# Bigger is Not Always Better: on the Quality of Hypotheses in Active Automata Learning

**Rick Smetsers** R.SMETSERS@CS.RU.NL
**Michele Volpato** M.VOLPATO@CS.RU.NL
**Frits Vaandrager** F.VAANDRAGER@CS.RU.NL
*Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands*

**Sicco Verwer** S.E.VERWER@TUDELFT.NL
*Institute for Intelligent Systems and Software, Delft University of Technology, The Netherlands*

**Editor:** Alexander Clark, Makoto Kanazawa and Ryo Yoshinaka

## Abstract

In Angluin's $L^*$ algorithm a learner constructs a sequence of hypotheses in order to learn a regular language. Each hypothesis is consistent with a larger set of observations and is described by a bigger model. From a behavioral perspective, however, a hypothesis is not always better than the previous one, in the sense that the minimal length of a counterexample that distinguishes a hypothesis from the target language may decrease. We present a simple modification of the $L^*$ algorithm that ensures that for subsequent hypotheses the minimal length of a counterexample never decreases, which implies that the distance to the target language never increases in a corresponding ultrametric. Preliminary experimental evidence suggests that our algorithm speeds up learning in practical applications by reducing the number of equivalence queries.

**Keywords:** Active learning, automata learning, distance metrics

## 1. Introduction

Automata learning techniques have become increasingly important for their applications to a wide variety of software engineering problems, especially in the analysis and testing of complex systems. Recently, they have been successfully applied for security protocol testing (Shu and Lee, 2007), for the analysis of botnet command and control protocols (Cho et al., 2010), in regression testing of telecommunication protocols (Hungar et al., 2003), and in conformance testing of communication protocols (Aarts et al., 2014).

Automata learning aims to identify an unknown target language from examples of its members and nonmembers (Gold, 1967). In *active* automata learning, introduced in the seminal work by Angluin (1987), a *learner* identifies the language with the help of an *oracle* (in contrast to *passive* learning, where the learner is provided with data). Angluin's $L^*$ algorithm is characterized by the iterative alternation between two phases. In the first phase, the learner poses *membership queries* to construct a hypothesis. In the second phase it asks an *equivalence query* to determine if the hypothesis correctly describes the language. The oracle either signals success (if the hypothesis correctly describes the language) or provides a counterexample that distinguishes the hypothesis and the language. The algorithm iterates in this way until it finds a hypothesis that correctly describes the target language. In the learning process, each successive hypothesis is described by a bigger model.

In this paper, we show that a bigger model is not always better. Different notions of quality exist for a hypothesis and we argue that a valid metric for quality should be based on its *behaviour*, i.e. the strings in its language. In systems engineering, a potential bug in the far-away future is less troubling than a potential bug today (de Alfaro et al., 2003). Based on this observation, we study a well-known metric based on minimal-length counterexamples and we show that the quality of successive hypotheses may decrease in such a setting. To correct for this, we propose a simple modification to $L^*$ that finds a counterexample at the cost of a membership query if this is the case. As a result, we make sure that each hypothesis is at least as good as the previous one, and we possibly decrease the number of equivalence queries required in the learning process. We give preliminary experimental evidence that in a realistic setting our modification speeds up learning, because in practice, equivalence queries are typically expensive to answer (Smeenk, 2012). In a case study, we show that our modification helps in learning a piece of industrial control software used in Océ printers.

Recently, the $L^*$ algorithm has been adapted for learning the behaviour of reactive systems (see Steffen et al., 2011; Shahbaz and Groz, 2009). In practice, when learning such systems, it is impossible to exhaustively search for counterexamples; we have to stop learning at some point. As a result, the oracle is not always perfect and it is possible that the final hypothesis is incorrect. Contrary to the original $L^*$ algorithm, our modification guarantees that in such a case the final hypothesis will behave correctly for at least as long as all previous hypotheses.

**Related work** Improvements of $L^*$ have been investigated before, see Balcázar et al. (1997) for an overview of early work. Optimisations for large alphabets are discussed in Irfan et al. (2012), and practical optimisations that improve both the number of queries and time required to answer them are given in Bauer et al. (2012) and Irfan et al. (2010). Distance metrics for automata have been studied extensively before, but the majority of this work has been done in a setting where statistical information is available (for an overview, see de la Higuera, 2010, chap. 5). To the best of our knowledge, no earlier work has compared successive hypotheses produced by $L^*$ from the perspective of their counterexamples. Length-bounded counterexamples have been used in Ipate (2012), and minimal-length counterexamples in Birkendorf et al. (2000) and Ibarra and Jiang (1991). These are, however, strong assumptions. Instead, our modification works in a standard $L^*$ setting.

**Outline** We recall regular languages, Angluin's $L^*$ algorithm and metric spaces in Section 2. Then, in Section 3, we define a metric for comparing languages, and we show that $L^*$ may construct hypotheses that are worse than previous ones according to this metric. Section 4 provides an algorithm that solves this problem and some preliminary experimental results. We conclude our work in Section 5.

## 2. Preliminaries

**Definitions** Let $\Sigma$ be a finite *alphabet* of *symbols*. A *string* over $\Sigma$ is a finite sequence of symbols. We denote with $\Sigma^*$ the set of all strings over $\Sigma$. The empty string is denoted by

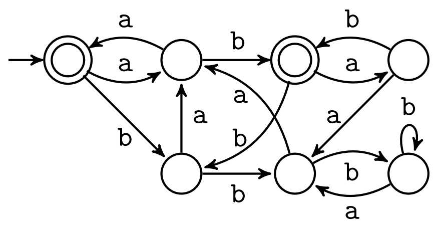


Figure 1: A canonical DFA over the alphabet $\{\mathtt{a}, \mathtt{b}\}$. Elements of $Q$ are represented by nodes, and elements of $F$ by double circle nodes. The initial state is indicated by the non-labelled arrow. An edge between states $p$ and $q$, labeled with an element $a$ of the alphabet, is present if and only if $\delta(p, a) = q$.

$\epsilon$. A *language* is a subset of $\Sigma^*$. We denote $\mathcal{L}_\Sigma$ the set of all languages over the alphabet $\Sigma$, and we shorten it to $\mathcal{L}$ if $\Sigma$ is clear or not significant in the context.

A *regular* language is any language that is accepted by some *deterministic finite automaton* (DFA), which is a tuple $A = \langle \Sigma, Q, q_0, F, \delta \rangle$, where $\Sigma$ is a finite alphabet, $Q$ is a finite set of *states*, $q_0 \in Q$ is the *initial* state, $F \subseteq Q$ is the set of *accepting* states, and $\delta : Q \times \Sigma \to Q$ is the *transition function* between states. We extend $\delta$ to $Q \times \Sigma^* \to Q$ in the usual way. The language accepted by $A$ is the set of strings $u$ such that $\delta(q_0, u) \in F$, and is denoted $L_A$. Two DFAs $A$ and $A'$ are *equivalent*, denoted $A \equiv A'$ if they accept the same language, i.e $L_A = L_{A'}$. A DFA is *canonical* if no other equivalent DFA has fewer states. In the rest of the paper, all DFAs are considered to be canonical, unless specified otherwise. Figure 1 shows an example canonical DFA that we will use throughout this paper.

We say that a string $u$ *distinguishes* $A$ and $A'$ if $u \in L_A \iff u \notin L_{A'}$. Such a string is called a *distinguishing string* for $L_A$ and $L_{A'}$. A *minimal-length distinguishing string* for two languages $L$ and $L'$ is defined as a string $v$, such that, for each string $w$, $|w| < |v|$ implies $w \in L \iff w \in L'$. There exist efficient algorithms for finding a minimal-length distinguishing string between two DFAs (see for example Dovier et al. 2001).

**Learning regular languages with $L^*$** Angluin's $L^*$ algorithm is an efficient algorithm where a *learner* identifies an unknown regular language $L$ with the help of an *oracle*. The oracle answers two types of queries about the target language. In a *membership query* the learner asks if a string is in the language. After having posed a number of membership queries, the learner constructs a canonical DFA $A_H$ that is consistent with all the replies given by the oracle so far. The language $H$ that this DFA accepts is the learner's *hypothesis* for the target language. Depending on the context, the word "hypothesis" is either used to refer to a language $H$, or to the canonical DFA that accepts this language $A_H$. In an *equivalence query* the learner asks about the correctness of $H$. The oracle replies positively if $H = L$. If this is not the case, the oracle returns a distinguishing string that shows that the hypothesis is incorrect. Such a string is called a *counterexample*. An oracle that answers these two types of queries is known as a *minimally adequate teacher*.

The $L^*$ algorithm maintains an *observation table* in which it stores the answers to all membership queries posed so far. The observation table consists of a set of *access strings* to the states of the hypothesis DFA, their *one-symbol extensions*, and a set of *distinguishing suffixes*. An observation table can be visualized as a table that has the access strings and their one-symbol extensions as its row labels and the distinguishing suffixes as its column labels. A cell has a value of 1 if and only if the concatenation of the corresponding prefix (access sequence or one-symbol extension) and (distinguishing) suffix is in the language. Otherwise, a cell has a value of 0. If all cells are filled, the table is *complete*. Example complete observation tables are shown in Table 1.

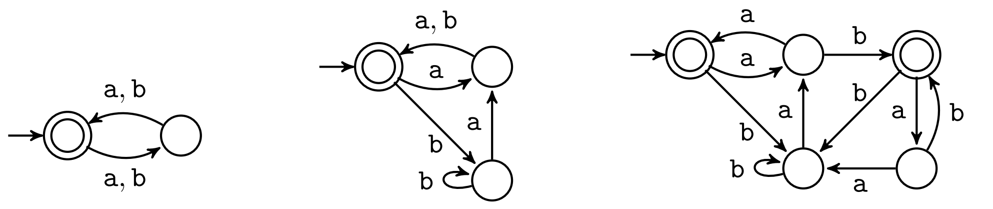
In order to construct a hypothesis DFA from the observation table, it needs to be *closed* and *consistent*. An observation table is *closed* if for each row labeled with a one-symbol extension there exists a row with an access string that has an identical value in every column. An observation table is *consistent* if for all rows labeled with access strings that have an identical value in every column, it holds that the rows labeled by their one-symbol extensions have an identical value in every row as well.

**Algorithm 1:** The $L^*$ algorithm

```
Initialize observation table;
Make table closed and consistent;
repeat
    Construct hypothesis $H$;
    Ask equivalence query for $H$, let $c$ be the response;
    if $c$ contains a counterexample then
        Handle counterexample $c$;
        Make the table closed and consistent;
    end
until $c$ does not contain a counterexample;
```

We provide a high-level description of $L^*$ (Algorithm 1). For a more detailed description we refer to de la Higuera (2010, chap. 9). First, the observation table is initialized such that the set of access strings and the set of distinguishing strings both contain $\epsilon$, and the algorithms asks membership queries for $\epsilon$ and $\Sigma$ (the one-symbol extensions of $\epsilon$). Then, the table is checked for closedness and consistency. If the table is not closed or not consistent, an element from the one-symbol extensions is added to the access strings, and its one-symbol extensions are added to the table (a new state is added). This process is repeated until the table is closed and consistent. Once the table is closed and consistent, a hypothesis DFA $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ is constructed in the following way:

- $Q$ contains exactly one state for every access sequence. This ensures that for every pair of states, the corresponding rows have different values in at least one column;
- $q_0$ is the state in $Q$ for access sequence $\epsilon$;
- $F$ contains states in $Q$ for access sequences that are in the language;
- The one-symbol extensions are used to define $\delta$, where $\delta(\delta(q_0, s), a) = \delta(q_0, t)$ if and only if (one-symbol extension) $s \cdot a$ and (access string) $t$ have identical values in every column.



(*a*) Initial hypothesis.
(*b*) Hypothesis after handling ba.
(*c*) Hypothesis after handling abaa.

Figure 2: Starting the learning process.

The hypothesis is presented to the oracle in an equivalence query. If the oracle replies positively, the learner has successfully learned the target language and the algorithm terminates. Otherwise, the oracle provides a counterexample and the algorithm modifies the observation table. There are many different strategies for handling a counterexample (see e.g. Maler and Pnueli, 1995; Rivest and Schapire, 1993; Shahbaz and Groz, 2009; Steffen et al., 2011). In this paper we use the strategy described by Steffen et al. (2011), but our contribution is independent from the one that is used. After handling a counterexample the table is not closed anymore, so an element from the one-symbol extensions is added to the access strings (a new state is added), and more membership queries are asked to obtain a new hypothesis. The algorithm iterates in this fashion until it produces a correct hypothesis. Each hypothesis in the learning process is described by a canonical DFA, and each successive hypothesis has more states than the previous one. Assume that the canonical DFA accepting the target language has $n$ states, then the algorithm clearly terminates, because the number of equivalence queries is limited by $n$. An example run of $L^*$ is described in Example 1.

**Example 1** *We show how $L^*$ constructs the first three hypotheses while learning the language described by Figure 1. We start by asking membership queries for $\epsilon$,* `a` *and* `b` *and we store the answers in an observation table. We find that the table is not closed, so we add* `a` *to $S$ and we ask membership queries for* `aa` *and* `ab`*. Then, we construct the table shown in Table 1(a) and the corresponding hypothesis shown in Figure 2(a). This hypothesis is presented as an equivalence query. The oracle replies that our hypothesis evaluates* `ba` *incorrectly. We handle this counterexample in the table and after asking more membership queries, we construct the table shown in Table 1(b) and we present the hypothesis shown in Figure 2(b) as an equivalence query. This time, the oracle presents* `abaa` *as a counterexample. We use this information to construct the table shown in Table 1(c) and the hypothesis shown in Figure 2(c).*

**Metric spaces** In order to reason about the quality of hypotheses produced by $L^*$, we need a function to compare them. A function is called a *metric*, if it satisfies the conditions in Definition 1.

Table 1: Observation tables. Rows are labeled by access strings (top) and their one-symbol extensions (bottom), columns are labeled with distinguishing suffixes.

(*a*) Initial observation table.

| | $\epsilon$ |
|---|---|
| $\epsilon$ | 1 |
| `a` | 0 |
| `b` | 0 |
| `aa` | 1 |
| `ab` | 1 |

(*b*) Observation table after handling `ba`.

| | $\epsilon$ | `a` |
|---|---|---|
| $\epsilon$ | 1 | 0 |
| `a` | 0 | 1 |
| `b` | 0 | 0 |
| `aa` | 1 | 0 |
| `ab` | 1 | 0 |
| `ba` | 0 | 1 |
| `bb` | 0 | 0 |

(*c*) Observation table after handing `abaa`.

| | $\epsilon$ | `a` | `aa` |
|---|---|---|---|
| $\epsilon$ | 1 | 0 | 1 |
| `a` | 0 | 1 | 0 |
| `b` | 0 | 0 | 1 |
| `ab` | 1 | 0 | 0 |
| `aba` | 0 | 0 | 0 |
| `aa` | 1 | 0 | 1 |
| `ba` | 0 | 1 | 0 |
| `bb` | 0 | 0 | 1 |
| `abb` | 0 | 0 | 1 |
| `abaa` | 0 | 0 | 1 |
| `abab` | 1 | 0 | 0 |

**Definition 1** *Let $X$ be a set, then a function $d : X \times X \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, is a* metric *on $X$ if:*

1. $d(x, y) = 0 \iff x = y$ *(identity);*
2. $d(x, y) = d(y, x)$ *(symmetry);*
3. $d(x, y) \le d(x, z) + d(z, y)$ *(triangle inequality).*

*A metric is an* ultrametric *if in addition it satisfies a stronger version of triangle inequality:*

4. $d(x, y) \le \max(d(x, z), d(z, y))$ *(strong triangle inequality).*

Note that any metric is nonnegative. Given a set $X$ and a metric $d$ on $X$, the pair $\langle X, d\rangle$ is called a *metric space*. If $d$ is an ultrametric on $X$, we call $\langle X, d\rangle$ an *ultrametric space*. In metric and ultrametric spaces, the function $d$ provides the set $X$ with the concept of distance between its elements. Given three elements $x, y, z \in X$, we say that $x$ is *closer* to $z$ than $y$ if $d(x, z) < d(y, z)$. In an ultrametric space $\langle X, d\rangle$, for any triple of elements $x, y, z \in X$, two of the distances among them are equal and the third one is equal or smaller (Lemma 2).

**Lemma 2** *Let $\langle X, d\rangle$ be an ultrametric space and let $x, y, z$ be elements of $X$, then:*

$$d(x, y) \neq d(y, z) \implies d(x, z) = \max(d(x, y), d(y, z)).$$

**Proof** Assume $d(x, y) > d(y, z)$. Then, because of strong triangle inequality, we obtain that $d(x, z) \le \max(d(x, y), d(y, z)) = d(x, y)$ and that $d(x, y) \le \max(d(y, z), d(x, z)) = d(x, z)$ (because $d(y, z) < d(x, y)$). Thus $d(x, y) = d(x, z)$. ■

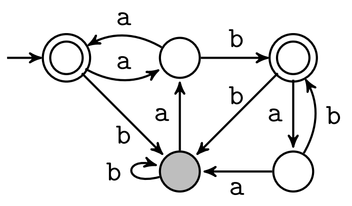
## 3. Languages in a Metric Space

In the process of learning a language, different notions of quality exist for a hypothesis. The $L^*$ algorithm guarantees that each hypothesis explains all observations from membership queries seen so far. Moreover, each successive hypothesis is based on more observations than the previous one, because a counterexample adds new information. As a result, each hypothesis has more states than the previous one (see Figure 2, for example).

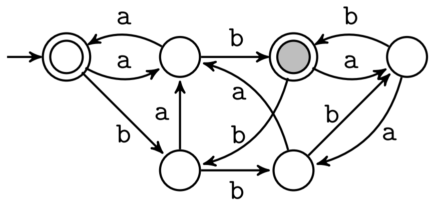
In this section, we show that a bigger model is not always better. In fact, the size difference between a hypothesis and the target DFA is not a valid metric. Let $A_H$ and $A_{H'}$ be DFAs with $m$ and $n$ states respectively, and let $H, H' \in \mathcal{L}$ be the languages that they accept. Then it is possible that $m = n$ and $H \neq H'$. Hence, a function on the number of states is not a valid metric on $\mathcal{L}$, because it does not satisfy the identity axiom of Definition 1.

We argue that a valid metric should be based on the *behaviour* of a hypothesis, in terms of the strings in its language. Assume that, with regard to such a metric, a hypothesis is more distant to the target language than the previous one. Then, if one can detect this divergence, there must be some information that can be used to improve the hypothesis, without the need of asking an equivalence query. We recall a well-known metric for comparing two languages (see de Bakker and Zucker, 1982, for example). Intuitively, this metric is based on the minimal length of distinguishing strings: languages are more distant if they are distinguished by shorter strings. Example 2 shows that in $L^*$, the minimal length of a counterexample for hypotheses may decrease. As a result, the distance to the target language increases.

**Example 2** *Let us continue the process for learning the language represented by Figure 1. The starting point is the hypothesis accepted by Figure 3(a). In this hypothesis all strings of length 4 are evaluated correctly and we are presented* `bbbaa` *as a counterexample. While handling* `bbbaa` *something peculiar happens: our new hypothesis (Figure 3(b)) incorrectly changes behaviour for* `bbbb`*. Hence, an error has been introduced, and the quality of our hypothesis has decreased.*



(*a*) Third hypothesis. The string `bbbb` is evaluated correctly.



(*b*) Fourth hypothesis. The string `bbbb` is evaluated incorrectly.

Figure 3: After handling `bbbaa` (a minimal-length counterexample), a shorter string (`bbbb`) incorrectly ends in an accepting state. Grey nodes denote the state that `bbbb` ends in.

Given two different languages $L_1$ and $L_2$, let $u$ be a minimal-length string that distinguishes them. Clearly, $u$ distinguishes any DFA accepting $L_1$ from any DFA accepting $L_2$ and vice versa. In Definition 3 we present a metric to compare $L_1$ and $L_2$ based on the length of this string. Similar metrics have been used in literature, e.g. in concurrency theory (de Bakker and Zucker, 1982). Note that, if two languages are not equal, there must exist a distinguishing string. The intuition is that, the longer this string is, the closer $L_1$ and $L_2$ are.

**Definition 3** *Let $L_1$ and $L_2$ be two languages. The ultrametric $d$ is the function*

$$d(L_1, L_2) = \begin{cases} 0 & \text{if } L_1 = L_2 \\ 2^{-n} & \text{otherwise} \end{cases}$$

*where $n$ is the minimal length of a string that distinguishes $L_1$ and $L_2$.*

**Lemma 4** *The pair $\langle \mathcal{L}, d \rangle$ with $d$ defined by Definition 3 is an ultrametric space.*

**Proof** We prove that the pair $\langle \mathcal{L}, d \rangle$ with $d$ defined by Definition 3 satisfies the three axioms of an ultrametric. Let $L_1$, $L_2$ and $L_3$ be languages in $\mathcal{L}$. Then

i) $d(L_1, L_2) = 0 \iff L_1 = L_2$: ($\Leftarrow$) by definition;
($\Rightarrow$) it must hold that $L_1 = L_2$ because $2^{-|u_{12}|}$, where $u_{12}$ is a minimal-length string that distinguish $L_2$ from $L_2$, can never evaluate to 0;

ii) $d(L_1, L_2) = d(L_2, L_1)$: the strings that distinguish $L_1$ from $L_2$ are the same that distinguish $L_2$ from $L_1$;

iii) $d(L_1, L_2) \leq \max(d(L_2, L_3), d(L_3, L_1))$: if any two among $L_1$, $L_2$ and $L_3$ are equal, then it holds trivially. Otherwise, if they are all different, then let $u_{12}$, $u_{23}$ and $u_{31}$ be minimal-length strings that pairwise distinguish them. We show that $|u_{12}| \geq \min(|u_{23}|, |u_{31}|)$ (which is equivalent to $d(L_1, L_2) \leq \max(d(L_2, L_3), d(L_3, L_1))$). By contradiction, assume $|u_{12}| < \min(|u_{23}|, |u_{31}|)$. By definition of minimal-length distinguishing strings, $\forall w$ such that $(|w| < |u_{23}|) \wedge (|w| < |u_{31}|)$ it holds that $(w \in L_2 \iff w \in L_3) \wedge (w \in L_3 \iff w \in L_1)$, in particular for $w = u_{12}$. But, then, $u_{12}$ is not a distinguishing string for $L_1$ and $L_2$ which is a contradiction.

■

In Example 2 we have used minimal-length counterexamples to illustrate that $L^*$ can produce a hypothesis that is more distant to the target language than the previous one. Unfortunately, in the process of learning an unknown target language, finding a minimal-length counterexample for a hypothesis is a difficult task. We can however make use of the strong triangle inequality property of ultrametrics (see Lemma 2) to check a hypothesis' relative distance to the target at the cost of a single membership query. By Lemma 2, for the ultrametric described in Definition 3 this means that for any triple of elements, two of the minimal-length distinguishing strings will have equal lengths and the third will be of equal or *greater* length.

**Lemma 5** *Let $\langle \mathcal{L}, d \rangle$ be the ultrametric space with $d$ defined by Definition 3. Let $L, H$ and $H'$ be languages in $\mathcal{L}$ with $H \neq H'$ and let $v$ be any minimal-length distinguishing string for $H$ and $H'$. Then:*

$$d(L, H) < d(L, H') \implies (v \in H \iff v \in L)$$

**Proof** Let $u$ and $u'$ be minimal-length strings distinguishing $L$ from $H$ and $H'$, respectively. Then by Definition 3, $d(L, H) < d(L, H')$ implies that $|u| > |u'|$. Moreover, by Lemma 2 $d(L, H) < d(L, H')$ implies that $d(L, H') = d(H, H')$ which in turn implies that $|v| = |u'|$, and hence that $|u| > |v|$. It follows that $v \in H \iff v \in L$, because $u$ is a minimal-length distinguishing string. ■

Assume that $H$ and $H'$ are successive hypotheses for $L$ and that $v$ is a minimal-length string that distinguishes the two hypotheses. Then to verify that $H'$ is at least as close as $H$ it suffices to ask a membership query for $v$. If $H$ evaluates $v$ incorrectly, then according to Lemma 5, $H'$ is at least as close as $H$. If $H$ evaluates $v$ correctly, we cannot be sure that this is the case, but we have found a new counterexample for $H'$ without asking an equivalence query. The algorithm that we present in Section 4 makes use of this result to guarantee that the distance to $L$ does not increase.

## 4. Monitoring the Quality of Hypotheses

In the previous section we have seen that $L^*$ can change the behaviour of strings that happened to be handled correctly before. As a result, the distance of a hypothesis to the target language can increase. In this section, we propose a modification to $L^*$ which guarantees that the distance to the target language does not increase in the ultrametric space $\langle \mathcal{L}, d \rangle$, with $d$ defined in Definition 3.

**Definition 6** *Let $H$ and $H'$ be successive hypotheses in process of identifying an unknown language $L$, and let $d$ be the ultrametric defined in Definition 3, then $H'$ is* stable *if and only if $d(H, L) \geq d(H', L)$.*

Our modification to the $L^*$ algorithm is shown in Algorithm 2. In the algorithm we maintain a *stable* hypothesis $H$ that is known to be the best we have seen so far according to Definition 6. The main idea is that a *candidate* hypothesis $H'$ is refined until it is at least as good as the stable hypothesis before asking an equivalence query. As a result, each stable hypothesis is at least as good as the previous one.

We start by constructing the initial stable hypothesis $H$ that we present in an equivalence query. If the oracle replies positively, the learner has successfully learned the target language and the algorithm terminates. If this is not the case we obtain a counterexample $c$, which we use to construct a candidate hypothesis $H'$. Then, we use a standard algorithm to obtain a minimal-length string $v$ that distinguishes the stable hypothesis and the candidate hypothesis.

According to Lemma 5, we are sure that the candidate ($H'$) is at least as good as the previous stable hypothesis ($H$) if $v$ is evaluated correctly by the candidate $H'$. If, however, $v$ is evaluated correctly by the previous stable hypothesis $H$ (and the condition

**Algorithm 2:** Modified $L^*$ algorithm

```
Initialize observation table;
Make table closed and consistent;
repeat
    Construct stable hypothesis $H$;
    Ask equivalence query for $H$, let $c$ be the response;
    if $c$ contains a counterexample then
        Handle counterexample $c$;
        Make table closed and consistent;
        repeat
            Construct candidate hypothesis $H'$;
            Obtain minimal-length string $v$ that distinguishes $H$ and $H'$;
            if $v$ distinguishes $H'$ and $L$ then
                Handle counterexample $v$;
                Make table closed and consistent;
            end
        until $v$ distinguishes $H$ and $L$;
    end
until $c$ does not contain a counterexample;
```

is not met), then we cannot guarantee that the candidate has improved. In this case, we have however found a counterexample ($v$) without asking an equivalence query. We use this counterexample to construct a new (and bigger) candidate $H'$ (following the standard $L^*$ procedure) and we again ask for a minimal-length string $v$ that distinguishes the new candidate from the stable hypothesis. The algorithm iterates in this way until it finds a minimal-length distinguishing string $v$ that is evaluated correctly by the candidate. In this case we break the loop, promote the candidate to stable and ask for the next equivalence query. The correctness and termination of our algorithm are proven by Theorem 7.

**Theorem 7** *The execution of Algorithm 2 terminates, and each stable hypothesis is at least as good as the previous one.*

**Proof** First, note that the inner loop handles a counterexample in the regular way. Hence, each candidate hypothesis has more states than any previous hypothesis. We show that the number of iterations in the inner loop is finite, which proves that the algorithm terminates, given the termination of the original $L^*$ algorithm.

Let $H'$ be the candidate hypothesis obtained from $H$ by handling $c$, and let $v$ be a minimal-length distinguishing string for $H'$ and $H$. Let $u$ and $u'$ be (unknown) minimal-length counterexamples of $H$ and $H'$ respectively, and note that $u$ can never be longer than $c$. Moreover, note that $v$ can never be longer than $c$ because, by construction, $c \in H \iff c \notin H'$ and $v$ is a string that distinguishes $H$ and $H'$.

If $v$ distinguishes $H$ and $L$, then $v$ is a counterexample for $H$. Hence, by contraposition of Lemma 5, $d(L, H) \geq d(L, H')$. The algorithm breaks out of the inner loop and promotes $H'$ to stable. If, instead $v$ distinguishes $H'$ and $L$, then it might be the case that $|u| > |u'|$

(and hence that $d(L, H) < d(L, H')$). Thus, given that $H'$ evaluates $v$ incorrectly, $v$ can be used as a counterexample obtaining a new hypothesis $H''$ and a new minimal-length distinguishing string $v''$ for $H''$ and $H$. Note that $v$ does not distinguish $H''$ and $L$, and hence that $v \neq v''$. Consequently, in successive loops, the algorithm never processes $v$ again, because all hypotheses are consistent with the membership queries asked so far. Given that there are a finite number of strings that are not longer than $c$, the inner loop terminates. ■

In Example 3 we show that, by using Algorithm 2, each stable hypothesis is at least as good as the previous one. As a result, we reduce the number of equivalence queries required in learning the language represented by Figure 1.

**Example 3** *Let us continue the process for learning the language represented by the DFA in Figure 1. Our starting point is the hypothesis in Figure 3(*a*). When this hypothesis is presented as an equivalence query, the oracle provides* `bbbaa` *as a counterexample. By handling this counterexample, we obtain the hypothesis in Figure 3(*b*). The original $L^*$ would present this hypothesis to the oracle in an equivalence query. Instead, Algorithm 2 first verifies that the hypothesis is at least as good as the previous one. To do so, it finds a minimal-length string (*`bbbb`*) that distinguishes it from the previous hypothesis (Figures 3(*a*)). Then, it asks a membership query to check if the behaviour has changed. The membership query returns* `false`*, and so does the previous hypothesis. Hence, an error was introduced in the current hypothesis. Instead of presenting the hypothesis as an equivalence query, it is refined by handling* `bbbb`*.*

*For the refined hypothesis, the algorithm performs another inequivalence check. This time, a minimal-length string that distinguishes the current hypothesis from the previous one is* `bbbaa`*, the initial counterexample. This string is evaluated incorrectly in the previous hypothesis (and correctly in the current one). Therefore, the algorithm exits the loop and continues learning. The current hypothesis is, presented as an equivalence query, and the oracle replies affirmatively. Algorithm 2 required four equivalence queries to learn the target language, one less than the original $L^*$ algorithm. Moreover, the algorithm guarantees that each hypothesis is at least as good as the previous one, contrary to the original $L^*$ algorithm.*

## 4.1. Experimental Results

In this section, we give preliminary experimental evidence that our algorithm speeds up learning. We have implemented Algorithm 2 on top of $L^*$ in LearnLib, a state-of-the-art tool for active automata learning (Raffelt et al., 2009; Merten et al., 2011), and we have compared the number of equivalence queries that it requires. Compared to an implementation of the original $L^*$ algorithm, Algorithm 2 requires approximately 4% less equivalence queries. The experiments were conducted on randomly generated DFAs with a varying number of states (100–2000) and alphabet (2–20).

To show that the contributions of this paper are suited for practical learning problems as well, we have applied them in a more realistic setting. Recently, automata learning techniques have become increasingly important for the construction of models for software components. Smeenk (2012) used the $L^*$ algorithm to learn the behaviour of the Engine Status Manager (ESM), a piece of industrial software that controls the transition from one status to another in Océ printers[1]. Learning the behaviour of this software is hard because of the many details involved. A key challenge that the authors faced was the task of searching for a counterexample: more than 263 million sequences of input actions were not enough to fully learn the behaviour of the system. As a result, the learning process was terminated before the correct hypothesis was found. In total, the time required for learning exceeded 19 hours, of which over 7 hours were spent on searching for counterexamples. Altogether, 131 hypotheses were generated. The partially correct final hypothesis had 3,326 states.

Using the distance metric described in Section 3 we were able to verify that the partially correct final hypothesis was the best one seen so far. However, by comparing intermediate hypotheses (Algorithm 2), we have found a counterexample for four hypotheses without having to search for them using an equivalence query. In these cases, a minimal-length distinguishing string for two successive hypotheses had incorrectly changed its behaviour. With the use of our algorithm, we would have been able to detect these mistakes. As a result, it is highly likely that our algorithm would have reduced the time required to learn the final hypothesis. This case study shows the implications of our contributions in practice: behaviour-based metrics can provide useful information about hypotheses in the learning process. Using this information, we reduce the number of equivalence queries required.

## 5. Conclusions and Future Work

Bigger hypotheses in active automata learning are not always better. In this paper, we have shown that different notions of quality exist for a hypothesis. We have argued that a valid metric should be based on the *behaviour* of a hypothesis. To the best of our knowledge, our work is the first to address the quality of hypotheses in active learning from such a solid, theoretical perspective.

Using a well-known metric based on minimal-length counterexamples, we have shown that the quality of successive hypotheses produced by $L^*$ may decrease. To correct this, we have proposed a simple modification to $L^*$ that makes sure that each hypothesis is at least as good as the previous one.

Experiments and a case study have provided preliminary evidence that our contributions are effective in practice. Moreover, they have shown that behaviour-based metrics can provide useful information about the learning process, that can be used to reduce the number of equivalence queries required in active learning.

The results of this paper may provide insights in the problem of finding counterexamples for an hypothesis. In a realistic setting, where the help of an oracle is unavailable, we have to search for counterexamples by posing membership queries. In our experiments, we have shown that a minimal-length distinguishing string for successive hypotheses has a relatively high chance to be a counterexample. In future work, we wish to investigate if other distinguishing strings are good candidates as well. A search strategy based on these strings might find a counterexample more quickly.

1. A detailed description of the case study, with models and statistics, is available at `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/ESM/`.

## Acknowledgments

This work is partially supported by NWO project 628.001.009 on Learning Extended State Machine for Malware Analysis (LEMMA) and STW project 11763 on Integrating Testing And Learning of Interface Automata (ITALIA).

## References

Fides Aarts, Harco Kuppens, Jan Tretmans, Frits Vaandrager, and Sicco Verwer. Improving active mealy machine learning for protocol conformance testing. *Machine Learning*, 96 (1-2):189–224, 2014. doi: 10.1007/s10994-013-5405-0.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106, 1987. doi: 10.1016/0890-5401(87)90052-6.

José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. Springer US, 1997. doi: 10.1007/978-1-4613-3394-4_2.

Oliver Bauer, Johannes Neubauer, Bernhard Steffen, and Falk Howar. Reusing system states by active learning algorithms. In Alessandro Moschitti and Riccardo Scandariato, editors, *Eternal Systems*, volume 255 of *Communications in Computer and Information Science*, pages 61–78. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-28033-7_6.

Andreas Birkendorf, Andreas Böker, and Hans Ulrich Simon. Learning deterministic finite automata from smallest counterexamples. *Discrete Mathematics*, 13(4):465–491, 2000.

Chia Yuan Cho, Domagoj Babić, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 426–439, New York, NY, USA, 2010. ACM. doi: 10.1145/1866307.1866355.

Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In JosC.M. Baeten, JanKarel Lenstra, Joachim Parrow, and GerhardJ. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer Berlin Heidelberg, 2003. doi: 10.1007/3-540-45061-0_79.

Jaco W. de Bakker and Jeffery I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(12):70 – 120, 1982. doi: 10.1016/S0019-9958(82) 91250-5.

Colin de la Higuera. *Grammatical inference.* Cambridge University Press, Cambridge, UK, 2010.

Agostino Dovier, Carla Piazza, and Alberto Policriti. A fast bisimulation algorithm. In Grard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 79–90. Springer Berlin Heidelberg, 2001. doi: 10.1007/3-540-44585-4_8.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. doi: 10.1016/S0019-9958(67)91165-5.

Hardi Hungar, Oliver Niese, and Bernhard Steffen. Domain-specific optimization in automata learning. In Jr. Hunt, WarrenA. and Fabio Somenzi, editors, *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 315–327. Springer Berlin Heidelberg, 2003. doi: 10.1007/978-3-540-45069-6_31.

Oscar H. Ibarra and Tao Jiang. Learning regular languages from counterexamples. *Journal of Computer and System Sciences*, 43(2):299–316, October 1991. doi: 10.1016/0022-0000(91)90016-X.

Florentin Ipate. Learning finite cover automata from queries. *Journal of Computer and System Sciences*, 78(1):221 – 244, 2012. doi: 10.1016/j.jcss.2011.04.002.

Muhammad Naeem Irfan, Roland Groz, and Catherine Oriat. Optimising angluin algorithm l* by minimising the number of membership queries to process counterexamples. In *Zulu Workshop*, page 134, 2010.

Muhammad-Naeem Irfan, Roland Groz, and Catherine Oriat. Improving model inference of black box components having large input test set. In *Proceedings of ICGI*, volume 21 of *JMLR Workshop and Conference Proceedings*, pages 133–138, 2012.

Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995. doi: 10.1006/inco.1995.1070.

Maik Merten, Bernhard Steffen, Falk Howar, and Tiziana Margaria. Next generation learnlib. In Parosh A. Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *Lecture Notes in Computer Science*, pages 220–223. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-19835-9_18.

Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. Learnlib: a framework for extrapolating behavioral models. *Software Tools for Technology Transfer*, 11(5): 393–407, 2009. doi: 10.1007/s10009-009-0111-8.

Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299 – 347, 1993. doi: 10.1006/inco.1993.1021.

Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In Ana Cavalcanti and DennisR. Dams, editors, *FM 2009: Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 207–222. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-05089-3_14.

Guoqiang Shu and David Lee. Testing security properties of protocol implementations - a machine learning based approach. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, pages 25–25, June 2007. doi: 10.1109/ICDCS.2007.147.

Wouter Smeenk. *Applying Automata Learning to Complex Industrial Software*. Radboud University Nijmegen, 2012. Master's thesis.

Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In Marco Bernardo and Valrie Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-21455-4_8.