Bachelor's Thesis Computing Science

Radboud University Nijmegen

The Lsharp Algorithm for Deterministic Finite Automata

Author:
Martijn Sanders
s1028766

First supervisor/assessor:
Prof. Dr. Frits Vaandrager

Second assessor:
Dr. Jurriaan Rot

May 29, 2024

Abstract

We present $L^{\#}$ for DFAs, an active automata learning algorithm. The $L^{\#}$ for DFAs algorithm is an adaptation of the $L^{\#}$ algorithm, originally designed for learning Mealy Machines, to learn Deterministic Finite Automata (DFAs). The adaptation primarily involves accommodating for the differences in information provided by the teacher in the MAT framework and the differences in properties of DFAs and Mealy Machines. Notably, the introduction of unknown states in the observation tree enables the algorithm to function effectively with reduced information per query compared to Mealy machines. Additionally, the concept of accepting states in DFAs introduces a new potential separating sequence, the empty string. The algorithm is implemented in Rust, by adapting the code of the original algorithm, and compared to existing active learning algorithms for DFAs. Results indicate that the $L^{\#}$ for DFAs variants using Adaptive Distinguishing Sequences (ADS) are competitive with state-of-the-art algorithms for DFAs.

Contents

Chapter 1

Introduction

Automata theory is a branch of theoretical computer science, which studies the behaviour of abstract machines. An automaton is an abstract model of a machine that performs computations on an input by shifting through states. The way the automaton shifts through the states is decided by a transition function, which at each state determines the next state on the basis of the current state and the input. Different software systems can be modeled by automata.
Automata learning focuses on understanding the behaviour of a computer system in minimum time and effort. The goal is to construct an automaton of the system under test by providing inputs and observing the outputs. This technique can be applied to systems of which the code is known (white-box) or unknown (black-box). This can be done by passive learning, collecting and analysing runs of the system. Or by active learning, where input is provided to the system and its response is evaluated. In this research we consider active learning techniques for black-box systems. Active automata learning has various applications. Active Automata Learning has been particularly effective to find bugs in protocol implementations. For this a model of a protocol implementation can be learned and then compared to the RFC (document describing the official desired behaviour of the protocol), potentially revealing inconsistencies. Another application of active automata learning is checking refactored implementations of legacy software. In this case both implementations are learned and checked if they behave the same. However interactions with the system under test often require much time, thus algorithms that require less interactions are needed.
This is where the $L^{\#}$ algorithm comes in, presented by Frits Vaandrager, Bharat Garhewal, Jurriaan Rot and Thorsten Wißmann in their paper "A New Approach for Active Automata Learning Based on Apartness*" [15], the $L^{\#}$ algorithm is an active learning algorithm for Mealy machines, which is as efficient as state-of-the-art learning algorithms. The algorithm is build on the Minimally Adequate Teacher (MAT) framework established by Dana Angluin [1]. In this model the learning algorithm (the learner) can ask two types of queries about an unknown regular language L ($\mathcal{M}$ is the minimal DFA that accepts L). The first type of query the learner can pose is a *membership query*: The learner can

choose a word $w \in I^*$ over the input alphabet and pose the query "Is the word $w \in I^*$ accepted by $\mathcal{M}$?". The teacher responds to a membership query with a simple yes or no, indicating the presence of the word w in the set $L(\mathcal{M})$. The second type of query the learner can pose to the teacher is an *equivalence query*: The learner conjectures a hypothesis $\mathcal{H}$, a DFA the learner believes describes the observed behaviour, the teacher responds with either yes, indicating that the hypothesis is correct (the language accepted by DFA $\mathcal{H}$ is equal to the language accepted by $\mathcal{M}$), or it responds with a counterexample. A counterexample is a word $\sigma \in I$ for which the accepting property of the hypothesis $\mathcal{H}$ and the DFA $\mathcal{M}$ differ.

Mealy machines are a variation of Finite State Machines (FSM), which are different from DFAs. In contrast to DFAs, Mealy machines have associated output symbols to transitions, and Mealy machines do not contain a set of accepting states. This difference gives rise to a modification in the MAT framework. Where the teacher responds to membership queries in the case of DFAs with only the accepting property of the destination state, the teacher gives the string of all output symbols in the case of a Mealy Machine. This same difference in responds holds for counterexamples. This means that the learner of a DFA obtains considerably less information about the hidden machine per query than it would in the case of a Mealy Machine. And for this reason the $L^\#$ algorithm for Mealy Machines does not work for DFAs without adaptations that accommodate for this lack of knowledge.

In this thesis we present $L^\#$ for DFAs, an adaptation of $L^\#$ such that it learns Deterministic Finite Automata. For this transformation we adapted the definitions and proofs of the original paper. Note that this thesis is not self contained and we often refer to Vaandrager et al. [15] for background information about the $L^\#$ algorithm. The adaptations to the $L^\#$ algorithm presented in this thesis are necessary to ensure that the $L^\#$ algorithm for DFAs learns the correct DFA, while still maintaining the same query and symbol complexity as the original $L^\#$ algorithm. We also adapted the implementation of $L^\#$ in Rust to an implementation of $L^\#$ for DFAs. Our experiments suggest that, like the original algorithm, the adapted algorithm is competitive with other algorithms implemented in LearnLib and AALpy. The development of a version of $L^\#$ for DFAs is interesting for various reasons. The main reason is that DFAs are fundamental in Computer Science. Another reason is that register automata, a richer modelling framework where input symbols carry data parameters, which are crucial if we want to scale automata learning to richer settings, are often defined as direct extensions of DFAs. Thus adapting $L^\#$ to DFAs will be a first step toward generalizing $L^\#$ to richer modeling frameworks.

The rest of this thesis is organized as follows. It starts with a preliminary chapter, where some concepts and definitions needed for the $L^\#$ for DFAs algorithm are introduced. This is followed by the research part of the thesis, which starts with an introduction to the Minimally Adequate Teacher (MAT) model. Then the $L^\#$ algorithm for DFAs is presented, an example run is shown, the correctness is proven and the complexity is studied. In the results

chapter the performance of an implementation of the algorithm written in Rust is compared to some of the best known active automata learning algorithms for DFAs implemented in both LearnLib and AALpy. In the second to last section there is looked at related work, here the algorithm is compared with the QSM algorithm, which is another active learning algorithm for DFAs that uses observation trees as the primary datastructure. And the thesis ends with some conclusions and suggestions for future work.

Chapter 2

Preliminaries

In this chapter and the rest of the thesis a lot of information and definitions will come from the paper "A New Approach for Active Automata Learning Based on Apartness" [15], as the algorithm presented in this thesis is an adaption of the $L^\#$ algorithm presented by Vaandrager et al. [15]. Some other definitions, like the definition of a DFA come from the book "Introduction to Automata Theory, Languages, and Computation" [4]. A DFA is a finite-state machine that recognizes a certain language. A language is a subset of all possible words of an alphabet. This is formally defined below. The primary datastructure of the $L^\#$ algorithm is a partial DFA. The notation for partial maps is fixed here.

Definition 2.1 (Partial Map)
*A **partial map** $f\colon X \rightharpoonup Y$ is a function from X to Y where f is not necessarily defined for all of its input values.*
f defined on x: $f(x)\downarrow \iff \exists y \in Y : f(x) = y$
f undefined on x: $f(x)\uparrow \iff \nexists y \in Y : f(x) = y$

An undefined value in a partial transition map represents lack of knowledge. The hidden DFA is complete, so if a value is undefined, this value can be obtained from the teacher. The composition of partial maps $f\colon X \rightharpoonup Y$ and $g\colon Y \rightharpoonup Z$ is denoted by $g \circ f\colon X \rightharpoonup Z$ and $(g \circ f)(x)\downarrow \iff f(x)\downarrow$ and $g(f(x))\downarrow$.

Definition 2.2 (Alphabet)
*An **alphabet** I is a finite set of letters. Sequences of these letters are **words**. The set of all possible words from the alphabet I is denoted by I^*. We use ϵ to denote the empty word, which is always in I^*.*

Definition 2.3 (Language)
*A **language** L over an alphabet I is a subset of I^*, that is $L \subseteq I^*$.*

A Deterministic Finite Automaton (DFA) is defined as follows.

Definition 2.4 (DFA)
*A **Deterministic Finite Automaton** $\mathcal{M}$ over an alphabet I is a tuple $\mathcal{M} = (Q^{\mathcal{M}}, q_0^{\mathcal{M}}, F^{\mathcal{M}}, \delta^{\mathcal{M}})$, where*

- $Q^{\mathcal{M}}$ *is a finite set of* ***states***
- $q_0^{\mathcal{M}} \in Q^{\mathcal{M}}$ *is the* ***initial state***
- $F^{\mathcal{M}} \colon Q^{\mathcal{M}} \rightharpoonup \mathbb{B}$, *where* $\mathbb{B} = \{0, 1\}$, *is an* ***accepting state (final state) function****. A state $q \in Q^{\mathcal{M}}$ is accepting (final) whenever $F^{\mathcal{M}}(q) = 1$.*
- $\delta^{\mathcal{M}} \colon Q^{\mathcal{M}} \times I \rightharpoonup Q^{\mathcal{M}}$ *is a* ***transition function***

Intuitively, a DFA works as follows: at any time a DFA $\mathcal{M}$ is in a state $q^{\mathcal{M}} \in Q^{\mathcal{M}}$, starting in the initial state $q_0^{\mathcal{M}}$. When it reads an input symbol $i \in I$, $\mathcal{M}$ shifts to the successor state $q'^{\mathcal{M}} = \delta^{\mathcal{M}}(q^{\mathcal{M}}, i)$, determined by the transition function $\delta^{\mathcal{M}}$. The transition function takes a state $q \in Q$ and an input $i \in I$ and generates the resulting state. The transition function is generalized to input words of arbitrary length by composing δ with itself:

$$\delta^{\mathcal{M}}(q, \epsilon) = q$$
$$\delta^{\mathcal{M}}(q, w \cdot a) = \delta^{\mathcal{M}}(\delta^{\mathcal{M}}(q, w), a) \; \forall q \in Q^{\mathcal{M}}, w \in I^*, a \in I$$

A DFA $\mathcal{M}$ accepts a word $w \in I^*$ if and only if $\mathcal{M}$ ends up in an accepting state after processing w, $F^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, w)) = 1$. The language of an automaton is the set of words that it accepts, this is formally defined as:

Definition 2.5 (Language of a DFA)
The ***language of a state*** $q \in Q^{\mathcal{M}}$ *is:*
$L(q) = \{w \in I^* | F^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, w)) = 1\}$.
The ***language of a DFA*** $\mathcal{M}$ *is:*
$L(\mathcal{M}) = L(q_0^{\mathcal{M}}) = \{w \in I^* | F^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, w)) = 1\}$.

Definition 2.6 (Completeness)
An automaton $\mathcal{M}$ is ***complete*** *if $\delta^{\mathcal{M}}$ and $F^{\mathcal{M}}$ are total, i.e. $\forall q \in Q^{\mathcal{M}}$, $\forall i \in I$ $\delta^{\mathcal{M}}(q, i)\downarrow$ and $F^{\mathcal{M}}(q)\downarrow$.*

Note that in this thesis we assume that the hidden DFA of the teacher is complete.

Definition 2.7 (Semantics)
The ***semantics*** *of a state $q \in Q$ is a map $[\![q]\!] \colon I^* \rightharpoonup \mathbb{B}$, indicating the language it accepts, defined by $[\![q]\!](\sigma) = F(\delta(q, \sigma))$ States q, q' in possibly different DFAs are* ***equivalent****, written $q \approx q'$, if $[\![q]\!](\sigma) = [\![q']\!](\sigma)$ $\forall \sigma \in I^*$. DFAs $\mathcal{M}$ and $\mathcal{N}$ are* ***equivalent*** *if their respective initial states are equivalent: $q_0^{\mathcal{M}} \approx q_0^{\mathcal{N}}$.*

In this thesis the maps between DFAs that are considered preserve existing accepting properties and transitions and possibly extend the knowledge of the transitions. This is more formally put below.

Definition 2.8
For DFAs $\mathcal{M}$ 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*

- $f(q_0^{\mathcal{M}}) = q_0^{\mathcal{N}}$
- $F^{\mathcal{M}}(q)\downarrow \implies F^{\mathcal{M}}(q) = F^{\mathcal{N}}(f(q))$
- $\delta^{\mathcal{M}}(q,i)\downarrow \implies \delta^{\mathcal{M}}(q,i) = \delta^{\mathcal{N}}(f(q),i)$

The datastructure of the $L^{\#}$ algorithm is an observation tree. This observation tree contains all observed behaviour during learning. An observation tree is comparable to a tree datastructure, where each vertex has at most n successors, with n the size of the input alphabet. The root of the tree is the initial state and there are only transitions that go further down the tree, an observation tree does not have loops, this means that for every state in the observation tree there exists a unique path/sequence to arrive at this particular state. In the original algorithm, that learns Mealy Machines, the observation tree is a partial Mealy Machine. To adapt the $L^{\#}$ algorithm to work for DFAs, the observation tree is adapted such that it is a partial DFA. A state $t \in Q^{\mathcal{T}}$ is an accepting (not-accepting) state whenever the sequence of inputs leading to t is known to be accepted (not accepted). Aside from accepting and non-accepting states, an observation tree can additionally contain states for which it is not known if the state (that it corresponds to in the hidden DFA) is accepting or not, for these states $q \in Q^{\mathcal{T}}$ the accepting state function is not yet defined ($F^{\mathcal{T}}(q)\uparrow$). For the observation trees constructed by $L^{\#}$ for DFAs the accepting property of leave nodes is known. If there exists a functional simulation from the tree to the DFA then it is an observation tree for the DFA. The $L^{\#}$ algorithm uses the observation tree to construct a hypothesis DFA $\mathcal{H}$. Now the formal definition of an observation tree is given.

Definition 2.9 (Observation Tree)
A DFA $\mathcal{T} = (Q^{\mathcal{T}}, q_0^{\mathcal{T}}, F^{\mathcal{T}}, \delta^{\mathcal{T}})$ is a ***tree*** *if for each $q \in Q^{\mathcal{T}}$ there is a unique sequence $\sigma \in I^*$ s.t. $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) = q$. This sequence of inputs leading to q is denoted as* $\mathsf{access}(q)$*. In an observation tree $\mathcal{T}$, there can be states $q \in Q^{\mathcal{T}}$ for which there is no information whether q is accepting or not ($F^{\mathcal{T}}(q)\uparrow$), we call these states* ***unknown states****. A tree $\mathcal{T}$ is an* ***observation tree*** *for a DFA $\mathcal{M}$ if there is a functional simulation $f\colon \mathcal{T} \to \mathcal{M}$.*

To distinguish states in the observation tree, the learner can analyse the tree to conclude that certain states cannot be mapped to the same states in the hidden DFA $\mathcal{M}$ by a functional simulation. For this Vaandrager et al. [15] use a concept called apartness, a constructive form of inequality, to prove that certain states are not equivalent in $\mathcal{M}$. We use this same concept, where we alter its definition slightly to accommodate for the unknown states in the observation tree.

Definition 2.10 (Apartness)
For a DFA $\mathcal{M}$, states $q, p \in Q^{\mathcal{M}}$ are said to be ***apart*** *(written $q \# p$) if there*

is some $\sigma \in I^*$ *such that* $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,\sigma))\downarrow$, $F^{\mathcal{T}}(\delta^{\mathcal{T}}(p,\sigma))\downarrow$, *and one of* $\delta^{\mathcal{T}}(q,\sigma)$, $\delta^{\mathcal{T}}(p,\sigma)$ *is accepting and the other is not accepting* ($[\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma)$). *In this case* σ *is called a* ***witness/separating sequence*** *of* $q \# p$ *and this is notated as* $\sigma \vdash q \# p$.

Note that when an input $\sigma \in I^*$ leads to an unknown state from a state $q \in Q^{\mathcal{T}}$ in the observation tree, σ can not be a witness proving apartness between state q and any other state in the tree (if $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,\sigma))\uparrow$ then $\sigma \nvdash q \# r \ \forall r \in Q^{\mathcal{T}}$).

Some properties of the apartness relation $\# \subseteq Q \times Q$ are:

- irreflexive:

$$\neg(q \# q)$$

- symmetric:

$$q \# p \implies p \# q$$

- weak co-transitive:

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

The weak co-transitivity is a weaker version of *co-transitivity*, stating that if $\sigma \vdash r \# r'$ and q has the transitions for σ and the accepting property for the destination state, then q must be apart from at least one of r and r'. The weak co-transitivity property is used by $L^{\#}$ during learning.

Lemma 2.11 (Weak co-transitivity)
In every DFA $\mathcal{M}$,

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

Proof. The witness $\sigma \vdash r \# r'$ implies that $F(\delta(r,\sigma))\downarrow$, $F(\delta(r',\sigma))\downarrow$, and $[\![q]\!](\sigma) \neq [\![p]\!](\sigma)$. Since $F(\delta(q,\sigma))\downarrow$, $\neg(r \# q) \wedge \neg(r' \# q)$ leads to the contradiction

$$[\![q]\!](\sigma) = [\![r]\!](\sigma) \neq [\![r']\!](\sigma) = [\![q]\!](\sigma)$$

□

Another thing to notice is that an accepting state and a non-accepting state are always apart.

Claim 2.12
An accepting state and a non-accepting state are apart. States $t_i, t_j \in Q^{\mathcal{T}}$ *for which* $F^{\mathcal{T}}(t_i) = 1$ *and* $F^{\mathcal{T}}(t_j) = 0$ *are apart* ($t_i \# t_j$).

Proof. For $\epsilon \in I^*$ if $F^{\mathcal{T}}(\delta(t_i,\epsilon))\downarrow$, $F^{\mathcal{T}}(\delta(t_j,\epsilon))\downarrow$ and $[\![t_i]\!](\epsilon) = 1 \neq 0 = [\![t_j]\!](\epsilon)$ $\implies t_i \# t_j$

□

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 2.13
For a functional simulation $f\colon \mathcal{T} \to \mathcal{M}$,

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

Proof. Assume $\sigma \vdash q \# p$ for $q, p \in \mathcal{T}$ $\implies$ $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,\sigma))\downarrow$, $F^{\mathcal{T}}(\delta^{\mathcal{T}}(p,\sigma))\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma)$. $\implies$ $\delta^{\mathcal{M}}(f(q),\sigma)\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) = [\![f(q)]\!]^{\mathcal{M}}(\sigma)$ and similarly $\delta^{\mathcal{M}}(f(p),\sigma)\downarrow$ and $[\![p]\!]^{\mathcal{T}}(\sigma) = [\![f(p)]\!]^{\mathcal{M}}(\sigma)$.
$\implies$ $[\![f(q)]\!]^{\mathcal{M}}(\sigma) = [\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma) = [\![f(p)]\!]^{\mathcal{M}}(\sigma)$
$\implies$ $[\![f(q)]\!]^{\mathcal{M}} \neq [\![f(p)]\!]^{\mathcal{M}}$ $\implies$ $f(q) \not\approx f(p)$ □

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

2.1 Example of Apartness in the Observation Tree

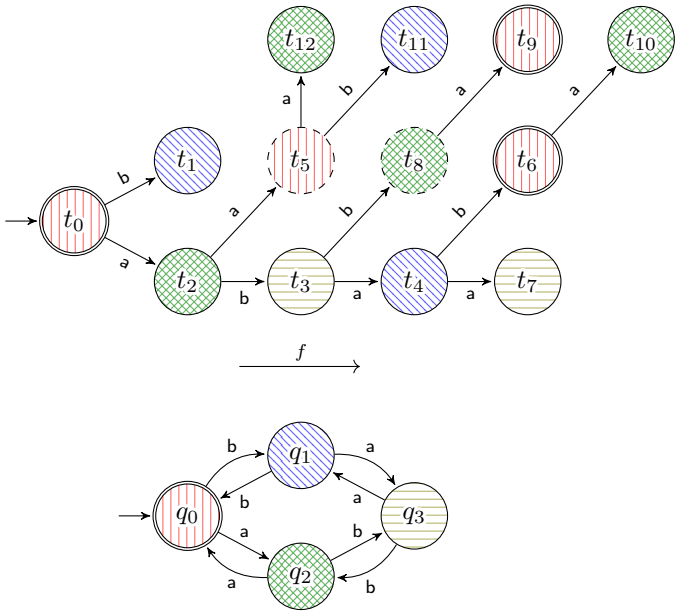


Figure 2.1: An observation tree (above) for a DFA (below).

Figure 2.1 shows an observation tree for the DFA displayed below. The functional simulation f is indicated via coloring of the states. Accepting states

are indicated by the double circle and unknown states are indicated by the dashed circle.

An observation tree $\mathcal{T}$ for the hidden DFA $\mathcal{M}$ of the teacher can be constructed by the learner by performing membership and equivalence queries. With every membership query the learner can add new transitions and states to the observation tree. The functional simulation f from $\mathcal{T}$ to $\mathcal{M}$, which maps every state of the tree to a state in the hidden DFA, indicated by the colors in Figure 2.1, and every transition in the tree to a transition in the hidden DFA, is unknown to the learner. However, the observation tree itself is known by the learner, and the $L^{\#}$ algorithm uses apartness to distinguish certain states in the tree.

For the observation tree of Figure 2.1 we may derive i.a. the following apartness pairs and corresponding witnesses:

$$\epsilon \vdash t_0 \# t_2 \qquad \epsilon \vdash t_0 \# t_3 \qquad \epsilon \vdash t_0 \# t_4$$

$$a\,b \vdash t_2 \# t_3 \qquad b \vdash t_2 \# t_4 \qquad b\,a \vdash t_3 \# t_4$$

$$a \vdash t_5 \# t_8 \qquad a \vdash t_6 \# t_8$$

Note that although the accepting property of a state can be unknown, we can sometimes deduct that this state is apart from other states by using apartness.

The apartness pairs shown in this example show that the states t_0, t_2, t_3 and t_4 are pairwise apart. This means that using this observation tree the learner can already conclude that the hidden DFA has to contain at least four states.

Chapter 3

Research

In this Chapter the main contribution of the thesis is presented: the $L^\#$ algorithm is adapted to $L^\#$ for DFAs, a learning algorithm that correctly learns any regular set from any minimally adequate teacher. All adaptions and the $L^\#$ algorithm for DFAs are described in this part of the thesis. This chapter starts with a brief introduction to active automata learning in the Minimally Adequate Teacher (MAT) model, this model was established by Dana Angluin [1]. There are some common assumptions stated and the problem is formally defined. After this the $L^\#$ algorithm for DFAs is described.

3.1 Active Automata Learning in the MAT Framework

In this thesis we explore Active Automata Learning in the MAT framework, this framework was established by Angluin in 1987 [1]. In active automata learning there is a learning algorithm (the learner) who has the goal to determine a hidden DFA $\mathcal{M}$ (an unknown regular language L) over a known alphabet I. To accomplish this it is assumed that the unknown regular language is presented to the learner by a minimally adequate teacher (MAT), which can answer two types of queries about the language. The first type of query the learner can pose is a *membership query*: The learner can select a word $w \in I^*$ and pose the query "Is word w in $L(\mathcal{M})$?", this is answered by the teacher with a simple yes or no, indicating the presence of the word w in the set $L(\mathcal{M})$. As described by both Angluin [1] and Isberner [7], when the teacher can only answer membership queries, it is generally not possible for the learner to determine the correct DFA in a polynomial number of membership queries. Furthermore, when using only membership queries the learner can never be certain that the correct DFA is learned. This is due to the fact that the number of posed membership queries is finite, therefore there are at any point infinitely many DFAs whose answers to these membership queries are consistent with the observations. Consequently, Angluin proposed that a minimally adequate teacher must also respond to a

second type of query. The second type of query a learner can pose to the teacher is an *equivalence query*: The learner conjectures a hypothesis DFA $\mathcal{H}$ to an *Equivalence Query* and the teacher responds with either yes, when the hypothesis is correct (the language accepted by DFA $\mathcal{H}$ is equal to the language accepted by $\mathcal{M}$, $L(\mathcal{H}) \approx L(\mathcal{M})$), or it gives a counterexample. A counterexample is a word $\sigma \in I$ for which the accepting property of the hypothesis $\mathcal{H}$ and the DFA $\mathcal{M}$ differ ($[\![q_0^{\mathcal{M}}]\!]^{\mathcal{M}}(\sigma) \neq [\![q_0^{\mathcal{H}}]\!]^{\mathcal{H}}(\sigma)$). A counterexample is thus presented by the teacher and the learner is not in control over the counterexamples it receives. Note that in the case of an incorrect conjecture, any of the possible counterexamples may be given by a minimally adequate Teacher, and that different counterexamples may influence the performance of the learner.

After a counterexample, the learner knows the hypothesis is not correct and can alter the hypothesis such that it is never a hypothesis again. For the remainder of this thesis input alphabet I and the hidden DFA $\mathcal{M}$ are fixed.

The MAT model, with the availability of both the membership queries and the equivalence queries forms the foundation of an algorithmic framework, where almost all active automata learning algorithms are based on. This framework is referred to as the "learning loop" by Malte Isberner [7]. This learning loop is shown in Algorithm 1 (also from Isberner [7]). Firstly the learner constructs the initial hypothesis using only membership queries. This initial hypothesis is then posed as an equivalence query to the teacher. If the teacher responds with success, the learner is done and returns the hypothesis $\mathcal{H}$. If a counterexample is provided to the learner, the algorithm ends up in a loop, where it can alter the hypothesis $\mathcal{H}$ using the provided counterexample and additional membership queries, to again pose an equivalence query with a hypothesis $\mathcal{H}'$, until an equivalence query indicates success.

This shows that an algorithm with a "learning loop" only terminates when the learner has found the correct DFA. The definition of the equivalence queries guarantees the correctness of the result of the algorithm upon termination. This means that correctness of the algorithm is proven by showing that the algorithm terminates.

Algorithm 1 The "learning loop"

Require: Access to a MAT with a target DFA $\mathcal{M}$
Ensure: Hypothesis $\mathcal{H}$ satisfying $\mathcal{H} \approx \mathcal{M}$
  Build initial hypothesis $\mathcal{H}$ using membership queries
  while EquivQuery ($\mathcal{H}$) does not indicate success
    Let $w \in I^*$ be the provided counterexample
    Refine $\mathcal{H}$ using membership queries, taking w into account
  end while
  return Final Hypothesis $\mathcal{H}$

The $L^\#$ for DFAs algorithm is build on this framework and is a strategy to formulate a hypothesis that indicates success. The queries that the learner can pose to the teacher are more formally defined below.

MemberQuery(σ): For $\sigma \in I^*$, the teacher replies with yes if $\sigma \in L(\mathcal{M})$ and no otherwise. The output of the teacher is of the form $o \in \mathbb{B} = \{0, 1\}$:

$$F^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma)) \in \mathbb{B}$$

Note that we assume the hidden DFA $\mathcal{M}$ to be complete, thus the answer of the teacher is always yes or no and never not defined.

EquivQuery($\mathcal{H}$): For a complete DFA $\mathcal{H}$, the teacher replies `yes` if $\mathcal{H} \approx \mathcal{M}$ (the language accepted by $\mathcal{H}$ and $\mathcal{M}$ are the same) or `no`, providing a *counterexample*, some $\sigma \in I^*$ that is in the language of $\mathcal{H}$ and not in the language of $\mathcal{M}$ or the other way around ($[\![q_0^{\mathcal{M}}]\!]^{\mathcal{M}}(\sigma) \neq [\![q_0^{\mathcal{H}}]\!]^{\mathcal{H}}(\sigma)$).

3.2 Structuring the Observation Tree

The datastructure of the $L^\#$ algorithm is an observation tree, for the $L^\#$ for DFAs algorithm we use the same datastructure with a few adaptions to make it work for DFAs. Here we shortly explain the observation tree datastructure of the $L^\#$ algorithm and explain the adaptions to make it work for DFAs. An observation tree for the hidden DFA $\mathcal{M}$ is of the form: $\mathcal{T} = (Q^{\mathcal{T}}, q_0^{\mathcal{T}}, F^{\mathcal{T}}, \delta^{\mathcal{T}})$. The observation tree $\mathcal{T}$ contains all information gained from queries so far, the learner possesses no additional information about the hidden DFA and it's separate states. So the apartness relation between states is solely based on the information in the observation tree. The observation tree $\mathcal{T}$ of $L^\#$ initially consists of an initial state, this is the root of the tree. For $L^\#$ for DFAs the acceptance property of this root state is initially unknown. During execution the algorithm builds $\mathcal{T}$, extending the tree with every membership and equivalence query. The input σ for a membership query is added as a path or extension of a path in the observation tree, with the query response indicating whether the last state in this path is accepting or not. The input word σ of the membership query can be of arbitrary length and thus could potentially exceed the length of the path it extends in the tree with a difference greater than one.

The output provided by the teacher for a membership query consists only of the accepting property of the last state of the input word, this implies that there may be multiple states between the states in the observation tree and this last state for which the answer to the query does not provide any information.

This means that we will add states to the observation tree for which the accepting property is not known, the unknown states ($q \in Q^{\mathcal{T}}$ unknown $\iff F^{\mathcal{T}}(q)\uparrow$).

Similarly, every negative response to a EquivQuery provides a counterexample that is introduced as a path in the observation tree. The acceptance status of the last state along this path is determined by whether the last hypothesis $\mathcal{H}$ posed in the equivalence query accepts the counterexample or not; specifically,

the counterexample is accepted by the hypothesis if and only if it is not accepted by the hidden DFA $\mathcal{M}$. For all other states along the counterexample's path, the teacher's response does not yield any information. Consequently, counterexamples can add unknown states to the observation tree in the same way membership queries can. When a state is unknown in the observation tree and the response to a query gives information about the accepting property of this state, this new information is also added to the tree. This is done by updating the acceptance property of the state from unknown to the provided acceptance property. From the observation tree, the algorithm can construct a DFA $\mathcal{H}$ to conjecture to an equivalence query. By constantly adding information to the observation tree $L^\#$ works towards an observation tree, that when converted with a functional simulation to a DFA, is equivalent to the hidden DFA $\mathcal{M}$.

The $L^\#$ for DFAs algorithm structures the observation tree in the same three sets as the original algorithm, with only one additional constraint. The additional constraint is in the fact that the set of basis states can not contain any unknown states. The sets are defined below.

1. The set of states $S \subseteq Q^{\mathcal{T}}$, which already have been fully identified, states for which the tree contains information that proofs they have to represent distinct states in the hidden DFA. This implies that all states in S are pairwise apart: $\forall p, q \in S, p \neq q \colon p \# q$. The set S is called the *basis*. Initially, $S := \{q_0^{\mathcal{T}}\}$, and throughout the execution S can be extended, and forms a sub tree of $\mathcal{T}$. The $L^\#$ for DFAs algorithm ensures that at any point in execution the acceptance property is defined for all states in the basis ($\forall q \in S \; F^{\mathcal{T}}(q)\!\downarrow$).

2. The states $F \subseteq Q^{\mathcal{T}}$, the *frontier*, the set of immediate non-basis successors (neighbors) of the basis states. The frontier is the set from which the next node to be added to S is chosen.
   $F \;:=\; \{q' \in Q \setminus S \mid \exists q \in S, i \in I : q' = \delta(q, i)\}$.

3. The remaining states $Q \setminus (S \cup F)$.

Lemma 3.1
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 $\mathcal{T}'$.

Proof. $p \# q$ in $\mathcal{T} \implies \exists \sigma \in I^*$ s.t. $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q, \sigma))\!\downarrow$, $F^{\mathcal{T}}(\delta^{\mathcal{T}}(p, \sigma))\!\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma)$
And $\mathcal{T} \subseteq \mathcal{T}' \implies F^{\mathcal{T}'}(\delta^{\mathcal{T}'}(q, \sigma))\!\downarrow$, $F^{\mathcal{T}'}(\delta^{\mathcal{T}'}(p, \sigma))\!\downarrow$ and $[\![q]\!]^{\mathcal{T}'}(\sigma) \neq [\![p]\!]^{\mathcal{T}'}(\sigma)$
$\implies p \# q$ in $\mathcal{T}'$ □

So during the execution of $L^\#$ the observation tree and the apartness relation grow.

3.3 Hypothesis Construction

The learner builds a hypothesis DFA to check if the hidden DFA is correctly learned. This is done by posing an equivalence query to the teacher containing this hypothesis. The learner constructs this hypothesis solely on the information in the observation tree $\mathcal{T}$, this can be done at almost any point. The $L^\#$ for DFAs algorithm constructs the hypothesis in the same way as the $L^\#$ algorithm for Mealy Machines, the only difference is in the fact that the states in the basis can be accepting and not accepting, and this property has to be represented in the hypothesis as well. As in the original algorithm, the set of states of a Hypothesis $\mathcal{H}$ is the the set of states for which the learner knows that it must be distinct states in the hidden DFA, this is the basis $S \subseteq \mathcal{T}$ ($Q^{\mathcal{H}} := S$). Where the property of being accepting or not is transferred to $\mathcal{H}$ for every base state, i.e. a state in the hypothesis is accepting if and only if the corresponding state in the observation tree is accepting ($\forall f(q) \in Q^{\mathcal{H}}\ F^{\mathcal{T}}(q) = F^{\mathcal{H}}(f(q))$). Note that $L^\#$ for DFAs ensures that for all states in the basis the acceptance property is known. The hypothesis should also contain transitions between the states. This is done analogous to $L^\#$. So the hypothesis contains the transitions between basis states in the observation tree $\mathcal{T}$ and the transitions in $\mathcal{T}$ from basis to frontier states in T are set by finding a base state for each frontier state, such that they are not apart in the tree. These ideas are formally defined as follows, and equal to $L^\#$, where only the constraint that the states in the hypothesis and the basis have the same accepting property is added. For a more elaborate explanation of this definition we refer to Vaandrager et al. [15].

Definition 3.2
Let $\mathcal{T}$ be an observation tree with basis S and frontier F.

1. *A DFA $\mathcal{H}$ **contains the basis** if $Q^{\mathcal{H}} = S$ where $F^{\mathcal{H}}(q^{\mathcal{H}}) = F^{\mathcal{T}}(q^{\mathcal{T}})$ and $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \mathsf{access}(q^{\mathcal{T}})) = q^{\mathcal{H}}$ for all $q^{\mathcal{T}} \in S$.*
2. *A **hypothesis** is a complete DFA $\mathcal{H}$ containing the basis such that $p' = \delta^{\mathcal{H}}(q, i)$ in $\mathcal{H}$ $(q \in S)$ and $p = \delta^{\mathcal{T}}(q, i)$ in $\mathcal{T}$ imply $\neg(p \# p')$ (in $\mathcal{T}$).*
3. *A hypothesis $\mathcal{H}$ is **consistent** if there is a functional simulation $f : \mathcal{T} \to \mathcal{H}$.*
4. *For a DFA $\mathcal{H}$ 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 $\mathcal{T}$).*

For a hypothesis to exist there can not be any frontier state that is apart from all basis states, because if such a frontier state would exists it could not be mapped to a basis state with a functional simulation. For a hypothesis to be unique the observation tree has to contain all transitions leaving the basis states and all frontier states can only have one basis state that it is not apart from, because then there is only one functional simulation possible. The criteria are formally defined below.

Definition 3.3 (Isolated)
*A state $p \in F \subset \mathcal{T}$ is **isolated** if it is apart from all states in $S \subset \mathcal{T}$ ($\forall q \in S$ $q \# p$).*

Definition 3.4 (Identified)
*A state $p \in F \subset \mathcal{T}$ is **identified** if it is apart from all states in $S \subset \mathcal{T}$ except for one ($\exists! q \in S$ such that $\neg(q \# p)$, $\forall q' \in S$ with $q' \neq q \implies q' \# p$).*

Definition 3.5 (Complete)
*The basis S is **complete** if each state in S has a transition for each input in I ($\forall q \in S\ \forall i \in I\ \delta^{\mathcal{T}}(q, i)\downarrow$).*

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.

The expanding property of the observation tree $\mathcal{T}$, which is achieved by adding all received information to it, implies that the hidden DFA is learned when the basis S is complete, contains the same number of states as the hidden DFA, and all frontier states are identified. This is formalised in the next theorem 3.7.

Theorem 3.7
Suppose $\mathcal{T}$ is an observation tree for a (hidden) DFA $\mathcal{M}$ 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 $\mathcal{H}$.

3.4 Main Loop of the Algorithm

In this section the main loop of $L^{\#}$ is adapted, such that it works for DFAs. The main loop of $L^{\#}$ for DFAs has four rules, similar to the original $L^{\#}$ algorithm. Because the observation tree of the $L^{\#}$ for DFAs algorithm can contain unknown states, which do not exists in the $L^{\#}$ algorithm for Mealy Machines, and we assume that the hidden DFA is complete (among other things this means that for every state the acceptance property is defined), we have to adapt the rules of $L^{\#}$ to ensure that the hypothesis does not contain unknown states. For the reason that the states in the hypothesis are the states in the basis of the observation tree, this adaption comes down to make sure that the basis will not contain unknown states when the hypothesis is build. In the $L^{\#}$ for DFAs algorithm we establish this by guaranteeing that the basis will never contain unknown states. States are only added to the basis by the first rule (R1), by modifying (R1) $L^{\#}$ for DFAs assures that when a state is added to the basis the acceptance property is known. The adapted (R1) comes down to:

(R1) Promotion: If there is an isolated state contained in the frontier F, this state is apart from all states in the basis S and therefore must represent a newly discovered state from the hidden DFA not yet present in S, therefor it is moved from F to S. If the acceptance property of the state is unknown, a membership query is done to obtain this information. If multiple frontier states are isolated, one of them is chosen arbitrarily.

The rules (R2), (R3) and (R4) can stay the same. Rule two (Extension) extends the frontier by adding a non-existing transition from the basis to a new frontier state. Rule three (Identification) extends the apartness relation, it adds at least one new apartness pair between a frontier state and a base state. The fourth rule (Equivalence), when applicable, builds a hypothesis and checks whether this hypothesis is correct, if not a counterexample is provided and the apartness relation is extended. If the hypothesis is correct, the algorithm has found the correct DFA and terminates. For a more formal and detailed definition of rules (R2), (R3) and (R4), we refer to Vaandrager et al. [15]. These rules are applied non-deterministically until none of them can be applied anymore. The $L^{\#}$ for DFAs algorithm is presented in pseudocode in Algorithm 2, where the Dijkstra's guarded command notation is used to show that the rules are applied non-deterministically.

Algorithm 2 Overall $L^{\#}$ for DFAs algorithm

procedure LSharp for DFAs
  do q isolated, for some $q \in F \rightarrow$ ▷ Rule (R1)
    if $F^{\mathcal{T}}(q)\uparrow$ **then**
      MemberQuery(access(q))
    end if
    $S \leftarrow S \cup \{q\}$
  [] $\delta^{\mathcal{T}}(q, i)\uparrow$, for some $q \in S, i \in I \rightarrow$ ▷ Rule (R2)
    MemberQuery(access(q) i)
  [] $\neg(q \# r)$, $\neg(q \# r')$, for some $q \in F$, $r, r' \in S$, $r \neq r' \rightarrow$ ▷ Rule (R3)
    $\sigma \leftarrow$ witness of $r \# r'$
    MemberQuery(access(q) σ)
  [] F has no isolated states and basis S is complete $\rightarrow$ ▷ Rule (R4)
    $\mathcal{H} \leftarrow$ BuildHypothesis
    $(b, \sigma) \leftarrow$ CheckConsistency($\mathcal{H}$)
    if $b =$ `yes` **then**
      $(b, \rho) \leftarrow$ EquivQuery($\mathcal{H}$)
      if $b =$ `yes` **then: return** $\mathcal{H}$
      else: $\sigma \leftarrow$ shortest prefix of ρ such that $\delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma) \# \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$ (in $\mathcal{T}$)
    end if
    ProcCounterEx($\mathcal{H}, \sigma$)
  end do
end procedure

The algorithms CheckConsistency($\mathcal{H}$), ProcCounterEx($\mathcal{H}, \sigma$) and BuildHypothesis introduced by Vaandrager et al. [15] are used by $L^{\#}$ for DFAs, and for now are assumed to be correct and work as described below. Each of them is explained and proven later in the thesis That is BuildHypothesis chooses one of the possible hypotheses (3.6), CheckConsistency($\mathcal{H}$) checks if

the hypothesis $\mathcal{H}$ is consistent with the observation tree, and if not, provides $\sigma \in I^*$ leading to a conflict (3.10), and ProcCounterEx$(\mathcal{H}, \sigma)$, where $\mathcal{H}$ contains the basis and σ leads to a conflict, extends the observation tree $\mathcal{T}$ such that $\mathcal{H}$ can never be a hypothesis again (3.11). We prove the correctness of $L^\#$ for DFAs later in the thesis by showing that the algorithm terminates.

3.5 Example Run of $L^\#$ for DFAs

In this example run of the $L^\#$ algorithm for DFAs, the algorithm learns a 4-state DFA, that only accepts words over the input alphabet with an even number of both a's and b's (Figure 3.1). Be aware of the fact that this is one of the potential runs of the algorithm for this DFA, as the rules can be applied in arbitrary order and the counterexamples are randomly generated by the teacher.

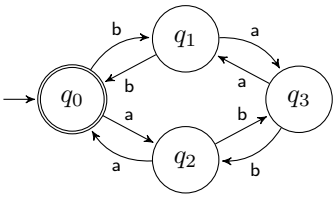


Figure 3.1: The hidden DFA.

The algorithm constructs an observation tree $\mathcal{T}$, which initially consists of a single state for which the accepting property is not yet known. During execution of the algorithm the observation tree is gradually extended. The final tree is shown in figure 3.5.

1. Initially, the root of the tree is the only state in the tree and thus apart from all other states. Rule 1 (R1) is applied to promote this state to the basis, because the accepting property of the initial state S_0 is unknown, a membership query for the empty word is posed to the teacher. This is answered by the teacher with 'yes', so the initial state is accepting. And (R1) promotes S_0 to the basis.

2. Rule 2 (extension) is applied twice to explore basis state S_0, by posing two membership queries, this leads to two new frontier states in the observation tree $\mathcal{T}$, S_1 with access sequence a, that is not accepting and S_2 with access sequence b, that is also not accepting.

3. Rule 1 (promotion) is applied on S_1. State S_1 is apart from S_0, because their accepting property differ and thus S_1 is apart from all basis states, so S_1 is moved from the frontier F to basis S.

4. Rule 2 (extension) is applied twice to explore new basis state S_1. Two membership queries are posed to the teacher, starting with the access

sequence for state S_1 (a), followed by an input symbol for which the path is not yet in the tree. The first membership query is aa, this is answered with 'yes' indicating the state reached by this input is accepting, this new accepting frontier state S_3 is again added to observation tree $\mathcal{T}$. The second one is ab, which is replied with 'no' and is the next not-accepting state S_4 added to the frontier of $\mathcal{T}$.

5. The basis is complete and the frontier contains no isolated states. In fact, all frontier states have been identified. Therefore, the learner applies rule 4 (equivalence), it builds the first hypothesis $\mathcal{H}_1$ (Figure 3.2) from the observation tree and poses an equivalence query to the teacher.

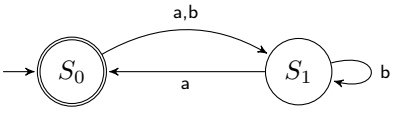


Figure 3.2: The first hypothesis.

6. In this specific run the teacher responds with the counterexample $aabb$, which is accepted by the teacher and rejected by the hypothesis. The counterexample is added to the observation tree. In $\mathcal{T}$ the state reached by aa was already known to be accepting, the counterexample provided extends this path with bb, which means that the accepting property of the state S_6 $aabb$ will be accepting, but their is no information about the accepting property of state S_5 aab, this state will be added as an unknown state to $\mathcal{T}$ (unknown states are indicated with a dashed circle).

7. Counterexample processing, which we will not explain in detail here, leads to another membership query bb. This brings the conflict back to state S_2. State S_2 is apart from both S_0 and S_1, the witness for this, that is provided by the counterexample, is b.

8. State S_2 is an isolated frontier state, hence rule 1 (promotion) is applied which moves S_2 from the frontier to the basis.

9. Rule 2 (extension) is applied to explore basis state S_2, because state S_7 with access sequence bb is already added to the observation tree by the counterexample processing algorithm, only a membership query for ba is posed to the teacher. This is answered with 'no', so a not-accepting state S_8 is added to the tree.

10. Rule 3 (identification) is applied twice to identify both frontier states S_4 and S_8, which are both not-accepting states, so by definition already apart from S_0, so the witness b for $S_1 \# S_2$ is used to identify the frontier states. The membership query bab is posed to the teacher to identify state S_8, which is answered with 'no'. Not-accepting state S_9 is added to the tree

and thus $S_8 \# S_2$. The membership query *abb* is posed to the teacher to identify state S_4, which is answered with 'no' as well. So not-accepting state S_{10} is added to the tree and thus $S_4 \# S_2$.

11. The learner now applies rule 4 (equivalence), because the basis is complete and the frontier contains no isolated states. In fact all frontier states have been identified. It builds hypothesis $\mathcal{H}_2$ (Figure 3.3) from the observation tree and poses an equivalence query to the teacher.

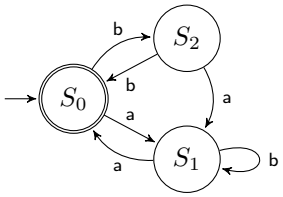


Figure 3.3: The second hypothesis.

12. The teacher responds to this equivalence query with the counterexample *baa*, which is not-accepted by the hidden DFA, but is accepted by $\mathcal{H}_2$. Not-accepting state S_{11} with access sequence *baa* is added to the tree. And it is obvious that the conflict occurs in frontier state S_8.

13. Rule 1 (promotion) is applied to add frontier state S_8 to the basis.

14. Rule 3 (identification) is applied three times to identify frontier states S_4, S_9 and S_{11}. For S_4 separating sequence a is used, thus membership query *aba* is posed and adds a not-accepting state S_{12} to the tree, and makes S_4 only identifiable with state S_8. For S_9 the same separating sequence a is used and MemberQuery(*baba*) adds an accepting state S_{13} with access sequence *baba* to the tree, which makes S_9 apart from all basis states except S_0. Membership query *baaa* is used to identify S_{11}, this adds a not-accepting state S_{14} to the tree and S_{11} is not identified yet as it is not apart from both basis state S_2 and S_8. Rule 3 (identification) is applied again to identify S_{11}, where the witness b for $S_2 \# S_8$ is used. The teacher replies 'yes' to the membership query *baab* and accepting state S_{15} is added to the tree, which makes $S_{11} \# S_8$.

15. Rule 4 (equivalence) is applied once more to make a hypothesis $\mathcal{H}_3$ (Figure 3.4). Which confirms that the hypothesis and the hidden DFA are equivalent.

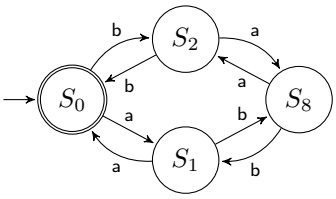


Figure 3.4: The last hypothesis.

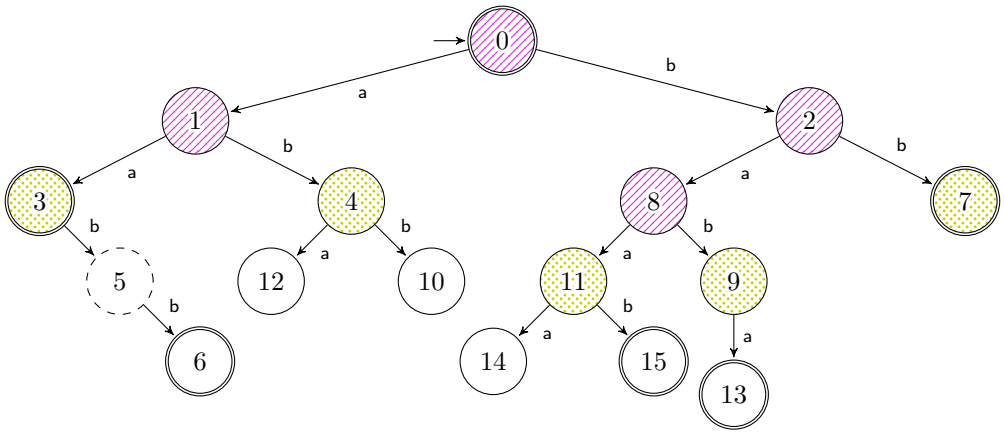


Figure 3.5: Final observation tree

3.6 Termination of the Algorithm

This section is dedicated to show that the algorithm will terminate. The learner will have the correct DFA when $L^{\#}$ terminates, because the algorithm only terminates when the teacher indicates that the hypothesis in the equivalence query is equivalent to the hidden DFA. So the correctness of the algorithm comes down to proving that the algorithm terminates. The $L^{\#}$ algorithm does this by showing that the four rules of $L^{\#}$ (R1) (introduced in section 3.4), (R2), (R3) and (R4) (introduced in Vaandrager et al. [15]) extend the basis S, the frontier F, or the apartness relation $\#$ restricted to $S \times F$ of the observation tree, while these three are all bounded by the hidden DFA $\mathcal{M}$. To prove this, Vaandrager et al. [15] introduce a norm $N(\mathcal{T})$, that is defined as follows.

$$N(\mathcal{T}) = \frac{|S| \cdot (|S|+1)}{2} + |\{(q,i) \in S \times I \mid \delta^{\mathcal{T}}(q,i)\downarrow\}| + |\{(q,q') \in S \times F \mid q \# q'\}| \tag{3.1}$$

Vaandrager et al. [15] show that every rule application increases the norm

$N(\mathcal{T})$, since $L^\#$ for DFAs uses an adapted version of (R1), while the other rules stay the same, we only have to show that the new (R1) still increases the norm.

The (R1) used for $L^\#$ for DFAs moves a state from F to S and possibly poses a membership query for this state, with this it extends the basis S, and thus increases the first summand. The application of (R1) reduces the number of apartness pairs between basis states and frontier states and thus reduces the third summand, but because the first summand in quadratic in the number of base states (R1) still increases the norm. The additional membership query that $L^\#$ for DFAs potentially poses may give new apartness pairs between this new base state and frontier states and thus increase the third summand again. In the end (R1) for DFAs increases the norm. The rules (R2), (R3) and (R4) stay the same as for the $L^\#$ algorithm and thus also increase the norm, so $L^\#$ for DFAs, like $L^\#$, increases the norm with each rule. We refer to Vaandrager et al. [15] for an explanation on how (R2), (R3) and (R4) increase the norm.

Theorem 3.8
Every rule application in $L^\#$ increases the norm $N(\mathcal{T})$ in (3.1).

The norm $N(\mathcal{T})$ is bounded by the norm of the hidden DFA (Theorem 3.9). Since every rule application increases the norm, the number of rule applications is bounded as well.

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

The algorithm only terminates when the hidden DFA is found. The $L^\#$ algorithm for DFAs will terminate for the same reasons that $L^\#$ terminates. That is $L^\#$ for DFAs can always apply one of the rules (R1), (R2), or (R4) and thus never halts, and the number of rule applications is bounded. When the norm $N(\mathcal{T})$ reaches the bound of the hidden DFA, only rule (R4) is applicable and the hypothesis queried by (R4) at this point will be accepted by the teacher. This implies that the learner discovered the hidden DFA within $\mathcal{O}(k \cdot n^2)$ rule applications. The complexity in terms of the input parameters is studied in Section 3.10.

3.7 Consistency Checking

The $L^\#$ algorithm checks the consistency of a hypothesis $\mathcal{H}$ with their CheckConsistency($\mathcal{H}$) algorithm. If the hypothesis $\mathcal{H}$ is consistent with the observation tree, i.e. there exists a functional simulation from the observation tree to the hypothesis, the CheckConsistency($\mathcal{H}$) algorithm returns yes and the $L^\#$ algorithm poses the EquivQuery for $\mathcal{H}$ to the teacher. If a functional simulation $f : \mathcal{T} \to \mathcal{H}$ does not exist, a word $\sigma \in I^*$ leading to a conflict is provided by the consistency check algorithm, without the use of a membership or equivalence query to the teacher. For $L^\#$ for DFAs we use this algorithm to check the consistency of the hypothesis, since it can be directly converted to

work for the $L^\#$ for DFAs algorithm we refer to Vaandrager et al. [15] for a more detailed description. The algorithm is shown in Algorithm 3.

Lemma 3.10
The algorithm that checks if hypothesis $\mathcal{H}$ is consistent with observation tree $\mathcal{T}$ 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 $\mathcal{T}$).*

Algorithm 3 Check if hypothesis $\mathcal{H}$ is consistent with observation tree $\mathcal{T}$

```
procedure CheckConsistency(H)
    Q ← new queue ⊆ S × S
    enqueue(Q, (q_0^T, q_0^H)))
     while (q, r) ← dequeue(Q)
        if q # r then: return no: access(q)
        for all δ^T(q, i)↓ in T do
            enqueue(Q, (δ^T(q, i), δ^H(r, i)))
        end for
     end while
    return yes
end procedure
```

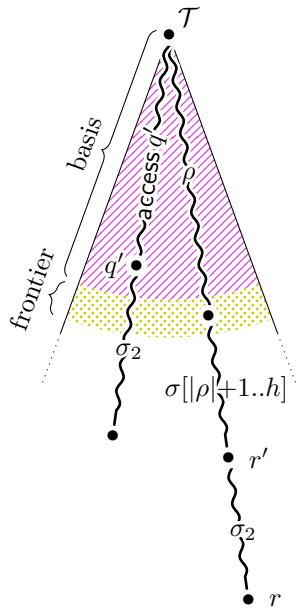
3.8 Counterexample Processing

When a counterexample is obtained, the $L^\#$ algorithm processes this to find the frontier state in which the conflict occurs. This conflict indicates that the functional simulation send one of the frontier states to a basis state in $\mathcal{H}$, while they are distinct states in the hidden DFA, causing a wrong transition in $\mathcal{H}$. To process σ, a recursive procedure is constructed, where the length of σ is reduced using binary search. When counterexample processing is finished $\mathcal{H}$ will not be a hypothesis for the observation tree anymore. Counterexample processing in $L^\#$ requires $\mathcal{O}(\log m)$ queries to analyze a counterexample of length m. For a more precise explanation of the procedure for counterexample processing we refer to Vaandrager et al. [15], their algorithm is used for $L^\#$ for DFAs. The algorithm for counterexample processing is shown in pseudo-code in Algorithm 4.

Lemma 3.11
Suppose basis S is complete, $\mathcal{H}$ is a complete DFA containing the basis, and $\sigma \in I^*$ leads to a conflict. Then ProcCounterEx($\mathcal{H}, \sigma$) *terminates and is correct, that is after terminating the hypothesis $\mathcal{H}$ cannot be a hypothesis for the observation tree $\mathcal{T}$ again. To accomplish this the algorithm poses at most $\mathcal{O}(\log_2 |\sigma|)$ membership queries.*

Algorithm 4 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 $q \# r$
    MemberQuery(access(q') σ_2 η)
    if $q' \# r'$ **then**
      ProcCounterEx ($\mathcal{H}$, σ_1)
    else
      ProcCounterEx ($\mathcal{H}$, access(q') σ_2)
    end if
  end if
end procedure



3.9 Adaptive Distinguishing Sequences

Adaptive Distinguishing Sequences (ADS) are dynamically constructed sequences of inputs designed to efficiently differentiate between distinct states of the target automaton. The $L^{\#}$ algorithm uses adaptive output queries, which actually are decision graphs (not a predetermined sequences), where after the initial input the received outputs decide the path in the graph and thus the next input, to reduce the number of queries in practice.

To use these adaptive queries in $L^{\#}$ for DFAs we have to extend the MAT framework. This extension of the framework amounts to the teacher being able to provide the learner with more information. So should the teacher answer a membership query with not only a simple 'yes' or 'no' indicating the acceptance of the word, but return the whole string of accepting properties for each state of the path. This not only applies for answering membership queries, but also returning the whole string of accepting properties of a counterexample. Note that when the MAT framework is extended as described above, unknown states are eliminated from the observation tree all together.

With this the observation tree of $L^{\#}$ for DFAs is essentially the same as the observation tree that is used by $L^{\#}$ for Mealy Machines. This means that when we extend the framework further like Vaandrager et al. do in the $L^{\#}$ paper, we can use ADS to extend rules (R2) and (R3) in the same way as it is done by the

$L^\#$ algorithm to maximize the expected number of apartness pairs per query. This further extension entails that we assume that the teacher can output the accepting property of a state, then receive additional input, transition to the subsequent state, and output the acceptance property of that state, allowing for arbitrary iterations. For a more detailed description of adaptive distinguishing sequences and how the $L^\#$ algorithm constructs an optimal ADS from the observation tree we refer to Vaandrager et al. [15] and to the example in section 3.9.1.

Proposition 3.12
$L^\#_{\text{ADS}}$ for DFAs is defined by replacing the membership queries in rule 2 and rule 3 of the $L^\#$ for DFAs algorithm with:

(R2') MemberQuery(access(q) i Ads(S)) *in (R2)*

(R3') MemberQuery(access(q) Ads($\{b \in S \mid \neg(b \# q)\}$)) *in (R3).*

In $L^\#$ for DFAs when (R2) or (R3) is applied the norm increases, this holds for $L^\#_{\text{ADS}}$ when (R2') and (R3') are applied as well. This means that $L^\#_{\text{ADS}}$ terminates and therefor is correct.

When the learning framework cannot be extended such that the teacher can answer equivalence queries with the whole string of accepting properties for each state of the path. However the teacher is able to output the accepting property of a state receive additional inputs, transition to the subsequent states, and output the acceptance property of each of these destination states. Note that the observation tree can contain unknown states in this scenario. Then $L^\#$ for DFAs can use a combination of both ADS and separating sequences. It can build the adaptive query as normal, where transitions to unknown states are seen as not existing transitions in the tree. This means that paths with unknown states are cut short and thus apartness pairs might be lost. This thus gives the possibility that there is no possible ADS in the tree that guarantees a new apartness pair and thus would possibly not increase the norm. When this is the case, the ADS tree is not used for the membership query and (R2)/(R3) is used instead of (R2')/(R3').

3.9.1 ADS Example

To make the concept of adaptive distinguishing sequences in $L^\#$ for DFAs more visual, a practical example is elaborated. Consider the observation tree of Figure 3.6(left). The basis for this tree consists of $|\{t_0, t_1, t_2, t_3\}| = 4$ states, which are pairwise apart (separating sequences are a, aa and ab). This leads to the frontier $F = \{t_4, t_5, t_7, t_9, t_{11}\}$. The expected award of the basis of this tree is $E(\{t_0, t_1, t_2, t_3\}) = 3$ and Ads(U) Figure 3.6(right) is the constructed decision tree over U. With this single decision tree the frontier states can be identified. The ADS starts with input a. If the response is 'no' ($0 \in \mathbb{B}$) then the frontier state is either t_0 or t_2, and the state is identified with a subsequent input a. Similarly, if the response to the initial input a is 'yes' ($1 \in \mathbb{B}$) then the

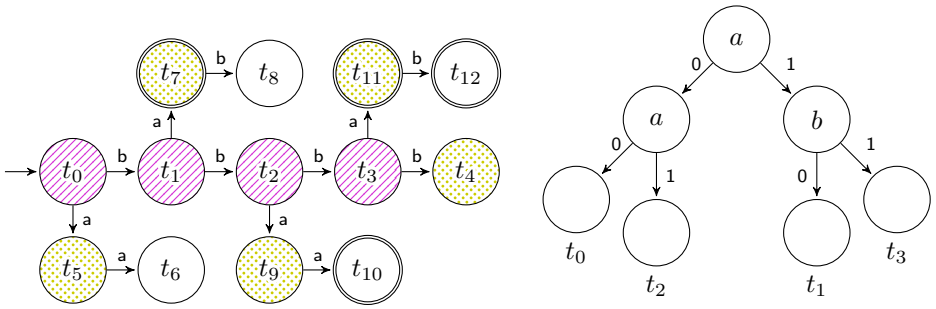


Figure 3.6: An observation tree (left) and an ADS for its basis (right)

frontier state is either t_1 or t_3, and the state is identified by a subsequent input b. So frontier state t_4 can be identified with a single (extended) membership query that starts with the access sequence for t_4 ($bbbb$) followed by the ADS of Figure 3.6(right). If separating sequences where used, at least 2 membership queries would be needed. This is an adapted version of an example found in the $L^\#$ paper [15].

3.10 Complexity

In this section the complexity of the $L^\#$ for DFAs algorithm is reported. But first the manner in which the complexity of an active learning algorithm in the MAT framework can be computed is elaborated. The complexity of $L^\#$ for DFAs is given in 3.10.2.

3.10.1 Complexity Measures in the MAT Framework

The complexity of an active automata learning algorithm is generally computed in terms of the query complexity, that is the asymptotic number of membership and equivalence queries needed by the learner to construct the correct model. Whereas the complexity of other algorithms is normally computed in time spent on computations. This choice is motivated by the fact that, in active automata learning algorithms, computation time is usually insignificant compared to the time needed for queries. Isberner et al. [7] showed that looking only at the query complexity is in some applications insufficient. As in numerous real-world applications of active automata learning, the cost of a membership query is linear in the length of the word.

Consequently, the following complexity measures are used to evaluate the performance of a learning algorithm.

- *Membership Query Complexity* is the total number of membership queries posed by the algorithm.

- *Equivalence Query Complexity* is the total number of equivalence queries posed by the algorithm.
- *Symbol Complexity* is the total number of symbols contained in all membership and equivalence queries.

Where these complexity measures can be dependent on the following input parameters: the number of states of the hidden DFA $\mathcal{M}$ (n), the size of the input alphabet (k) and the length of the longest counterexample returned by the teacher to an equivalence query (m).

3.10.2 The Complexity of $L^\#$ for DFAs

As the $L^\#$ for DFAs algorithm is very close to the original $L^\#$ algorithm, the complexity of both algorithms is the same. This holds true even though learners in the MAT framework for DFAs receive less information per query compared to those for Mealy machines. Similarly to $L^\#$ for Mealy Machines, we can define a strategic $L^\#$ for DFAs that postpones equivalence queries as long as possible to minimize the number of posed equivalence queries in practice.

Definition 3.13 (Strategic $L^\#$ for DFAs)
Strategic $L^\#$ for DFAs *(resp. $L^\#_{\text{ADS}}$ for DFAs) is the special case of Algorithm 2 where rule (R4) is only applied if none of the other rules is applicable.*

This gives the same query complexity for the $L^\#$ for DFAs algorithm as for the $L^\#$ algorithm, which is equal to the query and symbol complexities of the best known active learning algorithms, such as Rivest & Schapire's algorithm [13, 14], the observation pack algorithm [5], the TTT algorithm [8, 7], and the ADT algorithm [3].

Theorem 3.14 (Query Complexity)
Strategic $L^\#$ for DFAs (resp. $L^\#_{\text{ADS}}$ for DFAs) learns the correct DFA within $\mathcal{O}(kn^2 + n \log m)$ membership queries and at most $n - 1$ equivalence queries.

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

3.11 Experimental Evaluation

In this chapter we present the results of an evaluation of an implementation of $L^\#$ for DFAs [1]. The implementation of $L^\#$ for DFAs was written in rust and is an adaption of the $L^\#$ implementation as presented by Vaandrager et al. [15]. Here we run three different versions of the $L^\#$ for DFAs algorithm, the standard $L^\#$ for DFAs algorithm (2), the $L^\#$ for DFAs algorithm using adaptive queries ($L^\#_{ADS}$ for DFAs) and the $L^\#$ for DFAs version that uses the combination of

[1] https://gitlab.science.ru.nl/sanders/thelsharpalgorithmfordfas

both adaptive queries and separating sequences.

Target Models
The models that were learned in this thesis are randomly generated DFAs. This is mainly due to the absence of realistic DFA benchmark models. The random DFAs are generated by AALpy [10], a fixed model size of 60 is chosen and the input alphabet size is varied between 2 and 100, (2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 60, 80, 100). For each alphabet size five DFAs were randomly generated.

Equivalence Queries
A random word oracle was used to test the equivalence of the system under learning with the hypothesis. This generates a word from the input alphabet of random length and than checks if the hypothesis and the SUL responses are equal. If the responses are different, the counterexample is given to the learner. If the responses are the same the process of generating a word of random length over the input alphabet and checking the responses is repeated for a fixed number of times. If no counterexample is found within this fixed quantity the DFAs are said to be equivalent. In our test setup, the length of a word was chosen arbitrarily between 5 and 300. And the maximum number of checks performed was 10,000.

Metrics
The figures labeled 3.7 and 3.8 respectively show the average number of queries and symbols required by the learner for each alphabet size, based on testing five DFAs with that particular alphabet size.

Comparison
The implementations of $L^\#$ for DFAs in rust are compared to algorithms for DFAs implemented in AALpy [10]: The L^* algorithm using Rivest and Schapire's counterexample processing [13, 14], and Kearns and Vazirani's algorithm [9]. As well as two algorithms implemented in LearnLib [6]: Kearns and Vazirani's algorithm [9] and the TTT algorithm [7, 8].

3.11.1 Results and Discussion

The results of the experiments are displayed in figure 3.7 and figure 3.8. In figure 3.7 the mean number of queries is plotted against the alphabet size, and in figure 3.8 the mean number of symbols is plotted against the alphabet size. In terms of the membership queries used for learning, the $L^\#$ for DFAs algorithm using only separating sequences seems to be comparable with the Kearns and Vazirani's algorithm [9] of AALpy and the Kearns and Vazirani's algorithm [9] of LearnLib [6], as well as the TTT algorithm [7, 8] implemented in LearnLib [6]. For these four algorithms there does not seem to be a lot of difference in performance on these benchmarks. Only the Rivest and Schapire's Algorithm [13, 14] seems to perform poorer than the others. Both the $L^\#$ for DFAs algorithm using the combination of ADS and Separating sequences and the one using only ADS

seem to consistently need less membership queries then all the other algorithms tested. This seems to imply that most of the information in the tree is close to the frontiers and that enough information is thus also available for Adaptive Distinguishing Sequences that can not search down a path with unknown states. Note that both these algorithms have an advantage over the others, in the sense that their teacher provides more information per query due to the extended framework. This also seems to imply that the $L^{\#}$ algorithm [15] when converted to work for DFAs stays competitive with state-of-the-art learning algorithms.

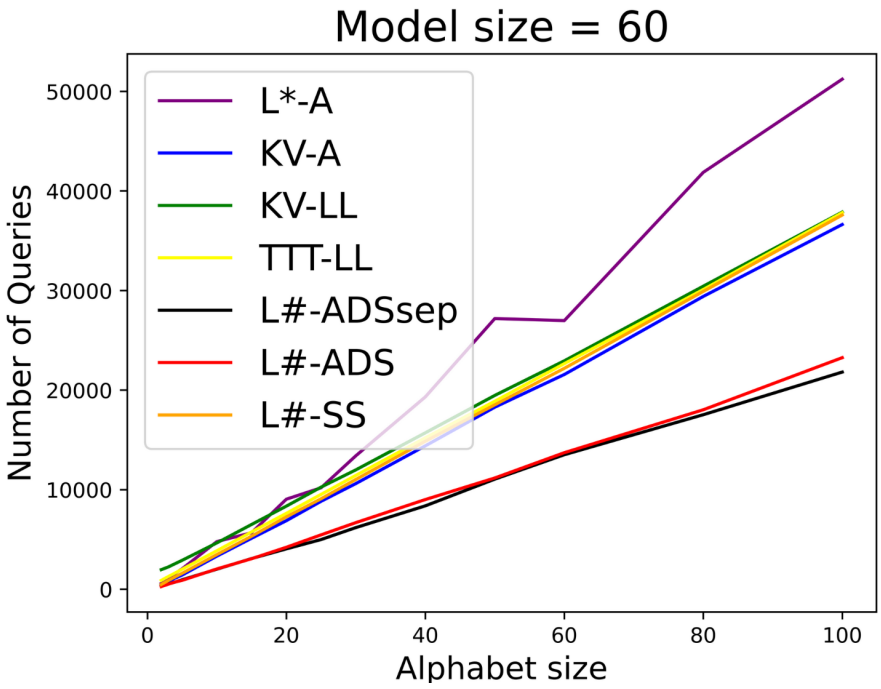


Figure 3.7: Membership queries used during learning

In terms of the number of symbols used for learning, the results in figure 3.8 show something different. All algorithms seem to be pretty competitive with each other. And both Kearns and Vazirani's algorithm [9] and L^* Algorithm with Rivest and Schapire's counterexample processing [13, 14] implemented in AALpy do a lot better than expected.

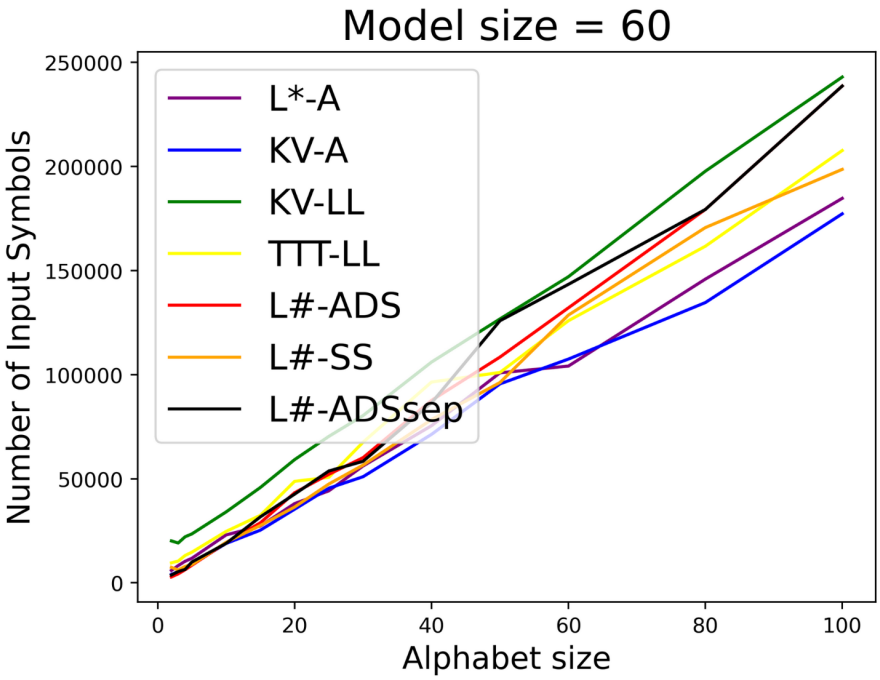


Figure 3.8: Symbols used during learning

Chapter 4

Related Work

One active learning algorithm that is comparable with the $L^\#$ algorithm for DFAs is the Query-driven State Merging (QSM) algorithm [2]. The datastructure of the QSM algorithm, the Prefix Tree Acceptor (PTA), is similar to the observation tree used by $L^\#$ for DFAs. The PTA is a tree containing all obtained words that are accepted by the hidden DFA, all states between the root and this accepting state are assumed to be accepting as well. This means that the tree in the QSM algorithm only contains accepting states, which is different from the observation tree in $L^\#$ for DFAs. The QSM algorithm then tries to generalize the tree by merging two different states of the tree, the QSM algorithm checks the compatibility of the merging by looking at the negative examples (words that are rejected by the hidden DFA) it has observed. As $L^\#$ for DFAs, QSM maintains a set of states that is learned with certainty already, the basis, in QSM the consolidated states (or red states when blue-fringe is used), the states for which no compatible merging can be found between any of its predecessor states. The QSM algorithm with the blue-fringe extension also maintains a set of blue-fringe states (frontiers), the direct successors of the red states, which are first considered for merging. As in $L^\#$ for DFAs, when a blue state is found incompatible with all red states it is promoted to be a red state and the blue-fringe is updated and the process is iterated. When an intermediate automaton is compatible with the available scenarios, a membership query is done to obtain additional information about the new automaton. Although there are similarities between to two algorithms, there are some substantial differences as well. The QSM algorithm for instance, uses no equivalence queries at all, the convergence of the algorithm to the correct automaton is instead guaranteed when the learner receives a sample as input that includes a 'characteristic sample', as defined in both the RPNI [11] and the QSM [2] algorithms. Another major difference is the fact that the QSM algorithm needs a nonempty initial collection of scenarios from which it builds the PTA. Whereas $L^\#$ for DFAs starts without any information about the model to learn and obtains all information itself with member and equivalence queries.

Other work that is related to the $L^\#$ for DFAs algorithm is "Adaptation

d'un algorithme d'apprentissage d'automates" by Zielinsky [16], where the $L^{\#}$ algorithm is adapted to work for DFAs as well. However in this work the adapted version of $L^{\#}$ seems to need more queries than the L^{*} algorithm, which is inconsistent with our findings. This could be do to the observation tree containing only accepting and not-accepting states and no unknown states. It could also be caused by some other difference in the implementation.

Chapter 5

Conclusions

In this thesis we addressed the problem of adapting the $L^\#$ algorithm that learns Mealy Machines to an algorithm that can learn Deterministic Finite Automata. The primary modification was the introduction of unknown states in the observation tree, due to the fact that the teacher provides less information per query, than in the usual setting for Mealy machines. The fact that states in DFAs have an accepting property turned out to cause only a minor change, where the empty string was introduced as a possible separating sequence. In general the adaption of the algorithm to DFAs was rather straightforward. In particular, the subroutines of the algorithm, i.e. counterexample processing, consistency checking and build hypothesis stayed essentially the same. The algorithm for DFAs is implemented in rust and compared to other active learning algorithms for DFAs. The results showed that the $L^\#$ for DFAs variant with ADS is at least competitive with the best algorithms at the moment.

5.1 Future Work

A few things that are in line with this thesis and can still be done are:

- The $L^\#$ algorithm could be generalised to richer modeling frameworks, for instance Moore machines, Nondeterministic Finite Automata, or register automata.
- The construction of Adaptive Distinguishing Sequences for observation trees with unknown states could be defined. For this the most obvious solution would be to add unknown states to both output branches in the ADS tree. For this the Expected award function should be adapted, such that the optimal tree can be found and thus increasing the norm will be guaranteed.
- The construction of Adaptive Distinguishing Sequences in general observation trees could be studied to optimize efficiency. Building all possible

ADS trees takes a lot of time and when the automata sizes increase the time to build all ADS trees will only increase more. One could for example examine the states for which the ADS tree is constructed and verify if these states are apart in the tree, proceeding with building the ADS tree only when they are.

- When comparing the time the implementation of $L^\#$ for DFAs needs for learning models to the time the algorithms implemented in AALpy and LearnLib need to learn these same models, there seems to be a big difference. So can AALpy and LearnLib learn models of sizes around 600 states in a matter of minutes, whereas the $L^\#$ implementation for DFAs used in this thesis takes hours to learn this same model. It could be interesting to look into possible explanations why the $L^\#$ for DFAs implementation needs more time.

- To improve the complexity of $L^\#$ for Deterministic Finite Automata, in the case separating sequences are used, an expected reward function could be constructed to get the maximal number of new apartness pairs per query. An example of such a function that could sort the separating sequences on number of expected apartness pairs is:

  - $E(X, \sigma) = \frac{2(|Y_\sigma^A| \times |Y_\sigma^N|)}{|Y|}$
    - $Y = \{y \in X | F^{\mathcal{T}}(\delta^{\mathcal{T}}(y, \sigma))\downarrow\}$
    - σ is the separating sequence
    - $Y_\sigma^A = \{y \in Y | F^{\mathcal{T}}(\delta^{\mathcal{T}}(y, \sigma)) = 1\}$
    - $Y_\sigma^N = \{y \in Y | F^{\mathcal{T}}(\delta^{\mathcal{T}}(y, \sigma)) = 0\}$

- The adaption of $L^\#$ to DFAs in this thesis could be compared to the adaption of Zielinsky [16], to see where the two implementations differ and why they seem to perform differently.

- There could be looked at why both algorithms implemented in AALpy seem to perform better in terms of input symbols on the random automata generated in AALpy than expected.

Bibliography

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[2] Pierre Dupont, Bernard Lambeau, Christophe Damas, and Axel Lamsweerde. The QSM algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22:77–115, 02 2008.

[3] Markus Theo Frohme. Active automata learning with adaptive distinguishing sequences. *CoRR*, abs/1902.01139, 2019.

[4] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[5] F. Howar. *Active learning of interface programs*. PhD thesis, University of Dortmund, June 2012.

[6] M. Isberner, F. Howar, and B. Steffen. The open-source learnlib - A framework for active automata learning. in *CAV, LNCS*, 9206:487–495, 2015.

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

[8] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 307–322, Cham, 2014. Springer International Publishing.

[9] Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory*. MIT Press, 1994.

[10] Edi Muškardin, Bernhard Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. Aalpy: an active automata learning library. *Innovations in Systems and Software Engineering*, 18:1–10, 03 2022.

[11] Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, 01 1992.

[12] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104, pages 167–183, 1981.

[13] R.L. Rivest and R.E. Schapire. 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*, pages 411–420. ACM, 1989.

[14] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.

[15] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022.

[16] Pierre Zielinsky. Adaptation d'un algorithme d'apprentissage d'automates. MSc Thesis, University of Mons, Belgium. 2022.

Appendix A

Appendix

The proofs presented by Vaandrager et al. [15] are used and adapted to accommodate for the necessary changes that are needed to transform the algorithm to work for DFAs.

Proof of 2.11

The witness $\sigma \vdash r \# r'$ implies that $F(\delta(r,\sigma))\downarrow$, $F(\delta(r',\sigma))\downarrow$, and $[\![q]\!](\sigma) \neq [\![p]\!](\sigma)$. Since $F(\delta(q,\sigma))\downarrow$, $\neg(r \# q) \wedge \neg(r' \# q)$ leads to the contradiction

$$[\![q]\!](\sigma)=[\![r]\!](\sigma) \neq [\![r']\!](\sigma)=[\![q]\!](\sigma)$$

Proof of 2.12

For $\epsilon \in I$ if $F^{\mathcal{T}}(\delta(t_i,\epsilon))\downarrow$, $F^{\mathcal{T}}(\delta(t_j,\epsilon))\downarrow$ and $[\![t_i]\!](\epsilon) = 1 \neq 0 = [\![t_j]\!](\epsilon) \implies t_i \# t_j$

Proof of 2.13

Assume $\sigma \vdash q \# p$ for $q,p \in \mathcal{T} \implies F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,\sigma))\downarrow$, $F^{\mathcal{T}}(\delta^{\mathcal{T}}(p,\sigma))\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma)$. $\implies \delta^{\mathcal{M}}(f(q),\sigma)\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) = [\![f(q)]\!]^{\mathcal{M}}(\sigma)$ and similarly $\delta^{\mathcal{M}}(f(p),\sigma)\downarrow$ and $[\![p]\!]^{\mathcal{T}}(\sigma) = [\![f(p)]\!]^{\mathcal{M}}(\sigma)$.
$\implies [\![f(q)]\!]^{\mathcal{M}}(\sigma) = [\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma) = [\![f(p)]\!]^{\mathcal{M}}(\sigma)$
$\implies [\![f(q)]\!]^{\mathcal{M}} \neq [\![f(p)]\!]^{\mathcal{M}} \implies f(q) \not\approx f(p)$

Proof of 3.1

$p \# q$ in $\mathcal{T} \implies \exists \sigma \in I^*$ s.t. $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,\sigma))\downarrow$, $F^{\mathcal{T}}(\delta^{\mathcal{T}}(p,\sigma))\downarrow$ and $[\![q]\!]^{\mathcal{T}}(\sigma) \neq [\![p]\!]^{\mathcal{T}}(\sigma)$
And $\mathcal{T} \subseteq \mathcal{T}' \implies F^{\mathcal{T}'}(\delta^{\mathcal{T}'}(q,\sigma))\downarrow$, $F^{\mathcal{T}'}(\delta^{\mathcal{T}'}(p,\sigma))\downarrow$ and $[\![q]\!]^{\mathcal{T}'}(\sigma) \neq [\![p]\!]^{\mathcal{T}'}(\sigma)$
$\implies p \# q$ in $\mathcal{T}'$

Proof of 3.6

This proof can stay unchanged from the proof presented in Vaandrager et al. [15], and thus we refer to the proof presented there.

Proof of 3.7

In the proof of 3.7, we characterize equivalence of DFAs via *bisimulations*.

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

$$q_0^{\mathcal{M}} \; R \; q_0^{\mathcal{N}} \text{ and } q \; R \; r \wedge q' = \delta^{\mathcal{M}}(q,i) \;\Rightarrow\; \exists r' \colon r' = \delta^{\mathcal{N}}(r,i) \wedge F^{\mathcal{N}}(r') = F^{\mathcal{M}}(q') \wedge q' \; R \; r'$$

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

Lemma A.2
Given complete DFAs $\mathcal{M}$ 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 [12], is again easy to prove.

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

We now come to the actual proof of 3.7:

Proof of 3.7. Let f be a functional simulation 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 functional simulation 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 $F^{\mathcal{H}}(\delta^{\mathcal{H}}(q,i)) = F^{\mathcal{M}}(\delta^{\mathcal{M}}(r,i))$ and $(q',r') \in R$. We consider two cases:

   (a) $\delta^{\mathcal{T}}(q,i) \in S$. Then, by construction of $\mathcal{H}$, $F^{\mathcal{H}}(\delta^{\mathcal{H}}(q,i)) = F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i))$ and $q' = \delta^{\mathcal{T}}(q,i)$. Moreover, as f is a functional simulation from $\mathcal{T}$ to $\mathcal{M}$, $f(q') = \delta^{\mathcal{M}}(f(q),i)$ and $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i)) = F^{\mathcal{M}}(\delta^{\mathcal{M}}(f(q),i))$. By definition of R, $f(q) \approx^{\mathcal{M}} r$. Hence, by Lemma A.2, $F^{\mathcal{M}}(\delta^{\mathcal{M}}(f(q),i)) =$

$F^{\mathcal{M}}(\delta^{\mathcal{M}}(r,i))$ and $\delta^{\mathcal{M}}(f(q),i) \approx^{\mathcal{M}} r'$. By combining the derived equalities we obtain:

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

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 $\mathcal{H}$, if $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i))\downarrow$ then $F^{\mathcal{H}}(\delta^{\mathcal{H}}(q,i)) = F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i))$ 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 functional simulation from $\mathcal{T}$ to $\mathcal{M}$, $f(q'') = \delta^{\mathcal{M}}(f(q),i)$ and $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i)) = F^{\mathcal{M}}(\delta^{\mathcal{M}}(f(q),i))$. By definition of R, $f(q) \approx^{\mathcal{M}} r$. Hence, by Lemma A.2, $F^{\mathcal{M}}(\delta^{\mathcal{M}}(f(q),i)) = F^{\mathcal{M}}(\delta^{\mathcal{M}}(r,i))$ and $\delta^{\mathcal{M}}(f(q),i) \approx^{\mathcal{M}} r'$. By combining the derived qualities we obtain:

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

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.
If $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i)))\uparrow$, the accepting property of the frontier state $q'' \in \mathcal{T}$ is not known, but the fact that q'' is identified and the fact that $|S|$ is the number of equivalence classes of $\approx^{\mathcal{M}}$ implies that the accepting property of q'' can only be the same as the accepting property of the unique basis state it is not apart from. (Because if the accepting properties would differ, the states are apart and 2.13 gives that q'' is another equivalence class in $\mathcal{M}$, but this gives a contradiction with $|S|$ is the number of equivalence classes of $\approx^{\mathcal{M}}$.). Thus $F^{\mathcal{T}}(q'')$ is $F^{\mathcal{T}}(s)$ for the unique state $s \in S$ which is not apart from q''. This implies that $F^{\mathcal{H}}(\delta^{\mathcal{H}}(q,i)) = F^{\mathcal{T}}(q'') = F^{\mathcal{T}}(s)$ and $F^{\mathcal{T}}(s) = F^{\mathcal{M}}(\delta^{\mathcal{M}}(q,i))$. And thus the same conclusion as for $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q,i)))\downarrow$ can be drawn. Note that this is only the case because $|S|$ = #equivalence classes M and q'' identified.

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

Proof of 3.8

This proof can stay unchanged from the proof presented in Vaandrager et al. [15], and thus we refer to the proof presented there.

Proof of 3.9

This proof can stay unchanged from the proof presented in Vaandrager et al. [15], and thus we refer to the proof presented there.

Proof of 3.10

The breadth-first search in Algorithm 3 verifies whether there is a functional simulation $f\colon \mathcal{T} \to \mathcal{H}$. Since $\mathcal{H}$ is deterministic (like all DFAs 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 (2.8).

- 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 2.13 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 $p = \delta^{\mathcal{T}}(q, i)$ in $\mathcal{T}$ and $p' = \delta^{\mathcal{H}}(f(q), i)$ in $\mathcal{H}$. Since the basis S is complete in $\mathcal{T}$, we have $\delta^{\mathcal{T}}(f(q), i)\!\downarrow$ and we have either $F^{\mathcal{T}}(\delta^{\mathcal{T}}(f(q), i))\!\downarrow$ which guarantees $F^{\mathcal{T}}(\delta^{\mathcal{T}}(f(q), i)) = F^{\mathcal{T}}(\delta^{\mathcal{T}}(q, i))$, or we have $F^{\mathcal{T}}(\delta^{\mathcal{T}}(f(q), i))\!\uparrow$ which implies that i can not be a witness for $f(q)$, so in both cases we have $i \not\Vdash q \# f(q)$. 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 3.11

Termination of the counterexample processing algorithm is proven 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 $\rho =$ unique prefix of σ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \in F$ and $h = \frac{|\rho|+|\sigma|}{2}$ defined as in Algorithm 4. The decomposition of $\sigma = \sigma_1 \cdot \sigma_2$, gives

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

  By definition of the hypothesis it holds that $q' := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1) \in S$, this gives 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, the following holds:

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

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

Let $MQ(n)$ denote the maximal number of membership queries performed during a run of Algorithm 4 with $n = d(\sigma)$. Then, using the above observations, we may show by induction on $d(\sigma)$ that

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

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

For correctness, let q and r as in Algorithm 4:
$q := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma)$
$r := \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma)$
such that $\eta \vdash q \,\#\, r$ for some $\eta \in I^*$, i.e. one of $\delta^{\mathcal{T}}(q, \eta), \delta^{\mathcal{T}}(r, \eta)$ is accepting and one is not accepting $(F^{\mathcal{T}}(\delta^{\mathcal{T}}(q, \eta)) \neq F^{\mathcal{T}}(\delta^{\mathcal{T}}(r, \eta)))$.

- 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, σ decomposes into $\sigma = \alpha\, i$ with $\alpha \in I^*$ and $i \in I$.
  Let:
  $q' = \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \alpha)$
  $r' = \delta^{\mathcal{H}}(q_0^{\mathcal{T}}, \alpha)$.
  Since $\mathcal{T}$ is a tree, it necessarily holds that $\alpha = \mathsf{access}(r')$. $(r \in S \cup F \to r' \in S)$
  Hence,
  $$q' = \delta^{\mathcal{H}}(q_0, \alpha) = \delta^{\mathcal{H}}(q_0, \mathsf{access}(r')) = r'.$$

It holds that $q = \delta^{\mathcal{H}}(q', i)$ in $\mathcal{H}$ and $r = \delta^{\mathcal{T}}(r', i)$ 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', r' as in Algorithm 4.
  $q' := \delta^{\mathcal{H}}(q_0^{\mathcal{H}}, \sigma_1) \in S$
  $r' := \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma_1)$
  After MemberQuery($\mathsf{access}(q')\,\sigma_2\,\eta$), we have $F^{\mathcal{T}}(\delta^{\mathcal{T}}(q', \sigma_2\,\eta))\!\downarrow$ 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 it necessarily holds that

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

     and thus also

     $$[\![\delta^{\mathcal{T}}(q', \sigma_2)]\!]^{\mathcal{T}}(\eta) = [\![\delta^{\mathcal{T}}(r', \sigma_2)]\!]^{\mathcal{T}}(\eta). \qquad (*)$$

     It is verified that $\mathsf{access}(q')\,\sigma_2$ can be passed to ProcCounterEx:

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

     But on the other hand:

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

     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 3.12

This proof can stay unchanged from the proof presented in Vaandrager et al. [15], and thus we refer to the proof presented there.

Proof of 3.14

Strategic $L^{\#}_{\text{ADS}}$ for DFAs makes the same amount of membership queries and equivalence queries as strategic $L^{\#}$ for DFAs, so it is sufficient to discuss strategic $L^{\#}$ for DFAs.

In the strategic $L^{\#}$ for DFAs, 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 $\mathcal{O}(n \log m)$ membership queries. By Theorem 3.8 and Theorem 3.9 there are at most $\mathcal{O}(kn^2)$ rule applications during a run of $L^\#$ for DFAs. Since applications of rule (R1) require at most one membership query, and each application of rule (R2) and (R3) requires exactly one membership query, this means that applications of rules (R1), (R2) and (R3) will require $\mathcal{O}(kn^2)$ membership queries. Altogether, $L^\#$ for DFAs will require $\mathcal{O}(kn^2 + n \log m)$ membership queries.

Proof of 3.15

This proof can stay unchanged from the proof presented in Vaandrager et al. [15], and thus we refer to the proof presented there.