A New Approach for Active Automata Learning Based on Apartness*

Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann

Institute for Computing and Information Sciences, Radboud University, Nijmegen, the Netherlands

Abstract. We present $L^\#$, a new and simple approach to active automata learning. Instead of focusing on equivalence of observations, like the L^* algorithm and its descendants, $L^\#$ takes a different perspective: it tries to establish *apartness*, a constructive form of inequality. $L^\#$ does not require auxiliary notions such as observation tables or discrimination trees, but operates directly on tree-shaped automata. $L^\#$ has the same asymptotic query and symbol complexities as the best existing learning algorithms, but we show that adaptive distinguishing sequences can be naturally integrated to boost the performance of $L^\#$ in practice. Experiments with a prototype implementation, written in Rust, suggest that $L^\#$ is competitive with existing algorithms.

Keywords: $L^\#$ algorithm · active automata learning · Mealy machine · apartness relation · adaptive distinguishing sequence · observation tree · conformance testing

1 Introduction

In 1987, Dana Angluin published a seminal paper [5], in which she showed that the class of regular languages can be learned efficiently using queries. In Angluin's approach of a *minimally adequate teacher (MAT)*, learning is viewed as a game in which a learner has to infer a deterministic finite automaton (DFA) for an unknown regular language L by asking queries to a teacher. The learner may pose two types of queries: "Is the word w in L?" (*membership queries*), and "Is the language recognized by DFA H equal to L?" (*equivalence queries*). In case of a *no* answer to an equivalence query, the teacher supplies a counterexample that distinguishes hypothesis H from L. The L^* algorithm proposed by Angluin [5] is able to learn L by asking a polynomial number of membership and equivalence queries (polynomial in the size of the corresponding canonical DFA).

Angluin's approach triggered a lot of subsequent research on active automata learning and has numerous applications in the area of software and hardware analysis, for instance for generating conformance test suites of software components [28], finding bugs in implementations of security-critical protocols [22,23,21],

* Research supported by NWO TOP project 612.001.852 "Grey-box learning of Interfaces for Refactoring Legacy Software (GIRLS)".

learning interfaces of classes in software libraries [33], inferring interface protocols of legacy software components [8], and checking that a legacy component and a refactored implementation have the same behavior [56]. We refer to [64,34] for surveys and further references.

Since 1987, major improvements of the original L^* algorithm have been proposed, for instance by [53,54,38,41,57,35,45,51,32,37,25]. Yet, all these improvements are variations of L^* in the sense that they approximate the Nerode congruence by means of refinement. Isberner [36] shows that these *descendants* of L^* can be described in a single, general framework.[1]

Variations of L^* have also been used as a basis for learning extensions of DFAs such as Mealy machines [48], I/O automata [2], non-deterministic automata [16], alternating automata [6], register automata [1,17], nominal automata [46], symbolic automata [40,7], weighted automata [14,11,30], Mealy machines with timers [65], visibly pushdown automata [36], and categorical generalisations of automata [63,29,12,18]. It is fair to say that L^*-like algorithms completely dominate the research area of active automata learning.

In this paper we present $L^{\#}$, a fresh approach to automata learning that differs from L^* and its descendants. Instead of focusing on equivalence of observations, $L^{\#}$ tries to establish *apartness*, a constructive form of inequality [62,26]. The notion of apartness is standard in constructive real analysis and goes back to Brouwer, with Heyting giving an axiomatic treatment in [31]. This change in perspective has several key consequences, developed and presented in this paper:

- $L^{\#}$ does not maintain auxiliary data structures such as observation tables or discrimination trees, but operates directly on the observation tree. This tree is a partial Mealy machine itself, and is very close to an actual hypothesis that can be submitted to the teacher. As a result, our algorithm is *simple*.
- The asymptotic query complexity of $L^{\#}$ is $\mathcal{O}(kn^2 + n \log m)$ and the asymptotic symbol complexity[2] is $\mathcal{O}(kmn^2 + nm \log m)$. Here k is the number of input symbols, n is the number of states, and m is the length of the longest counterexample. These are the *same asymptotic complexities* as the best existing (L^*-like) learning algorithms [53,54,32,37,36,25].
- The use of observation trees as primary data structure makes it easy to *integrate concepts from conformance testing to improve the performance* of $L^{\#}$. In particular, adaptive distinguishing sequences [39], which we can compute directly from the observation tree, turn out to be an effective boost in practice, even if their use does not affect asymptotic complexities. Through $L^{\#}$ testing and learning become even more intertwined [13,4].

[1] Except for the ZQ algorithm of [51], which was developed independently, and the ADT algorithm of [25], that was developed later and uses adaptive distinguishing sequences which are not covered in Isberner's framework.

[2] The symbol complexity is the number of input symbols required to learn an automaton. This is a relevant measure for practical learning scenarios, where the total time needed to learn a model is proportional to the number of input symbols.

– Experiments on benchmarks of [47], with a *prototype implementation* written in Rust, suggest that $L^{\#}$ is competitive with existing, highly optimized algorithms implemented in LearnLib [52].

Related work. A few other algorithms have been proposed that follow a different approach than L^*. Meinke [43,44] developed a dual approach where, instead of starting with a maximally coarse approximating relation and refining it during learning, one starts with a maximally fine relation and coarsens it by merging equivalence classes. Although Meinke reports superior performance in the application to learning-based testing, these algorithms have exponential worst-case query complexities. Using ideas from [54], Groz et al. [27] use a combination of homing sequences and characterization sets to develop an algorithm for active model learning that does not require the ability to reset the system. Via an extensive experimental evaluation involving benchmarks from [47] they show that the performance of their algorithm is competitive with the L^* descendant of [57], but there can be huge differences in the performance of their algorithm for models that are similar in size and structure. Several authors have explored the use of SAT and SMT solvers for obtaining learning algorithms, see for instance [50,59], but these approaches suffer from fundamental scalability problems. In a recent paper, Soucha & Bogdanov [61] outline an active learning algorithm which also takes the observation tree as the primary data structure, and use results from conformance testing to speed up learning. They report that an implementation of their approach outperforms standard learning algorithms like L^*, but they have no explicit apartness relation and associated theoretical framework. It is precisely this theoretical underpinning which allowed us to establish complexity and correctness results, and define efficient procedures for counterexample processing and computing adaptive distinguishing sequences.

In the present paper, we first define partial Mealy machines, observation trees, and apartness (Section 2). Then, we present the full $L^{\#}$ algorithm (Section 3) and benchmark our prototype implementation (Section 4). The proofs of all theorems can be found in Appendix A and complete benchmark results in Appendix B.

2 Partial Mealy Machines and Apartness

The $L^{\#}$ algorithm learns a hidden (complete) Mealy machine, and its primary data structure 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 mean that f is defined on x, that is, $\exists y \in Y\colon f(x) = y$, and conversely write $f(x){\uparrow}$ if f is undefined for x. Often, we 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 we have $(g \circ f)(x){\downarrow}$ iff $f(x){\downarrow}$ and $g(f(x)){\downarrow}$. There is a partial order on $X \rightharpoonup Y$ 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 $f(x) = g(x)$.

Throughout this paper, we fix a finite set I of *inputs* and a set O of *outputs*.

Definition 2.1. *A* ***Mealy machine*** *is a tuple* $\mathcal{M} = (Q, q_0, \delta, \lambda)$*, where*
- Q *is a finite set of* ***states*** *and* $q_0 \in Q$ *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 disambiguate 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 call* $\mathcal{M}$ ***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. *The semantics of a state* q *is a map* $[\![q]\!] \colon I^* \rightharpoonup O^*$ *defined by* $[\![q]\!](\sigma) = \lambda(q, \sigma)$*. States* q, q' *in possibly different Mealy machines are* ***equivalent****, written* $q \approx q'$*, if* $[\![q]\!] = [\![q']\!]$*. Mealy machines* $\mathcal{M}$ *and* $\mathcal{N}$ *are* ***equivalent*** *if their respective initial states are equivalent:* $q_0^{\mathcal{M}} \approx q_0^{\mathcal{N}}$*.*

In our learning setting, an *undefined* value in the partial transition map represents lack of knowledge. We consider maps between Mealy machines that preserve existing transitions, but possibly extend the knowledge of transitions:

Definition 2.3. *For Mealy machines* $\mathcal{M}$ *and* $\mathcal{N}$*, a* ***functional simulation*** $f \colon \mathcal{M} \to \mathcal{N}$ *is a map* $f \colon Q^{\mathcal{M}} \to Q^{\mathcal{N}}$ *with*

$$f(q_0^{\mathcal{M}}) = q_0^{\mathcal{N}} \qquad \text{and} \qquad q \xrightarrow{i/o} q' \text{ implies } f(q) \xrightarrow{i/o} f(q').$$

Intuitively, a functional simulation preserves transitions. In the literature, a functional simulation is also called *refinement mapping* [3].

Lemma 2.4. *For a functional simulation* $f \colon \mathcal{M} \to \mathcal{N}$ *and* $q \in Q^{\mathcal{M}}$*, we have* $[\![q]\!] \sqsubseteq [\![f(q)]\!]$*.*

For a given machine $\mathcal{M}$, an observation tree is simply a Mealy machine itself which represents the inputs and outputs we have observed so far during learning. Using functional simulations, we define it formally as follows.

Definition 2.5 ((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* $f \colon \mathcal{T} \to \mathcal{M}$*.*

Figure 1 shows an observation tree for the Mealy machine displayed on the right. The functional simulation f is indicated via coloring of the states.

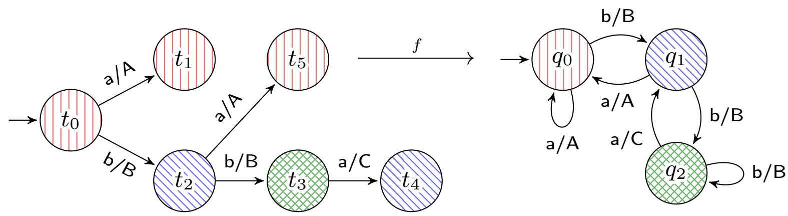


Fig. 1: An observation tree (left) for a Mealy machine (right).

By performing output and equivalence queries, the learner can build an observation tree for the unknown Mealy machine $\mathcal{M}$ of the teacher. However, the learner does not know the functional simulation. Nevertheless, by analysis of the observation tree, the learner may infer that certain states in the tree cannot have the same color, that is, they cannot be mapped to same states of $\mathcal{M}$ by a functional simulation. In this analysis, the concept of *apartness*, a constructive form of inequality, plays a crucial role [62,26]. A similar concept has previously been studied in the context of automata learning under the name *inequivalence constraints* in work on passive learning of DFAs, see for instance [15,24].

Definition 2.6. *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 $[\![q]\!](\sigma)\downarrow$, $[\![p]\!](\sigma)\downarrow$, and $[\![q]\!](\sigma) \neq [\![p]\!](\sigma)$. 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. A witness is also called *separating sequence* [60]. For the observation tree of Figure 1 we may derive the following apartness pairs and corresponding witnesses:

$$a \vdash t_0 \# t_3 \qquad a \vdash t_2 \# t_3 \qquad b\ a \vdash t_0 \# t_2$$

The apartness of states $q \# p$ expresses that there is a conflict in their semantics, and consequently, apart states can never be identified by a functional simulation:

Lemma 2.7. *For a functional simulation $f : \mathcal{T} \to \mathcal{M}$,*

$$q \# p \text{ in } \mathcal{T} \qquad \Longrightarrow \qquad f(q) \not\approx f(p) \text{ in } \mathcal{M} \qquad \text{for all } q, p \in Q^{\mathcal{T}}.$$

Thus, whenever states are apart in the observation tree $\mathcal{T}$, the learner knows that these are distinct states in the hidden Mealy machine $\mathcal{M}$.

The apartness relation satisfies a weaker version of *co-transitivity*, stating that if $\sigma \vdash r \# r'$ and q has the transitions for σ, then q must be apart from at least one of r and r', or maybe even both:

Lemma 2.8 (Weak co-transitivity). *In every Mealy machine $\mathcal{M}$,*

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

We use the weak co-transitivity property during learning. For instance in Fig. 1, by posing the output query aba, consisting of the access sequence for t_1 concatenated with the witness ba for $t_0 \# t_2$, co-transitivity ensures that $t_0 \# t_1$ or $t_2 \# t_1$. By inspecting the outputs, the learner may conclude that $t_2 \# t_1$.

3 Learning Algorithm

The task solved by $L^{\#}$ is to find a strategy for the learner in the following game:

Definition 3.1. *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 $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \in O^*$.*[3]
EquivQuery($\mathcal{H}$): *For a complete Mealy machine $\mathcal{H}$, the teacher replies* `yes` *if $\mathcal{H} \approx \mathcal{M}$ or* `no`*, providing some $\sigma \in I^*$ with $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \neq \lambda^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$.*

Our $L^{\#}$ algorithm operates on an observation tree $\mathcal{T} = (Q, q_0, \delta, \lambda)$ for the unknown complete Mealy machine $\mathcal{M}$, where $\mathcal{T}$ contains the results of all output and equivalence queries so far. An observation tree is similar to the *cache* which is commonly used in implementations of L^*-based learning algorithms to store the answers to previously asked queries, avoiding duplicates [10,42]. But whereas for L^*-based learning algorithms the cache is an auxiliary data structure and only used for efficiency reasons, it is a first-class citizen in $L^{\#}$.

Remark 3.2. The learner has no information about the teacher's hidden Mealy machine. In particular, whenever we write #, we always refer to the apartness relation *on the observation tree $\mathcal{T}$*.

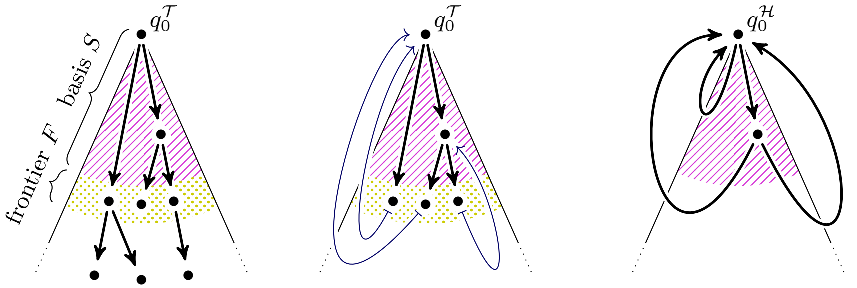
The observation tree is structured in a very similar way as Dijkstra's shortest path algorithm [19] structures a graph. Recall that during the execution of Dijkstra's algorithm 'the nodes are subdivided into three sets' [19]:

1. the nodes S to which a shortest path from the initial node is known. S initially only contains the initial node and grows from there.
2. the nodes F from which the next node to be added to S will be selected.
3. the remaining nodes.

This scheme adapts to the observation tree as follows and is visualized in Fig. 2a.

1. The states $S \subseteq Q^{\mathcal{T}}$, which already have been fully identified, i.e. the learner found out that these must represent distinct states in the teacher's hidden Mealy machine. We call S the *basis*. Initially, $S := \{q_0^{\mathcal{T}}\}$, and throughout the execution S forms a subtree of $\mathcal{T}$ and all states in S are pairwise apart: $\forall p, q \in S, p \neq q \colon p \# q$.

[3] In fact, later on we will assume that the teacher responds to slightly more general output queries to enable the use of *adaptive distinguishing sequences*, see Section 3.5.



(a) $\mathcal{T}$, S, and F (b) A choice $h\colon F \to S$ (c) Hypothesis $\mathcal{H}$ for h

Fig. 2: From the observation tree to the hypothesis ($|I| = 2$)

2. the *frontier* $F \subseteq Q^{\mathcal{T}}$, from which the next node to be added to S is chosen. 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 $Q \setminus (S \cup F)$.

Initially, $\mathcal{T}$ consists of only an initial state $q_0^{\mathcal{T}}$ with no transitions. For every OutputQuery(σ) during the execution, the input $\sigma \in I^*$ and the corresponding response of type O^* is added automatically to the observation tree $\mathcal{T}$, and similarly every negative response to a EquivQuery leads to new states and transitions in the observation tree. With every extension $\mathcal{T}'$ of the observation tree $\mathcal{T}$, the apartness relation can only grow: whenever $p \# q$ in $\mathcal{T}$, then still $p \# q$ in $\mathcal{T}'$. Thus, along the learning game, $\mathcal{T}$ and $\#$ grow steadily:

Assumption 3.3 *We implicitly require that via output and equivalence queries, the observation tree $\mathcal{T}$ and the basis S are gradually extended, with the frontier F automatically moving along while S grows.*

3.1 Hypothesis construction

At almost any point during the learning game, the learner can come up with a hypothesis $\mathcal{H}$ based on the knowledge in the observation tree $\mathcal{T}$. Since the basis S contains the states already discovered, the set of states of such a hypothesis is simply set to $Q^{\mathcal{H}} := S$, and it contains every transition between basis states (in $\mathcal{T}$). The hypothesis must also reflect the transitions in $\mathcal{T}$ that leave the basis S, i.e. the transitions to the frontier. Those are resolved by finding for every frontier state a base state, for which the learner conjectures that they are equivalent states in the hidden Mealy machine. This choice boils down to a map $h\colon F \to S$ ($\mapsto$ in Fig. 2b). Then, a transition $q \xrightarrow{i/o} p$ in $\mathcal{T}$ with $q \in S$, $p \in F$ leads to a transition $q \xrightarrow{i/o} h(p)$ in $\mathcal{H}$ (Fig. 2c). These ideas are formally defined as follows.

Definition 3.4. *Let $\mathcal{T}$ be an observation tree with basis S and frontier F.*

1. *A Mealy machine $\mathcal{H}$* ***contains the basis*** *if $Q^{\mathcal{H}} = S$ and $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q)) = q$ for all $q \in S$.*

2. *A **hypothesis** is a complete Mealy machine $\mathcal{H}$ containing the basis such that $q \xrightarrow{i/o'} p'$ in $\mathcal{H}$ ($q \in S$) and $q \xrightarrow{i/o} p$ in $\mathcal{T}$ imply $o = o'$ and $\neg(p \# p')$ (in $\mathcal{T}$).*
3. *A hypothesis $\mathcal{H}$ is **consistent** if there is a functional simulation $f\colon \mathcal{T} \to \mathcal{H}$.*
4. *For a Mealy machine $\mathcal{H}$ containing the basis, an input sequence $\sigma \in I^*$ is said to **lead to a conflict** if $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) \# \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$ (in $\mathcal{T}$).*

Intuitively, the first three notions describe how confident we are in the correctness of the 'back loops' in $\mathcal{H}$ obtained from a choice $h\colon F \to S$. Notion 1 does not provide any warranty, notion 2 asserts that $\neg(q \# h(q))$ for all $q \in F$, and notion 3 (by definition) means that $\mathcal{T}$ is an observation tree for $\mathcal{H}$, that is, all observations so far are consistent with the hypothesis $\mathcal{H}$. The learner can verify the consistency of a hypothesis without querying the teacher (algorithm is in Section 3.3 below). The existence and uniqueness of a hypothesis are related to criteria on $\mathcal{T}$:

Definition 3.5. *In an observation tree $\mathcal{T}$, a state in F is 1. **isolated** if it is apart from all states in S and 2. is **identified** if it is apart from all states in S except one. 3. The basis S is **complete** if each state in S has a transition for each input in I.*

Lemma 3.6. *For an observation tree $\mathcal{T}$, if F has no isolated states then there exists a hypothesis $\mathcal{H}$ for $\mathcal{T}$. If S is complete and all states in F are identified then the hypothesis is unique.*

With a growing observation tree $\mathcal{T}$, the hidden Mealy machine is found as soon as the basis is big enough:

Theorem 3.7. *Suppose $\mathcal{T}$ is an observation tree for a (hidden) Mealy machine $\mathcal{M}$ such that S is complete, all states in F are identified, and $|S|$ is the number of equivalence classes of $\approx^{\mathcal{M}}$. Then $\mathcal{H} \approx \mathcal{M}$ for the unique hypothesis $\mathcal{H}$.*

The theorem itself is not necessary for the correctness of $L^{\#}$, but guarantees feasibility of learning.

3.2 Main loop of the algorithm

The $L^{\#}$ algorithm is listed in Algorithm 1 in pseudocode. The code uses Dijkstra's guarded command notation [20], which means that the following rules are applied non-deterministically until none of them can be applied anymore:

(R1) If F contains an isolated state, then this means that 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 $\mathsf{access}(q)\ i$ will add the generated i successor, implicitly extending the frontier F.

Algorithm 1 Overall $L^{\#}$ algorithm

```
procedure LSharp
    do q isolated, for some q ∈ F →                                   ▷ Rule (R1)
        S ← S ∪ {q}
    [] δ^T(q, i) ↑, for some q ∈ S, i ∈ I →                           ▷ Rule (R2)
        OutputQuery(access(q) i)
    [] ¬(q # r), ¬(q # r'), for some q ∈ F, r, r' ∈ S, r ≠ r' →       ▷ Rule (R3)
        σ ← witness of r # r'
        OutputQuery(access(q) σ)
    [] F has no isolated states and basis S is complete →             ▷ Rule (R4)
        H ← BuildHypothesis
        (b, σ) ← CheckConsistency(H)
        if b = yes then
            (b, ρ) ← EquivQuery(H)
            if b = yes then: return H
            else: σ ← shortest prefix of ρ such that δ^H(q_0^H, σ) # δ^T(q_0^T, σ) (in T)
        end if
        ProcCounterEx(H, σ)
    end do
end procedure
```

(R3) When $q \in F$ is a state in the frontier that is not yet identified, then there are at least two states in S that are not apart from q. In this 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 (Lemma 2.8).

(R4) When F has no isolated states and S is complete, BuildHypothesis picks a hypothesis $\mathcal{H}$ (at least one exists Lemma 3.6). If $\mathcal{H}$ is not consistent with observation tree $\mathcal{T}$ we get a conflict σ for free. Otherwise, we pose an equivalence query for $\mathcal{H}$. If the hypothesis is correct, $L^{\#}$ terminates, and otherwise we obtain a counterexample ρ. The counterexample decomposes into two words $\sigma\eta$, where σ leads to a conflict and η witnesses it. The conflict σ means that one of the frontier states was merged with an apart basis state in $\mathcal{H}$, causing a wrong transition in $\mathcal{H}$. Since σ can be very long, the task of ProcCounterEx(σ) is to shorten σ until we know which frontier state caused the conflict. So after ProcCounterEx, $\mathcal{H}$ is not a hypothesis for the updated $\mathcal{T}$ anymore.

We will show the correctness of $L^{\#}$ in a top-down approach discussing the subroutines later and only assuming now that:

1. BuildHypothesis picks one of the possible hypotheses (Lemma 3.6)
2. CheckConsistency($\mathcal{H}$) tells if there is a functional simulation $\mathcal{T} \to \mathcal{H}$, and if not, provides $\sigma \in I^*$ leading to a conflict (Lemma 3.10 below).
3. If $\mathcal{H}$ contains the basis and σ leads to a conflict, then ProcCounterEx($\mathcal{H}, \sigma$), extends $\mathcal{T}$ such that $\mathcal{H}$ is not a hypothesis anymore (Lemma 3.11 below).

Whenever the algorithm terminates, the learner has found the correct model. Therefore, correctness amounts to showing termination. The rough idea is that each rule will let S, F, or $\#$ restricted to $S \times F$ grow, and each of these sets are bounded by the hidden Mealy machine $\mathcal{M}$. We define the norm $N(\mathcal{T})$ by

$$\frac{|S| \cdot (|S|+1)}{2} \;+\; |\{(q,i) \in S \times I \mid \delta^{\mathcal{T}}(q,i){\downarrow}\}| \;+\; |\{(q,q') \in S \times F \mid q \# q'\}| \quad (1)$$

The first summand increases whenever a state is moved from F to S (R1); it is quadratic in $|S|$ because (R1) reduces the third summand. The second summand records the progress achieved by extending the frontier (R2). The third summand counts how much the states in the frontier are identified (R3). Rule (R4) extends the apartness relation, leading to an increase of the third summand.

Theorem 3.8. *Every rule application in $L^{\#}$ increases the norm $N(\mathcal{T})$ in* (1).

The norm $N(\mathcal{T})$ and therefore also the number of rule applications is bounded:

Theorem 3.9. *If $\mathcal{T}$ is an observation tree for $\mathcal{M}$ with n equivalence classes of states and $|I| = k$, then $N(\mathcal{T}) \leq \frac{1}{2} \cdot n \cdot (n+1) + kn + (n-1)(kn+1) \in \mathcal{O}(kn^2)$.*

At any point of execution, either rule (R1), (R2), or (R4) is applicable, so $L^{\#}$ never blocks. As soon as the norm $N(\mathcal{T})$ hits the bound, the only applicable rule is rule (R4) with the teacher accepting the hypothesis. Thus, the correct Mealy machine is learned within $\mathcal{O}(k \cdot n^2)$ rule applications. The complexity in terms of the input parameters is studied in Section 3.6.

We now continue defining the subroutines and proving them correct.

3.3 Consistency checking

A hypothesis $\mathcal{H}$ is not necessarily *consistent* with $\mathcal{T}$, in the sense of a functional simulation $\mathcal{T} \to \mathcal{H}$. Via a breadth-first search of the Cartesian product of $\mathcal{T}$ and $\mathcal{H}$ (Algorithm 2), we may check in time linear in the size of $\mathcal{T}$ whether a functional simulation $\mathcal{T} \to \mathcal{H}$ exists. In the negative case, we obtain $\sigma \in I^*$ leading to a conflict without any equivalence or output query to the teacher needed. Thus, this is also called 'counterexample milking' [10].

Lemma 3.10. *Algorithm 2 terminates and is correct, that is, if $\mathcal{H}$ is a hypothesis for $\mathcal{T}$ with a complete basis, then* CheckConsistency($\mathcal{H}$)

1. *returns* `yes`*, if $\mathcal{H}$ is consistent,*
2. *returns* `no` *and $\rho \in I^*$, if ρ leads to a conflict ($\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \# \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \rho)$ in $\mathcal{T}$).*

3.4 Counterexample processing

The L^* algorithm [5] performs $\mathcal{O}(m)$ queries to analyze a counterexample of length m. So if a teacher returns really long counterexamples, their analysis will dominate the learning process. Rivest & Schapire [53,54] improve counterexample

Algorithm 2 Check if hypothesis $\mathcal{H}$ is consistent with observation tree $\mathcal{T}$

procedure CheckConsistency($\mathcal{H}$)
  $Q \leftarrow$ `new` $queue \subseteq S \times S$
  $enqueue(Q, (q_0^{\mathcal{T}}, q_0^{\mathcal{H}}))$
  while $(q, r) \leftarrow dequeue(Q)$
    if $q \# r$ **then: return** `no`: $\mathsf{access}(q)$
    for all $q \xrightarrow{i/o} p$ in $\mathcal{T}$ **do**
      $enqueue(Q, (p, \delta^{\mathcal{H}}(r, i)))$
    end for
  end while
  return `yes`
end procedure

analysis of L^* using binary search, requiring only $\mathcal{O}(\log m)$ queries. A similar trick is applied in $L^{\#}$.

Suppose σ leads to a conflict $q \# r$ for $q = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$ and $r = \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$. Then, ProcCounterEx($\sigma$) (Algorithm 3) extends $\mathcal{T}$ such that $\mathcal{H}$ will never be a hypothesis for $\mathcal{T}$ again.

If $r \in S \cup F$, then the conflict $q \# r$ is obvious and $\mathcal{H}$ is not a hypothesis again. If otherwise $r \notin S \cup F$, the binary search will successively reduce the number of transitions of σ outside $S \cup F$ by a factor of 2 until we reach the above base case $S \cup F$. Let $\sigma_1 \sigma_2 := \sigma$ such that the run of σ_1 in $\mathcal{T}$ ends halfway between the frontier and r. By an additional output query, the binary search checks whether already σ_1 leads to a conflict. In the two cases, we can either avoid σ_1 or σ_2, so we reduce the number of transitions outside $S \cup F$ to half the amount. The precise argument is in:

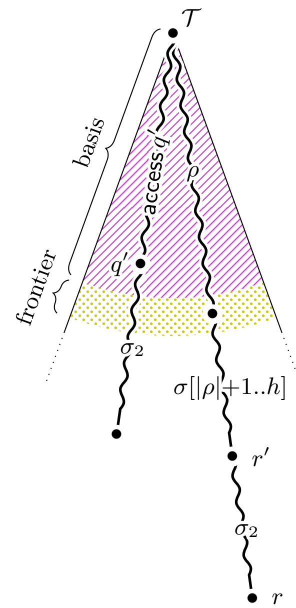
Lemma 3.11. *Suppose basis S is complete, $\mathcal{H}$ is a complete Mealy machine containing the basis, and $\sigma \in I^*$ leads to a conflict. Then* ProcCounterEx($\mathcal{H}, \sigma$) *terminates, performs at most $\mathcal{O}(\log_2 |\sigma|)$ output queries and is correct: upon termination, the machine $\mathcal{H}$ is not a hypothesis for $\mathcal{T}$ anymore.*

3.5 Adaptive distinguishing sequences

As an optimization in practice, we may extend the rules (R2) and (R3) by incorporating *adaptive distinguishing sequences* (Ads) into the respective output queries. Adaptive distinguishing sequences, which are commonly used in the area of conformance testing [39], are input sequences where the choice of an input may depend on the outputs received in response to previous inputs. Thus, strictly speaking, an ADS is a decision graph rather than a sequence. This mild extension of the learning framework reflects the actual black box behaviour of Mealy machines: for every input in I sent to the hidden Mealy machine, the learner observes the output O before sending the next input symbol. Use of adaptive distinguishing sequences may reduce the number of output queries that are required for the identification of frontier states.

Algorithm 3 Processing σ that leads to a conflict, i.e. $\delta^{\mathcal{H}}(q_0, \sigma) \# \delta^{\mathcal{T}}(q_0, \sigma)$

procedure ProcCounterEx($\mathcal{H}$, $\sigma \in I^*$)
  $q \leftarrow \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$
  $r \leftarrow \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$
  if $r \in S \cup F$ **then**
    return
  else
    $\rho \leftarrow$ unique prefix of σ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \in F$
    $h \leftarrow \lfloor \frac{|\rho|+|\sigma|}{2} \rfloor$
    $\sigma_1 \leftarrow \sigma[1..h]$
    $\sigma_2 \leftarrow \sigma[h+1..|\sigma|]$
    $q' \leftarrow \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1)$
    $r' \leftarrow \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma_1)$
    $\eta \leftarrow$ witness for $q \# r$
    OutputQuery($\mathsf{access}(q')\ \sigma_2\ \eta$)
    if $q' \# r'$ **then**
      ProcCounterEx ($\mathcal{H}$, σ_1)
    else
      ProcCounterEx ($\mathcal{H}$, $\mathsf{access}(q')\ \sigma_2$)
    end if
  end if
end procedure

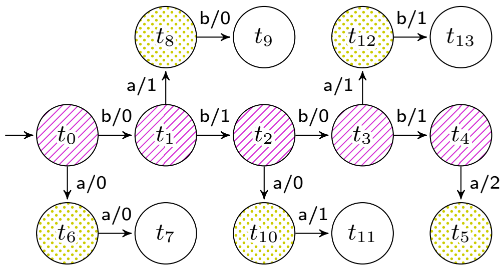


As an example, consider the observation tree of Figure 3(left). The basis for this tree consists of 5 states, which are pairwise apart (separating sequences are a, ab and aa). Frontier states can be identified by the *single* adaptive sequence of Figure 3(right). The ADS starts with input a. If the response is 2 we have identified our frontier state as t_4. If the response is 0 then the frontier state is either t_0 or t_2, and we may identify the state with a subsequent input a. Similarly, if the response is 1 then the frontier state is either t_1 or t_3, and we may identify the state by a subsequent input b. We can therefore identify (or isolate) frontier state t_5 with a single (extended) output query that starts with the access sequence for t_5 ($bbbba$) followed by the ADS of Figure 3(right). If we used separating sequences, we would need at least 2 output queries.

In the setting of $L^{\#}$, we can directly compute an optimal ADS from the current observation tree. To this end, we recursively define an *expected reward* function E, which sends a set $U \subseteq Q^{\mathcal{T}}$ of states to the maximal expected number of apartness pairs (in the absence of unexpected outputs).

$$E(U) = \max_{i \in inp(U)} \left(\sum_{o \in O} \frac{|U \xrightarrow{i/o}| \cdot (|U \xrightarrow{i}| - |U \xrightarrow{i/o}| + E(U \xrightarrow{i/o}))}{|U \xrightarrow{i}|} \right) \quad (2)$$

where $inp(U) := \{i \in I \mid \exists q \in U : \delta^{\mathcal{T}}(q, i)\downarrow\}$, $U \xrightarrow{i} := \{q \in U \mid \delta^{\mathcal{T}}(q, i)\downarrow\}$ and $U \xrightarrow{i/o} := \{q' \in Q^{\mathcal{T}} \mid \exists q \in U : q \xrightarrow{i/o} q'\}$. We define the maximum over the empty set to be 0. Then $\text{Ads}(U)$ is the decision tree constructed as follows:



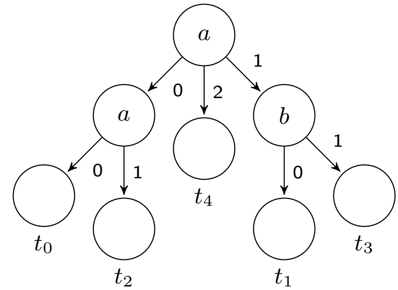


Fig. 3: An observation tree (left) and an ADS for its basis (right)

- If $U \xrightarrow{i} = \emptyset$ then $\textsc{Ads}(U)$ consists of a single node U without a label.
- If $U \xrightarrow{i} \neq \emptyset$ then $\textsc{Ads}(U)$ is constructed by choosing an input i that witnesses the maximum $E(U)$, creating a node U with label i, and, for each output o with $U \xrightarrow{i/o} \neq \emptyset$, adding an o-transition to $ADS(U \xrightarrow{i/o})$.

For the observation tree of Figure 3(left) we may compute $E(\{t_0, \ldots, t_4\}) = 4$ and obtain the decision tree of Figure 3(right) as ADS. Running the ADS from state t_5 will create 4 new apartness pairs with basis states (or 5 in case an unexpected output occurs, e.g. $a(1)b(2)$).

Proposition 3.12. *Define $L^{\#}_{\textsc{Ads}}$ by replacing the output queries in $L^{\#}$ with*

(R2') $\textsc{OutputQuery}(\mathsf{access}(q)\ i\ \textsc{Ads}(S))$ *in (R2) and*
(R3') $\textsc{OutputQuery}(\mathsf{access}(q)\ \textsc{Ads}(\{b \in S \mid \neg(b \# q)\}))$ *in (R3).*

Then, $L^{\#}_{\textsc{Ads}}$ lets the norm $N(\mathcal{T})$ grow for each rule application and thus is correct.

3.6 Complexity

Since equivalence queries are costly in practice and since processing of long counterexamples of length m requires $\mathcal{O}(\log m)$ output queries, it makes sense to postpone equivalence queries as long as possible:

Definition 3.13. ***Strategic*** *$L^{\#}$ (resp. $L^{\#}_{\textsc{Ads}}$) is the special case of Algorithm 1 where rule (R4) is only applied if none of the other rules is applicable.*

Then we obtain the following query complexity for the $L^{\#}$ algorithm.

Theorem 3.14. *Strategic $L^{\#}$ (resp. $L^{\#}_{\textsc{Ads}}$) 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 $L^{\#}$ equals the best known query complexity for active learning algorithms, as achieved by Rivest & Schapire's algorithm [53,54], the observation pack algorithm [32], the TTT algorithm [37,36], and the ADT algorithm [25].

In a black box learning setting in practice, answering an output query for $\sigma \in I^*$ grows linearly with the length σ. Therefore, the (asymptotic) total number of input symbols sent by the learner is also a metric for comparing learning algorithms:

Theorem 3.15. *Let $n \in \mathcal{O}(m)$. Then the strategic $L^\#$ algorithm learns the correct Mealy machine with $\mathcal{O}(kmn^2 + nm \log m)$ input symbols.*

This matches the asymptotic symbol complexity of the best known active learning algorithms. Although ProcCounterEx reduces the length of the sequence leading to the conflict, the witness of the conflict remains of size $\Theta(m)$ in the worst case. This means that we need $\mathcal{O}(m \log m)$ symbols to process a single counterexample and $\mathcal{O}(nm \log m)$ symbols to process all counterexamples.

4 Experimental Evaluation

In the previous sections, we have introduced and discussed the $L^\#$ algorithm. We now present a short experimental evaluation of the algorithm to demonstrate its performance when compared to other state-of-art algorithms. We run two versions of $L^\#$: the base version (Algorithm 1), and the ADS optimised variant ($L^\#_{\text{ADS}}$), and compare these with the (highly optimized) LearnLib[4] implementations of TTT, ADT,[5] and 'RS', by which we refer to L^* with Rivest-Schapire counterexample processing [53,54]. All source-code and data is available online.[6]

Implementing EquivQuery*:* We implement equivalence queries using conformance testing, which also makes output queries. We have fixed the testing tool to Hybrid-ADS[7] [58]. Hybrid-ADS has multiple configuration options, and we have set the state cover mode to "buggy", the number of extra states to check for to 10, the number of infix symbols to 10, and the mode of execution to "random", generating an infinite test-suite. Note that with these settings, the equivalence queries are not exact in general but approximated via random testing.

Data-set and metrics: We use a subset of the models available from the AutomataWiki (see [47]): we learn models for the SSH, TCP, and TLS protocols, alongside the BankCard models. The largest model in this subset has 66 states and 13 input symbols. We record the number of output queries and input symbols used during learning and testing, alongside the number of equivalence queries required to learn each model. An output query is a sequence $\sigma \in I^*$ of $|\sigma|$ input symbols and one *reset* symbol. A reset symbol returns the *system under test* (SUT) to its initial state. So *resets* denotes the number of output queries and *inputs* denotes the total number of symbols sent to the SUT. We believe that these metrics accurately portray the effort required to learn a model.

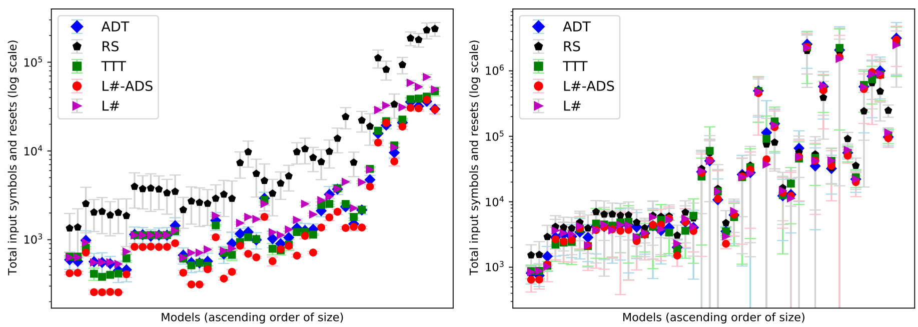
4 Available at https://learnlib.de/

5 The ADT algorithm makes use of some heuristics to guide the learning process, we have selected the "Best-Effort" settings.

6 Available at https://gitlab.science.ru.nl/bharat/automata-lib

7 Available at https://github.com/Jaxan/hybrid-ads

Experiment Set-up: All experiments were run on a Ryzen 3700X processor with 32GB of memory, running Linux. Each experiment refers to completely learning a model of the SUT. Due to the effects of randomization in the equivalence oracle, we repeat each experiment 100 times.



(a) Symbols used during learning phase (b) Symbols used both learning and testing

Fig. 4: Performance plots of the selected learning algorithms (lower is better.)

Results and Discussion Fig. 4a shows the total size of data sent by the learning algorithms via output queries – so both the number and the size of output queries are counted. In order to incorporate the equivalence queries, Fig. 4b shows the total size of data sent to the SUT during learning and testing. Note, in both plots the y-axis is log-scaled. The x-axis indicates the models, sorted in increasing number of states. The bars indicate standard deviation.

We can observe from the learning phase plot (Fig. 4a) that $L^{\#}$ expectedly does not perform better than the TTT and ADT algorithms, while the RS algorithm performs the worst among all four. However, $L^{\#}_{\text{ADS}}$ usually performs better than – or, at least, is competitive with – ADT and TTT. Furthermore, the error bars in the learning phase are very small, indicating that the measurements are stable. Generally, depending on the models a different algorithm is the fastest, but for every model, $L^{\#}_{\text{ADS}}$ is among the fastest, with and without the exclusion of the testing phase.

Fig. 4b presents the total number of input symbols and resets sent to the SUT. All algorithms seem to be very close in performance, which may be explained by the testing phase dominating the process. Indeed, Aslam et al. [8] experimentally demonstrated that it is largely the testing phase which influences learning effort.

The complete benchmark results (in Appendix B) show more detailed information of the learned models, and highlights the smallest number per column and model. We can see that the number of equivalence queries are roughly similar for almost all the algorithms, while $L^{\#}$ seems to perform better for some models in the learning phase.

5 Conclusions and Future Work

We presented $L^{\#}$, a new algorithm for the classical problem of active automata learning. The key idea behind the approach is to focus on establishing *apartness*, or inequivalence of states, instead of approximating equivalence as in L^* and its descendants. Concretely, the table/discrimination tree in L^*-like algorithms is replaced in $L^{\#}$ by an observation tree, together with an apartness relation. This change in perspective leads to a simple but effective algorithm, which reduces the total number of symbols required for learning when compared to state-of-the-art algorithms. In particular, the use of observation trees, which are essentially tree-shaped Mealy machines, enables a modular integration of testing techniques, such as the ADS method, to identify states. Although the asymptotic output query complexity of $L^{\#}$ is $\mathcal{O}(kn^2 + n \log m)$, in our experiments $L^{\#}$ only needs in between kn and $4kn$ output queries (resets) to learn the benchmark models (with $n \leq 66$), which means that on average $L^{\#}$ needs in between 1 and 4 output queries to identify a frontier state.

Of course there are also similarities between $L^{\#}$ and L^*. The basis of $L^{\#}$ is comparable to the top half of the L^* table: both in $L^{\#}$ and in ([54]'s version of) L^* these prefixes induce a spanning tree. The frontier of $L^{\#}$ is comparable to the bottom half of the L^* table. But whereas L^* constructs residual classes of the language, $L^{\#}$ builds an automaton directly from the observation tree. As a consequence, L^* asks redundant queries, and optimizations of L^* try to avoid this redundancy. In contrast, $L^{\#}$ does not even think about asking redundant queries since it operates directly on the observation tree and only poses queries that increase the norm.

There is still much work to do to improve our prototype implementation, to include additional conformance testing algorithms, and to extend the experimental evaluation to a richer set of benchmarks and algorithms. One issue that we need to address is scaling of $L^{\#}$ to bigger models. Our prototype implementation easily learns Mealy machines with hundreds of states, but fails to learn larger models such as the ESM benchmark of [58] (3410 states, 78 inputs) because the observation tree becomes too big ($\approx$25 million nodes will be required for the ESM). We see several ways to address this issue, e.g., pruning the observation tree, only keeping short ADSs to separate the basis states, storing parts of the tree on disk, distributing the tree over multiple processors (parallelizing the learning process), and using existing platforms for big graph processing [55].

Aslam et al. [9] report on experiments in which active learning techniques are applied to 202 industrial software components from ASML. Out of these, interface protocols could be successfully derived for 134 components (within a give time bound). One of the main conclusions of the study is that the equivalence checking phase (i.e. conformance testing of hypothesis models) is the bottleneck for scalability in industry. We believe that a tighter integration of learning and testing, as enabled by $L^{\#}$, will be key to address this challenging problem.

It will be interesting to extend $L^{\#}$ to richer frameworks such as register automata, symbolic automata and weighted automata. In fact, we discovered $L^{\#}$ while working on a grey-box learning algorithm for symbolic automata.

References

1. Aarts, F., Heidarian, F., Kuppens, H., Olsen, P., Vaandrager, F.: Automata learning through counterexample-guided abstraction refinement. In: Giannakopoulou, D., Méry, D. (eds.) 18th International Symposium on Formal Methods (FM 2012), Paris, France, August 27-31, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7436, pp. 10–27. Springer (Aug 2012), http://dx.doi.org/10.1007/978-3-642-32759-9_4
2. Aarts, F., Vaandrager, F.: Learning I/O automata. In: Gastin, P., Laroussinie, F. (eds.) 21st International Conference on Concurrency Theory (CONCUR), Paris, France, August 31st - September 3rd, 2010, Proceedings. Lecture Notes in Computer Science, vol. 6269, pp. 71–85. Springer (2010)
3. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991)
4. Aichernig, B.K., Mostowski, W., Mousavi, M.R., Tappler, M., Taromirad, M.: Model learning and model-based testing. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. Lecture Notes in Computer Science, vol. 11026, pp. 74–100. Springer (2018)
5. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
6. Angluin, D., Eisenstat, S., Fisman, D.: Learning regular languages via alternating automata. In: IJCAI. pp. 3308–3314. AAAI Press (2015)
7. Argyros, G., D'Antoni, L.: The learnability of symbolic automata. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 427–445. Springer (2018), https://doi.org/10.1007/978-3-319-96145-3_23
8. Aslam, K., Cleophas, L., Schiffelers, R.R.H., van den Brand, M.: Interface protocol inference to aid understanding legacy software components. Softw. Syst. Model. **19**(6), 1519–1540 (2020), https://doi.org/10.1007/s10270-020-00809-2
9. Aslam, K., Luo, Y., Schiffelers, R.R.H., van den Brand, M.: Interface protocol inference to aid understanding legacy software components. In: Hebig, R., Berger, T. (eds.) Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018. CEUR Workshop Proceedings, vol. 2245, pp. 6–11. CEUR-WS.org (2018)
10. Balcázar, J.L., Díaz, J., Gavaldà, R.: Algorithms for learning finite automata from queries: A unified view. In: Du, D., Ko, K. (eds.) Advances in Algorithms, Languages, and Complexity - In Honor of Ronald V. Book. pp. 53–72. Kluwer (1997)
11. Balle, B., Mohri, M.: Learning weighted automata. In: CAI. Lecture Notes in Computer Science, vol. 9270, pp. 1–21. Springer (2015)
12. Barlocco, S., Kupke, C., Rot, J.: Coalgebra learning via duality. In: FoSSaCS. Lecture Notes in Computer Science, vol. 11425, pp. 62–79. Springer (2019)
13. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) Fundamental Approaches to Software Engineering, 8th International

Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3442, pp. 175–189. Springer (2005)
14. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. SIAM J. Comput. **25**(6), 1268–1280 (Dec 1996). https://doi.org/10.1137/S009753979326091X, https://doi.org/10.1137/S009753979326091X
15. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Computers **21**(6), 592–597 (1972), https://doi.org/10.1109/TC.1972.5009015
16. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style learning of NFA. In: IJCAI. pp. 1004–1009 (2009)
17. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Formal Asp. Comput. **28**(2), 233–263 (2016)
18. Colcombet, T., Petrisan, D., Stabile, R.: Learning automata and transducers: A categorical approach. In: CSL. LIPIcs, vol. 183, pp. 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
19. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (Dec 1959), https://doi.org/10.1007/BF01386390
20. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (Aug 1975), https://doi.org/10.1145/360933.360975
21. Fiterău-Broştean, P., Howar, F.: Learning-based testing the sliding window behavior of TCP implementations. *in* FMICS, LNCS **10471**, 185–200 (2017)
22. Fiterău-Broştean, P., Janssen, R., Vaandrager, F.: Combining model learning and model checking to analyze TCP implementations. *in* CAV, LNCS **9780**, 454–471 (2016)
23. Fiterău-Broştean, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F., Verleg, P.: Model learning and model checking of SSH implementations. In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 142–151. SPIN 2017, ACM, New York, NY, USA (2017)
24. Florêncio, C.C., Verwer, S.: Regular inference as vertex coloring. Theor. Comput. Sci. **558**, 18–34 (2014), https://doi.org/10.1016/j.tcs.2014.09.023
25. Frohme, M.T.: Active automata learning with adaptive distinguishing sequences. CoRR **abs/1902.01139** (2019), http://arxiv.org/abs/1902.01139
26. Geuvers, H., Jacobs, B.: Relating apartness and bisimulation. CoRR **abs/2002.02512** (2020), https://arxiv.org/abs/2002.02512
27. Groz, R., Brémond, N., da Silva Simão, A., Oriat, C.: hW-inference: A heuristic approach to retrieve models through black box testing. J. Syst. Softw. **159** (2020), https://doi.org/10.1016/j.jss.2019.110426
28. Hagerer, A., Margaria, T., Niese, O., Steffen, B., Brune, G., Ide, H.D.: Efficient regression testing of CTI-systems: Testing a complex call-center solution. Annual review of communication, Int.Engineering Consortium (IEC) **55**, 1033–1040 (2001)
29. Heerdt, G.v.: CALF: Categorical Automata Learning Framework. Phd thesis, University College London (Oct 2020)
30. Heerdt, G.v., Kupke, C., Rot, J., Silva, A.: Learning weighted automata over principal ideal domains. In: Goubault-Larrecq, J., König, B. (eds.) Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings. vol. 12077, pp. 602–621. Springer (2020), https://doi.org/10.1007/978-3-030-45231-5_31

31. Heyting, A.: Zur intuitionistischen Axiomatik der projektiven Geometrie. Mathematische Annalen **98**, 491–538 (1927)
32. Howar, F.: Active learning of interface programs. Ph.D. thesis, University of Dortmund (Jun 2012)
33. Howar, F., Isberner, M., Steffen, B., Bauer, O., Jonsson, B.: Inferring semantic interfaces of data structures. In: ISoLA (1): Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I. Lecture Notes in Computer Science, vol. 7609, pp. 554–571. Springer (2012)
34. Howar, F., Steffen, B.: Active automata learning in practice. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. pp. 123–148. Springer International Publishing (2018)
35. Irfan, M.N., Oriat, C., Groz, R.: Angluin style finite state machine inference with non-optimal counterexamples. In: Proceedings of the First International Workshop on Model Inference In Testing. p. 11–19. MIIT '10, Association for Computing Machinery, New York, NY, USA (2010)
36. Isberner, M.: Foundations of active automata learning: an algorithmic perspective. Ph.D. thesis, Technical University Dortmund, Germany (2015), http://hdl.handle.net/2003/34282
37. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. pp. 307–322. Springer International Publishing, Cham (2014)
38. Kearns, M.J., Vazirani, U.V.: An introduction to computational learning theory. MIT Press (1994)
39. Lee, D., Yannakakis, M.: Testing finite-state machines: State identification and verification. IEEE Trans. Comput. **43**(3), 306–320 (1994)
40. Maler, O., Mens, I.: A generic algorithm for learning symbolic automata from membership queries. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10460, pp. 146–169. Springer (2017)
41. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Inf. Comput. **118**(2), 316–326 (1995), https://doi.org/10.1006/inco.1995.1070
42. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based relevance filtering for efficient system-level test-based model generation. Innov. Syst. Softw. Eng. **1**(2), 147–156 (2005), https://doi.org/10.1007/s11334-005-0016-y
43. Meinke, K.: CGE: A sequential learning algorithm for Mealy automata. In: Sempere, J., García, P. (eds.) Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6339, pp. 148–162. Springer (2010)
44. Meinke, K., Niu, F., Sindhu, M.A.: Learning-based software testing: A tutorial. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation - International Workshops, SARS 2011 and MLSC 2011, Held Under the Auspices of ISoLA 2011 in

Vienna, Austria, October 17-18, 2011. Revised Selected Papers. Communications in Computer and Information Science, vol. 336, pp. 200–219. Springer (2011)
45. Merten, M., Howar, F., Steffen, B., Margaria, T.: Automata learning with on-the-fly direct hypothesis construction. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation - International Workshops, SARS 2011 and MLSC 2011, Held Under the Auspices of ISoLA 2011 in Vienna, Austria, October 17-18, 2011. Revised Selected Papers. Communications in Computer and Information Science, vol. 336, pp. 248–260. Springer (2011)
46. Moerman, J., Sammartino, M., Silva, A., Klin, B., Szynwelski, M.: Learning nominal automata. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 613–625. ACM (2017), https://doi.org/10.1145/3009837.3009879
47. Neider, D., Smetsers, R., Vaandrager, F.W., Kuppens, H.: Benchmarks for automata learning and conformance testing. In: Margaria, T., Graf, S., Larsen, K.G. (eds.) Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 11200, pp. 390–416. Springer (2018)
48. Niese, O.: An Integrated Approach to Testing Complex Systems. Ph.D. thesis, University of Dortmund (2003)
49. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) 5^{th} GI Conference. Lecture Notes in Computer Science, vol. 104, pp. 167–183. Springer-Verlag (1981)
50. Petrenko, A., Avellaneda, F., Groz, R., Oriat, C.: From passive to active FSM inference via checking sequence construction. In: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (eds.) Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10533, pp. 126–141. Springer (2017)
51. Petrenko, A., Li, K., Groz, R., Hossen, K., Oriat, C.: Inferring approximated models for systems engineering. In: 15th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2014, Miami Beach, FL, USA, January 9-11, 2014. pp. 249–253. IEEE Computer Society (2014), https://doi.org/10.1109/HASE.2014.46
52. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. STTT **11**(5), 393–407 (2009)
53. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences (extended abstract). In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, 15-17 May 1989, Seattle, Washington, USA. pp. 411–420. ACM (1989)
54. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993), http://dx.doi.org/10.1006/inco.1993.1021
55. Sakr, S., Bonifati, A., Voigt, H., Iosup, A., Ammar, K., Angles, R., Aref, W., Arenas, M., Besta, M., Boncz, P.A., Daudjee, K., Valle, E.D., Dumbrava, S., Hartig, O., Haslhofer, B., Hegeman, T., Hidders, J., Hose, K., Iamnitchi, A., Kalavri, V., Kapp, H., Martens, W., Özsu, M.T., Peukert, E., Plantikow, S., Ragab, M., Ripeanu, M.R., Salihoglu, S., Schulz, C., Selmer, P., Sequeda, J.F., Shinavier, J., Szárnyas, G., Tommasini, R., Tumeo, A., Uta, A., Varbanescu, A.L., Wu, H.Y., Yakovets, N., Yan, D., Yoneki, E.: The future is big graphs: A community view on graph processing systems. Commun. ACM **64**(9), 62–71 (Aug 2021). https://doi.org/10.1145/3434642, https://doi.org/10.1145/3434642

56. Schuts, M., Hooman, J., Vaandrager, F.: Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In: Ábrahám, E., Huisman, M. (eds.) Proceedings 12th International Conference on integrated Formal Methods (iFM), Reykjavik, Iceland, June 1-3. LNCS, vol. 9681, pp. 311–325 (2016)
57. Shahbaz, M., Groz, R.: Inferring Mealy machines. In: Cavalcanti, A., Dams, D. (eds.) FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5850, pp. 207–222. Springer (2009)
58. Smeenk, W., Moerman, J., Vaandrager, F.W., Jansen, D.N.: Applying automata learning to embedded control software. In: Butler, M.J., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9407, pp. 67–83. Springer (2015), https://doi.org/10.1007/978-3-319-25423-4_5
59. Smetsers, R., Fiterau-Brostean, P., Vaandrager, F.W.: Model learning as a satisfiability modulo theories problem. In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10792, pp. 182–194. Springer (2018)
60. Smetsers, R., Moerman, J., Jansen, D.N.: Minimal separating sequences for all pairs of states. In: Dediu, A., Janousek, J., Martín-Vide, C., Truthe, B. (eds.) Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9618, pp. 181–193. Springer (2016), https://doi.org/10.1007/978-3-319-30000-9_14
61. Soucha, M., Bogdanov, K.: Observation tree approach: Active learning relying on testing. Comput. J. **63**(9), 1298–1310 (2020), https://doi.org/10.1093/comjnl/bxz056
62. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2 edn. (2000). https://doi.org/10.1017/CBO9781139168717
63. Urbat, H., Schröder, L.: Automata learning: An algebraic approach. In: LICS. pp. 900–914. ACM (2020)
64. Vaandrager, F.: Model learning. Communications of the ACM **60**(2), 86–95 (Feb 2017). https://doi.org/10.1145/2967606
65. Vaandrager, F., Bloem, R., Ebrahimi, M.: Learning Mealy machines with one timer. In: Leporati, A., Martín-Vide, C., Shapira, D., Zandron, C. (eds.) Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12638, pp. 157–170. Springer (2021)

A Omitted Proofs

Proof of Lemma 2.7

For the witness $\sigma \vdash q \# q'$, we have $\lambda^{\mathcal{T}}(q,\sigma){\downarrow}$ and $\lambda^{\mathcal{T}}(q',\sigma){\downarrow}$. Since f is a functional simulation, by Lemma 2.4 we have $\lambda^{\mathcal{M}}(f(q),\sigma){\downarrow}$ and $\lambda^{\mathcal{T}}(q,\sigma) = \lambda^{\mathcal{M}}(f(q),\sigma)$, and similarly $\lambda^{\mathcal{T}}(q',\sigma) = \lambda^{\mathcal{M}}(f(q'),\sigma)$. Hence,

$$\lambda^{\mathcal{M}}(f(q),\sigma) = \lambda^{\mathcal{T}}(q,\sigma) \neq \lambda^{\mathcal{T}}(q',\sigma) = \lambda^{\mathcal{M}}(f(q),\sigma)$$

which proves $[\![f(q)]\!] \neq [\![f(q')]\!]$.

Proof of Lemma 2.8

The witness $\sigma \vdash r \# r'$ implies that $\lambda(r,\sigma){\downarrow}$, $\lambda(r',\sigma){\downarrow}$, and $\lambda(r,\sigma) \neq \lambda(r',\sigma)$. Since $\lambda(q,\sigma){\downarrow}$, $\neg(r \# q) \wedge \neg(r' \# q)$ leads to the contradiction

$$\lambda(r,\sigma) \overset{\neg(r\#q)}{=} \lambda(q,\sigma) \overset{\neg(r'\#q)}{=} \lambda(r',\sigma). \qquad \square$$

Proof of Lemma 3.6

Consider the relation:

$$\Delta \subseteq (S \times I) \times S \qquad ((q,i),q') \in \Delta \quad \text{if} \quad \delta^{\mathcal{T}}(q,i){\uparrow} \ \text{ or } \ \neg(\delta^{\mathcal{T}}(q,i) \# q')$$

1. If there are no isolated states, then every $(q,i) \in S \times I$ is related to some $q' \in S$, so there is some functional relation $\delta^{\mathcal{H}} \subseteq \Delta$ of type $\delta^{\mathcal{H}} \colon S \times I \to S$.
2. If S is complete and all states in F are identified, then Δ is already a functional relation $\delta^{\mathcal{H}} = \Delta \colon S \times I \to S$. $\square$

Proof of Theorem 3.7

In the proof of Theorem 3.7, we characterize equivalence of Mealy machines via *bisimulations*.

Definition A.1. *A **bisimulation** between Mealy machines $\mathcal{M}$ and $\mathcal{N}$ is a relation $R \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ satisfying, for all $q \in Q^{\mathcal{M}}$, $r \in Q^{\mathcal{N}}$, $i \in I$, $o \in O$,*

$$q_0^{\mathcal{M}} \, R \, q_0^{\mathcal{N}} \qquad \textit{and} \qquad q \, R \, r \ \wedge \ q \xrightarrow{i/o} q' \ \Rightarrow \ \exists r' \colon r \xrightarrow{i/o} r' \ \wedge \ q' \, R \, r'$$

We write $\mathcal{M} \simeq \mathcal{N}$ if there exists a bisimulation relation between $\mathcal{M}$ and $\mathcal{N}$.

Lemma A.2. *Given complete Mealy machines $\mathcal{M}$ and $\mathcal{N}$, the equivalence relation $\approx \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ is a bisimulation.*

The next lemma, which is a variation of the classical result of [49], is again easy to prove.

Lemma A.3. *Let $\mathcal{M}$ and $\mathcal{N}$ be complete Mealy machines. Then $\mathcal{M} \simeq \mathcal{N}$ iff $\mathcal{M} \approx \mathcal{N}$.*

We now come to the actual proof of Theorem 3.7:

Proof (of Theorem 3.7). Let f be a refinement from $\mathcal{T}$ to $\mathcal{M}$. Define relation $R \subseteq S \times Q^{\mathcal{M}}$ by

$$(q, r) \in R \Leftrightarrow f(q) \approx^{\mathcal{M}} r.$$

We claim that R is a bisimulation between $\mathcal{H}$ and $\mathcal{M}$.

1. Since f is a refinement from $\mathcal{T}$ to $\mathcal{M}$, $f(q_0^{\mathcal{T}}) = q_0^{\mathcal{M}}$. By construction, $q_0^{\mathcal{T}} = q_0^{\mathcal{H}}$. Now the fact that equivalence relation $\approx^{\mathcal{M}}$ is reflexive implies $f(q_0^{\mathcal{H}}) \approx^{\mathcal{M}} q_0^{\mathcal{M}}$, and therefore $(q_0^{\mathcal{H}}, q_0^{\mathcal{M}}) \in R$.
2. Suppose $(q, r) \in R$ and $i \in I$. Let $q' = \delta^{\mathcal{H}}(q, i)$ and $r' = \delta^{\mathcal{M}}(r, i)$. We need to show that $\lambda^{\mathcal{H}}(q, i) = \lambda^{\mathcal{M}}(r, i)$ and $(q', r') \in R$. We consider two cases:
   (a) $\delta^{\mathcal{T}}(q, i) \in S$. Then, by construction of $\mathcal{H}$, $\lambda^{\mathcal{H}}(q, i) = \lambda^{\mathcal{T}}(q, i)$ and $q' = \delta^{\mathcal{T}}(q, i)$. Moreover, as f is a refinement from $\mathcal{T}$ to $\mathcal{M}$, $f(q') = \delta^{\mathcal{M}}(f(q), i)$ and $\lambda^{\mathcal{T}}(q, i) = \lambda^{\mathcal{M}}(f(q), i)$. By definition of R, $f(q) \approx^{\mathcal{M}} r$. Hence, by Lemma A.2, $\lambda^{\mathcal{M}}(f(q), i) = \lambda^{\mathcal{M}}(r, i)$ and $\delta^{\mathcal{M}}(f(q), i) \approx^{\mathcal{M}} r'$. By combining the derived equalities we obtain:

   $$\begin{aligned} \lambda^{\mathcal{H}}(q, i) = \lambda^{\mathcal{T}}(q, i) = \lambda^{\mathcal{M}}(f(q), i) = \lambda^{\mathcal{M}}(r, i), \\ f(q') = \delta^{\mathcal{M}}(f(q), i) \approx {}^{M} r'. \end{aligned}$$

   Hence by definition of R, $(q', r') \in R$, as required.
   (b) $\delta^{\mathcal{T}}(q, i) \in F$. Let $q'' = \delta^{\mathcal{T}}(q, i) \in F$. Then, by construction of $\mathcal{H}$, $\lambda^{\mathcal{H}}(q, i) = \lambda^{\mathcal{T}}(q, i)$ and q' is the unique state in S such that q'' and q' are not apart. By Lemma A.2, since all states of S are pairwise apart, all states in the image of s under f are in different equivalence classes of $\approx^{\mathcal{M}}$. Since $\approx^{\mathcal{M}}$ has as many equivalence classes as the number of states of S, each state of $\mathcal{M}$ belongs to the same equivalence class as $f(s)$, for some $s \in S$. Since q'' is apart from all states of S except q', $f(q'')$ does not belong to the same equivalence class as $f(s)$, for $s \in S \setminus \{q'\}$, by Lemma A.2. Hence, by the Sherlock Holmes principle, $f(q'') \approx^{\mathcal{M}} f(q')$. Since f is a refinement from $\mathcal{T}$ to $\mathcal{M}$, $f(q'') = \delta^{\mathcal{M}}(f(q), i)$ and $\lambda^{\mathcal{T}}(q, i) = \lambda^{\mathcal{M}}(f(q), i)$. By definition of R, $f(q) \approx^{\mathcal{M}} r$. Hence, by Lemma A.2, $\lambda^{\mathcal{M}}(f(q), i) = \lambda^{\mathcal{M}}(r, i)$ and $\delta^{\mathcal{M}}(f(q), i) \approx^{\mathcal{M}} r'$. By combining the derived equalities we obtain:

   $$\begin{aligned} \lambda^{\mathcal{H}}(q, i) &= \lambda^{\mathcal{T}}(q, i) = \lambda^{\mathcal{M}}(f(q), i) = \lambda^{\mathcal{M}}(r, i), \\ f(q') \approx^{\mathcal{M}} f(q'') &= \delta^{\mathcal{M}}(f(q), i) \approx {}^{M} r'. \end{aligned}$$

   As equivalence relation $\approx^{\mathcal{M}}$ is transitive, $f(q') \approx^{\mathcal{M}} r'$, and hence by definition of R, $(q', r') \in R$, as required.

The theorem now follows by application of Lemma A.3.

Proof of Theorem 3.8

In all cases, let $S, F, \mathcal{T}$ denote the values before and $S', F', \mathcal{T}'$ denote the values after the respective rule application. Also introduce abbreviations:

$$N_Q(\mathcal{T}) = \frac{|S| \cdot (|S|+1)}{2}$$

$$N_{\downarrow}(\mathcal{T}) = \{(q,i) \in S \times I \mid \delta(q,i){\downarrow}\}$$

$$N_{\#}(\mathcal{T}) = \{(q,q') \in S \times F \mid q \# q'\}$$

The total norm is:

$$N(\mathcal{T}) = N_Q(\mathcal{T}) + |N_{\downarrow}(\mathcal{T})| + |N_{\#}(\mathcal{T})|$$

1. If q is isolated and is thus moved from F to S, i.e. $S' := S \cup \{q\}$, then we have

$$\begin{aligned} N_Q(\mathcal{T}') &= \frac{|S'| \cdot (|S'|+1)}{2} = \frac{(|S|+1) \cdot (|S|+1+1)}{2} \\ &= \frac{(|S|+1) \cdot |S|}{2} + \frac{(|S|+1) \cdot 2}{2} = N_Q(\mathcal{T}) + |S| + 1 \end{aligned}$$

$$N_{\downarrow}(\mathcal{T}') \supseteq N_{\downarrow}(\mathcal{T})$$

   Finally we have

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

   and thus

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

   In total, $N(\mathcal{T}') \geq N(\mathcal{T}) + 1$.
2. In the second rule, let $\delta^{\mathcal{T}}(q,i)$ for some $q \in S$, $i \in I$. After the output query for $\mathsf{access}(q)\, i\, \textsc{Ads}(S)$, we have

$$N_Q(\mathcal{T}') = N_Q(\mathcal{T}) \qquad N_{\downarrow}(\mathcal{T}') = N_{\downarrow}(\mathcal{T}) \cup \{(q,i)\} \qquad N_{\#}(\mathcal{T}') \subseteq N_{\#}(\mathcal{T})$$

   and thus $N(\mathcal{T}') \geq N(\mathcal{T}) + 1$.
3. For the third rule, consider a state $q \in F$ and distinct $r, r' \in S$ with $\neg(q \# r)$ 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 $r \# q$ or $r' \# q$ by weak co-transitivity (Lemma 2.8). Thus,

$$N_{\#}(\mathcal{T}') \supseteq N_{\#}(\mathcal{T}) \cup \{(r,q)\} \qquad \text{or} \qquad N_{\#}(\mathcal{T}') \supseteq N_{\#}(\mathcal{T}) \cup \{(r',q)\}$$

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

4. If the fourth rule did not terminate the algorithm, we show that $\mathcal{H}$ is not a hypothesis for $\mathcal{T}'$ anymore. By Lemma 3.10 and by EquivQuery, we have in any case that $\sigma \in I^*$ is such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}'}(q_0^{\mathcal{T}'}, \sigma)$ (in a possibly extended observation tree $\mathcal{T}'$). Moreover, during this rule, in CheckConsistency and ProcCounterEx, the basis S is not modified: $S = S'$. Even though the observation tree has been updated since BuildHypothesis, $\mathcal{H}$ still meets the criteria of Lemma 3.11. Hence, after counter example processing, $\mathcal{H}$ is not a hypothesis for the updated $\mathcal{T}'$ 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 $\mathcal{T}'$. Hence $q \# r$, so that $r \neq q$ and consequently $r \in F$. Therefore, we obtain:
$$(q, r) \in N_{\#}(\mathcal{T}') \setminus N_{\#}(\mathcal{T}). \qquad \square$$

Proof of Theorem 3.9

Since there is a functional simulation $\mathcal{T} \to \mathcal{M}$, we have by Lemma 2.7 that

$$|S| \leq n.$$

The number of successors of the basis is bounded by $k \cdot n$:

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

The set $S \cup F$ (the basis and all its successor states) contains at most $kn + 1$ elements. Since each state in the frontier can be apart from at most $n - 1$ states in the basis, this means we have

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

In $\mathcal{O}$-notation this simplifies to

$$N(\mathcal{T}) \leq \frac{1}{2}n(n + 1) + kn + (n - 1)(kn + 1) \in \mathcal{O}(kn^2) \qquad \square$$

Proof of Lemma 3.10

The breadth-first search in Algorithm 2 verifies whether there is a functional simulation $f\colon \mathcal{T} \to \mathcal{H}$. Since $\mathcal{H}$ is deterministic (like all Mealy machines considered here) and since every state of $\mathcal{T}$ is reachable from the root, there is at most one functional simulation $\mathcal{T} \to \mathcal{H}$. Thus, consistency checking amounts to verifying whether the map

$$f\colon Q^{\mathcal{T}} \to Q^{\mathcal{H}} \qquad f(q) := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q))$$

is a functional simulation (Definition 2.3).

- If the procedure returns `no`, then $q \# f(q)$ for some $q \in \mathcal{T}$. Note that f is idempotent, because $\mathcal{H}$ contains S: $f(q) = f(f(q))$ (using $Q^{\mathcal{H}} \subseteq Q^{\mathcal{T}}$). If f was a functional simulation $\mathcal{T} \to \mathcal{H}$, this would lead to a contradiction: applying Lemma 2.7 to $q \# f(q)$ (in $\mathcal{T}$) implies that $f(q) \not\approx f(f(q)) = f(q)$ (in $\mathcal{M}$), a contradiction to the reflexivity of $\approx$.
- If the procedure returns `yes`, then $\neg(q \# f(q))$ for all $q \in Q^{\mathcal{T}}$. For the verification that f is a functional simulation, first note that we trivially have $f(q_0^{\mathcal{T}}) = q_0^{\mathcal{H}}$. For the preservation of transitions, consider $q \xrightarrow{i/o} p$ in $\mathcal{T}$ and $f(q) \xrightarrow{i/o'} p'$ in $\mathcal{H}$. Since the basis S is complete in $\mathcal{T}$, we have $\lambda^{\mathcal{T}}(f(q), i) = o'$. Thus $o = o'$, because otherwise we had $i \vdash q \# f(q)$. Note that $\mathsf{access}(p) = \mathsf{access}(q)\, i$ and so

$$f(p) = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(p)) = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q)\, i) = \delta^{\mathcal{H}}(f(q), i) = p'$$

and thus f is a functional simulation. □

Proof of Lemma 3.11

We prove termination by providing a bound on the number of recursive calls. For an input word $\sigma \in I^*$ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)\downarrow$, we define the *distance from the frontier* $d(\sigma) \in \mathbb{N}$ by:

$$d(\sigma) = |\sigma| - \max\{|\rho| \mid \rho \text{ prefix of } \sigma, \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \in S \cup F\}$$

Observe that:

- $d(\sigma) = 0$ iff $r := \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) \in S \cup F$.
- If $d(\sigma) > 0$ then $d(\sigma) = |\sigma| - |\rho| \geq 1$ with ρ defined as in Algorithm 3. For the decomposition $\sigma = \sigma_1 \cdot \sigma_2$, we have

$$d(\sigma_1) = h - |\rho| = \left\lfloor \frac{|\rho| + |\sigma|}{2} \right\rfloor - |\rho| = \left\lfloor |\rho| + \frac{|\sigma| - |\rho|}{2} \right\rfloor - |\rho| = \left\lfloor \frac{d(\sigma)}{2} \right\rfloor.$$

Since $q' := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1) \in S$ by definition, we have that $\delta^{\mathcal{T}}(q', i) \in S \cup F$ if i is the first character of σ_2. Note that if σ_2 is empty, then $d(\mathsf{access}(q')\, \sigma_2) = 0 \leq \frac{d(\sigma)}{2}$, trivially. So if σ_2 is not empty, then we have:

$$\begin{aligned} d(\mathsf{access}(q')\, \sigma_2) \;&\leq\; |\sigma_2| - 1 = |\sigma| - h - 1 = |\sigma| - \left\lfloor \frac{|\rho| + |\sigma|}{2} \right\rfloor - 1 \\ &= \left\lceil |\sigma| - \frac{|\rho| + |\sigma|}{2} \right\rceil - 1 \leq \left\lfloor |\sigma| - \frac{|\rho| + |\sigma|}{2} \right\rfloor = \left\lfloor \frac{|\sigma| - |\rho|}{2} \right\rfloor \leq \left\lfloor \frac{d(\sigma)}{2} \right\rfloor. \end{aligned}$$

So in any of the two recursive calls, if $\sigma' \in I^*$ denotes parameter passed to the recursive call, then we have $2 \cdot d(\sigma') \leq d(\sigma)$. This implies termination.

Let $OQ(n)$ denote the maximal number of output queries performed during a run of Algorithm 3 with $n = d(\sigma)$. Then, using the above observations, we may show by induction on $d(\sigma)$ that

$$OQ(n) \leq \begin{cases} 0 & \text{if } n = 0 \\ \log_2(2n) & \text{if } n > 0 \end{cases}$$

Since $d(\sigma) < |\sigma|$, this implies that the number of output queries is bounded by $\mathcal{O}(\log(|\sigma|))$.

For correctness, let $q := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$ and $r := \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$ as in Algorithm 3 such that $\eta \vdash q \# r$ for some $\eta \in I^*$, i.e. $\lambda^{\mathcal{T}}(q, \eta) \neq \lambda^{\mathcal{T}}(r, \eta)$

- In the case of $r \in S \cup F$, note that since $q_0^{\mathcal{H}} = q_0^{\mathcal{T}}$ and $q \# r$, we have $q_n \neq r_n$ and $|\sigma| \geq 1$. Hence, we can decompose $\sigma = \alpha\, i$ into $\alpha \in I^*$ and $i \in I$. Let $q' = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \alpha)$ and $r' = \delta^{\mathcal{H}}(q_0,^{\mathcal{T}}, \alpha)$. Since $\mathcal{T}$ is a tree, we necessarily have $\alpha = \mathsf{access}(r')$. Hence,

$$q' = \delta^{\mathcal{H}}(q_0, \alpha) = \delta^{\mathcal{H}}(q_0, \mathsf{access}(r')) = r'.$$

  We have $q' \xrightarrow{i} q$ in $\mathcal{H}$ and $r' \xrightarrow{i} r$ in $\mathcal{T}$ with $q' = r'$ but $q \# r$, hence $\mathcal{H}$ is not a hypothesis for $\mathcal{T}$.
- Let $\sigma = \sigma_1\, \sigma_2$ be the decomposition into $\sigma_1, \sigma_2 \in I^*$, and let $q' := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1) \in S$ and $r' := \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma_1)$ as in Algorithm 3. After OutputQuery, we have $\lambda^{\mathcal{T}}(q', \sigma_2\, \eta){\downarrow}$ and thus:
  1. If $q' \# r'$, then $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma_1)$, so σ_1 is a valid parameter to ProcCounterEx and shorter than σ, so by induction, $\mathcal{H}$ is not a hypothesis anymore after the recursive call.
  2. If $\neg(q' \# r')$, then we necessarily have that

$$\lambda^{\mathcal{T}}(q', \sigma_2\, \eta) = \lambda^{\mathcal{T}}(r', \sigma_2\, \eta)$$

  and thus also

$$\lambda^{\mathcal{T}}(\delta^{\mathcal{T}}(q', \sigma_2), \eta) = \lambda^{\mathcal{T}}(\delta^{\mathcal{T}}(r', \sigma_2), \eta). \tag{$*$}$$

  We verify that $\mathsf{access}(q')\, \sigma_2$ can be passed to ProcCounterEx:

$$\lambda^{\mathcal{T}}(\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q')\, \sigma_2), \eta) = \lambda^{\mathcal{T}}(\delta^{\mathcal{H}}(q', \sigma_2), \eta) = \lambda^{\mathcal{T}}(q, \eta)$$

  But on the other hand:

$$\begin{aligned} \lambda^{\mathcal{T}}(\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \mathsf{access}(q')\, \sigma_2), \eta) &= \lambda^{\mathcal{T}}(\delta^{\mathcal{T}}(q', \sigma_2), \eta) \overset{(*)}{=} \lambda^{\mathcal{T}}(\delta^{\mathcal{T}}(r', \sigma_2), \eta) \\ &= \lambda^{\mathcal{T}}(r, \eta) \neq \lambda^{\mathcal{T}}(q, \eta) \end{aligned}$$

  Hence, η is a witness for $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q')\, \sigma_2) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \mathsf{access}(q')\, \sigma_2)$ and invoking ProcCounterEx($\mathsf{access}(q')\, \sigma_2$) makes that $\mathcal{H}$ is not a hypothesis for $\mathcal{T}$ afterwards. □

Proof of Proposition 3.12

We first show that if there are at least two states to be distinguished, the expected reward is positive:

Lemma A.4. *Suppose that* $U \subseteq Q^{\mathcal{T}}$, $r, r' \in U$ *with* $r \# r'$. *Then* $E(U) > 0$.

Proof. Let $\sigma \in I^+$ be a witness for $r \# r'$, that is $\sigma \vdash r \# r'$. We prove $E(U) > 0$ by induction on the length of σ. Let $\sigma = i\ \sigma'$ for $i \in I$ and $\sigma' \in I^*$ and put $o = \lambda(r, i) \in O$, and $o' = \lambda(r', i) \in O$.

– If $o \neq o'$, then $\delta(r, i) \in U \xrightarrow{i/o}$ and $\delta(r', i) \notin U \xrightarrow{i/o}$, and so

$$|U \xrightarrow{i/o}| \geq 1 \text{ and } (|U \xrightarrow{i}| - |U \xrightarrow{i/o}|) \geq 1$$

Thus, the fraction in (2) is greater than 0 and consequently also $E(U) > 0$.

– If $o = o'$, then $\delta(r, i), \delta(r', i) \in U \xrightarrow{i/o}$ and $\sigma' \vdash \delta(r, i) \# \delta(r', i)$. Hence, we obtain $E(U \xrightarrow{i/o}) > 0$ by the induction hypothesis and so the fraction for i and o is greater than 0 and so $E(U) > 0$. □

Next we show that if the expected reward is positive, there exist, for every maximal path in $\text{ADS}(U)$, two states in U that are apart.

Lemma A.5. *If* $E(U) > 0$, π *is a path from the root of* $\text{ADS}(U)$ *to a leaf, and* $\sigma \in I^*$ *is the sequence of labels of the states occurring in* π, *then there are* $r, r' \in U$ *with* $\lambda(r, \sigma) \neq \lambda(r', \sigma)$.

Proof. We prove the claim by induction on the length (that is, the number of transitions) of π.

If the length of π is 0 then none of the states in U has an outgoing transition, which implies that $E(U) = 0$, which means that the statement of the lemma holds.

For the induction step, assume that the length of π is greater than 0. Then at least one state in U has an outgoing transition. Let i be the input that witnesses the maximum in $E(U) > 0$. Then the root of $\text{ADS}(U)$ has label i and thus $\sigma = i\ \sigma'$, for some σ'. Suppose that path π starts with an o-transition from U to $\text{ADS}(U \xrightarrow{i/o})$, where $U \xrightarrow{i/o} \neq \emptyset$. We consider two cases:

– If $|U \xrightarrow{i}| > |U \xrightarrow{i/o}|$, then there is some $o' \neq o$ with $U \xrightarrow{i/o'} \neq \emptyset$. This implies there exist states $r, r' \in U$ with $\lambda(r, i) = o$ and $\lambda(r, i) = o'$ and in particular $\lambda(r, \sigma) \neq \lambda(r', \sigma)$.

– If $|U \xrightarrow{i}| = |U \xrightarrow{i/o}|$ then $E(U \xrightarrow{i/o}) = E(U) > 0$. This means we can apply the induction hypothesis to obtain $p, p' \in U \xrightarrow{i/o}$ with $\lambda(p, \sigma') \neq \lambda(p', \sigma')$. By definition of $U \xrightarrow{i/o}$, this yields us $r, r' \in U$ with $r \xrightarrow{i/o} p$ and $r' \xrightarrow{i/o} p'$ and so $\lambda(r, \sigma) \neq \lambda(r, \sigma)$ for the composed $\sigma = i\ \sigma'$. □

We now come to the main proof of Proposition 3.12, i.e. that the updated output queries induce at least the same apartness pairs and so the norm $\mathcal{N}(\mathcal{T})$ grows with each rule application:

1. It is clear that $\textsc{OutputQuery}(\mathsf{access}(q)\ i\ \textsc{Ads}(S))$ in the updated (R2') discovers at least the same apartness pairs as $\textsc{OutputQuery}(\mathsf{access}(q)\ i)$ in plain $L^{\#}$.
2. For $U := \{b \in S \mid \neg(b \# q)\}$, we show that querying $\mathsf{access}(q)\ \textsc{Ads}(U)$ in the updated (R3') makes b apart from at least one $r \in U$. Whenever the rule (R3') is applied, then U contains two states that are apart, so $E(U) > 0$ by Lemma A.4. Let $\sigma \in I^+$ be the sequence that is sent in total to the teacher, i.e. σ is the path of $\textsc{Ads}(U)$ that is actually run.
   We distinguish two cases:
   - If σ does not reach to a leaf in the decision tree $\textsc{Ads}(U)$, then this means that the adaptive distinguished sequence terminated earlier because of an unexpected output $o \in O$. Concretely, this means that $\lambda(q, \sigma) \neq \lambda(b, \sigma)$ for all $b \in U$.
   - If σ reaches a leaf in the decision tree $\textsc{Ads}(U)$, then we obtain that there are $r, r' \in U$ with $\lambda(r, \sigma) \neq \lambda(r', \sigma)$, by Lemma A.5. After the output query, $\lambda(q, \sigma)\downarrow$, and so by weak co-transitivity (Lemma 2.8), either $q \# r$ or $q \# r'$ or both.

   Hence, $L^{\#}_{\textsc{Ads}}$ lets the norm $\mathcal{N}(\mathcal{T})$ grow with each rule application. By Theorem 3.9, $L^{\#}_{\textsc{Ads}}$ must have reached the correct hypothesis before $\mathcal{N}(\mathcal{T})$ exceeds the bound, hence it is correct. (Note that Theorem 3.9 is a general observation on observation trees and does not involve the algorithm at all).

Proof of Theorem 3.14

Strategic $L^{\#}_{\textsc{Ads}}$ makes the same amount of output queries and equivalence queries as strategic $L^{\#}$, so it is sufficient to discuss strategic $L^{\#}$.

In the strategic $L^{\#}$, every (non-terminating) application of rule (R4) leads to an isolated state in the frontier, i.e. increases the basis by one state before another equivalence query can be asked. Since the basis may contain at most n elements, this means that there are at most $n-1$ applications of rule (R4). Processing the counterexamples generated by the resulting consistency checks and equivalence queries of rule (R4) will require $\mathcal{O}(n \log m)$ output queries. By Theorem 3.8 and Theorem 3.9 there are at most $\mathcal{O}(kn^2)$ rule applications during a run of $L^{\#}$. Since applications of rule (R1) require no output queries, and each application of rule (R2) and (R3) requires exactly one output query, this means that applications of rules (R1), (R2) and (R3) will require $\mathcal{O}(kn^2)$ output queries. Altogether, $L^{\#}$ will require $\mathcal{O}(kn^2 + n \log m)$ output queries.

Proof of Theorem 3.15

During counterexample processing, we may create a witness of length at most m between a state q in the frontier and a state r in the basis. When we subsequently move state q to the basis, we have created a pair of states in the basis with a witness of length m. In fact, at any point during a run of $L^{\#}$, the length of a minimal witness that distinguishes a state from $F \cup S$ from a state of S will be at most m. Since $n \in \mathcal{O}(m)$, this implies that the number of symbols in any output query will be $\mathcal{O}(m)$. Since, by Theorem 3.14, there are $\mathcal{O}(kn^2 + n \log m)$ output queries, the result follows.

B Complete benchmarking results

In Tables 1 to 5, we list the number of queries for every model and every learning algorithm. See Section 4 for and description of the benchmark setup.

Table 1: Benchmark results (Part 1 of 5)

Model	Algorithm	EQs	Learn-Inputs	Learn-Resets	Test-Inputs	Test-Resets	Total
4_learnresult_SecureCodeAut_fix	$L^\#$	3.00	484.35	146.36	194.60	17.87	843.18
	$L^\#_{\mathrm{ADS}}$	3.00	322.58	96.04	207.26	18.13	644.01
$n = 4$ $k = 14$	TTT	2.95	495.57	149.24	210.96	16.78	872.55
	ADT	2.93	477.73	115.56	201.74	16.55	811.58
	RS	2.99	1214.37	132.19	170.30	14.47	1531.33
ASN_learnresult_SecureCodeAut_fix	$L^\#$	3.00	485.74	146.43	225.31	19.89	877.37
	$L^\#_{\mathrm{ADS}}$	3.00	327.35	96.74	206.05	17.67	647.81
$n = 4$ $k = 14$	TTT	2.94	495.52	149.38	216.96	17.21	879.07
	ADT	2.92	458.28	110.40	163.00	14.21	745.89
	RS	2.91	1252.11	127.60	159.23	13.15	1552.09
1_learnresult_MasterCard_fix	$L^\#$	3.00	728.63	201.45	81.59	9.19	1020.86
	$L^\#_{\mathrm{ADS}}$	3.75	576.70	141.09	359.43	29.03	1106.25
$n = 5$ $k = 15$	TTT	3.42	620.95	187.46	227.21	18.15	1053.77
	ADT	3.71	818.83	165.08	449.64	34.49	1468.04
	RS	3.32	2347.76	196.97	344.27	26.32	2915.32
OpenSSL_1.0.1j_client_regular	$L^\#$	6.00	439.25	114.64	2866.97	218.78	3639.64
	$L^\#_{\mathrm{ADS}}$	5.91	200.30	56.01	2244.26	175.18	2675.75
$n = 6$ $k = 7$	TTT	4.35	320.91	92.23	1679.45	128.02	2220.61
	ADT	4.89	475.37	89.39	2131.04	161.65	2857.45
	RS	4.60	1868.79	165.97	1979.22	149.00	4162.98
OpenSSL_1.0.1l_client_regular	$L^\#$	6.00	441.21	115.42	2576.04	198.39	3331.06
	$L^\#_{\mathrm{ADS}}$	5.90	199.65	55.88	2057.69	159.52	2472.74
$n = 6$ $k = 7$	TTT	4.54	295.63	85.73	1859.69	141.00	2382.05
	ADT	4.89	465.76	88.15	2446.06	182.21	3182.18
	RS	4.49	1915.47	161.12	1838.95	139.71	4055.25
OpenSSL_1.0.2_client_regular	$L^\#$	6.00	440.58	114.76	2442.08	188.35	3185.77
	$L^\#_{\mathrm{ADS}}$	5.93	201.51	56.45	2197.54	172.05	2627.55
$n = 6$ $k = 7$	TTT	5.31	312.40	89.52	1896.67	145.52	2444.11
	ADT	4.92	447.98	87.94	2237.95	167.92	2941.79
	RS	4.60	1727.94	165.39	1899.77	143.42	3936.52
RSA_BSAFE_Java_6.1.1_server_regular	$L^\#$	4.94	419.53	112.52	3444.36	264.15	4240.56
	$L^\#_{\mathrm{ADS}}$	4.95	198.28	56.81	3412.44	261.89	3929.42
$n = 6$ $k = 8$	TTT	4.84	322.00	94.00	3201.54	239.90	3857.44
	ADT	4.55	369.82	79.67	2777.73	213.04	3440.26
	RS	4.50	1840.03	181.51	2631.52	201.02	4854.08
miTLS_0.1.3_server_regular	$L^\#$	4.00	579.69	154.13	1329.16	104.97	2167.95
	$L^\#_{\mathrm{ADS}}$	4.35	320.61	87.79	1609.79	128.98	2147.17
$n = 6$ $k = 8$	TTT	3.99	481.64	136.55	1377.39	105.62	2101.20
	ADT	4.27	375.66	82.58	2227.62	170.22	2856.08
	RS	3.97	1703.94	156.19	1945.37	146.89	3952.39
10_learnresult_MasterCard_fix	$L^\#$	4.00	894.32	229.38	2399.60	177.44	3700.74
	$L^\#_{\mathrm{ADS}}$	4.00	668.80	163.13	2673.32	195.45	3700.70
$n = 6$ $k = 14$	TTT	3.95	895.90	229.52	3278.16	238.56	4642.14
	ADT	4.35	947.33	199.58	2871.02	208.77	4226.70
	RS	4.14	3681.22	288.22	2838.96	206.23	7014.63
4_learnresult_MAESTRO_fix	$L^\#$	4.00	897.31	229.19	3115.73	225.96	4468.19
	$L^\#_{\mathrm{ADS}}$	4.00	663.51	162.87	3035.95	223.70	4086.03
$n = 6$ $k = 14$	TTT	3.93	892.34	229.45	3152.39	228.05	4502.23
	ADT	4.45	941.28	196.62	2833.00	203.47	4174.37
	RS	4.18	3446.26	291.62	2509.68	179.91	6427.47

Table 2: Benchmark results (Part 2 of 5)

Model n k	Algorithm	EQs	Learn-Inputs	Learn-Resets	Test-Inputs	Test-Resets	Total
4_learnresult_PIN_fix	$L^{\#}$	4.00	894.15	229.08	2379.98	176.20	3679.41
	$L^{\#}_{\mathrm{ADS}}$	4.00	672.03	163.36	2761.73	203.21	3800.33
$n = 6$ $k = 14$	TTT	3.97	895.99	229.75	2979.36	216.20	4321.30
	ADT	4.39	905.19	189.10	2687.80	195.58	3977.67
	RS	4.16	3512.37	289.47	2476.66	179.00	6457.50
ASN_learnresult_MAESTRO_fix	$L^{\#}$	4.00	895.02	228.81	3092.95	224.88	4441.66
	$L^{\#}_{\mathrm{ADS}}$	4.00	664.85	162.74	2587.53	189.71	3604.83
$n = 6$ $k = 14$	TTT	3.99	892.60	229.66	3469.10	252.07	4843.43
	ADT	4.47	927.69	196.79	3159.81	228.45	4512.74
	RS	4.12	3418.29	285.89	2318.70	167.20	6190.08
Rabo_learnresult_MAESTRO_fix	$L^{\#}$	4.00	900.82	229.37	3015.36	220.35	4365.90
	$L^{\#}_{\mathrm{ADS}}$	4.00	667.36	162.94	2689.37	196.94	3716.61
$n = 6$ $k = 14$	TTT	3.97	896.20	229.75	3562.85	255.31	4944.11
	ADT	4.46	930.73	192.82	3050.85	220.12	4394.52
	RS	4.06	3081.30	280.15	2703.82	195.80	6261.07
Rabo_learnresult_SecureCode_Aut_fix	$L^{\#}$	4.00	1016.16	251.29	1531.42	111.57	2910.44
	$L^{\#}_{\mathrm{ADS}}$	4.00	737.42	174.67	1487.81	107.97	2507.87
$n = 6$ $k = 15$	TTT	3.92	999.53	249.37	1475.98	105.90	2830.78
	ADT	4.23	1196.91	243.05	2485.44	171.59	4096.99
	RS	3.69	3221.53	266.03	1257.26	89.63	4834.45
OpenSSL_1.0.2_server_regular	$L^{\#}$	5.64	491.62	135.43	2523.07	195.46	3345.58
	$L^{\#}_{\mathrm{ADS}}$	6.55	335.43	89.22	2543.01	194.99	3162.65
$n = 7$ $k = 7$	TTT	5.70	475.28	137.84	2398.72	182.75	3194.59
	ADT	5.59	557.41	112.91	2445.53	184.98	3300.83
	RS	4.89	1969.57	201.53	1673.30	128.78	3973.18
GnuTLS_3.3.12_client_regular	$L^{\#}$	4.92	562.16	149.79	4777.83	342.42	5832.20
	$L^{\#}_{\mathrm{ADS}}$	4.98	244.03	69.06	3853.00	279.03	4445.12
$n = 7$ $k = 8$	TTT	4.84	397.92	116.05	3395.21	247.80	4156.98
	ADT	4.61	455.24	97.04	3490.28	251.97	4294.53
	RS	4.68	2496.24	220.92	3202.27	231.87	6151.30
GnuTLS_3.3.12_server_regular	$L^{\#}$	4.99	563.83	149.99	2865.63	214.65	3794.10
	$L^{\#}_{\mathrm{ADS}}$	4.92	244.02	68.82	3877.60	280.35	4470.79
$n = 7$ $k = 8$	TTT	5.65	427.35	122.42	4400.93	319.48	5270.18
	ADT	4.67	448.58	98.50	2882.06	211.32	3640.46
	RS	4.74	2397.88	224.18	3073.54	223.68	5919.28
NSS_3.17.4_client_regular	$L^{\#}$	5.95	631.47	144.44	4239.68	324.29	5339.88
	$L^{\#}_{\mathrm{ADS}}$	5.92	378.23	86.18	4734.30	359.43	5558.14
$n = 7$ $k = 8$	TTT	4.92	402.21	106.06	2687.59	203.47	3399.33
	ADT	4.85	477.09	94.10	3204.14	244.10	4019.43
	RS	4.70	2335.73	220.75	3216.35	240.62	6013.45
Volksbank_learnresult_MAESTRO_fix	$L^{\#}$	4.90	1522.08	342.88	407.92	31.60	2304.48
	$L^{\#}_{\mathrm{ADS}}$	4.79	874.04	192.04	409.50	32.33	1507.91
$n = 7$ $k = 14$	TTT	3.70	1177.17	273.76	415.49	29.48	1895.90
	ADT	3.79	1412.84	233.27	347.55	24.04	2017.70
	RS	3.14	2683.45	230.76	141.85	10.72	3066.78
NSS_3.17.4_server_regular	$L^{\#}$	5.56	599.68	150.57	4424.54	334.87	5509.66
	$L^{\#}_{\mathrm{ADS}}$	5.40	288.12	76.45	4227.87	323.47	4915.91
$n = 8$ $k = 8$	TTT	5.15	561.69	150.21	2693.04	205.19	3610.13
	ADT	5.48	570.48	118.95	3918.63	293.35	4901.41
	RS	4.98	2974.54	267.54	3397.81	254.65	6894.54
RSA_BSAFE_C_4.0.4_server_regular	$L^{\#}$	4.93	626.48	169.71	3151.88	215.60	4163.67
	$L^{\#}_{\mathrm{ADS}}$	5.52	346.25	87.92	2944.89	201.13	3580.19
$n = 9$ $k = 8$	TTT	5.53	526.40	149.94	5109.60	330.75	6116.69
	ADT	4.35	764.37	140.03	2948.31	197.35	4050.06
	RS	4.24	2660.90	245.70	2549.10	173.86	5629.56

Table 3: Benchmark results (Part 3 of 5)

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
OpenSSL_1.0.2_client_full		$L^{\#}$	7.90	1233.59	307.07	26255.54	1824.25	29620.45
		$L^{\#}_{\text{ADS}}$	7.82	690.52	164.74	25024.71	1744.82	27624.79
$n = 9$	$k = 10$	TTT	5.98	735.55	204.92	21819.77	1537.01	24297.25
		ADT	6.24	965.68	198.12	25715.06	1772.96	28651.82
		RS	7.05	6788.28	562.07	22996.72	1565.32	31912.39
GnuTLS_3.3.12_client_full		$L^{\#}$	8.24	1445.77	356.32	37675.94	2645.50	42123.53
		$L^{\#}_{\text{ADS}}$	8.17	548.64	145.02	39712.47	2776.01	43182.14
$n = 9$	$k = 12$	TTT	7.82	832.63	233.49	55244.09	3674.93	59985.14
		ADT	6.87	1005.87	214.65	38305.05	2653.84	42179.41
		RS	7.61	9013.67	735.01	43531.60	2979.22	56259.50
GnuTLS_3.3.12_server_full		$L^{\#}$	5.55	1367.74	336.92	11819.63	825.78	14350.07
		$L^{\#}_{\text{ADS}}$	5.72	499.24	133.13	9782.28	682.94	11097.59
$n = 9$	$k = 12$	TTT	5.87	791.64	224.11	12443.06	866.25	14325.06
		ADT	5.38	810.39	182.30	9095.98	646.15	10734.82
		RS	4.93	5107.98	444.35	9598.71	657.93	15808.97
learnresult_fix		$L^{\#}$	5.51	2079.48	454.47	390.57	30.01	2954.53
		$L^{\#}_{\text{ADS}}$	5.99	1509.48	297.97	444.18	34.43	2286.06
$n = 9$	$k = 15$	TTT	6.55	2308.32	499.74	678.86	49.17	3536.09
		ADT	5.81	2538.33	394.99	562.09	38.64	3534.05
		RS	3.21	4298.31	321.66	118.97	8.88	4747.82
OpenSSL_1.0.1g_client_regular		$L^{\#}$	7.67	991.53	219.06	5585.07	397.77	7193.43
		$L^{\#}_{\text{ADS}}$	8.04	467.24	105.21	5436.12	386.92	6395.49
$n = 10$	$k = 7$	TTT	5.94	636.29	154.00	4711.84	328.12	5830.25
		ADT	6.41	860.05	164.77	4908.55	347.64	6281.01
		RS	4.98	3033.12	288.41	2684.53	191.09	6197.15
OpenSSL_1.0.1l_server_regular		$L^{\#}$	6.29	905.66	220.06	21326.23	1410.80	23862.75
		$L^{\#}_{\text{ADS}}$	6.78	636.09	145.03	22191.25	1478.07	24450.44
$n = 10$	$k = 7$	TTT	6.76	600.41	158.58	21566.25	1438.12	23763.36
		ADT	6.19	754.49	147.60	23622.86	1567.63	26092.58
		RS	6.08	3955.93	364.30	20465.68	1370.96	26156.87
OpenSSL_1.0.1j_server_regular		$L^{\#}$	6.36	1059.33	241.40	24161.17	1571.78	27033.68
		$L^{\#}_{\text{ADS}}$	6.77	699.95	150.75	28120.44	1819.17	30790.31
$n = 11$	$k = 7$	TTT	7.13	705.32	175.21	30815.09	2001.78	33697.40
		ADT	6.56	898.62	158.96	25157.72	1638.99	27854.29
		RS	6.15	4808.87	402.85	28722.31	1856.74	35790.77
GnuTLS_3.3.8_client_regular		$L^{\#}$	9.54	1364.90	303.80	451324.32	27124.13	480117.15
		$L^{\#}_{\text{ADS}}$	9.56	535.86	125.91	429999.75	25849.52	456511.04
$n = 11$	$k = 8$	TTT	9.23	987.33	230.63	462575.01	27776.77	491569.74
		ADT	9.27	1148.95	205.48	466411.19	28009.34	495774.96
		RS	9.73	8984.35	770.60	451463.11	27105.14	488323.20
NSS_3.17.4_client_full		$L^{\#}$	7.31	2078.19	453.81	32498.93	2145.23	37176.16
		$L^{\#}_{\text{ADS}}$	8.34	902.46	200.46	40699.57	2705.91	44508.40
$n = 11$	$k = 12$	TTT	6.70	965.71	254.42	84864.87	5507.70	91592.70
		ADT	6.96	1051.96	221.48	105782.28	6698.76	113754.48
		RS	6.91	9754.23	796.31	60972.89	3990.11	75513.54
GnuTLS_3.3.8_server_regular		$L^{\#}$	9.25	1571.72	355.04	136623.93	8584.27	147134.96
		$L^{\#}_{\text{ADS}}$	9.35	581.97	134.48	129102.33	8128.54	137947.32
$n = 12$	$k = 8$	TTT	8.60	914.31	225.65	157839.22	9885.96	168865.14
		ADT	8.09	1104.10	205.97	145767.53	9164.61	156242.21
		RS	7.50	7770.50	634.80	67966.55	4357.96	80729.81
TCP_FreeBSD_Client		$L^{\#}$	6.92	2289.74	492.68	11267.74	778.03	14828.19
		$L^{\#}_{\text{ADS}}$	6.77	1132.21	239.65	11605.24	799.53	13776.63
$n = 12$	$k = 10$	TTT	6.78	2015.94	446.30	9501.69	653.10	12617.03
		ADT	5.85	1819.95	296.72	9543.45	654.37	12314.49
		RS	5.41	6982.82	550.39	8233.50	554.58	16321.29

Table 4: Benchmark results (Part 4 of 5)

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
TCP_Windows8_Client		$L^{\#}$	5.66	2537.64	520.59	7698.67	527.57	11284.47
		$L^{\#}_{\mathrm{ADS}}$	6.50	1504.24	279.52	9831.82	672.51	12288.09
$n = 13$	$k = 10$	TTT	7.98	2085.67	442.94	15383.47	1045.90	18957.98
		ADT	7.58	2818.29	421.76	9039.31	631.45	12910.81
		RS	6.42	9128.32	725.32	8719.31	598.27	19171.22
TCP_Linux_Client		$L^{\#}$	8.86	3095.37	646.31	41814.35	2710.85	48266.88
		$L^{\#}_{\mathrm{ADS}}$	9.47	1728.00	336.20	43672.28	2842.91	48579.39
$n = 15$	$k = 10$	TTT	8.50	3147.93	645.35	39448.46	2576.82	45818.56
		ADT	8.92	3230.09	481.57	58246.62	3776.20	65734.48
		RS	7.23	12872.18	956.92	40579.28	2644.20	57052.58
GnuTLS_3.3.8_client_full		$L^{\#}$	12.71	3695.29	790.48	2118359.37	128248.24	2251093.38
		$L^{\#}_{\mathrm{ADS}}$	12.74	1107.82	248.56	2183860.24	132530.08	2317746.70
$n = 15$	$k = 12$	TTT	12.64	2052.11	473.23	2160486.33	130945.24	2293956.91
		ADT	11.70	1969.34	378.96	2406607.40	145753.37	2554709.07
		RS	11.74	22354.29	1926.29	1891570.94	114696.03	2030547.55
OpenSSL_1.0.1g_server_regular		$L^{\#}$	8.56	1872.45	394.68	37720.47	2384.62	42372.22
		$L^{\#}_{\mathrm{ADS}}$	9.83	1161.84	235.21	38480.72	2447.14	42324.91
$n = 16$	$k = 7$	TTT	9.03	1473.41	331.58	42366.02	2690.85	46861.86
		ADT	7.61	1274.51	230.77	31678.33	2013.93	35197.54
		RS	6.29	6823.62	598.54	42989.03	2677.14	53088.33
GnuTLS_3.3.8_server_full		$L^{\#}$	12.00	3663.81	774.90	555275.27	34664.72	594378.70
		$L^{\#}_{\mathrm{ADS}}$	11.55	1123.95	248.00	472976.53	29498.93	503847.41
$n = 16$	$k = 11$	TTT	11.32	1756.95	418.20	506809.47	31735.75	540720.37
		ADT	10.77	1827.97	335.26	539188.33	33643.54	574995.10
		RS	10.15	20495.94	1617.38	346658.16	21869.93	390641.41
DropBear		$L^{\#}$	9.20	5283.63	870.11	34251.78	2166.16	42571.68
		$L^{\#}_{\mathrm{ADS}}$	11.98	3411.55	553.86	29448.29	1904.79	35318.49
$n = 17$	$k = 13$	TTT	11.86	5372.69	905.68	33573.63	2171.21	42023.21
		ADT	10.36	4122.24	617.36	25764.85	1678.13	32182.58
		RS	6.59	17666.79	1258.94	13306.88	866.19	33098.80
OpenSSH		$L^{\#}$	19.79	25058.68	3907.52	1420810.47	80721.84	1530498.51
		$L^{\#}_{\mathrm{ADS}}$	21.93	10783.46	1665.03	1517837.57	86105.79	1616391.85
$n = 31$	$k = 22$	TTT	18.48	14408.62	2487.83	2106605.28	119174.68	2242676.41
		ADT	17.53	13826.78	1874.73	1953538.60	110583.56	2079823.67
		RS	10.69	104984.85	6635.57	1747643.65	98694.50	1957958.57
model4		$L^{\#}$	19.68	28731.52	3827.40	27858.50	1325.21	61742.63
		$L^{\#}_{\mathrm{ADS}}$	22.41	18868.28	1868.44	28438.01	1368.67	50543.40
$n = 34$	$k = 14$	TTT	21.09	18864.22	2793.27	32723.10	1589.43	55970.02
		ADT	20.92	18008.41	1505.27	34826.25	1647.89	55987.82
		RS	11.08	77936.81	4842.28	8359.44	403.81	91542.34
model1		$L^{\#}$	7.78	9679.96	1308.49	10548.64	655.90	22192.99
		$L^{\#}_{\mathrm{ADS}}$	7.85	6832.78	794.71	11622.10	724.60	19974.19
$n = 35$	$k = 15$	TTT	11.77	9879.92	1681.13	11402.54	713.65	23677.24
		ADT	8.94	8410.19	1159.94	11852.78	733.03	22155.94
		RS	4.85	31513.88	2046.84	1959.90	133.42	35654.04
TCP_Windows8_Server		$L^{\#}$	23.16	27848.73	3247.90	506861.24	28256.37	566214.24
		$L^{\#}_{\mathrm{ADS}}$	24.12	17035.46	1789.66	481803.64	26983.07	527611.83
$n = 38$	$k = 13$	TTT	27.49	19998.26	2579.75	554970.20	30983.03	608531.24
		ADT	24.63	19062.25	1839.28	522854.14	29153.41	572909.08
		RS	11.22	88537.94	5074.22	140818.07	7964.42	242394.65
TCP_FreeBSD_Server		$L^{\#}$	30.41	53430.17	5201.33	786916.89	41169.88	886718.27
		$L^{\#}_{\mathrm{ADS}}$	34.67	28090.92	2528.64	880079.60	45874.76	956573.92
$n = 55$	$k = 13$	TTT	35.49	34650.97	3621.54	668327.57	35255.36	741855.44
		ADT	33.79	32968.56	2622.59	751184.96	39159.92	825936.03
		RS	14.56	176695.43	9733.47	444159.86	23210.48	653799.24

Table 5: Benchmark results (Part 5 of 5)

Model		Algo-		Learn-		Test-		
n	k	rithm	EQs	Inputs	Resets	Inputs	Resets	Total
TCP_Linux_Server		$L^{\#}$	31.89	48035.66	4795.18	821110.22	43285.08	917226.14
		$L^{\#}_{\text{ADS}}$	36.95	27730.98	2514.78	785883.53	41525.04	857654.33
$n = 57$	$k = 12$	TTT	35.24	35176.91	3759.52	794674.37	41833.81	875444.61
		ADT	34.38	29541.56	2380.81	918691.24	48389.22	999002.83
		RS	14.72	169782.62	9407.62	291075.22	15466.60	485732.06
model3		$L^{\#}$	22.24	60706.81	7347.66	43379.00	2684.39	114117.86
		$L^{\#}_{\text{ADS}}$	26.37	33990.69	3703.85	52503.04	3245.99	93443.57
$n = 58$	$k = 22$	TTT	26.24	35771.47	5205.59	58231.15	3576.05	102784.26
		ADT	26.67	33035.42	3160.37	57865.50	3561.65	97622.94
		RS	12.14	215860.78	14273.09	16613.11	1019.75	247766.73
BitVise		$L^{\#}$	33.03	44794.28	4499.48	2306551.51	120942.82	2476788.09
		$L^{\#}_{\text{ADS}}$	39.47	26910.45	2404.41	2809658.31	146435.51	2985408.68
$n = 66$	$k = 13$	TTT	42.13	43145.21	3977.16	2559975.40	134109.12	2741206.89
		ADT	37.30	27293.16	2365.21	2975447.04	153177.77	3158283.18
		RS	17.47	223860.20	14090.38	2300385.66	118905.61	2657241.85