $wd \cdot f_i(wa)$ results in output 1 for each $d \in I_{min} - \{a\}$. The output query required for this is composed of the access sequence for q'_w concatenated with $f_i(dwa)$, this has length $m-1$.

Moreover, for each previously seen word $uaua$ with $u \in \{b,c\}^*$ and $|u| = log\ l$, we have a q'_w as a new candidate state for $\delta(q_\varepsilon, vd)$. To rule out q'_w, we need to ask the output query vd concatenated with $f_i(awa)$ because $f_i(awa) \vdash q_{sink}\ \#\ q'_w$, which has length m. Note that each q'_w needs to be ruled out because according to the construction each $\delta(q_\varepsilon, vd)$ leads to the sink and there does not exist a witness that show that we are in the sink, only witnesses that show that we are not in the sink. Let e indicate the e^{th} state q'_w that is found. For each e, $(k-2)/2 \cdot e$ queries of length at least $m-1$ are needed to identify the $\delta(q'_w, d)$ frontier states. This indicates that at least $\sum_{e=1}^{l} \frac{(k-2)}{2} \cdot e = \frac{1}{4} \cdot (k-2)l(l+1)$ queries are needed of at least $m-1$ symbols to identify frontier states $\delta(q'_w, d)$ and to rule out q_w as candidate for all previously found states $\delta(q'_v, d)$.

This indicates that at least $\frac{1}{4}(k-2)(m-1)l^2$, or $\frac{1}{64}(k-2)(m-1)n^2$ input symbols are needed for the identification output queries. For the counterexample processing queries, at least $\frac{1}{2}m \cdot log(\frac{m}{n} - 1)$ are needed per counterexample. By choosing $i \geq 2$ and $c_0 = \frac{1}{2}$, the theorem above holds. □

6 Conclusion and Future Work

We presented $L_Z^\#$, a variation on the $L^\#$ algorithm. The $L_Z^\#$ algorithm differs from $L^\#$ in several ways. First, we only use rules with long witnesses when no rules with short witnesses can be applied in the strategic version of $L_Z^\#$. Second, if a long witness exists after counterexample processing, this witness often contains a new state or can be shortened. By prioritizing the identification of the immediate successors of states that have long witnesses with other states, we often find a new state or a shorter witness than faster than by random exploration like in $L^\#$. Next, we exploit special cases in the observation tree with two new rules that always lead to a short witness or a new basis state. Finally, we use a technique from conformance testing, separating sequences, to ensure that all witnesses are short before we ask an equivalence query. Experimental evaluation shows that in scenarios where long witnesses frequently occur, strategic $L_Z^\#$ reduces the total number of symbols required for learning by 20.5%. Additionally, 55.3% fewer symbols are used in all queries with long witnesses. This implies that $L_Z^\#$ uses fewer output queries with long witnesses compared to $L^\#$. In models where long witnesses never occur, which is about half of the benchmark models, the performance of $L_Z^\#$ is comparable to $L^\#$.

The $L_Z^\#$ algorithm is a combination of several heuristics based on the idea that long witnesses should be used only when absolutely necessary. One could take this idea even further and design an algorithm that always applies the shortest possible witness first. Moreover, we are interested in the symbol complexity, and the number of symbols used during an output query depends on the access sequence and length of the witness. Another heuristic could be to always apply the shortest possible output query.

Moreover, the current order in which the rules are applied might not be the order that leads to the best results. For example, it is possible that applying rules (R5) and (R6) before rule (R3_{short}) leads to better results because rules (R5) and (R6) might find new basis states earlier than (R3_{short}). It would be interesting to investigate the optimal order in more detail.

Contrary to our expectations, the execution logs from the models used in Experiment 3 do not frequently lead to long witnesses after counterexample processing. This could be specific to the chosen model set. Because more than half of the models in Experiments 1 and 2 never used long witnesses after counterexample processing, it is possible that long witnesses after counterexample processing rarely occur in practice. If that is the case, then it might be more beneficial to design algorithms that focus on finding new states by exploring the frontier instead of asking equivalence queries. Therefore, it would be interesting to investigate how often long witnesses occur after counterexample processing in practice.

The only experiment that shows that $L_Z^\#$ outperforms $L^\#$ is Experiment 2. One could argue that this is also the least realistic experiment because the counterexamples returned by the teacher are ridiculously large. In practice, it is unlikely that the teacher always returns a counterexample with thousands of self-loops. However, if the teacher can only use execution logs to find counterexamples and a certain state is not reached often, it is not unreasonable that the counterexample for that state is several thousands of symbols long. Therefore, Experiment 2 tests a practically feasible situation.

A detailed comparison between $L^\#$ and other state-of-the-art active learning algorithms can be found in the paper by Vaandrager et al. [21]. This comparison uses Hybrid-ADS instead of the Wp-method. By comparing the total learning and testing symbols, we can observe that the Wp-method requires more testing queries. In future work, it would be interesting to test the new equivalence oracle with long counterexample generation on the other active learning algorithms or the Hybrid-ADS equivalence oracle on $L_Z^\#$.

Moreover, the experimental evaluation in [21] uses an already existing variation called strategic $L^{\#}_{\text{ADS}}$. This version uses adaptive distinguishing sequences which are input sequences where the next input depends on the outputs received in response to previous inputs [21]. These adaptive distinguishing sequences are appended to the sequences used in output queries in rules (R2) and (R3) of the $L^{\#}$ algorithm. These sequences can also be added to the rules new rules. It would be interesting to investigate the combined effect of the variations of the $L^{\#}$ algorithm.

The $L^{\#}$ prototype implementation has some scalability issues, the observation trees become too big when the model size grows [21]. Because $L^{\#}_{Z}$ uses the same framework as $L^{\#}$, $L^{\#}_{Z}$ has the same scalability issues. When improvements are made towards the scalability of $L^{\#}$, these improvements can easily be made to $L^{\#}_{Z}$ as well. Similarly, one goal of [21] is to apply $L^{\#}$ to richer frameworks, such as weighted automata or symbolic automata. If the learning algorithm often reaches a situation where it is forced to use a long witness after counterexample processing in these richer frameworks, then $L^{\#}_{Z}$ can be considered as an alternative approach which might lead to better results.

Moreover, in this thesis, we prove a lower bound query complexity for the general problem of learning Mealy machines. This construction is reused to prove the following lower bound query complexity for strategic $L^{\#}$.

There exists a constant $c_0 > 0$ *such that for every* $k \geq 6$,

$$\#outp(\textit{strategic } L^{\#}, n, k, m) \geq c_0 \cdot (k-2) \cdot n^2 + log(\frac{m}{n} - 1) \cdot \#equiv(\textit{strategic } L^{\#}, n, k, m)$$

Using this construction, we prove the following lower bound symbol complexity for strategic $L^{\#}$.

There exists a constant $c_0 > 0$ *such that for every* $k \geq 6$,

$$\#inp\_symb(\textit{strategic } L^{\#}, n, k, m) \geq c_0 \cdot \Big(\frac{1}{32} \cdot (k-2) \cdot (m-1) \cdot n^2 + m \cdot log(\frac{m}{n} - 1) \cdot \#equiv(\textit{strategic } L^{\#}, n, k, m) \Big)$$

In future research, it would be interesting to apply the construction described in Section 5.3 to algorithms like TTT [10] or observation pack [7]. Because these algorithms use a similar procedure to process counterexamples and use the result to extent their data structures, it is possible that the same lower bound can be proven. However, this does not immediately lead to a general lower bound proof without the parameter m. For such a proof, we need to show that the parameter m is unavoidable, requiring a different construction.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106. 1
2. Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990. 1
3. José L Balcázar, Josep Díaz, Ricard Gavalda, and Osamu Watanabe. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. Springer, 1997. 1, 1, 1, 5.1, 5.2
4. Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE transactions on software engineering*, (3):178–187, 1978. 2.3
5. Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975. 3.3
6. John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971. 2.3
7. Falk M Howar. *Active learning of interface programs*. PhD thesis, 2012. 1, 3.4, 6
8. Muhammad Naeem Irfan, Catherine Oriat, and Roland Groz. Angluin style finite state machine inference with non-optimal counterexamples. In *Proceedings of the First International Workshop on Model Inference In Testing*, pages 11–19, 2010. 1
9. Malte Isberner. *Foundations of active automata learning: an algorithmic perspective*. PhD thesis, 2015. 1
10. Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: a redundancy-free approach to active automata learning. In *International Conference on Runtime Verification*, pages 307–322. Springer, 2014. 1, 1, 3, 3.4, 5.1, 6
11. Marc Jasper, Malte Mues, Alnis Murtovi, Maximilian Schlüter, Falk Howar, Bernhard Steffen, Markus Schordan, Dennis Hendriks, Ramon Schiffelers, Harco Kuppens, et al. Rers 2019: combining synthesis with real-world models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 101–115. Springer, 2019. 4.3
12. Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994. 1
13. Fujiwara Bochmann Khendek, S Fujiwara, GV Bochmann, F Khendek, M Amalou, and A Ghedamsi. Test selection based on finite state models. *IEEE Transactions on software engineering*, 17(591-603):10–1109, 1991. 2.3, 4.1
14. Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995. 1
15. Edward F Moore et al. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956. 1, 2.3, 3.1
16. Daniel Neider, Rick Smetsers, Frits Vaandrager, and Harco Kuppens. Benchmarks for automata learning and conformance testing. In *Models, Mindsets, Meta: The What, the How, and the Why Not?*, pages 390–416. Springer, 2019. 1, 4.1, 8, 9, 9
17. Ronald L Rivest and Robert E Schapire. Inference of finite automata using homing sequences. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 411–420, 1989. 1, 1
18. Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In *International Symposium on Formal Methods*, pages 207–222. Springer, 2009. 5.1
19. Rick Smetsers, Joshua Moerman, and David N. Jansen. Minimal separating sequences for all pairs of states. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 181–193. Springer, 2016. 2.3, 4, 2.3, 2.3, 3.3, 3.3, A
20. Frits Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, 2017. 1
21. Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 223–243. Springer, 2022. 1, 1, 2, 2.1, 3.3, 3.3, 3.4, 4.1, 5.1, 5.4, 6, A, A
22. Nan Yang, Kousar Aslam, Ramon Schiffelers, Leonard Lensink, Dennis Hendriks, Loek Cleophas, and Alexander Serebrenik. Improving model inference in industry by combining active and passive learning. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 253–263. IEEE, 2019. 4.3

A Omitted Proofs

Lemma 3.4

Let $q, q' \in S$ with $i\rho \vdash q \#_l q'$ for some $i \in I, \rho \in I^+$. Moreover, let $r, r' \in S \cup F$ and $\delta^{\mathcal{T}}(q, i) = r, \delta^{\mathcal{T}}(q', i) = r'$. Moreover, let $C(r) = \{u\}$ and $C(r') = \{u'\}$ and $\sigma \vdash u \#_s u'$ for some $\sigma \in I^+$ with $|\sigma| < |S|$. The OutputQuery($\mathsf{access}(r)\ \sigma$) leads to one of the following cases:

1. $\lambda^{\mathcal{T}}(r, \sigma) \neq \lambda^{\mathcal{T}}(u, \sigma)$. In this case, $u \notin C(r)$, thus r is a new state.
2. $\lambda^{\mathcal{T}}(r, \sigma) = \lambda^{\mathcal{T}}(u, \sigma)$. In this case, we also need to execute the OutputQuery($\mathsf{access}(r')\ \sigma$) to ensure that we find a short witness or a new state. The following cases are possible:
   (a) $\lambda^{\mathcal{T}}(r', \sigma) \neq \lambda^{\mathcal{T}}(u', \sigma)$. In this case, $u' \notin C(r')$, thus r' is a new state.
   (b) $\lambda^{\mathcal{T}}(r', \sigma) = \lambda^{\mathcal{T}}(u', \sigma)$. Because $\lambda^{\mathcal{T}}(u, \sigma) \neq \lambda^{\mathcal{T}}(u', \sigma)$, it must hold that $\lambda^{\mathcal{T}}(r', \sigma) \neq \lambda^{\mathcal{T}}(r, \sigma)$. Thus, we have found the short witness $i\sigma$ for $q' \#_s q$ because $|i\sigma| = 1 + |\sigma| < 1 + |S| \leq |S|$.

All cases lead to either a short witness or a new state. □

Lemma 3.5

Let $r, r' \in F$ with $\sigma \in r \# r'$ for some $\sigma \in I^+$ and let $C(r) = \{q\} = C(r')$ with $q \in S$. The OutputQuery($\mathsf{access}(q)\ \sigma$) leads to one of the following cases:

1. $\lambda^{\mathcal{T}}(q, \sigma) \neq \lambda^{\mathcal{T}}(r, \sigma) \wedge \lambda^{\mathcal{T}}(q, \sigma) = \lambda^{\mathcal{T}}(r', \sigma)$. This indicates that $q \notin C(r)$ and thus is r apart from all other states in S.
2. $\lambda^{\mathcal{T}}(q, \sigma) = \lambda^{\mathcal{T}}(r, \sigma) \wedge \lambda^{\mathcal{T}}(q, \sigma) \neq \lambda^{\mathcal{T}}(r', \sigma)$. This indicates that $q \notin C(r')$ and thus is r apart from all other states in S.
3. $\lambda^{\mathcal{T}}(q, \sigma) \neq \lambda^{\mathcal{T}}(r, \sigma) \wedge \lambda^{\mathcal{T}}(q, \kappa) \neq \lambda^{\mathcal{T}}(r', \sigma)$. This indicates that both states are apart from all other states in S.
4. $\lambda^{\mathcal{T}}(q, \sigma) = \lambda^{\mathcal{T}}(r, \sigma) \wedge \lambda^{\mathcal{T}}(q, \sigma) = \lambda^{\mathcal{T}}(r', \sigma)$. This can never occur because it implies $\lambda^{\mathcal{T}}(r, \sigma) = \lambda^{\mathcal{T}}(r', \sigma)$ while we assumed that $\sigma \vdash r \# r'$.

All cases lead to a new state that can be added to the basis. □

Lemma 3.6

Suppose that we have a hypothesis $\mathcal{H}$ that is consistent with some observation tree $\mathcal{T}$. Moreover, suppose that there are two states $q, q' \in S$ and these states are equivalent in $\mathcal{H}$, i.e. $q^{\mathcal{H}} \approx q'^{\mathcal{H}}$. Because $q', q \in S$, there must exist some $\sigma \in I^+$ such that $\lambda^{\mathcal{T}}(q, \sigma) \neq \lambda^{\mathcal{T}}(q', \sigma)$, otherwise q and q' cannot both be in the basis S. Because $q^{\mathcal{H}} \approx q'^{\mathcal{H}}$, it must be the case that σ leads to conflict when run from either q or q'. This contradicts the assumption that $\mathcal{H}$ is consistent. Therefore, it must hold that if $\mathcal{H}$ is consistent, then $\mathcal{H}$ is minimal. □

Lemma 3.7

From Lemma 3.6, we know that because $\mathcal{H}$ is consistent, $\mathcal{H}$ is minimal. Therefore, all states in $\mathcal{H}$ is inequivalent and a separating sequence between the every pair of states exists.

The chosen algorithm to make a minimal splitting tree terminates and is correct (see [19]). The for-loop is guaranteed to terminate because there exist at most $|S|^2$ pairs (q', q) in the hypothesis, this proves termination. To prove correctness, we use a case distinction:

1. For $q', q \in S$, there exists some $\sigma \in I^+$ with $\sigma \vdash q' \# q$ and $|\sigma| \leq |S|$. In this case, this pair already satisfies the condition that the states are apart with a short witness.
2. For $q', q \in S$, there exists some $\sigma \in I^+$ with $\sigma \vdash q' \# q$ and $|\sigma| > |S|$. In this case, we use the separating sequence which can be read from the minimal splitting tree. This sequence has length at most $|S|$. We run OutputQuery($\mathsf{access}(q')$ *sepseq*) and OutputQuery($\mathsf{access}(q)$ *sepseq*) which leads to the following cases:

(a) $\lambda^{\mathcal{T}}(q', sepseq) \neq \lambda^{\mathcal{T}}(q, sepseq)$ and the output is consistent with the output expected by the hypothesis. In this case, *sepseq* is a valid witness that shows $q' \# q$ and $|sepseq| \leq |S|$.
(b) $\lambda^{\mathcal{T}}(q', sepseq) \neq \lambda^{\mathcal{T}}(q, sepseq)$ but the output is inconsistent with the output expected by the hypothesis. In this case, *sepseq* is a valid witness that shows $q' \# q$ and $|sepseq| \leq |S|$. However, the hypothesis is incorrect because it is inconsistent with the sequence *sepseq* in the observation tree. From this we know that $\mathcal{H}$ is no longer a hypothesis for $\mathcal{T}$.
(c) $\lambda^{\mathcal{T}}(q', sepseq) = \lambda^{\mathcal{T}}(q, sepseq)$. This indicates that the hypothesis is incorrect because the sequence *sepseq* leads to a different output in the hypothesis than in the target automaton. From this we know that $\mathcal{H}$ is no longer a hypothesis for $\mathcal{T}$.

Suppose that each iteration of the for-loop leads to valid short witness *sepseq*, then after the for-loop each witness for all pairs of states in the hypothesis has a length of at most $|S|$. If one of the iterations leads to an unexpected output, we immediately get the result that the hypothesis $\mathcal{H}$ is no longer a valid hypothesis for $\mathcal{T}$. Therefore, Algorithm 3 satisfies the conditions of the Lemma. □

Theorem 3.8

Note that this proof is very similar to the correctness proof for $L^{\#}$ (Theorem 3.8 in [21]), for completeness the whole proof is displayed here. In all cases, let $S, W, Z, F, \mathcal{T}$ denote the values before and $S', W', Z', F', \mathcal{T}'$ denote the values after the respective rule application.

R1$_W$ If there exists some state $r \in Z$ which has a short witness with all states $q \in W$, we move r from Z to W, i.e. $W' := W \cup \{r\}$ and $Z' := Z \setminus \{r\}$. This indicates that

$$\begin{aligned} N_W(\mathcal{T}') &= 2|S'|\cdot(|W'|+1) \\ &= 2(|W' \cup Z'|) \cdot (|W| + 1 + 1) \\ &= 2(|W \cup Z| + 1 - 1) \cdot (|W| + 1 + 1) \\ &= 2(|S|) \cdot (|W| + 1 + 1) \\ &= N_W(\mathcal{T}) + 2|S| \end{aligned}$$

$$\begin{aligned} N_Z(\mathcal{T}') &= |S'|\cdot(|Z'|+1) \\ &= (|W' \cup Z'|) \cdot (|Z| + 1 - 1) \\ &= (|W \cup Z| + 1 - 1) \cdot |Z| \\ &= |S|\cdot|Z| \\ &= \mathcal{N}_Z(\mathcal{T}) - |S| \end{aligned}$$

This has no effect on $N_{\downarrow}(\mathcal{T})$, $N_{\#_s - S \times S}(\mathcal{T})$ or $N_{\# - S \times F}(\mathcal{T})$. Therefore,

$$N(\mathcal{T}') = N(\mathcal{T}) + 2|S| - |S| = N(\mathcal{T}) + |S| > N(\mathcal{T})$$

R1$_Z$ If q is isolated, then q is moved from F to Z, i.e. $Z' := Z \cup \{q\}$.

$$\begin{aligned} N_Z(\mathcal{T}') &= |S'|\cdot(|Z'|+1) \\ &= |W' \cup Z'|\cdot(|Z'|+1) \\ &= (|W \cup Z|+1) \cdot (|Z| + 1 + 1) \\ &= (|S|+1) \cdot (|Z| + 1 + 1) \\ &= N_Z(\mathcal{T}) + |Z| + 2|S| + 2 \end{aligned}$$

$$N_W(\mathcal{T}') = N_W(\mathcal{T}) \qquad N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T}) \qquad N_{\#_s - S \times S}(\mathcal{T}') \supseteq N_{\#_s - S \times S}(\mathcal{T})$$

Finally we have

$$N_{\#-S\times F}(\mathcal{T}') \quad \supseteq \quad N_{\#-S\times F}(\mathcal{T}) \setminus (S \times \{q\})$$

and thus

$$|N_{\#-S\times F}(\mathcal{T}')| \quad \geq \quad |N_{\#-S\times F}(\mathcal{T})| - |S|.$$

In total,

$$N(\mathcal{T}') \geq N(\mathcal{T}) + |Z| + 2|S| + 2 \; - \; |S| = N(\mathcal{T}) + |Z| + |S| + 2$$

R2 For this rule, let $\delta^{\mathcal{T}}(q, i)$ for some $q \in S$, $i \in I$. After the output query for $\mathsf{access}(q)\, i$, we have

$$N_W(\mathcal{T}') = N_W(\mathcal{T}) \qquad N_Z(\mathcal{T}') = N_Z(\mathcal{T}) \qquad N_{\downarrow}(\mathcal{T}') = N_{\downarrow}(\mathcal{T}) \cup \{(q, i)\}$$

$$N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T}) \qquad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T})$$

Thus $N(\mathcal{T}') \geq N(\mathcal{T}) + 1$.

R3$_{short}$ For this rule, consider a state $q \in F$ and distinct $r, r' \in S$ with $r' \#_s r$ and $\neg(q \# r)$ and $\neg(q \# r')$. The algorithm performs the query

$$\textsc{OutputQuery}(\mathsf{access}(q)\ \sigma).$$

Hence, $\delta^{\mathcal{T}}(q, \sigma)\downarrow$ in the updated observation tree, which implies $r \#_s q$ or $r' \#_s q$ by weak co-transitivity (Lemma 2.5). Thus,

$$N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(r, q)\} \quad \textit{or} \quad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(r', q)\}$$

and therefore $|N_{\#-S\times F}(\mathcal{T}')| \geq |N_{\#-S\times F}(\mathcal{T})|+1$. Additionally, $N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components stay unchanged. Thus, the norm rises.

R3$_{long}$ For this rule, consider a state $r \in F$ and distinct $u, u' \in S$ with $\rho \vdash u \# u'$ and $\neg(r \# u)$ and $\neg(r \# u')$. The algorithm performs the query

$$\textsc{OutputQuery}(\mathsf{access}(r)\ \rho).$$

Hence, $\delta^{\mathcal{T}}(r, \rho)\downarrow$ in the updated observation tree, which implies $r \# u$ or $r \# u'$ by weak co-transitivity (Lemma 2.5). Thus,

$$N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(u, r)\} \quad \textit{or} \quad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(u', r)\}$$

and therefore $|N_{\#-S\times F}(\mathcal{T}')| \geq |N_{\#-S\times F}(\mathcal{T})|+1$. Additionally, $N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components stay unchanged. Thus, the norm rises.

R3$_{orig}$ For this rule, consider a state $q \in F$ and distinct $r, r' \in S$ with $r' \# r$ and $\neg(q \# r)$ and $\neg(q \# r')$. The algorithm performs the query

$$\textsc{OutputQuery}(\mathsf{access}(q)\ \sigma).$$

Hence, $\delta^{\mathcal{T}}(q, \sigma)\downarrow$ in the updated observation tree, which implies $r \#_s q$ or $r' \#_s q$ by weak co-transitivity (Lemma 2.5). Thus,

$$N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(r, q)\} \quad \textit{or} \quad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(r', q)\}$$

and therefore $|N_{\#-S\times F}(\mathcal{T}')| \geq |N_{\#-S\times F}(\mathcal{T})|+1$. Additionally, $N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components stay unchanged. Thus, the norm rises.

R4 If rule (R4) did not terminate the algorithm and did not enter the if-statement that leads to execution of ComputeSepSeq, we show that $\mathcal{H}$ is not a hypothesis for $\mathcal{T}'$ anymore. By Lemma 3.10 in [21]) and by EquivalenceQuery, we have that $\sigma \in I^+$ is such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}'}(q_0^{\mathcal{T}'}, \sigma)$. Moreover, the basis S is not modified during CheckConsistency and ProcessCounterexample: $S = S'$. Even though the observation tree has been updated since BuildHypothesis, $\mathcal{H}$ still meets the criteria of Lemma 3.11 in [21]. Therefore,

after counterexample processing, $\mathcal{H}$ is not a hypothesis for $\mathcal{T}'$ anymore, that is, there exist $p \in S$, $p \xrightarrow{i/o} q$ in $\mathcal{H}$, and $p \xrightarrow{i/o'} r$ in $\mathcal{T}'$ with $o \neq o'$ or $q \# r$. But the case $o \neq o'$ does not occur: since S is complete, and $\mathcal{H}$ is a hypothesis for $\mathcal{T}$, the transition $p \xrightarrow{i/o} q$ in $\mathcal{H}$ implies $p \xrightarrow{i/o} r$ in $\mathcal{T}$ and therefore also in the extension $\mathcal{T}'$. This indicates that $q \# r$ and $r \in F$. Thus,

$$(q, r) \in N_{\#-S\times F}(\mathcal{T}') \setminus N_{\#-S\times F}(\mathcal{T})$$

Moreover, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components of the norm stay unchanged. Therefore, the total norm increases.

Now consider the case where we did enter the if-statement that leads to execution of ComputeSepSeq. This can only occur when there exists at least one $\sigma \in I^+$ such that $\sigma \vdash q' \#_l q$ for some $q, q' \in S$. From Lemma 3.7, we know that after execution of ComputeSepSeq, we either have a witness of length at most $|S|$ for each pair of states that previously had a long witness or the hypothesis is no longer a valid hypothesis for the observation tree, in the last case we use the argument explained above to prove that the norm has increased. In the first case,

$$N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T}) \cup \{(q, q')\}$$

If the hypothesis is still a valid hypothesis for the observation tree, then the other parts of the norm have remained the same. This indicates that the total norm has increased because $|N_{\#_s-S\times S}(\mathcal{T}')| \geq |N_{\#_s-S\times S}(\mathcal{T})| + 1$.

(R5) Let $q, q' \in S$ with $i\rho \vdash q \#_l q'$ for some $i \in I, \rho \in I^+$. Moreover, let $r, r' \in S \cup F$ and $\delta^{\mathcal{T}}(q, i) = r, \delta^{\mathcal{T}}(q', i) = r'$. Moreover, let $C(r) = \{u\}$ and $C(r') = \{u'\}$ and $\sigma \vdash u \#_s u'$ for some $\sigma \in I^+$ with $|\sigma| < |S|$. From Lemma 3.4, we know that we find either a a new basis states or a short witness for two basis states.

If a new basis states is found, the following holds:

$$N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(q, r)\} \quad or \quad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(q', r')\}$$

and therefore $|N_{\#-S\times F}(\mathcal{T}')| \geq |N_{\#-S\times F}(\mathcal{T})|+1$. Additionally, $N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components stay unchanged. Thus, the norm rises.

The short witness that can be found if no new basis states are found is the witness $i\sigma \vdash q \#_s q'$. Thus,

$$N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T}) \cup \{(q, q')\}$$

and therefore $|N_{\#_s-S\times S}(\mathcal{T}')| \geq |N_{\#_s-S\times S}(\mathcal{T})|+1$. The other components of the norm stay unchanged. Thus, the norm rises.

(R6) Let $r, r' \in F$ with $\sigma \in r \# r'$ for some $\sigma \in I^+$. If $C(r) = \{q\} = C(r')$ with $q \in S$, then we can perform an output query from q with σ. From Lemma 3.4, we know that after applying rule (R5), either r or r' is apart from all other states in the basis. Thus,

$$N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(q, r)\} \quad or \quad N_{\#-S\times F}(\mathcal{T}') \supseteq N_{\#-S\times F}(\mathcal{T}) \cup \{(q, r')\}$$

and therefore $|N_{\#-S\times F}(\mathcal{T}')| \geq |N_{\#-S\times F}(\mathcal{T})|+1$. Additionally, $N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$, $N_{\#_s-S\times S}(\mathcal{T}') \supseteq N_{\#_s-S\times S}(\mathcal{T})$ and the other components stay unchanged. Thus, the norm rises. □

Theorem 3.10

In strategic $L^{\#}_Z$, every (non-terminating) application of rule (R4) that does not trigger execution of ComputeSepSeq leads to an isolated state in the frontier. An isolated state in the frontier indicates that the basis increases with one state. There are at most n states in the Mealy machine, therefore, there are at most $n - 1$ applications of (R4) that do not terminate or trigger execution

of ComputeSepSeq. In those cases, we always use ProcCounterEx, this requires $\mathcal{O}(n \log m)$ output queries.

Execution of ComputeSepSeq can only be triggered if there is at least one pair of states $(q, q') \in S \times S$ and some $\sigma \in I^+$ with $\sigma \vdash q \# q'$ and $|\sigma| > |S|$. Because,

$$|\{(q, q') \in S \times S \mid q \# q'\}| \leq (n-1)n$$

and because for each pair two output queries are performed, a total of at most $2(n-1)n$ output queries are required.

The maximum number of output queries for rule (R4) has now been discussed. Next, we consider for each of the rules the maximum number of output queries that can be performed. Since there is a functional simulation $\mathcal{T} \to \mathcal{M}$, we have $|S| \leq n$ (Lemma 2.7 in [21]).

(R1$_W$) & (R1$_Z$) These rules cannot execute output queries.
(R2) The number of successors of the basis is bounded by $k \cdot n$

$$|\{(q, i) \in S \times I \mid \delta(q, i)\downarrow\}| \leq kn$$

(R3$_{short}$), (R3$_{orig}$) &(R3$_{orig}$) The set $S \cup F$ contains at most $kn+1$ elements. Since each state in the frontier can be apart from at most $n-1$ states in the basis,

$$|\{(q, q') \in S \times F \mid q \# q'\}| \leq (n-1)(kn+1)$$

(R5) The number of new states that can be found is bounded by n. The number of short witness that can be found is bounded by $n(n-1)$ because each state can be apart from at most $n-1$ other states.

$$|\{(q, q') \in S \times S \mid q \# q'\}| \leq (n-1)n$$

Because two output queries are performed in this rule, the number of output queries performed by rule (R6) is at most

$$2(n(n-1)) + 2n = 2(n(n-1+1)) = 2n^2$$

(R6) The number of new states that can be found is bounded by n

Combining all these output queries leads to an upper bound of $\mathcal{O}(kn^2 + n \log m)$ queries. □

Theorem 3.11

During counterexample processing, the largest witness between a state $q \in F$ and $r \in S$ has a length of at most m. When the state q is moved to the basis, the witness between the states q and r still has a length of at most m. At any point in the $L_Z^\#$ algorithm after a counterexample is processed (the sub-routine ProcCounterEx has terminated after the equivalence query), all output queries are a combination of the access sequence and (a part) of the witness. Therefore, the input symbols used by all output queries, during counterexample processing and not during counterexample processing, are bounded by m. Combining this information with the fact that there are $\mathcal{O}(kn^2 + n \log m)$ output queries, it follows that $\mathcal{O}(kmn^2 + n \log m)$ input symbols are required. □

B Complete benchmarking results for Experiment 1

In Tables 1 and 2, we list the number of queries for every model and both learning algorithms. In Tables 3 and 4, we list the number of symbols used per rule for every model and both learning algorithms. Note that the number of symbols needed for rule (R2) is not present in Tables 3 and 4. Yellow values indicate the best value out of the two algorithms. Pink values indicate a value higher than 0 for new rules. Moreover, we have the following abbreviations:

- PCE: Symbols used during ProcCounterEx.
- CSS: Symbols used during ComputeSepSeq.
- SWQ: Symbols used during queries with short witnesses.
- LWQ: Symbols used during queries with long witnesses.

See Section 4.1 for and description of the benchmark setup.

Table 1: *Benchmark results (Part 1 of Experiment 1)*

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
4_learnresult_SecureCode Aut_fix		$L^\#$	2.96	451.80	138.46	220.20	28.32	838.78
$n = 4$	$k = 14$	$L^\#_Z$	2.96	451.72	138.42	219.76	28.30	838.20
ASN_learnresult_SecureCode Aut_fix		$L^\#$	2.96	451.64	138.40	220.92	28.40	839.36
$n = 4$	$k = 14$	$L^\#_Z$	2.96	451.62	138.40	221.42	28.42	839.86
1_learnresult_MasterCard_fix		$L^\#$	2.88	719.30	198.20	129.22	17.44	1064.16
$n = 5$	$k = 15$	$L^\#_Z$	3.80	731.18	198.82	540.42	65.62	1536.04
OpenSSL_1.0.1j_client_regular		$L^\#$	5.60	297.98	85.64	4422.94	542.34	5348.90
$n = 6$	$k = 7$	$L^\#_Z$	5.60	281.54	81.52	4406.88	540.28	5310.22
OpenSSL_1.0.1l_client_regular		$L^\#$	5.60	296.70	85.42	4408.16	540.14	5330.42
$n = 6$	$k = 7$	$L^\#_Z$	5.60	281.14	81.50	4422.52	542.04	5327.20
OpenSSL_1.0.2_client_regular		$L^\#$	5.60	302.16	86.52	4405.90	540.36	5334.94
$n = 6$	$k = 7$	$L^\#_Z$	5.60	281.46	81.50	4410.70	540.94	5314.60
RSA_BSAFE_Java_6.1.1_server_regular		$L^\#$	4.88	242.98	73.24	3590.48	456.64	4363.34
$n = 6$	$k = 8$	$L^\#_Z$	4.88	241.18	73.00	3589.80	456.80	4360.78
miTLS_0.1.3_server_regular		$L^\#$	4.28	419.44	114.78	1686.78	211.44	2432.44
$n = 6$	$k = 8$	$L^\#_Z$	4.10	383.90	106.92	1495.62	188.80	2175.24
10_learnresult_MasterCard_fix		$L^\#$	4.00	786.46	204.26	10527.18	1183.32	12701.22
$n = 6$	$k = 14$	$L^\#_Z$	4.00	782.06	203.42	10519.60	1182.44	12687.52
4_learnresult_MAESTRO_fix		$L^\#$	4.00	788.14	204.36	10553.78	1184.86	12731.14
$n = 6$	$k = 14$	$L^\#_Z$	4.00	780.24	202.86	10563.14	1186.02	12732.26
4_learnresult_PIN_fix		$L^\#$	4.00	786.68	204.02	10560.92	1185.62	12737.24
$n = 6$	$k = 14$	$L^\#_Z$	4.00	780.20	202.86	10558.06	1185.48	12726.60
ASN_learnresult_MAESTRO_fix		$L^\#$	4.00	787.08	204.12	10555.92	1185.08	12732.20
$n = 6$	$k = 14$	$L^\#_Z$	4.00	780.50	202.74	10559.46	1185.60	12728.30
Rabo_learnresult_MAESTRO_fix		$L^\#$	4.00	785.82	203.98	10555.20	1185.14	12730.14
$n = 6$	$k = 14$	$L^\#_Z$	4.00	780.92	202.82	10549.00	1184.56	12717.30
Rabo_learnresult_SecureCode_Aut_fix		$L^\#$	4.10	912.58	230.28	3761.96	440.50	5345.32
$n = 6$	$k = 15$	$L^\#_Z$	4.04	914.30	230.60	3421.30	402.52	4968.72
OpenSSL_1.0.2_server_regular		$L^\#$	5.82	391.46	112.16	3165.46	393.46	4062.54
$n = 7$	$k = 7$	$L^\#_Z$	5.82	333.44	98.88	2780.42	343.50	3556.24
GnuTLS_3.3.12_client_regular		$L^\#$	4.86	340.26	100.02	6337.54	735.20	7513.02
$n = 7$	$k = 8$	$L^\#_Z$	4.86	342.16	100.46	6344.94	735.98	7523.54
GnuTLS_3.3.12_server_regular		$L^\#$	4.86	338.74	99.58	6364.76	738.50	7541.58
$n = 7$	$k = 8$	$L^\#_Z$	4.86	342.10	100.46	6356.90	737.32	7536.78
NSS_3.17.4_client_regular		$L^\#$	5.84	346.48	91.08	7387.58	909.80	8734.94
$n = 7$	$k = 8$	$L^\#_Z$	5.84	335.06	89.00	7399.52	911.24	8734.82
Volksbank_learnresult_MAESTRO_fix		$L^\#$	4.58	1306.78	304.90	1054.46	121.74	2787.88
$n = 7$	$k = 14$	$L^\#_Z$	4.58	1463.24	332.90	1056.86	121.74	2974.74

Table 2: *Benchmark results (Part 2 of Experiment 1)*

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
NSS_3.17.4_server_regular		$L^\#$	5.50	392.16	111.54	8441.84	1049.06	9994.60
$n = 8$	$k = 8$	$L_Z^\#$	5.50	392.90	112.00	8446.60	1049.78	10001.28
RSA_BSAFE_C_4.0.4_server_regular		$L^\#$	4.86	482.42	135.02	5370.56	617.42	6605.42
$n = 9$	$k = 8$	$L_Z^\#$	4.84	463.94	131.46	5554.52	637.98	6787.90
OpenSSL_1.0.2_client_full		$L^\#$	7.88	678.56	183.86	138556.70	15800.42	155219.54
$n = 9$	$k = 10$	$L_Z^\#$	7.88	612.12	168.94	142231.38	16220.38	159232.82
GnuTLS_3.3.12_client_full		$L^\#$	8.06	586.92	169.52	177113.06	20303.38	198172.88
$n = 9$	$k = 12$	$L_Z^\#$	8.04	619.40	176.36	179154.84	20509.44	200460.04
GnuTLS_3.3.12_server_full		$L^\#$	6.00	701.02	196.08	33646.92	3777.06	38321.08
$n = 9$	$k = 12$	$L_Z^\#$	6.00	735.38	200.30	33938.04	3780.02	38653.74
learnresult_fix		$L^\#$	6.06	1963.52	433.48	2346.02	260.48	5003.50
$n = 9$	$k = 15$	$L_Z^\#$	6.40	1968.50	432.82	2476.66	274.82	5152.80
OpenSSL_1.0.1g_client_regular		$L^\#$	8.02	630.96	149.68	16062.80	1755.68	18599.12
$n = 10$	$k = 7$	$L_Z^\#$	7.30	577.78	140.24	14184.90	1541.06	16443.98
OpenSSL_1.0.1l_server_regular		$L^\#$	6.52	595.54	154.14	58758.70	6722.98	66231.36
$n = 10$	$k = 7$	$L_Z^\#$	6.70	499.80	134.00	52058.50	5959.86	58652.16
OpenSSL_1.0.1j_server_regular		$L^\#$	6.46	732.90	174.94	63043.72	6936.88	70888.44
$n = 11$	$k = 7$	$L_Z^\#$	6.78	593.30	148.50	58510.36	6439.04	65691.20
NSS_3.17.4_client_full		$L^\#$	8.10	780.90	198.54	581672.20	63974.04	646625.68
$n = 11$	$k = 12$	$L_Z^\#$	8.36	765.48	195.86	589479.66	64777.62	655218.62
GnuTLS_3.3.8_server_regular		$L^\#$	8.76	697.56	170.28	331223.40	35042.72	367133.96
$n = 12$	$k = 8$	$L_Z^\#$	8.78	714.16	173.72	333985.58	35303.70	370177.16
TCP_FreeBSD_Client		$L^\#$	6.00	1723.72	381.58	53832.50	5719.20	61657.00
$n = 12$	$k = 10$	$L_Z^\#$	5.94	1854.36	403.58	51392.32	5463.46	59113.72
TCP_Windows8_Client		$L^\#$	6.40	2177.04	456.50	28008.18	2915.22	33556.94
$n = 13$	$k = 10$	$L_Z^\#$	7.28	2229.70	459.44	25379.32	2617.06	30685.52
TCP_Linux_Client		$L^\#$	9.44	2520.92	529.86	106779.32	10995.08	120825.18
$n = 15$	$k = 10$	$L_Z^\#$	9.40	2647.92	551.54	119032.68	12206.34	134438.48
OpenSSL_1.0.1g_server_regular		$L^\#$	9.68	1243.90	274.52	100446.50	10452.38	112417.30
$n = 16$	$k = 7$	$L_Z^\#$	8.86	1001.10	229.88	101151.84	10526.62	112909.44
GnuTLS_3.3.8_server_full		$L^\#$	11.66	1292.70	307.62	2571821.42	259475.30	2832897.04
$n = 16$	$k = 11$	$L_Z^\#$	11.74	1367.40	318.56	2359096.82	236781.42	2597564.20
DropBear		$L^\#$	10.46	4617.72	780.26	47846.90	4405.64	57650.52
$n = 17$	$k = 13$	$L_Z^\#$	10.22	4628.12	785.50	56295.96	5123.54	66833.12
OpenSSH		$L^\#$	22.16	14475.98	2487.18	24764571.98	2221787.98	27003323.12
$n = 31$	$k = 22$	$L_Z^\#$	22.00	17015.02	2883.54	30756019.66	2740737.66	33516655.88
model4		$L^\#$	22.48	19116.16	2782.50	436858.92	35508.48	494266.06
$n = 34$	$k = 14$	$L_Z^\#$	24.30	18826.22	2726.52	425417.12	34645.78	481615.64
model1		$L^\#$	8.56	10041.34	1259.18	61738.08	4777.00	77815.60
$n = 35$	$k = 15$	$L_Z^\#$	8.50	8518.58	1256.40	58135.60	4929.62	72840.20
TCP_Windows8_Server		$L^\#$	25.48	21861.28	2681.56	775734.76	57356.32	857633.92
$n = 38$	$k = 13$	$L_Z^\#$	27.80	19954.64	2474.64	827779.24	61067.50	911276.02
TCP_FreeBSD_Server		$L^\#$	34.72	37533.18	3731.72	5914368.64	364101.80	6319735.34
$n = 55$	$k = 13$	$L_Z^\#$	37.46	34889.08	3553.80	7034153.96	433958.92	7506555.76
TCP_Linux_Server		$L^\#$	36.52	35857.94	3623.24	1848009.76	120080.64	2007571.58
$n = 57$	$k = 12$	$L_Z^\#$	37.00	34305.52	3502.44	1858274.46	119646.98	2015729.40
model3		$L^\#$	27.24	44603.90	5484.74	653788.60	47118.38	750995.62
$n = 58$	$k = 22$	$L_Z^\#$	28.46	41900.76	5436.76	786421.42	56847.06	890606.00
BitVise		$L^\#$	41.16	33572.72	3382.92	3979486.82	252034.76	4268477.22
$n = 66$	$k = 13$	$L_Z^\#$	43.42	32716.64	3375.60	4151358.82	272129.88	4459580.94

Table 3: *Benchmark results (Part 3 of Experiment 1)*

Model	Algo-	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb
n k	rithm	R3$_{short}$	R3$_{long}$	R3$_{orig}$	PCE	CSS	R5	R6	SWQ	LWQ
4_learnresult_SecureCode Aut_fix	$L^{\#}$	-	-	-	4.38	-	-	-	322.06	0.00
$n = 4$ $k = 14$	$L^{\#}_Z$	321.98	0.00	0.00	4.38	0.00	0.00	0.00	321.98	0.00
ASN_learnresult_SecureCode Aut_fix	$L^{\#}$	-	-	-	4.38	-	-	-	321.90	0.00
$n = 4$ $k = 14$	$L^{\#}_Z$	321.88	0.00	0.00	4.38	0.00	0.00	0.00	321.88	0.00
1_learnresult_MasterCard_fix	$L^{\#}$	-	-	-	9.64	-	-	-	516.94	0.00
$n = 5$ $k = 15$	$L^{\#}_Z$	525.62	0.00	0.00	11.04	0.00	0.00	0.00	525.62	0.00
OpenSSL_1.0.1j_client_regular	$L^{\#}$	-	-	-	4.06	-	-	-	224.68	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	213.30	0.00	0.00	4.06	0.00	0.00	0.00	213.30	0.00
OpenSSL_1.0.1l_client_regular	$L^{\#}$	-	-	-	4.06	-	-	-	222.84	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	213.00	0.00	0.00	4.06	0.00	0.00	0.00	213.00	0.00
OpenSSL_1.0.2_client_regular	$L^{\#}$	-	-	-	4.06	-	-	-	228.88	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	213.22	0.00	0.00	4.06	0.00	0.00	0.00	213.22	0.00
RSA_BSAFE_Java_6.1.1_server_regular	$L^{\#}$	-	-	-	0.24	-	-	-	155.62	0.00
$n = 6$ $k = 8$	$L^{\#}_Z$	152.40	0.00	0.00	0.24	0.00	0.00	0.00	152.40	0.00
miTLS_0.1.3_server_regular	$L^{\#}$	-	-	-	1.04	-	-	-	315.78	0.00
$n = 6$ $k = 8$	$L^{\#}_Z$	278.14	0.00	0.00	1.04	0.00	0.00	0.00	278.14	0.00
10_learnresult_MasterCard_fix	$L^{\#}$	-	-	-	5.52	-	-	-	568.78	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	564.30	0.00	0.00	5.52	0.00	0.00	0.00	564.30	0.00
4_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	5.42	-	-	-	569.64	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	561.96	0.00	0.00	5.42	0.00	0.00	0.00	561.96	0.00
4_learnresult_PIN_fix	$L^{\#}$	-	-	-	5.42	-	-	-	568.18	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	561.92	0.00	0.00	5.42	0.00	0.00	0.00	561.92	0.00
ASN_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	5.42	-	-	-	568.66	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	561.90	0.00	0.00	5.42	0.00	0.00	0.00	561.90	0.00
Rabo_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	5.42	-	-	-	567.54	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	562.40	0.00	0.00	5.42	0.00	0.00	0.00	562.40	0.00
Rabo_learnresult_SecureCode_Aut_fix	$L^{\#}$	-	-	-	6.80	-	-	-	659.30	0.00
$n = 6$ $k = 15$	$L^{\#}_Z$	659.98	0.00	0.00	6.88	0.00	0.00	0.00	659.98	0.00
OpenSSL_1.0.2_server_regular	$L^{\#}$	-	-	-	4.72	-	-	-	296.72	0.00
$n = 7$ $k = 7$	$L^{\#}_Z$	242.06	0.00	0.00	4.78	0.00	0.00	0.00	242.06	0.00
GnuTLS_3.3.12_client_regular	$L^{\#}$	-	-	-	3.20	-	-	-	223.88	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	226.26	0.00	0.00	3.20	0.00	0.00	0.00	226.26	0.00
GnuTLS_3.3.12_server_regular	$L^{\#}$	-	-	-	3.12	-	-	-	223.26	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	226.28	0.00	0.00	3.12	0.00	0.00	0.00	226.28	0.00
NSS_3.17.4_client_regular	$L^{\#}$	-	-	-	0.74	-	-	-	227.84	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	219.50	0.00	0.00	0.74	0.00	0.00	0.00	219.50	0.00
Volksbank_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	23.30	-	-	-	994.38	0.00
$n = 7$ $k = 14$	$L^{\#}_Z$	1159.38	0.00	0.00	23.12	0.00	0.00	0.00	1159.38	0.00
NSS_3.17.4_server_regular	$L^{\#}$	-	-	-	1.22	-	-	-	252.62	0.00
$n = 8$ $k = 8$	$L^{\#}_Z$	254.30	0.00	0.00	1.22	0.00	0.00	0.00	254.30	0.00
RSA_BSAFE_C_4.0.4_server_regular	$L^{\#}$	-	-	-	2.06	-	-	-	319.28	0.00
$n = 9$ $k = 8$	$L^{\#}_Z$	302.74	0.00	0.00	1.92	0.00	0.00	0.00	302.74	0.00

Table 4: *Benchmark results (Part 4 of Experiment 1)*

Model		Algo-	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb
n	k	rithm	R3$_{short}$	R3$_{long}$	R3$_{orig}$	PCE	CSS	R5	R6	SWQ	LWQ
OpenSSL_1.0.2_client_full		$L^\#$	-	-	-	6.02	-	-	-	533.92	0.00
$n = 9$	$k = 10$	$L^\#_Z$	473.18	0.00	0.00	6.24	0.00	0.00	0.00	473.18	0.00
GnuTLS_3.3.12_client_full		$L^\#$	-	-	-	5.56	-	-	-	411.60	0.00
$n = 9$	$k = 12$	$L^\#_Z$	447.58	0.00	0.00	5.34	0.00	0.00	0.00	447.58	0.00
GnuTLS_3.3.12_server_full		$L^\#$	-	-	-	3.50	-	-	-	454.02	0.00
$n = 9$	$k = 12$	$L^\#_Z$	483.32	0.00	0.00	3.28	0.00	0.00	0.00	483.32	0.00
learnresult_fix		$L^\#$	-	-	-	57.76	-	-	-	1486.04	0.00
$n = 9$	$k = 15$	$L^\#_Z$	1485.92	0.00	0.00	61.42	0.00	0.00	0.20	1486.12	0.00
OpenSSL_1.0.1g_client_regular		$L^\#$	-	-	-	9.92	-	-	-	475.14	0.00
$n = 10$	$k = 7$	$L^\#_Z$	422.96	0.00	0.00	9.94	0.00	0.00	0.12	423.08	0.00
OpenSSL_1.0.1l_server_regular		$L^\#$	-	-	-	6.08	-	-	-	467.38	0.00
$n = 10$	$k = 7$	$L^\#_Z$	377.28	0.00	0.00	6.32	0.00	0.00	0.00	377.28	0.00
OpenSSL_1.0.1j_server_regular		$L^\#$	-	-	-	5.58	-	-	-	567.08	0.00
$n = 11$	$k = 7$	$L^\#_Z$	434.54	0.00	0.00	5.90	0.00	0.00	0.00	434.54	0.00
NSS_3.17.4_client_full		$L^\#$	-	-	-	1.24	-	-	-	512.30	0.00
$n = 11$	$k = 12$	$L^\#_Z$	503.92	0.00	0.00	1.24	0.00	0.00	0.00	503.92	0.00
GnuTLS_3.3.8_server_regular		$L^\#$	-	-	-	24.58	-	-	-	514.38	0.00
$n = 12$	$k = 8$	$L^\#_Z$	530.10	0.00	0.00	24.72	0.00	0.00	0.00	530.10	0.00
TCP_FreeBSD_Client		$L^\#$	-	-	-	37.10	-	-	-	1364.76	0.00
$n = 12$	$k = 10$	$L^\#_Z$	1501.70	0.00	0.00	37.24	0.00	0.00	0.00	1501.70	0.00
TCP_Windows8_Client		$L^\#$	-	-	-	44.30	-	-	-	1785.72	0.00
$n = 13$	$k = 10$	$L^\#_Z$	1848.92	0.00	0.00	52.90	0.00	0.00	0.14	1849.06	0.00
TCP_Linux_Client		$L^\#$	-	-	-	67.90	-	-	-	2063.72	0.00
$n = 15$	$k = 10$	$L^\#_Z$	2205.18	0.00	0.00	72.24	0.00	0.00	0.24	2205.42	0.00
OpenSSL_1.0.1g_server_regular		$L^\#$	-	-	-	12.42	-	-	-	981.90	0.00
$n = 16$	$k = 7$	$L^\#_Z$	748.76	0.00	0.00	11.56	0.00	0.00	0.00	748.76	0.00
GnuTLS_3.3.8_server_full		$L^\#$	-	-	-	26.92	-	-	-	941.54	0.00
$n = 16$	$k = 11$	$L^\#_Z$	1013.72	0.00	0.00	27.40	0.00	0.00	0.08	1013.80	0.00
DropBear		$L^\#$	-	-	-	73.14	-	-	-	3679.56	0.00
$n = 17$	$k = 13$	$L^\#_Z$	3688.78	0.00	0.00	73.50	0.00	0.00	0.22	3689.00	0.00
OpenSSH		$L^\#$	-	-	-	242.04	-	-	-	11821.52	0.00
$n = 31$	$k = 22$	$L^\#_Z$	14452.28	0.00	0.00	228.74	0.00	0.00	0.12	14452.40	0.00
model4		$L^\#$	-	-	-	537.24	-	-	-	17183.72	11.34
$n = 34$	$k = 14$	$L^\#_Z$	16882.14	0.58	2.10	558.36	0.00	0.00	1.80	16883.94	2.68
model1		$L^\#$	-	-	-	178.82	-	-	-	6630.80	0.00
$n = 35$	$k = 15$	$L^\#_Z$	5660.52	0.00	0.00	154.78	0.00	0.00	0.28	5660.80	0.00
TCP_Windows8_Server		$L^\#$	-	-	-	546.46	-	-	-	19223.26	27.68
$n = 38$	$k = 13$	$L^\#_Z$	17318.84	0.32	0.00	571.00	0.00	0.08	2.40	17321.32	0.32
TCP_FreeBSD_Server		$L^\#$	-	-	-	1089.74	-	-	-	32699.42	16.26
$n = 55$	$k = 13$	$L^\#_Z$	30115.32	0.00	0.00	1096.12	0.00	0.00	5.86	30121.18	0.00
TCP_Linux_Server		$L^\#$	-	-	-	1104.28	-	-	-	31360.12	23.08
$n = 57$	$k = 12$	$L^\#_Z$	29809.72	1.04	0.00	1075.00	0.00	0.14	6.60	29816.46	1.04
model3		$L^\#$	-	-	-	780.92	-	-	-	37123.84	0.00
$n = 58$	$k = 22$	$L^\#_Z$	34836.56	0.00	0.00	753.26	0.00	0.00	6.64	34843.20	0.00
BitVise		$L^\#$	-	-	-	1091.70	-	-	-	28106.04	0.00
$n = 66$	$k = 13$	$L^\#_Z$	27449.58	0.00	0.00	1109.90	0.00	0.00	5.50	27455.08	0.00

C Complete benchmarking results for Experiment 2

C.1 Experiment Tables

In Tables 1 and 2, we list the number of queries for every model and both learning algorithms. In Tables 3 and 4, we list the number of symbols used per rule for every model and both learning algorithms. Note that the number of symbols needed for rule (R2) is not present in Tables 3 and 4. Yellow values indicate the best value out of the two algorithms. Pink values indicate a value higher than 0 for new rules. Moreover, we have the following abbreviations:

- PCE: Symbols used during ProcCounterEx.
- CSS: Symbols used during ComputeSepSeq.
- SWQ: Symbols used during queries with short witnesses.
- LWQ: Symbols used during queries with long witnesses.

See Section 4.2 for and description of the benchmark setup.

Table 5: *Benchmark results (Part 1 of Experiment 2)*

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
4_learnresult_SecureCode Aut_fix		$L^\#$	2.95	470.45	142.20	21770.60	19.15	22402.40
$n = 4$	$k = 14$	$L^\#_Z$	2.95	470.30	142.20	21869.90	19.05	22501.45
ASN_learnresult_SecureCode Aut_fix		$L^\#$	2.95	470.35	142.20	21867.80	19.00	22499.35
$n = 4$	$k = 14$	$L^\#_Z$	2.95	470.60	142.25	21667.15	19.00	22299.00
1_learnresult_MasterCard_fix		$L^\#$	2.55	4145.05	207.65	12924.05	3.90	17280.65
$n = 5$	$k = 15$	$L^\#_Z$	3.35	9077.15	216.50	32504.05	38.50	41836.20
OpenSSL_1.0.1j_client_regular		$L^\#$	5.60	40038.20	113.65	89003.55	398.10	129553.50
$n = 6$	$k = 7$	$L^\#_Z$	5.60	40019.25	109.70	89011.05	398.05	129538.05
OpenSSL_1.0.1l_client_regular		$L^\#$	5.60	40041.15	114.25	88943.10	395.75	129494.25
$n = 6$	$k = 7$	$L^\#_Z$	5.60	40019.30	109.70	88965.85	395.00	129489.85
OpenSSL_1.0.2_client_regular		$L^\#$	5.60	40040.85	114.30	89017.05	397.80	129570.00
$n = 6$	$k = 7$	$L^\#_Z$	5.60	40019.25	109.70	88960.80	395.30	129485.05
RSA_BSAFE_Java_6.1.1_server_regular		$L^\#$	4.85	233.50	71.15	70958.80	518.65	71782.10
$n = 6$	$k = 8$	$L^\#_Z$	4.85	231.65	70.90	70934.45	517.60	71754.60
miTLS_0.1.3_server_regular		$L^\#$	4.35	419.05	114.00	51255.25	228.80	52017.10
$n = 6$	$k = 8$	$L^\#_Z$	4.05	386.25	106.60	43344.00	141.65	43978.50
10_learnresult_MasterCard_fix		$L^\#$	4.00	19651.65	223.85	56930.80	436.45	77242.75
$n = 6$	$k = 14$	$L^\#_Z$	4.00	19492.55	222.60	56802.20	436.85	76954.20
4_learnresult_MAESTRO_fix		$L^\#$	4.00	19687.85	224.30	54478.85	421.65	74812.65
$n = 6$	$k = 14$	$L^\#_Z$	4.00	19650.55	222.95	54705.90	423.25	75002.65
4_learnresult_PIN_fix		$L^\#$	4.00	19632.00	224.10	54721.40	421.60	74999.10
$n = 6$	$k = 14$	$L^\#_Z$	4.00	19553.35	222.85	54582.50	422.20	74780.90
ASN_learnresult_MAESTRO_fix		$L^\#$	4.00	19538.50	223.85	54543.35	422.55	74728.25
$n = 6$	$k = 14$	$L^\#_Z$	4.00	19567.25	222.75	54533.65	422.05	74745.70
Rabo_learnresult_MAESTRO_fix		$L^\#$	4.00	19609.10	223.85	54627.65	421.70	74882.30
$n = 6$	$k = 14$	$L^\#_Z$	4.00	19704.25	222.90	54785.15	422.25	75134.55
Rabo_learnresult_SecureCode_Aut_fix		$L^\#$	3.80	21158.55	248.00	50951.45	314.70	72672.70
$n = 6$	$k = 15$	$L^\#_Z$	3.80	20817.65	248.30	50772.40	276.25	72114.60
OpenSSL_1.0.2_server_regular		$L^\#$	5.15	29865.25	132.50	79197.15	402.60	109597.50
$n = 7$	$k = 7$	$L^\#_Z$	4.90	25238.40	112.50	71033.95	358.25	96743.10
GnuTLS_3.3.12_client_regular		$L^\#$	5.00	36614.90	125.20	87704.95	1144.65	125589.70
$n = 7$	$k = 8$	$L^\#_Z$	5.00	36620.50	126.70	87608.30	1140.40	125495.90
GnuTLS_3.3.12_server_regular		$L^\#$	5.00	36614.05	125.05	87604.45	1140.90	125484.45
$n = 7$	$k = 8$	$L^\#_Z$	5.00	36620.45	126.70	87604.25	1140.15	125491.55
NSS_3.17.4_client_regular		$L^\#$	5.75	332.70	87.75	81373.30	735.20	82528.95
$n = 7$	$k = 8$	$L^\#_Z$	5.75	324.15	86.15	81294.90	731.20	82436.40
Volksbank_learnresult_MAESTRO_fix		$L^\#$	3.90	64226.75	330.70	43244.60	68.25	107870.30
$n = 7$	$k = 14$	$L^\#_Z$	3.90	63257.35	360.90	43687.75	68.10	107374.10

Table 6: *Benchmark results (Part 2 of Experiment 2)*

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
NSS_3.17.4_server_regular		$L^\#$	5.25	1185.05	113.15	82837.35	851.90	84987.45
$n = 8$	$k = 8$	$L^\#_Z$	5.25	1185.55	113.55	82922.10	853.95	85075.15
RSA_BSAFE_C_4.0.4_server_regular		$L^\#$	4.55	10850.50	143.55	66917.50	498.60	78410.15
$n = 9$	$k = 8$	$L^\#_Z$	4.55	10832.20	140.95	66861.25	497.20	78331.60
OpenSSL_1.0.2_client_full		$L^\#$	7.90	75997.40	233.85	466117.70	15749.80	558098.75
$n = 9$	$k = 10$	$L^\#_Z$	7.85	75865.10	218.45	456324.15	15358.35	547766.05
GnuTLS_3.3.12_client_full		$L^\#$	7.80	79081.25	226.25	529119.45	19025.10	627452.05
$n = 9$	$k = 12$	$L^\#_Z$	7.80	79370.20	233.25	530651.20	19031.85	629286.50
GnuTLS_3.3.12_server_full		$L^\#$	6.10	64645.55	234.45	213040.60	5734.25	283654.85
$n = 9$	$k = 12$	$L^\#_Z$	6.10	64192.35	238.15	214132.55	5742.60	284305.65
learnresult_fix		$L^\#$	5.20	235678.95	482.45	83422.60	80.65	319664.65
$n = 9$	$k = 15$	$L^\#_Z$	5.55	264640.75	485.35	89356.90	87.55	354570.55
OpenSSL_1.0.1g_client_regular		$L^\#$	7.55	73342.25	198.30	155612.30	1618.50	230771.35
$n = 10$	$k = 7$	$L^\#_Z$	7.30	71405.75	185.95	146121.65	1622.20	219335.55
OpenSSL_1.0.1l_server_regular		$L^\#$	5.95	49730.55	187.65	213704.85	5542.05	269165.10
$n = 10$	$k = 7$	$L^\#_Z$	6.15	52959.60	159.60	225104.80	5854.55	284078.55
OpenSSL_1.0.1j_server_regular		$L^\#$	5.95	50632.30	208.05	230802.65	6262.80	287905.80
$n = 11$	$k = 7$	$L^\#_Z$	6.15	53798.25	173.05	239835.40	6499.30	300306.00
NSS_3.17.4_client_full		$L^\#$	8.25	761.45	194.95	966278.30	38833.00	1006067.70
$n = 11$	$k = 12$	$L^\#_Z$	8.40	740.60	191.95	974697.55	39192.50	1014822.60
GnuTLS_3.3.8_server_regular		$L^\#$	9.25	174716.85	232.20	6280164.65	270617.40	6725731.10
$n = 12$	$k = 8$	$L^\#_Z$	9.20	174058.65	233.85	5506608.40	236910.10	5917811.00
TCP_FreeBSD_Client		$L^\#$	5.60	166206.75	426.05	201998.60	5502.10	374133.50
$n = 12$	$k = 10$	$L^\#_Z$	5.80	161753.15	450.65	200731.80	5187.95	368123.55
TCP_Windows8_Client		$L^\#$	5.75	123820.40	491.20	177227.75	4239.00	305778.35
$n = 13$	$k = 10$	$L^\#_Z$	6.80	149366.60	495.95	167149.45	2488.10	319500.10
TCP_Linux_Client		$L^\#$	9.05	310154.80	596.15	624607.25	20601.25	955959.45
$n = 15$	$k = 10$	$L^\#_Z$	9.25	266255.35	618.65	585317.60	18682.40	870874.00
OpenSSL_1.0.1g_server_regular		$L^\#$	8.45	82427.10	328.95	409084.00	11832.25	503672.30
$n = 16$	$k = 7$	$L^\#_Z$	8.25	93635.35	271.55	373816.25	10493.00	478216.15
GnuTLS_3.3.8_server_full		$L^\#$	12.10	232381.50	394.75	18917108.60	816420.90	19966305.75
$n = 16$	$k = 11$	$L^\#_Z$	12.10	223136.65	403.30	19396319.85	832857.20	20452717.00
DropBear		$L^\#$	9.30	120753.15	850.75	404551.25	9766.05	535921.20
$n = 17$	$k = 13$	$L^\#_Z$	8.70	116923.85	847.95	402671.60	10136.40	530579.80
OpenSSH		$L^\#$	20.65	997288.65	2754.10	51840878.10	2113925.75	54954846.60
$n = 31$	$k = 22$	$L^\#_Z$	19.70	945336.45	3101.65	47348003.00	1921904.30	50218345.40
model4		$L^\#$	19.30	6274271.85	3088.90	905988.10	19524.95	7202873.80
$n = 34$	$k = 14$	$L^\#_Z$	20.15	3176350.40	2947.05	978352.35	21374.60	4179024.40
model1		$L^\#$	7.50	3358937.20	1347.50	201587.60	1576.50	3563448.80
$n = 35$	$k = 15$	$L^\#_Z$	7.05	2119981.35	1343.50	178970.10	1438.25	2301733.20
TCP_Windows8_Server		$L^\#$	21.30	2624927.80	2754.80	6442047.55	225459.10	9295189.25
$n = 38$	$k = 13$	$L^\#_Z$	24.50	1431502.65	2593.00	11264730.50	403585.80	13102411.95
TCP_FreeBSD_Server		$L^\#$	31.50	2642757.60	3883.15	88458363.95	2985917.60	94090922.30
$n = 55$	$k = 13$	$L^\#_Z$	32.60	1873471.60	3733.10	68806685.15	2335220.50	73019110.35
TCP_Linux_Server		$L^\#$	31.55	3635506.20	3826.70	16691944.80	550637.90	20881915.60
$n = 57$	$k = 12$	$L^\#_Z$	32.10	2198960.55	3641.65	16732001.75	548516.65	19483120.60
model3		$L^\#$	21.50	7858213.95	5855.45	1379757.20	33508.25	9277334.85
$n = 58$	$k = 22$	$L^\#_Z$	23.30	3959319.70	5618.50	1402521.25	32847.80	5400307.25
BitVise		$L^\#$	37.25	4422275.40	3484.85	81197112.95	2737983.00	88360856.20
$n = 66$	$k = 13$	$L^\#_Z$	38.75	3404269.80	3503.00	55373884.50	1875407.35	60657064.65

Table 7: *Benchmark results (Part 3 of Experiment 2)*

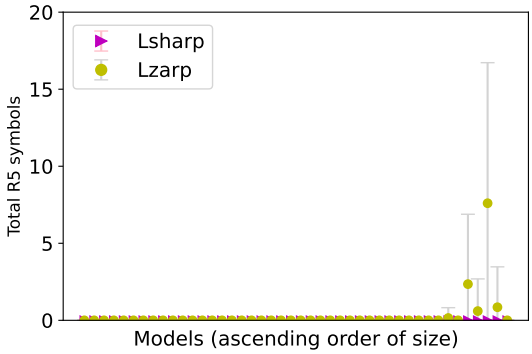
Model	Algo-	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb
n k	rithm	R3$_{short}$	R3$_{long}$	R3$_{orig}$	PCE	CSS	R5	R6	SWQ	LWQ
4_learnresult_SecureCode Aut_fix	$L^{\#}$	-	-	-	15.25	-	-	-	328.90	0.00
$n = 4$ $k = 14$	$L^{\#}_Z$	328.75	0.00	0.00	15.25	0.00	0.00	0.00	328.75	0.00
ASN_learnresult_SecureCode Aut_fix	$L^{\#}$	-	-	-	15.25	-	-	-	328.80	0.00
$n = 4$ $k = 14$	$L^{\#}_Z$	329.05	0.00	0.00	15.25	0.00	0.00	0.00	329.05	0.00
1_learnresult_MasterCard_fix	$L^{\#}$	-	-	-	3434.35	-	-	-	518.85	0.00
$n = 5$ $k = 15$	$L^{\#}_Z$	525.35	0.00	0.00	8359.45	0.00	0.00	0.00	525.35	0.00
OpenSSL_1.0.1j_client_regular	$L^{\#}$	-	-	-	39752.75	-	-	-	221.00	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	202.05	0.00	0.00	39752.75	0.00	0.00	0.00	202.05	0.00
OpenSSL_1.0.1l_client_regular	$L^{\#}$	-	-	-	39752.75	-	-	-	223.55	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	202.10	0.00	0.00	39752.75	0.00	0.00	0.00	202.10	0.00
OpenSSL_1.0.2_client_regular	$L^{\#}$	-	-	-	39752.75	-	-	-	224.15	0.00
$n = 6$ $k = 7$	$L^{\#}_Z$	202.05	0.00	0.00	39752.75	0.00	0.00	0.00	202.05	0.00
RSA_BSAFE_Java_6.1.1_server_regular	$L^{\#}$	-	-	-	0.20	-	-	-	145.95	0.00
$n = 6$ $k = 8$	$L^{\#}_Z$	143.90	0.00	0.00	0.20	0.00	0.00	0.00	143.90	0.00
miTLS_0.1.3_server_regular	$L^{\#}$	-	-	-	0.10	-	-	-	318.85	0.00
$n = 6$ $k = 8$	$L^{\#}_Z$	283.55	0.00	0.00	0.10	0.00	0.00	0.00	283.55	0.00
10_learnresult_MasterCard_fix	$L^{\#}$	-	-	-	18852.75	-	-	-	579.40	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	571.70	0.00	0.00	18700.35	0.00	0.00	0.00	571.70	0.00
4_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	18888.25	-	-	-	579.35	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	571.00	0.00	0.00	18858.30	0.00	0.00	0.00	571.00	0.00
4_learnresult_PIN_fix	$L^{\#}$	-	-	-	18833.20	-	-	-	578.55	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	573.10	0.00	0.00	18758.40	0.00	0.00	0.00	573.10	0.00
ASN_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	18738.10	-	-	-	579.55	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	570.90	0.00	0.00	18775.35	0.00	0.00	0.00	570.90	0.00
Rabo_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	18809.25	-	-	-	579.45	0.00
$n = 6$ $k = 14$	$L^{\#}_Z$	573.05	0.00	0.00	18909.35	0.00	0.00	0.00	573.05	0.00
Rabo_learnresult_SecureCode_Aut_fix	$L^{\#}$	-	-	-	20244.05	-	-	-	666.60	0.00
$n = 6$ $k = 15$	$L^{\#}_Z$	669.05	0.00	0.00	19899.35	0.00	0.00	0.00	669.05	0.00
OpenSSL_1.0.2_server_regular	$L^{\#}$	-	-	-	29495.70	-	-	-	281.85	0.00
$n = 7$ $k = 7$	$L^{\#}_Z$	202.45	0.00	0.00	24946.90	0.00	0.00	0.00	202.45	0.00
GnuTLS_3.3.12_client_regular	$L^{\#}$	-	-	-	36298.65	-	-	-	210.40	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	212.95	0.00	0.00	36298.65	0.00	0.00	0.00	212.95	0.00
GnuTLS_3.3.12_server_regular	$L^{\#}$	-	-	-	36298.65	-	-	-	209.60	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	212.90	0.00	0.00	36298.65	0.00	0.00	0.00	212.90	0.00
NSS_3.17.4_client_regular	$L^{\#}$	-	-	-	0.70	-	-	-	220.05	0.00
$n = 7$ $k = 8$	$L^{\#}_Z$	212.10	0.00	0.00	0.70	0.00	0.00	0.00	212.10	0.00
Volksbank_learnresult_MAESTRO_fix	$L^{\#}$	-	-	-	62942.05	-	-	-	995.30	0.00
$n = 7$ $k = 14$	$L^{\#}_Z$	1164.60	0.00	0.00	61809.10	0.00	0.00	0.20	1164.80	0.00
NSS_3.17.4_server_regular	$L^{\#}$	-	-	-	802.35	-	-	-	245.45	0.00
$n = 8$ $k = 8$	$L^{\#}_Z$	244.40	0.00	0.00	802.35	0.00	0.00	0.00	244.40	0.00
RSA_BSAFE_C_4.0.4_server_regular	$L^{\#}$	-	-	-	10364.35	-	-	-	324.40	0.00
$n = 9$ $k = 8$	$L^{\#}_Z$	308.20	0.00	0.00	10364.50	0.00	0.00	0.00	308.20	0.00

Table 8: *Benchmark results (Part 4 of Experiment 2)*

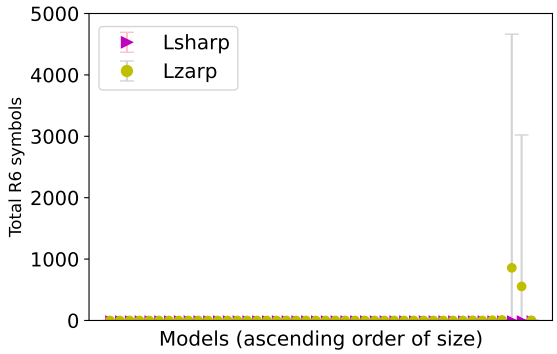
Model	Algo-	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb
n k	rithm	R3$_{short}$	R3$_{long}$	R3$_{orig}$	PCE	CSS	R5	R6	SWQ	LWQ
OpenSSL_1.0.2_client_full	$L^\#$	-	-	-	75338.95	-	-	-	526.90	0.00
$n = 9$ $k = 10$	$L^\#_Z$	454.80	0.00	0.00	75276.65	0.00	0.00	0.00	454.80	0.00
GnuTLS_3.3.12_client_full	$L^\#$	-	-	-	78515.25	-	-	-	404.10	0.00
$n = 9$ $k = 12$	$L^\#_Z$	432.95	0.00	0.00	78766.90	0.00	0.00	0.00	432.95	0.00
GnuTLS_3.3.12_server_full	$L^\#$	-	-	-	63215.40	-	-	-	444.85	751.65
$n = 9$ $k = 12$	$L^\#_Z$	470.80	0.00	0.00	63476.30	0.00	0.00	0.00	470.80	0.00
learnresult_fix	$L^\#$	-	-	-	233734.70	-	-	-	1512.50	0.00
$n = 9$ $k = 15$	$L^\#_Z$	1497.85	0.00	0.00	262713.05	0.00	0.00	0.25	1498.10	0.00
OpenSSL_1.0.1g_client_regular	$L^\#$	-	-	-	72728.30	-	-	-	473.20	0.00
$n = 10$ $k = 7$	$L^\#_Z$	410.65	0.00	0.00	70849.30	0.00	0.00	0.00	410.65	0.00
OpenSSL_1.0.1l_server_regular	$L^\#$	-	-	-	47925.55	-	-	-	474.85	1202.90
$n = 10$ $k = 7$	$L^\#_Z$	340.30	150.35	451.05	51897.15	0.00	0.00	0.00	340.30	601.40
OpenSSL_1.0.1j_server_regular	$L^\#$	-	-	-	49043.05	-	-	-	576.40	851.95
$n = 11$ $k = 7$	$L^\#_Z$	391.50	150.35	601.35	52498.10	0.00	0.00	0.00	391.50	751.70
NSS_3.17.4_client_full	$L^\#$	-	-	-	1.70	-	-	-	500.50	0.00
$n = 11$ $k = 12$	$L^\#_Z$	488.95	0.00	0.00	1.45	0.00	0.00	0.00	488.95	0.00
GnuTLS_3.3.8_server_regular	$L^\#$	-	-	-	168820.00	-	-	-	444.80	5310.65
$n = 12$ $k = 8$	$L^\#_Z$	449.75	951.90	2504.60	170003.35	0.90	0.00	0.00	450.65	3456.50
TCP_FreeBSD_Client	$L^\#$	-	-	-	123049.45	-	-	-	1311.35	41527.35
$n = 12$ $k = 10$	$L^\#_Z$	1426.20	7263.95	32911.10	119848.40	0.00	0.00	0.20	1426.40	40175.05
TCP_Windows8_Client	$L^\#$	-	-	-	118341.45	-	-	-	1736.70	3407.85
$n = 13$ $k = 10$	$L^\#_Z$	1717.40	0.00	0.00	147333.50	0.00	0.00	0.00	1717.40	0.00
TCP_Linux_Client	$L^\#$	-	-	-	242924.85	-	-	-	1900.90	64955.40
$n = 15$ $k = 10$	$L^\#_Z$	2055.40	3806.80	12520.75	247516.75	0.75	0.00	0.20	2056.35	16327.55
OpenSSL_1.0.1g_server_regular	$L^\#$	-	-	-	80348.45	-	-	-	981.95	852.20
$n = 16$ $k = 7$	$L^\#_Z$	679.40	0.00	0.00	92712.65	0.00	0.00	0.00	679.40	0.00
GnuTLS_3.3.8_server_full	$L^\#$	-	-	-	220745.15	-	-	-	822.20	10522.35
$n = 16$ $k = 11$	$L^\#_Z$	886.25	951.95	1903.65	219087.35	1.20	0.00	0.00	887.45	2855.60
DropBear	$L^\#$	-	-	-	116256.15	-	-	-	3630.90	0.00
$n = 17$ $k = 13$	$L^\#_Z$	3642.95	0.00	0.00	112419.25	0.00	0.00	0.00	3642.95	0.00
OpenSSH	$L^\#$	-	-	-	784011.25	-	-	-	11710.30	199085.85
$n = 31$ $k = 22$	$L^\#_Z$	14453.70	26557.90	174464.95	727360.00	0.00	0.00	0.00	14453.70	201022.85
model4	$L^\#$	-	-	-	2780517.25	-	-	-	14486.75	3477785.60
$n = 34$ $k = 14$	$L^\#_Z$	15776.85	149928.50	391108.10	2618027.10	9.95	0.15	2.45	15789.40	541036.60
model1	$L^\#$	-	-	-	1214831.70	-	-	-	5041.80	2135969.40
$n = 35$ $k = 15$	$L^\#_Z$	5201.95	339805.75	768734.65	1003425.85	9.05	0.00	0.55	5211.55	1108540.40
TCP_Windows8_Server	$L^\#$	-	-	-	1143035.65	-	-	-	17415.65	1462571.10
$n = 38$ $k = 13$	$L^\#_Z$	16748.40	2319.80	20898.15	1389662.95	0.50	2.35	4.10	16755.35	23217.95
TCP_FreeBSD_Server	$L^\#$	-	-	-	1399865.90	-	-	-	31721.40	1207663.20
$n = 55$ $k = 13$	$L^\#_Z$	29538.00	71658.40	252464.25	1516333.40	16.35	0.60	7.90	29562.85	324122.65
TCP_Linux_Server	$L^\#$	-	-	-	1831560.50	-	-	-	29662.85	1771159.95
$n = 57$ $k = 12$	$L^\#_Z$	28657.75	42469.65	160007.05	1963816.15	0.85	7.60	857.60	28672.40	203328.10
model3	$L^\#$	-	-	-	2910981.10	-	-	-	30642.15	4910140.90
$n = 58$ $k = 22$	$L^\#_Z$	31252.10	237682.35	954608.25	2728964.55	10.70	0.85	554.40	31267.15	1192841.50
BitVise	$L^\#$	-	-	-	2722023.95	-	-	-	22220.50	1674309.25
$n = 66$ $k = 13$	$L^\#_Z$	23679.65	214847.85	472566.05	2689479.50	50.15	0.00	3.15	23732.95	687413.90

C.2 Additional Figures Experiment 2

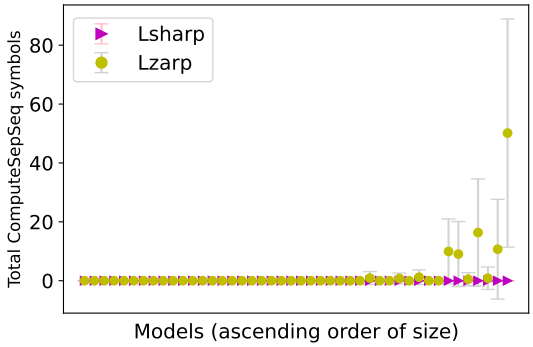
In Figures 14a, 14b and 14c additional information about the symbols used in different rules of the $L_Z^{\#}$ algorithm are displayed.



(a) Symbols used in rule (R5)



(b) Symbols used in rule (R6)



(c) Symbols used during ComputeSepSeq

Fig. 14: *Additional rule uses for Experiment 2 and algorithm* $L_Z^{\#}$

C.3 List of Benchmark Models Where Long Witnesses Are Not Possible

In Table 9, all models from the benchmarks used in Experiments 1 and 2 are listed where each pair of states has a witness of length 1.

	Model
1	10_learnresult_MasterCard_fix
2	1_learnresult_MasterCard_fix
3	4_learnresult_MAESTRO_fix
4	4_learnresult_PIN_fix
5	4_learnresult_SecureCode Aut_fix
6	ASN_learnresult_MAESTRO_fix
7	ASN_learnresult_SecureCode Aut_fix
8	GnuTLS_3.3.12_client_regular
9	GnuTLS_3.3.12_server_regular
10	learnresult_fix
11	miTLS_0.1.3_server_regular
12	NSS_3.17.4_client_regular
13	NSS_3.17.4_server_regular
14	OpenSSL_1.0.1g_client_regular
15	OpenSSL_1.0.1j_client_regular
16	OpenSSL_1.0.1l_client_regular
17	OpenSSL_1.0.2_client_regular
18	OpenSSL_1.0.2_server_regular
19	Rabo_learnresult_MAESTRO_fix
20	Rabo_learnresult_SecureCode_Aut_fix
21	RSA_BSAFE_C_4.0.4_server_regular
22	RSA_BSAFE_Java_6.1.1_server_regular
23	Volksbank_learnresult_MAESTRO_fix

Table 9: *List of models in the benchmarks from [16] where each pair of states has a witness of length 1*

D Complete benchmarking results for Experiment 3

In Table 1, we list the number of queries for every model and both learning algorithms. In Table 2, we list the number of symbols used per rule for every model and both learning algorithms. Note that the number of symbols needed for rule (R2) is not present in Tables 2. Yellow values indicate the best value out of the two algorithms. Pink values indicate a value higher than 0 for new rules. Moreover, we have the following abbreviations:

- PCE: Symbols used during ProcCounterEx.
- CSS: Symbols used during ComputeSepSeq.
- SWQ: Symbols used during queries with short witnesses.
- LWQ: Symbols used during queries with long witnesses.

See Section 4.3 for and description of the benchmark setup.

Table 10: *Benchmark results (Part 1 of Experiment 3)*

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
m164		$L^\#$	5.00	137368.68	30389.74	3861.68	847.72	172467.82
$n = 43$	$k = 102$	$L^\#_Z$	5.00	137081.08	30386.84	4198.56	921.64	172588.12
m183		$L^\#$	4.00	1478.70	430.94	6003.90	1821.80	9735.34
$n = 9$	$k = 12$	$L^\#_Z$	4.00	1573.30	452.66	5998.58	1820.24	9844.78
m217		$L^\#$	5.00	137759.40	22036.26	44750.94	7345.84	211892.44
$n = 14$	$k = 159$	$L^\#_Z$	5.00	137815.72	22032.56	44752.92	7346.04	211947.24
m54		$L^\#$	11.52	4652.70	1199.24	194246.16	47050.20	247148.30
$n = 27$	$k = 10$	$L^\#_Z$	11.62	4688.44	1210.18	207712.34	50242.44	263853.40
m95		$L^\#$	6.00	152417.34	25789.76	0.00	0.00	178207.10
$n = 33$	$k = 99$	$L^\#_Z$	6.00	151090.00	25773.00	0.00	0.00	176863.00

Table 11: *Benchmark results (Part 2 of Experiment 3)*

Model		Algo-	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb	Symb
n	k	rithm	R3$_{short}$	R3$_{long}$	R3$_{orig}$	PCE	CSS	R5	R6	SWQ	LWQ
m183		$L^\#$	-	-	-	13.14	-	-	-	1556.94	0.00
$n = 9$	$k = 12$	$L^\#_Z$	1674.62	0.00	0.00	13.14	0.00	0.00	0.00	1674.62	0.00
m217		$L^\#$	-	-	-	174.92	-	-	-	145336.62	0.00
$n = 14$	$k = 159$	$L^\#_Z$	145376.18	0.00	0.00	174.92	0.00	0.00	0.00	145376.18	0.00
m54		$L^\#$	-	-	-	41.82	-	-	-	4835.34	0.00
$n = 27$	$k = 10$	$L^\#_Z$	4890.28	0.00	0.00	42.06	0.00	0.00	0.00	4890.28	0.00
m95		$L^\#$	-	-	-	82.64	-	-	-	157797.90	0.00
$n = 33$	$k = 99$	$L^\#_Z$	155644.00	0.00	0.00	76.00	0.00	0.00	0.00	155644.00	0.00
m164		$L^\#$	-	-	-	6.76	-	-	-	145430.18	0.00
$n = 43$	$k = 102$	$L^\#_Z$	144181.72	0.00	0.00	3.06	0.00	0.00	0.00	144181.72	0.00