Inferring Mealy Machines

Muzammil Shahbaz and Roland Groz

Grenoble Universities,
F-38402 St Martin d'Hères Cedex, France
{muzammil,groz}@imag.fr

Abstract. Automata learning techniques are getting significant importance for their applications in a wide variety of software engineering problems, especially in the analysis and testing of complex systems. In recent studies, a previous learning approach [1] has been extended to synthesize Mealy machine models which are specifically tailored for I/O based systems. In this paper, we discuss the inference of Mealy machines and propose improvements that reduces the worst-time learning complexity of the existing algorithm. The gain over the complexity of the proposed algorithm has also been confirmed by experimentation on a large set of finite state machines.

1 Introduction

The field of automata learning has made a significant impact on a wide area of software engineering problems. For example, the learning techniques have been used to study the unknown behaviors of the system [2], testing the system [3], verifying interesting properties [4], building specification and maintaining the applications [5]. The use of such techniques actually ease the traditional practice of model driven engineering for the systems that consist of third-party components. The fact that these components come from different sources and have gone through various revisions before they are actually used, the components usually do not come with the formal and up-to-date specifications. Normally, the users of the components have to confront with their informal or insufficient information that hinders the direct applications of formal validation approaches. The application of automata learning techniques is a solution to synthesize the behavior models of the components, so that they could be used to analyze, test and validate the overall system using formal approaches.

Among various learning approaches, a well-known approach which has remained a major focus of the applied research in learning is the classical procedure, called L^* (aka Angluin's algorithm) [1]. Under this view, a component is assumed to be an unknown regular language whose alphabet is known. Then, the algorithm is applied to infer a Deterministic Finite Automaton (DFA) that models the unknown language in polynomial time (under certain assumptions).

Recent studies on reactive systems, e.g, telecom services, web-based applications, data acquisition modules, embedded system controllers etc, advocate the need of learning other forms of automata. This is due to the fact that complex systems characterize their behaviors in terms of input/output (i/o). Typically, such systems receive inputs from the environment, take decisions on internal transitions, perform computations and finally produce the corresponding outputs to the environment. Arguably, DFA models are not appropriate for modeling such systems since they lack the structure of i/o based behavior modeling. The more natural modeling of such systems is through Mealy machines that is much more concise compared to DFA models. Moreover, it is observed that a DFA model normally contains far more states than a Mealy machine if they model the same problem [6] [7]. Thus, efforts of learning Mealy machines are beneficial in terms of learning the state space of the problem to cater the complexity. We refer to the previous studies [7][5][6] for a detailed comparison of DFA and Mealy machine modeling of reactive systems.

In this paper, we discuss the learning of Mealy machines using the settings from Angluin's algorithm. There are many works which formally and informally present the adaptation of Angluin's algorithm to learn Mealy machines. However, we propose here some modifications in the adapted algorithm that brings down the complexity of learning Mealy machines significantly in some contexts.

The paper is organized as follows. Section 2 provides the basics of Angluin's algorithm informally. Section 3 discusses how Angluin's algorithm can be adapted for Mealy machine inference and how the adaptation can further be improved. Section 4 presents the adapted algorithm to learn Mealy machines, its complexity and its illustration on an example. Section 5 presents our improvements on the adapted algorithm, its complexity and its illustration on an example. Section 6 compares the complexity of the two algorithms on a finite state machine workbench experimentally. Section 7 concludes the paper.

2 Overview of Angluin's Algorithm

We refer to the original paper [1] for the complete discussion on learning DFA using Angluin's algorithm L^*. Here, we describe the algorithm informally.

The learning algorithm L^* starts by asking *membership queries* over the known alphabet Σ of the language to check whether certain strings from Σ^* are accepted or rejected by the language. The result of each such query in terms of $''1''$ (accepted) or $''0''$ (rejected) is recorded as an observation in a table. These queries are asked iteratively until the conditions on the observation table, i.e., it must be *closed* and *consistent*, are satisfied. The algorithm then conjectures a DFA based upon the observations recorded in the table. It then asks an *equivalence query* to a so called *oracle*, that knows the unknown language, to verify whether the conjecture is equivalent to the target DFA. The oracle validates the conjecture if it is correct or replies with a counterexample otherwise. A counterexample is a sequence that distinguishes the conjecture with the target DFA. The algorithm processes the counterexample in the table and performs another run of asking membership queries to construct a "better" conjecture. The algorithm iterates in this fashion until it produces a correct conjecture that is isomorphic to the target DFA.

Let $|\Sigma|$ be the size of the alphabet Σ, n be the total number of states in the target DFA and m be the length of the longest counterexample provided by the oracle, then the worst case complexity of Angluin's algorithm is $O(|\Sigma|mn^2)$.

3 From DFA to Mealy Machine

It is observed that Angluin's algorithm L^* can be used to learn Mealy machines through model transformation techniques. A simple way is to define a mapping from inputs and outputs of the machine to letters in a DFA's alphabet Σ. This can be done either by taking inputs and outputs as letters, i.e., $\Sigma = I \cup O$ [5] or by considering couples of inputs and outputs as letters, i.e., $\Sigma = I \times O$ [8]. But these methods increase the size of Σ and thus raise complexity problems because the algorithm is polynomial on these factors. However, there is a straightforward implication of L^* on learning Mealy machines by slightly modifying the structure of the observation table. The idea is to record the behaviors of the system as output strings in the table instead of recording just "1" and "0", as in the case of language inference. Similarly, we can modify the related concepts, such as making the table closed and consistent and making a conjecture from the table etc. For processing counterexamples in the table, we can also easily adapt the corresponding method from L^*. This adaptation of L^* to learn Mealy machines has already been discussed, formally [7] [9] [10] [3], and informally [6] [11].

However, our contribution in the inference of Mealy machines is the proposal of a new method for processing counterexamples that consequently reduces the complexity of the algorithm. The complexity analysis shows that by using our method for processing counterexamples, the algorithm for learning Mealy machines requires less number of queries, compared to the adapted method.

4 Inferring Mealy Machines

Definition 1. *A Mealy Machine is a sextuple* $(Q, I, O, \delta, \lambda, q_0)$*, where* Q *is the non-empty finite set of states,* $q_0 \in Q$ *is the initial state,* I *is the finite set of input symbols,* O *is the finite set of output symbols,* $\delta : Q \times I \to Q$ *is the transition function,* $\lambda : Q \times I \to O$ *is the output function.*

Definition 1 provides the formal definition of (deterministic) Mealy machines. When a Mealy machine is in the current (source) state $q \in Q$ and receives $i \in I$, it moves to the target state specified by $\delta(q, i)$ and produces an output given by $\lambda(q, i)$. The functions δ and λ are extended from symbols to strings in the standard way. We consider that the Mealy machines are input-enabled, i.e., $dom(\delta) = dom(\lambda) = Q \times I$. We denote by $suff^k(\omega)$, the suffix of a string ω of length k. Let $\omega = a \cdot b \cdots x \cdot y \cdot z$, then $suff^3(\omega) = x \cdot y \cdot z$. An example of a Mealy machine over the sets $I = \{a, b\}$ and $O = \{x, y\}$ is shown in Figure 1.

Now, we detail the learning of Mealy machines using the settings from Angluin's algorithm L^*, that has also been mentioned in the existing works. As for

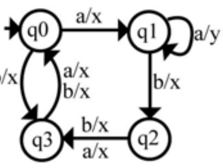

Fig. 1. Example of a Mealy Machine

DFA learning, the two main assumptions in learning Mealy machines are i) The basic input set I is known, and ii) The machine can be reset before each query.

The algorithm asks *output queries* [3] that are strings from I^+ and obtain the corresponding output strings from the machine. This is similar to the concept of membership queries in L^*. The difference is that instead of 1 or 0, the machine replies with the complete output string. Let $\omega \in I^+$, i.e., an input string of the query, then the machine replies to the query with $\lambda(q_0, \omega)$. The response to each query is recorded in the observation table. The queries are asked iteratively until the conditions on the observation table, i.e., it must be *closed* and *consistent*, are satisfied. This follows by making the Mealy machine conjecture from the table. The algorithm then asks equivalence query to the oracle. If the oracle says "yes", i.e., the conjecture is correct, then the algorithm terminates the procedure by outputting the conjecture. If the oracle replies with a counterexample, then the algorithm processes the counterexample in the table and refines the conjecture.

The formal description of learning Mealy machines is given in the subsequent sections. We denote by $\mathcal{M} = \{Q_{\mathcal{M}}, I, O, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}}, q_{0\mathcal{M}}\}$ the unknown Mealy machine model that has a minimum number of states. We assume that the input/output interfaces of the machines are accessible, i.e., the input interface from where an input can be sent and the output interface from where an output can be observed.

4.1 Observation Table

We denote by ${L_M}^*$ the learning algorithm for Mealy machines. At any given time, ${L_M}^*$ has information about a finite collection of input strings from I^+ and their corresponding output strings from O^+. This information is organized into an observation table, denoted by (S_M, E_M, T_M). The structure of the table is directly imported from Angluin's algorithm L^*. Let S_M and E_M be non-empty finite sets of finite strings over I. S_M is a prefix-closed set that always contains an empty string ϵ. E_M is a suffix-closed set (except $\epsilon \notin E_M$). Let T_M be a finite function that maps $(S_M \cup S_M \cdot I) \times E_M$ to O^+. If $s \in S_M \cup S_M \cdot I$ and $e \in E_M$, then $T_M(s, e)$ contains the output string from $\lambda_{\mathcal{M}}(q_{0\mathcal{M}}, s \cdot e)$. The rows of the table consist of the elements of $S_M \cup S_M \cdot I$ and the columns consist of the elements of E_M.

Since S_M and E_M are non-empty sets, the table is initialized by $S_M = \{\epsilon\}$ and $E_M = I$, i.e., every input symbol makes one column in the table, with the entry for a row $s \in S_M \cup S_M \cdot I$ and a column $e \in E_M$ equals to $T_M(s, e)$. The equivalence of rows is defined with respect to the strings in E_M. Suppose $s, t \in S_M \cup S_M \cdot I$ are two rows, then s and t are equivalent, denoted by $s \cong_{E_M} t$, if and only if $T_M(s, e) = T_M(t, e)$, for all $e \in E_M$. We denote by $[s]$ the equivalence class of rows that also includes s. An example of the observation table (S_M, E_M, T_M) for learning the Mealy machine in Figure 1 is given in Table 1.

Table 1. Example of the Observation Table (S_M, E_M, T_M)

		E_M	
		a	b
S_M	ϵ	x	x
$S_M \cdot I$	a	y	x
	b	x	x

The algorithm $L_M{}^*$ eventually uses the observation table (S_M, E_M, T_M) to build a Mealy machine conjecture. The strings or prefixes in S_M are the potential states of the conjecture, and the strings or suffixes in E_M distinguish these states from each other.

To build a valid Mealy machine conjecture from the observations, the table must satisfy two conditions. The first condition is that the table must be *closed*, i.e., for each $t \in S_M \cdot I$, there exists an $s \in S_M$, such that $s \cong_{E_M} t$. If the table is not closed, then a potential state that can be observed in the table would not appear in the conjecture. The second condition is that the table must be *consistent*, i.e., for each $s, t \in S_M$ such that $s \cong_{E_M} t$, it holds that $s \cdot i \cong_{E_M} t \cdot i$, for all $i \in I$. If the table is not consistent then two seemingly equivalent states in the conjecture may point to different target states for the same input.

When the observation table (S_M, E_M, T_M) is closed and consistent, then a Mealy machine conjecture can be constructed as follows:

Definition 2. *Let (S_M, E_M, T_M) be a closed and consistent observation table, then the Mealy machine conjecture $M_M = (Q_M, I, O, \delta_M, \lambda_M, q_{0M})$ is defined, where*

- $Q_M = \{[s] | s \in S_M\}$
- $q_{0M} = [\epsilon]$
- $\delta_M([s], i) = [s \cdot i], \forall s \in S_M, i \in I$
- $\lambda_M([s], i) = T_M(s, i), \forall i \in I$

To see that M_M is well defined, note that S_M is a non-empty prefix-closed set and it contains at least one row ϵ, hence Q_M and q_{0M} are well-defined. For all $s, t \in S_M$ such that $s \cong_{E_M} t$, we have $[s] = [t]$. Since the table is consistent, for all $i \in I$, $[s \cdot i] = [t \cdot i]$ holds. Since the table is closed, there exists $u \in S_M$ such that $[u] = [s \cdot i] = [t \cdot i]$ holds. Hence δ_M is well defined. Since E_M is non-empty and $E_M \supseteq I$ always hold. If there exists $s, t \in S_M$ such that $s \cong_{E_M} t$, then for all $i \in I$, $T_M(s, i) = T_M(t, i)$. Hence, λ_M is well defined.

Theorem 1. *If (S_M, E_M, T_M) is a closed and consistent observation table, then the Mealy machine conjecture M_M from (S_M, E_M, T_M) is consistent with the finite function T_M. That is, for every $s \in S_M \cup S_M \cdot I$ and $e \in E_M$, $\lambda_M(\delta_M(q_{0_M}, s), e) = T_M(s, e)$. Any other Mealy machine consistent with T_M but inequivalent to M_M must have more states.*

Theorem 1 claims the correctness of the conjecture. Niese [7] has given a formal proof of the correctness, which is a simple adaptation of the proofs in Angluin's algorithm, in which the range of the output function is replaced by O^+. Note that the conjecture is proved to be consistent with the observation table by exhibiting the prefix-closed and suffix-closed properties of S_M and E_M respectively. Moreover, the conjecture is the minimum machine by construction.

4.2 The Algorithm $L_M{}^*$

The algorithm $L_M{}^*$ starts by initializing (S_M, E_M, T_M) with $S_M = \{\epsilon\}$ and $E_M = I$. To determine T_M, it asks output queries constructed from the table. For each $s \in S_M \cup S_M \cdot I$ and $e \in E_M$, a query is constructed as $s \cdot e$. The corresponding output string of the machine, i.e., $\lambda_{\mathcal{M}}(q_{0_{\mathcal{M}}}, s \cdot e)$, is recorded with the help of the function T_M. Note that the table is prefix-closed which means that $\lambda_{\mathcal{M}}(q_{0_{\mathcal{M}}}, s)$ can be derived from the observations already recorded in the table. Therefore, $L_M{}^*$ records only the suffix of the output string of the length of e in the table as $T_M(s, e) = \mathit{suff}^{|e|}(\lambda_{\mathcal{M}}(q_{0_{\mathcal{M}}}, s \cdot e))$.

After filling the table with the result of the queries, $L_M{}^*$ checks if the table is closed and consistent. If it is not closed, then $L_M{}^*$ finds $t \in S_M \cdot I$ such that $t \ncong_{E_M} s$, for all $s \in S_M$. Then, it moves t to S_M and $T_M(t \cdot i, e)$ is determined for all $i \in I, e \in E_M$ in $S_M \cdot I$. If the table is not consistent, then $L_M{}^*$ finds $s, t \in S_M, e \in E_M$ and $i \in I$ such that $s \cong_{E_M} t$, but $T_M(s \cdot i, e) \neq T_M(t \cdot i, e)$. Then, it adds the string $i \cdot e$ to E_M and extends the table by asking output queries for the missing elements.

When the table is closed and consistent, $L_M{}^*$ makes a Mealy machine conjecture M_M from the table according to Definition 2.

4.3 Example

We illustrate the algorithm $L_M{}^*$ on the Mealy machine $\mathcal{M}$ given in Figure 1. The algorithm initializes (S_M, E_M, T_M) with $S_M = \{\epsilon\}$ and $S_M \cdot I = E_M = \{a, b\}$. Then, it asks the output queries to fill the table, as shown in Table 1. When the table is filled, $L_M{}^*$ checks if it is closed and consistent.

Table 1 is not closed since the row a in $S_M \cdot I$ is not equivalent to any row in S_M. Therefore, $L_M{}^*$ moves the row a to S_M and extends the table accordingly. Then, $L_M{}^*$ asks the output queries for the missing elements of the table. Table 2 shows the resulting observation table.

The new table is closed and consistent, so $L_M{}^*$ makes the conjecture $M_M{}^{(1)} = (Q_{M_M{}^{(1)}}, I, O, \delta_{M_M{}^{(1)}}, \lambda_{M_M{}^{(1)}}, q_{0_{M_M{}^{(1)}}})$ from Table 2. The conjecture $M_M{}^{(1)}$ is shown in Figure 2.

Table 2. Closed and Consistent Observation Table (S_M, E_M, T_M) for learning $\mathcal{M}$ in Figure 1

	a	b
ϵ	x	x
a	y	x
b	x	x
$a \cdot a$	y	x
$a \cdot b$	x	x

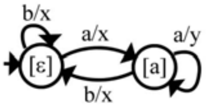

Fig. 2. The conjecture ${M_M}^{(1)}$ from Table 2

Now, ${L_M}^*$ asks an equivalence query to the oracle. Since, the conjecture ${M_M}^{(1)}$ is not correct, the oracle replies with a counterexample. The methods for processing counterexamples are discussed in the following sections. We shall illustrate the methods with the help of the same example. We provide here a counterexample that will be used in their illustrations.

Let $a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$ be a counterexample for ${M_M}^{(1)}$, since

- $\lambda_{{M_M}^{(1)}}(q_{0{M_M}^{(1)}}, a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a) = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot y$ and
- $\lambda_{\mathcal{M}}(q_{0\mathcal{M}}, a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a) = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$.

We choose a long counterexample to better illustrate the methods and to realize how they work when the counterexamples of arbitrary lengths are provided. In practice[1], it is not sure whether we obtain always the shortest counterexample.

4.4 Processing Counterexamples in ${L_M}^*$

Angluin's algorithm L^* provides a method for processing a counterexample in the observation table, so that the conjecture is refined with at least one more state. For the algorithm ${L_M}^*$, we can adapt Angluin's method straightforwardly. The adapted method is described as follows.

Directly Adapted Method from L^*. Let $M_M = (Q_M, I, O, \delta_M, \lambda_M, q_{0M})$ be the conjecture from a closed and consistent observation table (S_M, E_M, T_M) for learning the machine $\mathcal{M}$. Let ν be a string from I^+ as a counterexample such that $\lambda_M(q_{0M}, \nu) \neq \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, \nu)$. Then, ${L_M}^*$ adds all the prefixes of ν to S_M and extends (S_M, E_M, T_M) accordingly. The algorithm makes another run of output queries until (S_M, E_M, T_M) is closed and consistent, followed by making a new conjecture.

4.5 Complexity

We analyze the total number of output queries asked by ${L_M}^*$ in the worst case by the factors $|I|$, i.e., the size of I, n, i.e., the number of states of the minimum

[1] There are many frameworks that have been proposed to replace the oracle in Angluin's settings. The interested reader is referred to the works [12] [4] [13] for the comprehensive discussions on the limits of learning without oracles.

Table 3. The Observation Tables (S_M, E_M, T_M) for processing the counterexample $a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$ for ${M_M}^{(1)}$ using the adapted method from L^*. The boxes in the tables show the rows which make the tables inconsistent.

	a	b
$\boxed{\epsilon}$	x	x
a	y	x
$\boxed{a \cdot b}$	x	x
$a \cdot b \cdot a$	x	x
$a \cdot b \cdot a \cdot b$	x	x
$a \cdot b \cdot a \cdot b \cdot b$	x	x
$a \cdot b \cdot a \cdot b \cdot b \cdot a$	x	x
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$	y	x
b	x	x
$a \cdot a$	y	x
$a \cdot b \cdot b$	x	x
$a \cdot b \cdot a \cdot a$	x	x
$a \cdot b \cdot a \cdot b \cdot a$	y	x
$a \cdot b \cdot a \cdot b \cdot b \cdot b$	x	x
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot b$	x	x
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot a$	y	x
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot b$	x	x

(i) Adding the prefixes of $a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$ to S_M

	a	b	$a \cdot a$
ϵ	x	x	$x \cdot y$
a	y	x	$y \cdot y$
$\boxed{a \cdot b}$	x	x	$x \cdot x$
$\boxed{a \cdot b \cdot a}$	x	x	$x \cdot x$
$a \cdot b \cdot a \cdot b$	x	x	$x \cdot y$
$a \cdot b \cdot a \cdot b \cdot b$	x	x	$x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot a$	x	x	$x \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$	y	x	$y \cdot y$
b	x	x	$x \cdot x$
$a \cdot a$	y	x	$y \cdot y$
$a \cdot b \cdot b$	x	x	$x \cdot x$
$a \cdot b \cdot a \cdot a$	x	x	$x \cdot y$
$a \cdot b \cdot a \cdot b \cdot a$	y	x	$y \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot b$	x	x	$x \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot b$	x	x	$x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot a$	y	x	$y \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot b$	x	x	$x \cdot x$

(ii) Adding $a \cdot a$ to E_M

	a	b	$a \cdot a$	$a \cdot a \cdot a$	$b \cdot a \cdot a$
ϵ	x	x	$x \cdot y$	$x \cdot y \cdot y$	$x \cdot x \cdot x$
a	y	x	$y \cdot y$	$y \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot x$	$x \cdot x \cdot x$
$a \cdot b \cdot a$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot y$
$a \cdot b \cdot a \cdot b$	x	x	$x \cdot y$	$x \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot a$	x	x	$x \cdot y$	$x \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$	y	x	$y \cdot y$	$y \cdot y \cdot y$	$x \cdot x \cdot x$
b	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot y$
$a \cdot a$	y	x	$y \cdot y$	$y \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot y$
$a \cdot b \cdot a \cdot a$	x	x	$x \cdot y$	$x \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot a$	y	x	$y \cdot y$	$y \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot b$	x	x	$x \cdot y$	$x \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot y$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot a$	y	x	$y \cdot y$	$y \cdot y \cdot y$	$x \cdot x \cdot x$
$a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot x$	$x \cdot x \cdot x$

(iii) Adding $a \cdot a \cdot a$ and $b \cdot a \cdot a$ to E_M

machine $\mathcal{M}$ and m, i.e., the maximum length of any counterexample provided for learning $\mathcal{M}$.

Initially, S_M contains one element. Each time (S_M, E_M, T_M) is found not closed, one element is added to S_M. This introduces a new row to S_M, so a new state in the conjecture. This can happen for at most $n - 1$ times. For each counterexample of length at most m, there can be at most m strings that are added to S_M, and there can be at most $n - 1$ counterexamples to distinguish n states. Thus, the size of S_M cannot exceed $n + m(n - 1)$.

Initially, E_M contains $|I|$ elements. Each time (S_M, E_M, T_M) is found not consistent, one element is added to E_M. This can happen for at most $n-1$ times to distinguish n states. Thus, the size of E_M cannot exceed $|I| + n - 1$.

Thus, ${L_M}^*$ produces a correct conjecture by asking maximum $(S_M \cup S_M \cdot I) \times E_M = O(|I|^2 nm + |I|mn^2)$ output queries.

4.6 Example

For the conjecture ${M_M}^{(1)}$ in Figure 2 for learning the Mealy machine $\mathcal{M}$ in Figure 1, we have a counterexample as $\nu = a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$. According to the adapted method for processing counterexample, ${L_M}^*$ adds all the prefixes of ν, i.e., a, $a \cdot b$, $a \cdot b \cdot a$, $a \cdot b \cdot a \cdot b$, $a \cdot b \cdot a \cdot b \cdot b$, $a \cdot b \cdot a \cdot b \cdot b \cdot a$, and $a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$ to S_M and extends $S_M \cdot I$ accordingly. The table is then filled with the missing elements by asking output queries. Table 3 (i) shows the resulting observation table. Then, ${L_M}^*$ checks if the table is closed and consistent.

Table 3 (i) is closed but not consistent since $\epsilon \cong_{E_M} a \cdot b$, but $T_M(\epsilon \cdot a, a) \neq T_M(a \cdot b \cdot a, a)$. To make the table consistent, the string $a \cdot a$ is added to E_M and the table is filled accordingly. Table 3 (ii) shows the resulting observation table, in which the rows ϵ and $a \cdot b$ have become different. Now, ${L_M}^*$ checks if Table 3 (ii) is closed and consistent.

Table 3 (ii) is closed but not consistent since $a \cdot b \cong_{E_M} a \cdot b \cdot a$ but $T_M(a \cdot b \cdot a, a \cdot a) \neq T_M(a \cdot b \cdot a \cdot a, a \cdot a)$. To make the table consistent, the string $a \cdot a \cdot a$

is added to E_M. For the same rows, the other reason for inconsistency is due to $T_M(a \cdot b \cdot b, a \cdot a) \neq T_M(a \cdot b \cdot a \cdot b, a \cdot a)$. Therefore, the string $b \cdot a \cdot a$ is also added to E_M and the table is filled accordingly. Table 3 (iii) shows the resulting observation table, in which the rows $a \cdot b$ and $a \cdot b \cdot a$ have become different.

Table 3 (iii) is closed and consistent, and thus ${L_M}^*$ terminates by making a conjecture isomorphic to $\mathcal{M}$. The total number of output queries asked by ${L_M}^*$ is 85.

5 Improvements to Mealy Machine Inference

We propose improvements to the algorithm of learning Mealy machines by providing a new method for processing counterexamples in the observation table (S_M, E_M, T_M). The complexity calculations and the experimental results of our proposal show a significant reduction in the output queries that the algorithm asks during the learning procedure. We denote the algorithm with the improved method for processing counterexamples by ${L_M}^+$. In this section, we describe the idea of our improvements and the complete algorithm with its complexity, correctness and example illustration.

5.1 Motivation

Rivest & Schapire [14] observed that the basic Angluin's algorithm L^* can be improved by removing consistency check of the observation table. Consistency is checked only when two rows in the upper part of the table are found equivalent. That means, if the rows of the table remain inequivalent, then inconsistency will never occur and the condition will always hold trivially. They observed that the rows become equivalent in the table due to the improper handling of counterexamples. A counterexample is an experiment that distinguishes two or more equivalent rows (or states) in the table and thereby causes an increase in the size of the column. However, L^* does not follow this scheme directly, rather it adds a new row for each prefix of the counterexample in the table, assuming that all are potential states of the conjecture. Later, the rows are filled with the help of membership queries (no new column is added yet). This is where an inconsistency can occur in the table if the two rows become equivalent but their future behaviors are not equivalent. Thus, the two rows must be distinguished by adding a distinguishing sequence as a column in the table.

Rivest & Schapire proposed a method for processing counterexamples, which does not add the prefixes in the table. Thus, the rows remain inequivalent during the whole learning process. Their method consists in finding a distinguishing sequence from the counterexample and directly add the sequence in the columns. However, their method requires a relaxation on the prefix-closed and suffix-closed properties of the table, which are in fact the vital properties for having a consistent conjecture from the table [1]. If the table does not have such properties then the new conjecture might not be consistent with the table, and therefore, might still classify the previous counterexamples incorrectly. Balcazar et al. [15]

argued that by using the method of Rivest & Schapire, one can obtain the same counterexample to answer several equivalence queries in L^*. In addition, Berg & Raffelt [16] compiled the results from Balcazar et al. [15] and explained the complete method of Rivest & Schapire.

Our improvement in the algorithm for learning Mealy machines is inspired by Rivest & Schapire's idea. We also suggest to keep only inequivalent rows in S_M so that inconsistencies can never occur. However, we propose a new method for processing counterexamples such that it does not import the same problem as in the case of Rivest & Schapire. Our method for processing counterexample keeps (S_M, E_M, T_M) prefix-closed and suffix-closed, and therefore, the new conjecture is always consistent with the observations in (S_M, E_M, T_M), according to Theorem 1.

5.2 The Algorithm $L_M{}^+$

In the algorithm $L_M{}^+$, the definition of the observation table (S_M, E_M, T_M), described in Section 4.1, and the basic flow of the algorithm, described in Section 4.2, remain unchanged. However, the additional property of (S_M, E_M, T_M) is that all the rows in S_M are inequivalent, i.e., for all $s, t \in S_M$, $s \ncong_{E_M} t$. This means $L_M{}^+$ does not need to check for consistency because it always trivially holds. However, $L_M{}^+$ processes counterexamples according to the new method, which is described in the following.

5.3 Processing Counterexamples in $L_M{}^+$

Let $M_M = (Q_M, I, O, \delta_M, \lambda_M, q_{0M})$ be the conjecture from the closed (and consistent) observation table (S_M, E_M, T_M) for learning the machine $\mathcal{M}$. Let ν be a string from I^+ as a counterexample such that $\lambda_M(q_{0M}, \nu) \neq \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, \nu)$. The main objective of a counterexample is to distinguish the conjecture from the unknown machine. That means, the counterexample must contain a distinguishing sequence to distinguish at least two seemingly equivalent states of the conjecture; so that when applying the distinguishing sequence on these states, they become different.

In our method of processing counterexample, we look for the distinguishing sequence in the counterexample and add the sequence directly to E_M. Then, the two seemingly equivalent rows[2] in S_M become different. For this purpose, we divide ν into its appropriate prefix and suffix such that the suffix contains the distinguishing sequence. The division occurs in the following way.

We divide ν by looking at its longest prefix in $S_M \cup S_M \cdot I$ and take the remaining string as the suffix. Let $\nu = u \cdot v$ such that $u \in S_M \cup S_M \cdot I$. If there exists $u' \in S_M \cup S_M \cdot I$ another prefix of ν then $|u| > |u'|$, i.e., u is the longest prefix of ν in $S_M \cup S_M \cdot I$. The idea of selecting u from the observation table is that u is the access string that is already known such that $\lambda_M(q_{0M}, u) = \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, u)$. The fact that ν is a counterexample then $\lambda_M(q_{0M}, u \cdot v) \neq \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, u \cdot v)$ must

[2] Recall that the rows in S_M represent the states of the conjecture.

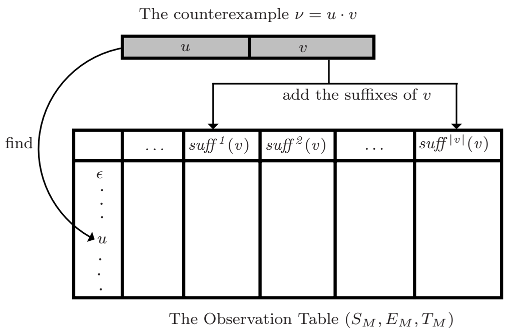

Fig. 3. Conceptual view of the method for processing counterexamples in $L_M{}^+$

hold. That means, v contains the distinguishing sequence to distinguish two rows in S_M. So, it is sufficient to add v to E_M. In fact, we add all the suffixes of v such that E_M remains suffix-closed.

Figure 3 provides a conceptual view of the method for processing a counterexample ν. It shows that ν is divided into the prefix u and the suffix v, such that $u \in S_M \cup S_M \cdot I$. Then, ν is processed by adding all the suffixes of v to E_M. The correctness proof of the method is given in the following section.

5.4 Correctness

Let $M_M = (Q_M, I, O, \delta_M, \lambda_M, q_{0M})$ be the conjecture from the closed (and consistent) observation table (S_M, E_M, T_M). Let $\nu = u \cdot i \cdot v$ be the counterexample for M_M such that $\lambda_M(q_{0M}, u \cdot i \cdot v) \neq \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, u \cdot i \cdot v)$. Let $u \cdot i$ be the longest prefix of ν in $S_M \cup S_M \cdot I$ and v be the corresponding suffix of ν. If ν is a counterexample then it must distinguish $[u \cdot i]$ from a seemingly equivalent state, i.e., $\lambda_{\mathcal{M}}(q_{0\mathcal{M}}, u \cdot i \cdot v) \neq \lambda_{\mathcal{M}}(q_{0\mathcal{M}}, t \cdot v)$, for some $t \in S_M$ such that $[t] = [u \cdot i]$. Thus, v contains a distinguishing sequence for the rows $u \cdot i$ and t.

Suppose we process ν in (S_M, E_M, T_M) by adding all the suffixes of v to E_M. Let us name the table as (S'_M, E'_M, T'_M) after this addition. Later, we ask output queries to fill the missing elements of the table (S'_M, E'_M, T'_M). Then, E'_M contains the distinguishing sequence that distinguishes the rows t and $u \cdot i$ in (S'_M, E'_M, T'_M). That is, there must exist some experiment $e \in E'_M$ such that $T'_M(t, e) \neq T'_M(u \cdot i, e)$. This implies that $u \cdot i \ncong_{E'_M} t$. In fact, $u \cdot i \in S'_M \cdot I$, since $t \in S'_M$ and there cannot be two equivalent rows in S'_M. If $u \cdot i \in S'_M \cdot I$ then trivially $u \in S'_M$. Moreover, in the table (S_M, E_M, T_M), if $u \cdot i \ncong_{E_M} s$, for $s \in S_M$, then in the extended table (S'_M, E'_M, T'_M), $u \cdot i \ncong_{E'_M} s$ also holds, for $s \in S'_M$. Therefore, $u \cdot i$ is a row in (S'_M, E'_M, T'_M) that is inequivalent to any row in S'_M. This makes the table not closed. Thus, making the table closed will move $u \cdot i$ to S'_M. Since, u is already in S'_M, this operation keeps (S'_M, E'_M, T'_M) prefix-closed. Since, S'_M is extended by one row, the new conjecture M'_M from the closed (S'_M, E'_M, T'_M) will contain at least one more state than M_M.

It is simple to check whether (S'_M, E'_M, T'_M) is suffix-closed, since E'_M is extended from E_M, which is suffix-closed, and E'_M contains the suffixes of v. Thus, (S'_M, E'_M, T'_M) is suffix-closed.

This proves the correctness of the method, since (S'_M, E'_M, T'_M) is a closed (and consistent) observation table that is prefix-closed and suffix-closed and contains the prefix $u \cdot i$ and the suffix v of the counterexample ν. Therefore, the conjecture M'_M from (S'_M, E'_M, T'_M) will be consistent with the function T'_M (Theorem 1) that will find at least one more state. $\square$

Theorem 2. *Let (S_M, E_M, T_M) be a closed (and consistent) observation table and M_M be the conjecture from (S_M, E_M, T_M). Let $\nu = u \cdot i \cdot v$ be the counterexample for M_M, where $u \cdot i$ is in $S_M \cup S_M \cdot I$. Let the table be extended as (S'_M, E'_M, T'_M) by adding all the suffixes of v to E_M, then the closed (and consistent) observation table (S'_M, E'_M, T'_M) is prefix-closed and suffix-closed. The conjecture M'_M from (S'_M, E'_M, T'_M) will be consistent with T'_M and must have at least one more state than M_M.*

5.5 Complexity

We analyze the total number of output queries asked by ${L_M}^+$ in the worst case by the factors $|I|$, i.e., the size of I, n, i.e., the number of states of the minimum machine $\mathcal{M}$ and m, i.e., the maximum length of any counterexample provided for learning $\mathcal{M}$.

The size of S_M increases monotonically up to the limit of n as the algorithm runs. The only operation that extends S_M is making the table *closed*. Every time (S_M, E_M, T_M) is not closed, one element is added to S_M. This introduces a new row to S_M, so a new state in the conjecture. This can happen at most $n-1$ times, since it keeps one element initially. Hence, the size of S_M is at most n.

E_M contains $|I|$ elements initially. If a counterexample is provided then at most m suffixes are added to E_M. There can be provided at most $n-1$ counterexamples to distinguish n states, thus the maximum size of E_M cannot exceed $|I| + m(n-1)$.

Thus, ${L_M}^+$ produces a correct conjecture by asking maximum $(S_M \cup S_M \cdot I) \times E_M = O(|I|^2 n + |I| m n^2)$ output queries.

5.6 Example

We illustrate the algorithm ${L_M}^+$ on the Mealy machine $\mathcal{M}$ given in Figure 1. Since, ${L_M}^+$ is only different from ${L_M}^*$ with respect to the method for processing counterexamples, the initial run of ${L_M}^+$ is same as described in Section 4.3. So, ${L_M}^+$ finds a closed (and consistent) table as Table 2 and draws the conjecture ${M_M}^{(1)}$, shown in Figure 2, from Table 2. Here, we illustrate how ${L_M}^+$ processes counterexamples to refine the conjecture.

For the conjecture ${M_M}^{(1)}$, we have a counterexample as $\nu = a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$. According to the improved method for processing counterexample, ${L_M}^+$ finds the longest prefix u of the counterexample in $S_M \cup S_M \cdot I$ in Table 2. The prefix

Table 4. The Observation Tables (S_M, E_M, T_M) for processing the counterexample $a \cdot b \cdot a \cdot b \cdot b \cdot a \cdot a$ for ${M_M}^{(1)}$ using the improved method. The boxes in the tables show the rows which make the tables not closed.

	a	b	$a \cdot a$	$b \cdot a \cdot a$	$b \cdot b \cdot a \cdot a$	$a \cdot b \cdot b \cdot a \cdot a$
ϵ	x	x	$x \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$
a	y	x	$y \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x$	$y \cdot x \cdot x \cdot x \cdot x$
$\boxed{b}$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x \cdot y$
$a \cdot a$	y	x	$y \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x$	$y \cdot x \cdot x \cdot x \cdot x$
$\boxed{a \cdot b}$	x	x	$x \cdot x$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$

(i) Adding the suffixes of $a \cdot b \cdot b \cdot a \cdot a$ to E_M

	a	b	$a \cdot a$	$b \cdot a \cdot a$	$b \cdot b \cdot a \cdot a$	$a \cdot b \cdot b \cdot a \cdot a$
ϵ	x	x	$x \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$
a	y	x	$y \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x$	$y \cdot x \cdot x \cdot x \cdot x$
b	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x \cdot y$
$a \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$
$a \cdot a$	y	x	$y \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x$	$y \cdot x \cdot x \cdot x \cdot x$
$b \cdot a$	x	x	$x \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$
$b \cdot b$	x	x	$x \cdot y$	$x \cdot x \cdot x$	$x \cdot x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x \cdot x$
$a \cdot b \cdot a$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x \cdot y$
$a \cdot b \cdot b$	x	x	$x \cdot x$	$x \cdot x \cdot y$	$x \cdot x \cdot x \cdot x$	$x \cdot x \cdot x \cdot x \cdot y$

(ii) Moving the rows b and $a \cdot b$ to S_M

$u = a \cdot b$ is the longest prefix found, so the remaining suffix is $v = a \cdot b \cdot b \cdot a \cdot a$. The algorithm adds all the suffixes of v, i.e., a, $a \cdot a$, $b \cdot a \cdot a$, $b \cdot b \cdot a \cdot a$ and $a \cdot b \cdot b \cdot a \cdot a$ to E_M. The table is filled by asking output queries for the missing elements. Table 4 (i) is the resulting observation table. Then, ${L_M}^+$ checks if the table is closed.

Table 4 (i) is not closed since the rows b and $a \cdot b$ are not equivalent to any rows in S_M. Hence, the rows b and $a \cdot b$ are moved to S_M and the table is extended accordingly. The table is filled by asking output queries for the missing elements. Table 4 (ii) is the resulting observation table. Now, ${L_M}^+$ checks whether Table 4 (ii) is closed.

Table 4 (ii) is closed, and thus ${L_M}^+$ terminates by making a conjecture isomorphic to $\mathcal{M}$. The total number of output queries asked by ${L_M}^+$ is 54.

6 Experimentation

We have performed an experimental evaluation to compare the adaptation of Angluin's algorithm for learning Mealy machines ${L_M}^*$ with our improved algorithm ${L_M}^+$. The worst case theoretical complexity analysis has shown that ${L_M}^+$ outperforms ${L_M}^*$ in terms of number of output queries. It is interesting to evaluate the average case complexity of the algorithms when the input sets, the number of states and the length of counterexamples are of arbitrary sizes.

The examples in the experiments are the synthetic finite state models of real world systems (e.g., Vending machine, ATM and ABP protocols, Mailing Systems etc) that are shipped with *Edinburgh Concurrency Workbench (CWB)* [17]. CWB is a tool for manipulating, analyzing and verifying concurrent systems. The examples in the workbench have also been used to investigate the applicability of Angluin's algorithm in learning reactive systems [18]. These examples were originally modeled as Non-Deterministic Finite Automata (NFA), with partial transition relations, in which every state is a final state. Therefore, we have transferred first each example to its corresponding DFA. The resulting DFA contains every state as final, plus one non-final (sink) state which loops itself for all inputs.

All inputs from a state that are invalid (missing transitions in the original NFA) are directed to the sink state. Then, we learn the Mealy machines models of the CWB examples using both algorithms one by one. We have also simulated an oracle so that the algorithms could ask equivalence queries for conjectures until they find correct models. The oracle obtains a counterexample by calculating a symmetric difference between the original example and the provided conjecture.

The number of output queries asked by the algorithms are given in Table 5. The first column labels the example. The second column shows the size of the input set I. The third column shows the minimum number of states in the example when modeled as DFA and Mealy machines. The fourth and fifth columns show the number of output queries asked by ${L_M}^*$ and ${L_M}^+$, respectively. The last column shows the reduction factor in queries asked by ${L_M}^+$ against ${L_M}^*$, i.e., $\frac{no.\ of\ output\ queries\ in\ {L_M}^*}{no.\ of\ output\ queries\ in\ {L_M}^+} - 1$.

Table 5. Comparison of ${L_M}^*$ with ${L_M}^+$ on the examples of CWB workbench. The examples are listed in ascending order with respect to the number of states.

Examples	$\|I\|$	No. of States	No. of Output Queries		Reduction Factor
		DFA / Mealy (min)	${L_M}^*$	${L_M}^+$	
ABP-Lossy	3	11	754	340	1.22
Peterson2	3	11	910	374	1.43
Small	5	11	462	392	0.18
VM	5	11	836	392	1.13
Buff3	3	12	580	259	1.24
Shed2	6	13	824	790	0.04
ABP-Safe	3	19	2336	754	2.1
TMR1	5	19	1396	1728	-0.2
VMnew	4	29	2595	1404	0.85
CSPROT	5	44	4864	3094	0.57

The experiments have been conducted on 10 CWB examples. All examples are of different sizes in terms of number of states and input set size. The results show that ${L_M}^+$ outperformed ${L_M}^*$ in almost all the examples. The greatest reduction factor achieved is 2.1 on the example *ABP-Safe*. However, there is only one example *TMR1* in which ${L_M}^+$ has performed negatively. This is because the implementation of our oracle provides arbitrary counterexamples that could influence the number of output queries in few cases.

Apart from the CWB workbench, we have also experimented on arbitrary random Mealy machines. We generated a set of 1500 machines with sizes ranging between 1 and 500 states and input size up to 15. The average reduction factor we achieved on this set is 1.32. We also studied the query complexity with respect to the relation of input size with the number of states. Up to 1250 machines were generated with larger inputs and relatively fewer states. We achieved the best result on this set with the average reduction factor of 1.66.

As we know from our worst case complexity analysis, ${L_M}^+$ performs better than ${L_M}^*$. We have experimentally confirmed the difference in complexity of the two algorithms on the finite state machine workbench, as well as on a large set of arbitrary random Mealy machines.

7 Conclusion and Perspectives

We have presented two algorithms for inferring Mealy machines, namely ${L_M}^*$ and ${L_M}^+$. The algorithm ${L_M}^*$ is a straightforward adaptation from the algorithm L^*. The algorithm ${L_M}^+$ is our proposal that contains a new method for processing counterexamples. The complexity calculations of the two algorithms shows that ${L_M}^+$ has a gain on the number of output queries over ${L_M}^*$.

The crux of the complexity comparison comes from the fact that when we deal with real systems, they work on huge data sets as their possible inputs. When these systems are learned, the size of the input set I becomes large enough to cripple the learning procedure. In most cases, $|I|$ is a dominant factor over the number of the states n. Therefore, when we look on the parts of the complexity calculations which exhibit a difference, i.e., $|I|^2nm$ for ${L_M}^*$ and $|I|^2n$ for ${L_M}^+$, then it is obvious that ${L_M}^+$ has a clear gain over ${L_M}^*$ as $|I|$ grows[3].

Another aspect of the complexity gain of ${L_M}^+$ comes from the fact that it is not easy to obtain always "smart" counterexamples that are short and yet can find the difference between the black box machine and the conjecture. Normally, we obtain counterexamples of arbitrary lengths in practice (without assuming a perfect oracle). They are usually long input strings that run over the same states of the black box machine many times to exhibit the difference. When ${L_M}^*$ processes such counterexamples in the observation table by adding all the prefixes of the counterexample to S_M, it adds unnecessarily as many states as the length of the counterexample. This follows the extension of the table due to $S_M \cdot I$. However, after filling the table with output queries, it is realized that only few prefixes in S_M are the potential states. On the contrary, the method for processing counterexample in ${L_M}^+$ consists in adding the suffixes of only a part of the counterexample to E_M. Then, ${L_M}^+$ finds the exact rows through output queries which must be the potential states and then moves the rows to S_M (see Section 5.4). So, the length of a counterexample m is less worrisome when applying ${L_M}^+$. As m becomes large, ${L_M}^+$ has more gain over ${L_M}^*$.

From the above discussion, we conclude that ${L_M}^+$ outperforms ${L_M}^*$, notably when the size of the input set I and the length of the counterexamples m are large. We have also confirmed the gain of ${L_M}^+$ over ${L_M}^*$ by experimentation on *CWB* [17] workbench of synthetic finite state models of real world systems, as well as on the random machines, where m, I and n are of different sizes.

The research in the approach of combining learning and testing has remained our major focus in the recent past [13]. We intend to continue our research in these directions to explore the benefits of our approach in disciplines, such as learning other forms of automata and its application on the integrated systems of black box components.

[3] Contrary to CWB examples which are small enough to realize the impact of the input size on the complexity, the experiments with random machines with large input sizes provides a good confidence on the gain.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 2, 87–106 (1987)
2. Muccini, H., Polini, A., Ricci, F., Bertolino, A.: Monitoring architectural properties in dynamic component-based systems. In: Schmidt, H.W., Crnković, I., Heineman, G.T., Stafford, J.A. (eds.) CBSE 2007. LNCS, vol. 4608, pp. 124–139. Springer, Heidelberg (2007)
3. Shu, G., Lee, D.: Testing security properties of protocol implementations-a machine learning based approach. In: ICDCS, p. 25. IEEE Computer Society, Los Alamitos (2007)
4. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. In: Proceedings of FORTE 1999, Beijing, China (1999)
5. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003)
6. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: HLDVT, pp. 95–100. IEEE Computer Society, Los Alamitos (2004)
7. Niese, O.: An Integrated Approach to Testing Complex Systems. PhD thesis, University of Dortmund (2003)
8. Mäkinen, E., Systä, T.: MAS - an interactive synthesizer to support behavioral modelling in UML. In: ICSE 2001, pp. 15–24. IEEE Computer Society, Los Alamitos (2001)
9. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005)
10. Li, K., Groz, R., Shahbaz, M.: Integration testing of components guided by incremental state machine learning. In: TAIC PART, pp. 59–70. IEEE Computer Society, Los Alamitos (2006)
11. Pena, J.M., Oliveira, A.L.: A new algorithm for the reduction of incompletely specified finite state machines. In: ICCAD, pp. 482–489. ACM, New York (1998)
12. Frazier, M., Goldman, S., Mishra, N., Pitt, L.: Learning from a consistently ignorant teacher. J. Comput. Syst. Sci. 52(3), 471–492 (1996)
13. Shahbaz, M.: Reverse Engineering Enhanced State Models of Black Box Components to support Integration Testing. PhD thesis, Grenoble Universities (2008)
14. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. In: Machine Learning: From Theory to Applications, pp. 51–73 (1993)
15. Balcazar, J.L., Diaz, J., Gavalda, R.: Algorithms for learning finite automata from queries: A unified view. In: AALC, pp. 53–72 (1997)
16. Berg, T., Raffelt, H.: Model checking. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 557–603. Springer, Heidelberg (2005)
17. Moller, F., Stevens, P.: Edinburgh Concurrency Workbench User Manual, Version 7.1 (2004), `http://homepages.inf.ed.ac.uk/perdita/cwb/`
18. Berg, T., Jonsson, B., Leucker, M., Saksena, M.: Insights to angluin's learning. Electr. Notes Theor. Comput. Sci. 118, 3–18 (2005)