Example-1: A Working Example of Model Learning

Shahbaz Ali (Pakistan)
Email: bazgikian@yahoo.com
October 13, 2019

Abstract

In this section, we explain the *Model Learning* process (a.k.a active automata learning) with the help of an example. First, we refresh our knowledge on basic concepts like learning algorithms (the learner), data structures (especially observation table which has been used in the example), etc then we explain the model learning process using an example, as discussed in the section 3. In this example, we learn a DFA model shown in Figure 4 which accepts the regular language with an even number of 0's and an even number of 1's (the numbers of 0's and 1's are not required to be equal). In this running example, we use a well-known Angluin's L* algorithm for posing MQs and EQs, and to handle counterexample, we use the approach of Rivest & Schapire to refine the hypothesis.

1 Basic Algorithm

1.1 The Angluin L* Algorithm

In the context of active learning, Dana Angluin proposes a query learning algorithm [1], L^* for inferring DFA which describes regular sets. The learning of DFA with L* algorithm is a typical example of polynomial-time learning via queries. Many efficient active learning algorithms have been designed and developed since then, and most of them follow Angluin's approach of MAT framework. The main components of the L^* algorithm have been shown in Figure 1. Learner poses two types of queries:

1. *Membership Queries (MQs):* With *MQs*, the learner asks whether the input sequence belongs to the target system. The answer is *"Yes"* if it belongs to, otherwise *"No"*.
2. *Equivalence Queries (EQs):* With *EQs*, the learner asks whether the hypothesized/conjectured automaton $\mathcal{H}$ is correct i.e., $\mathcal{H} \approx \mathcal{M}$. The teacher who knows about the system completely answers *"Yes"* if this is the case otherwise it answers *"No"* and provides a counterexample that differentiates $\mathcal{H}$ from $\mathcal{M}$ such that $\mathcal{H} \neq \mathcal{M}$.

The learner records the received responses (0 or 1) into a data structure called an *observation table* (section: 2.1). In this learning process when the table becomes *closed* and *consistent* then learner builds an automaton called conjecture/hypothesis with the help of a filled observation table.

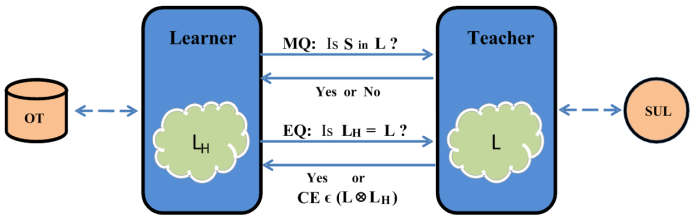


Figure 1: Components of L* Algorithm

Now, to check whether this hypothesis is equivalent to the model of black-box SUL, the algorithm depends upon the existence of some oracle. To check the validity, the learner uses *EQs* (*yes/counterexample query*). If the black-box SUL is not equivalent to the hypothesis, then the oracle generates a counterexample which is a string from input set that is accepted by the black box and rejected by hypothesis and vice-versa. By analyzing the counterexample in an intelligent way, the learner refines the observation table and constructs an improved version of the hypothesis. The whole process continues in a loop until the learner returns an automaton whose behavior matches with the black-box SUL.

For a regular set, the Angluin's L^* returns a minimal DFA $\mathcal{M}$ provided that the reply of every query is given correctly w.r.t the target regular set. Moreover, L^* obtains $\mathcal{M}$ in polynomial time so, the regular set class, which is represented by DFA, is polynomial-time learnable.

2 Data Structures

Model learning algorithms normally vary in two aspects: (1) the data structures used for storing responses of queries and realizing the black-box abstraction, (2) and how learning algorithms handle counterexamples. The basic learning algorithm L* and its variants used two types of data structures: *observation table* and *discrimination tree.*

2.1 Observation Table

An observation table $\mathcal{T}$ is the most well-known data structure, and it was introduced first by Gold[2]. Later on, Angluin used this data structure in her seminal algorithm L* for organizing the information collected during the interaction with the SUL. Different learning algorithms have used variants of observation table for other modeling formalisms.

Let Σ be the input alphabet, and Ω be the corresponding output alphabet of the SUL model. The problem of learning model (i.e., constructing Q, q_0, Σ, Ω) from *queries* and *observations* is called *black-box learning.* The observation table $\mathcal{T}$ may be a dynamic two-dimensional array that is characterized by upper and lower parts. The entries in the table are the values from the output alphabet set Ω. The rows and columns of this table are indexed by strings over Σ^*. As learning the model of an SUL is an incremental process, so the table is allowed to grow accordingly over time. Formally, it can be represented, as a triple over an alphabet Σ, by (S, E, T). Here, *S* and *E* are sets of strings over Σ which are non-empty and finite. During the learning process, the table is updated continuously by the learner, and it can be visualized by three distinguished indexing sets: (1) $S \subseteq \Sigma^*$ is *prefix-closed* set of input strings. The values in this part (upper left of $\mathcal{T}$) are used to represent the states of minimal state-model, (2) $S.\Sigma$ is also a *prefix-closed* set of input strings. The values in this part (lower-left of $\mathcal{T}$) are used in building transitions of state-model, and (3) $E \subseteq \Sigma^*$ is a *suffix-closed* set of input strings. The rows and columns of $\mathcal{T}$ are indexed by the sets $(S \ U \ S.\Sigma)$ and *E* respectively.

In the above expression (S, E, T) of an observation table, *T* is a finite function which is defined (for Angluin's Lerner L*) as $\mathcal{T} : ((SUS.\Sigma)\ x\ E) \rightarrow \{0, 1\}$. For any row $s \in (SUS.\Sigma)$, column $e \in E$, the function $T(s, e)$ represents the corresponding cell in table. The value of T(s,e) is "1" if the string s.e is accepted by the target SUL and "0" in other case. In order to construct an automaton from the filled observation table, $\mathcal{T}$ must fulfil the properties of *closed* and *consistent.* Let, $\mathcal{T} : ((SUS.\Sigma) \times E) \rightarrow \Omega$ is a two dimensional observation table. (1) $\mathcal{T}$ is **closed** if $\overline{s} \in S$ and $\sigma \in \Sigma$, there exists $\overline{q} \in S$ such that the two rows $\mathcal{T}[\overline{s}.\sigma]$ and $\mathcal{T}[\overline{q}]$ becomes equal i.e., $\mathcal{T}[\overline{s}.\sigma] = \mathcal{T}[\overline{q}]$. (2) $\mathcal{T}$ is **consistent** if whenever $\overline{p}, \overline{q} \in S$ such that $\mathcal{T}[\overline{p}] = \mathcal{T}[\overline{q}]$ then for $\forall\ \sigma \in \Sigma$ we have $\mathcal{T}[\overline{p}.\sigma] = \mathcal{T}[\overline{q}.\sigma]$.

2.2 Discrimination Tree

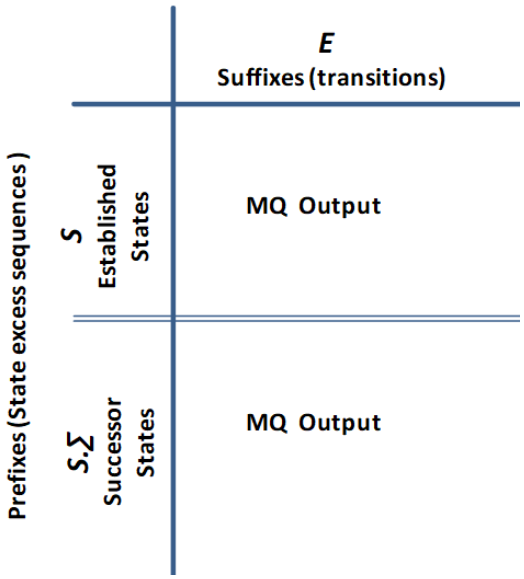
The second data structure used by many learning algorithms is a discrimination tree (DT) which is a decision tree for determining the equivalence of states. It is a classification scheme for distinguishing between states and first introduced by Kearns & Vazirani[3].

Figure 3b shows a discrimination tree that is obtained by applying *Observation Pack* algorithm on SUL having formal behavior shown in Figure 3a. Here, leaves represent the states of hypothesis and discriminators labeled the inner nodes. Further, every node holds two children, i.e., 0-child (represented by the dashed line) and 1-child (represented by a solid line). Two well-known operations which are used to get information from a discrimination tree are: (1) *"Sifting"* and (2) computing *"lowest common ancestor (LCA)".* The "sifting" explores the tree for classification, and "LCA" emphasizes the separation of classes presented in a tree.

For further details about discrimination tree data structure, we refer the interested readers to[4-6] and to our survey paper with the title name as: "Model Learning: A Survey of Foundations, Tools and Applications".

3 A Running Example for Model Learning

In this section, we explain the active model learning process with the help of an example. In this example, we learn a DFA model shown in Figure 4 which accepts the regular language with an even number of 0's and an even number of 1's (the numbers of 0's and 1's are not required to be equal). The formal definition of this DFA is as: SUL states (Q) are $\{q_0, q_1, q_2, q_3\}$, initial state is q_0, final state (F) is also q_0, input alphabet (Σ) is $\{0, 1\}$, and transition function (δ)

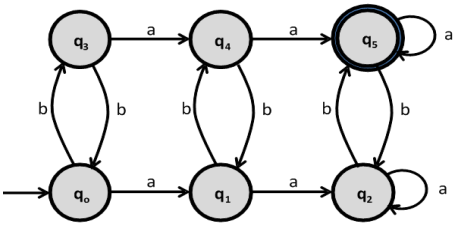


(a) Layout of OT

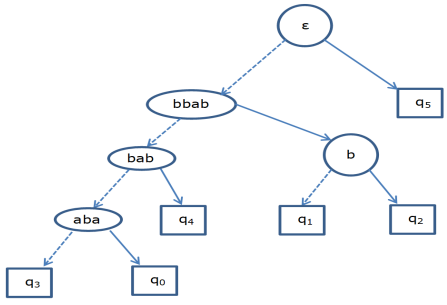
	ε	bbab	bab	b	aba
ε	0	0	0	0	1
a	0	1	0	0	1
ab	0	0	1	0	0
aba	1	0	1	0	0
aa	0	1	0	1	1
b	0	0	0	0	0
aab	1	0	1	1	0
abab	0	1	0	0	1
abb	0	1	0	0	1
abaa	1	0	1	1	0
aaa	0	1	0	0	1
ba	0	0	1	0	0
bb	0	0	0	0	1

(b) An Example of Filled OT

Figure 2: An Observation Table



(a) A Simple DFA



(b) Discrimination Tree using *Observation Pack* algo [4].

Figure 3: A simple DFA with Discrimination Tree

for this SUL is $\delta(q_0, 1) = q_1$, $\delta(q_0, 0) = q_2$, $\delta(q_1, 1) = q_0$, $\delta(q_1, 0) = q_3$, $\delta(q_2, 0) = q_0$, $\delta(q_2, 1) = q_3$, $\delta(q_3, 1) = q_2$, $\delta(q_3, 0) = q_1$.

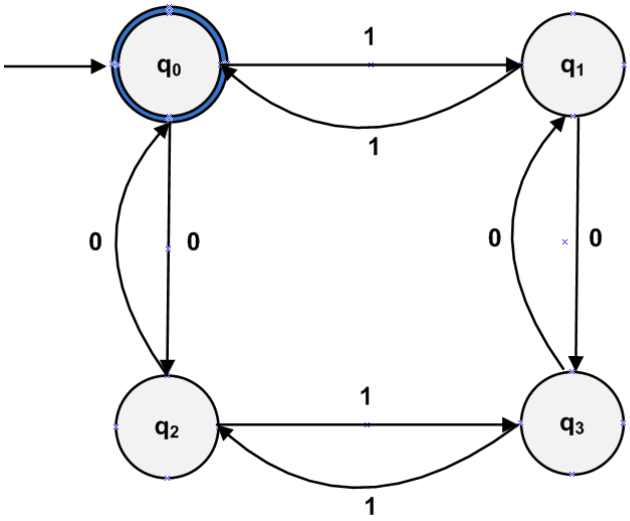


Figure 4: A Simple DFA

We explain model learning process using L* algorithm which is used for active learning of DFAs. The learner (L* algorithm), makes three assumptions for learning DFAs: (1) the SUL (in this case a DFA model shown in Figure 4) can be represented as a DFA, i.e, $D = \{(Q,\ q_0,\ \Sigma,\ \delta,\ F)\}$, (2) the learner knows the alphabet of the SUL (or D), and (3) there is an oracle (a.k.a, teacher) who knows the SUL (or D). It learns a DFA with the minimal number of states that accepts the same language of *D*. During the learning process, the learner poses two kinds of queries to the teacher for getting information about *D*: membership queries and equivalence queries. A membership query asks whether a string 'str' is accepted by *D*, i.e., $str \in L(D)$; and equivalence query asks whether a hypothesis DFA H accepts the same language of *D*, i.e., $L(H) = L(D)$. During the learning process, the learner stores the results of membership queries (*MQs*) in an *Observation Table (OT)*, which also has been discussed in 2.1. Formally, an *OT* is a triple over an alphabet Σ, and can be denoted as (P, E, T), where $P \subseteq \Sigma^*$ is a *prefix-closed* set of finite strings (i.e., for each string $str \in P$, all the prefixes of *str* are also in P) and P corresponds to the row indexes of the observation table; $E \subseteq \Sigma^*$ is a *suffix-closed* set (i.e., for each string $str \in E$, all its suffix strings are also present in E) and E corresponds to the column indexes of the observation table; T is a mapping function which is defined as: $T(str, str') = 1$ if str is a string in P or any string in P but appended with a symbol in Σ, and str' is a string in E and $str.str'$ (which is the concatenation of *str* and str') is a string accepted by*D* (or in other words by SUL); otherwise $T(str, str') = 0$. Intuitively, the mapping function T maps the table cell indexed by row *str* and column indexed by str' to either 1 or 0 depending upon whether the string $str.str'$ is accepted by*D* (i.e., SUL) or not. As discussed in the section 2.1, that the rows of the observation table are separated into two categories and their corresponding row indexes are denoted by S and $S.\Sigma$ i.e., $P = S \cup S.\Sigma$. And, for each row index $str' \in S.\Sigma$, there exist a string $str \in S$ and a symbol $e \in \Sigma$ such that $str.e = str'$. In this observation table, the learner (L* algo) categorizes strings on the basis of Myhill-Nerode Theorem[7].

Definition 3.1 ***(Myhill-Nerode Relation)****. Let $D = \{(Q,\ q_0,\ \Sigma,\ \delta,\ F)\}$ be a DFA with every state as reachable. A Myhill-Nerode Relation $\equiv_D$ on Σ^* w.r.t D, is an equivalence relation and defined as: $str_1 \equiv_D str_2 \overset{def}{\iff} \delta(q_0, str_1) = \delta(q_0, str_2)$ for any two strings $str_1, str_2 \in \Sigma^*$.*

As Myhill-Nerode Relation is an equivalence relation[1], i.e., it is reflexive, symmetric and transitive by definition. This equivalence relation also satisfies following three additional properties[7-8].

(1) *Right Congruence Property*: Right congruence property holds for this equivalence relation. For $str_1, str_2 \in$

[1]In mathematics, an equivalence relation is a binary relation. It is reflexive, transitive and symmetric. For example, the relation "is equal to" is the canonical example of an equivalence relation because for any objects x, y, and z, we have (1) $x = x$ (reflexive property), (2) if $x = y$ then $y = x$ (symmetric property), and (3) if $x = y$ and $y = z$ then $x = z$ (transitive property).

Σ^* and $i \in \Sigma$, $str_1 \equiv_D str_2 \Rightarrow str_1.i \equiv_D str_2.i$. As we have $\delta(q_0, str_1) = \delta(q_0, str_2)$, so it means that str_1 and str_2 lead to the same state. Let 'q' be the same state, so we can write as $\delta(q_0, str_1.i) = \delta(q, i) = \delta(q_0, str_2.i)$.

(2) *Refinement of L(D)*: The equivalence relation refines the language of *DFA D*, i.e., $str_1 \equiv_D str_2 \Rightarrow str_1 \in L(D) \iff str_2 \in L(D)$. As $\delta(q_0, str_1) = \delta(q_0, str_2)$ moves to same state (say 'f'), and if 'f' is the accepting state (final state), then both str_1 and str_2 are accepted by *D* (i.e., by DFA). Using this and the previous property, we can conclude that $str_1.str'$ is accepted by *DFA D iff* $str_2.str'$ is also accepted by *DFA D*, $\forall\ str' \in \Sigma^*$.

(3) *Formation of Equivalence Classes*: $\forall\ str_1, str_2 \in \Sigma^*$, if there exist $\delta(q_0, str_1) = \delta(q_0, str_2)$, i.e., str_1 and str_2 transit to same state from initial state, then we can say that all such strings belong to one equivalence class. Formally, it can be written as $\{str \in \Sigma^* \mid \delta(q_0, str) = q\}$ i.e., there is exacly one equivalence class corresponding to each state 'q' in *D*. We denote the equivalence class for string str_1 as $[str_1]_D$ (here str_1 is called a *'representing string'* for the equivalence class $[str_1]_D$), and formally, it can be defined as: $[str_1]_D \stackrel{def}{=} \{str_2 \mid str_2 \equiv_D str_1\}$. Thus, because of reflexive, transitive and symmetric properties, any equivalence relation partition the underlying set into disjoint equivalence classes. Two elements of the given set are equivalent to each other *iff* they belong to the same equivalence class.

Theorem 3.1 ***(Myhill-Nerode Theorem)****. A language $L_R \in \Sigma^*$ is said to be regular iff there exist a Myhill-Neorde relation for L_R.*

Finite automata and regular expressions are two different ways to represent regular languages. In theoretical computer science and formal language theory, a regular language is a formal language that can be expressed with a *regular expression* or by a *finite automaton*. So, a Myhill-Nerode relation is equivalent to a DFA (finite automaton) or regular expression. To obtain (or learn) an unknown *DFA D* using L* algorithm (the learner), our task is to find Myhill-Nerode relation from the observation table (which is used by L*).

Definition 3.2 ***(Bit-Vector)****. In an observation table (P, E, T), we denote Row(str) as a bit-vector which represents the values for the row indexed by $str \in P$. The values present in Row(str) is the concatenation of all cell values provided by the mapping function $T(str, str')$, where $str' \in E$.*

During the learning process, the learner (L* algorithm) poses *MQs* to SUL(teacher) and saves the responses in the observation table. As soon as the observation table becomes *closed* and *consistent*, the learner constructs a hypothesis (a candidate DFA) from the observation table using Myhill-Nerode theorem. Thus, to make observation table as *closed* and *consistent*, is the objective of L*.

Definition 3.3 ***(Closed Property of OT)****. An observation table is said to be closed if, $\forall\ str \in S$ and $i \in \Sigma$, there is a string $str' \in P$ such that $Row(str \cdot i) = Row(str')$. It is important to note here that str' can be indexed by S and $S \cdot \Sigma$.*

Definition 3.4 ***(Consistent Property of OT)****. An observation table is said to be consistent if whenever $str_1, str_2 \in S$ such that $Row(str_1) = Row(str_2)$, then $Row(str_1 \cdot i) = Row(str_2 \cdot i)$ for all $i \in \Sigma$.*

3.1 Rules for Constructing a Hypothesis

When the table becomes closed and consistent then the learner constructs a hypothesis (a candidate DFA) i.e., H=$(Q_h, q_h, \Sigma, \delta_h, F_h)$ using Myhill-Nerode Theorem in the following way:

(1) Q_h consist of different states. It contains exactly one state for each different row indexed by a string in S. Equivalent rows (having same bit-vectors) in OT correspond to the *same state* because they have same equivalence class.

(2) q_h is the hypothesis initial state which corresponds to the row indexed by the empty row string $\langle\rangle$, i.e., $Row(\langle\rangle)$.

(3) δ is the transition function for the constructed hypothesis. And, $\forall\ s \in Q_h$ which corresponds to a row indexed by string $str \in S$ and a symbol $i \in \Sigma$, $\delta_h(s, i) = s_p$, where s_p is the state for row indexed by the string $str \cdot i$ in P.

(4) F_h is the final or accepted state for the constructed hypothesis. A state s will lie in F_h *iff* its corresponding row indexed by string $str \in S$ such that $T(str, \langle\rangle) = 1$.

3.2 Counter-Example and Witness Suffix

After constructing the hypothesis *H*, the learner puts an equivalence query (EQ) to the teacher to check whether *L(H)* is equivalent to *L(D)*. If the hypothesis *H* accepts the same language as that of *DFA D* i.e., *L(H)* $\approx$ *L(D)* , then the learner returns the hypothesis *H* as the final model and terminates the learning experiment. In other case, i.e., *L(H)* $\neq$ *L(D)*, then the teacher returns a counterexample string $ce \in \Sigma^*$ such that either $ce \in L(D) \wedge ce \notin L(H)$ is true or $ce \in L(H) \wedge ce \notin L(D)$ is true, but not both are true at the same time. In case of counterexample (ce) found, the learner analyzes it in an intelligent way to find a *witness suffix (WS)*. A witness suffix is a string that classified the two strings, which were previously placed in same equivalence class during learning, when appended to the strings it separate them (the two string) into two equivalence classes under the Myhill-Nerode Relation [8]. To explain it with an example, let *str* be a string which is obtained by concatenating two strings str_0 and str_1, i.e., $str = str_0 \cdot str_1$. Let, *q* be the state reached from initial state q_0 by traversing the string str_0, i,e, $\delta(q_0, str_0) = q$. Now, we can call str_1 as the *witness suffix (WS)* of *str*, denoted by *WS(str)*, if $\delta(q, str_1) = q'$ and $q' \neq \delta(q_0, str)$. During learning, once the witness suffix WS(ce) is obtained, the learner L* uses it and refine the observation table. And, using the refined OT, the learner constructs the improved version of hypothesis *H*. This process continues until, we get the final model which satisfies the condition of *L(H) = L(D)*.

3.2.1 Learning Complexity of the Learner (L*)

A learning algorithm's complexity is normally calculated in terms of the required number of membership and equivalence queries. In her seminal work, Angluin [1] proved this fact that as long as *L(D)* is regular, L* returns a minimum DFA which accepts the same language of *D*. During the whole learning process, the learner (L*) poses at most *(n-1)* equivalence queries and $O(|\Sigma|n^2 + nlogm)$ membership queries (*MQs*), where *m* is the maximum length of counterexample strings returned by the oracle (teacher) and *n* represents the number of states of the minimal DFA.

3.2.2 Improving Learning Complexity of the Learner (L*)

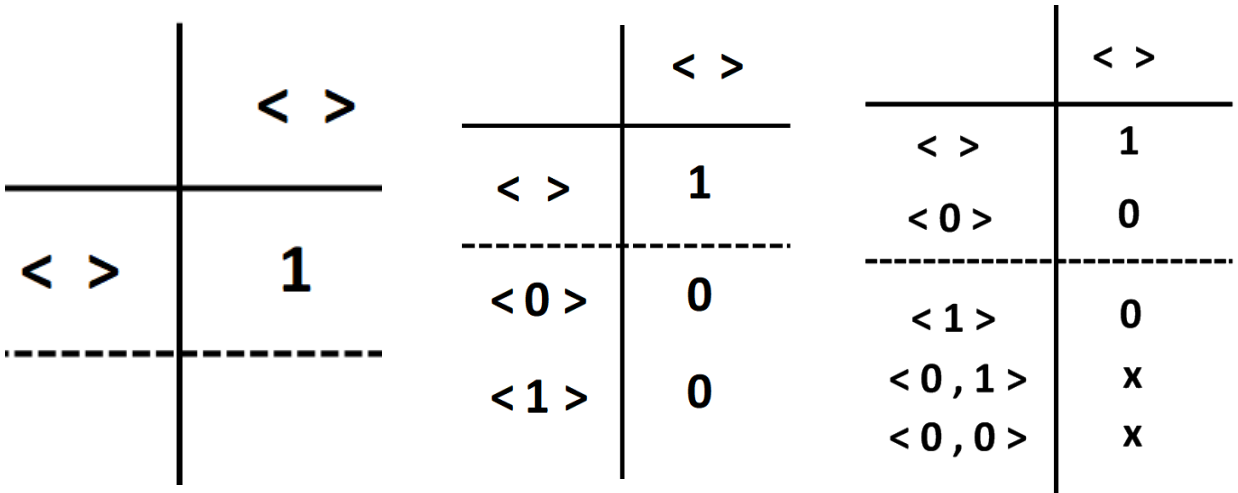
Since 1987, a number of changes and extensions were made in the original L* algorithm. For example, this algorithm was extended by Niese [9] to model Mealy machines. Besides, Kearns & Vazirani [3] improved the L* algorithm by replacing the observation table with discrimination tree which reduces the number of MQs. As in finding counterexample process, the learner, i.e., L* adds all the prefixes of a counterexample in rows of the table and again starts posing MQs to refine the hypothesis. Counterexample generator may produce a long, not minimal counterexamples which result in posing of numerous redundant MQs. Rivest & Schapire [10] observed an inefficiency in L^* algorithm regarding counterexample's prefixes which were being added as rows in the observation table. They pointed out that it is unnecessary to add all the prefixes of a counterexample to the observation table as rows. This discrepancy can be removed by adding a single and well-selected suffix to table as a column. In this running example, we use L* algorithm for posing MQs and EQs, and to handle counterexample, we use the approach of Rivest & Schapire to refine the hypothesis.

3.3 A Running Example

Suppose the *DFA D*, which is shown in Figure 4, is the one to be learned (SUL). At the beginning, the learner (L*) fills the row and column indexes by an empty string $\langle\rangle$. It poses *MQ* to the teacher (SUL which is *D* in this case) by asking whether the empty string $\langle\rangle$ is accepted by *DFA D*. At this moment, the status of observation table is as, $S = \{\langle\rangle\},\ S.\Sigma = \varnothing$, and $E = \{\langle\rangle\}$. The empty string is accepted by *D* (i.e., response of *MQ* from teacher) and thus the value of cell indexed by $(\langle\rangle, \langle\rangle)$ is 1, as shown in the Figure 5a.

Next, for each symbol of Σ, it poses a *MQ* and inserts the results into the lower-part (below dashed line) of observation table i.e., into $S.\Sigma$. In this case, it poses one *MQ* for the string $\langle 0\rangle = \langle 0\rangle.\langle\rangle$ and second *MQ* for the string $\langle 1\rangle = \langle 1\rangle.\langle\rangle$. The answers (responses of these *MQs* from teacher) are both 0, as shown in Figure 5b. At this stage, we write the status of observation table as: $S = \{\langle\rangle\},\ S.\Sigma = \{\langle 0\rangle, \langle 1\rangle\}$, and $E = \{\langle\rangle\}$.

The table shown in Figure 5b is not *closed* because the $Row(\langle 0\rangle) = 0$ (i.e., from lower-part of OT) is not indexed by any string in *S* (i.e., in upper-part of OT). Thus the learning algorithm moves the $Row(\langle 0\rangle)$ from $S.\Sigma$ to S, and $\forall\ a \in \Sigma$, adds $(\langle 0\rangle \cdot\ a)$ to $S.\Sigma$ part (lower-part) of OT, as shown in Figure 5c. This table is not *complete*, and to complete it, the learner asks *MQs* for string $\langle 0, 1\rangle = \langle 0, 1\rangle \cdot \langle\rangle$ and for string $\langle 0, 0\rangle = \langle 0, 0\rangle \cdot \langle\rangle$ and inserts the



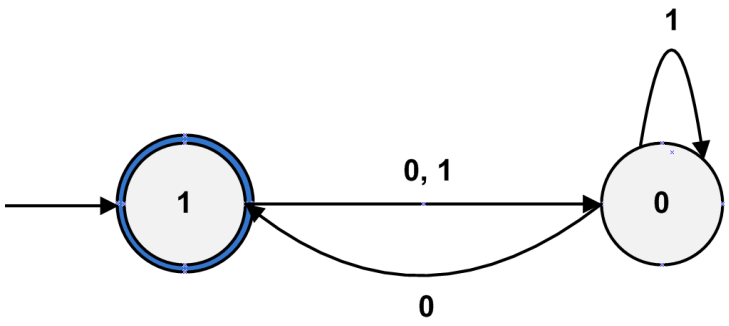
(a) First OT (b) Second OT (c) Third OT

Figure 5: Initial Observation Tables (OTs). The rows above the dashed line are the S rows and the rows below the dashed line are the $S.\Sigma$ rows.

results to $S.\Sigma$ of OT. The complete table is shown in Figure 6a, for which $S = \{\langle\rangle, \langle 0\rangle\}$, $S.\Sigma = \{\langle 1\rangle, \langle 0,1\rangle, \langle 0,0\rangle\}$, and $E = \{\langle\rangle\}$.

	< >
< >	1
< 0 >	0
< 1 >	0
< 0 , 1 >	0
< 0 , 0 >	1

(a) Fourth OT.
(Closed and Consistent)



(b) First hypothesis

Figure 6: First Closed and Consistent OT with First Hypothesis

As the table shown in Figure 6a is *closed* and *consistent*, so L* constructs the first hypothesis DFA, as shown in Figure 6b by using the following procedure:

(1) The two rows in S rows (upper-part of OT), i.e., $state1 = [\langle\rangle]_r$ and $state0 = [\langle 0\rangle]_r$ represent the two states of hypothesis and labelled as 1 and 0. (2) Note that state 1 is the *initial state* because its row index is $\langle\rangle$, and it is also an *accepting state* (final state) because its cell value indexed by the column index $\langle\rangle$ is 1. (3) The two transitions labelled with 0 and 1, start from state 1 and move to state 0 due to the fact that rows indexed by $\langle\rangle \cdot 0 = \langle 0\rangle$ and indexed by $\langle\rangle \cdot 1 = \langle 1\rangle$ are 0. The transition labeled 1, start from state 0 and remains in the same state, i.e., state 0 because the representing string for state 0 is $\langle 0\rangle$ and the row indexed by row index $\langle 0,1\rangle = \langle 0\rangle \cdot 1$ is 0. And, the

transition labeled 0, start from state 0 and move to the state 1, because the row indexed by row index $\langle 0, 0\rangle = \langle 0\rangle \cdot 0$ is 1.

	< >	< 1 >
< >	1	x
< 0 >	0	x
< 1 >	0	x
< 0 , 1 >	0	x
< 0 , 0 >	1	x

(a) Fifth OT

	< >	< 1 >
< >	1	0
< 0 >	0	0
< 1 >	0	1
< 0 , 1 >	0	0
< 0 , 0 >	1	0

(b) Sixth OT

	< >	< 1 >
< >	1	0
< 0 >	0	0
< 1 >	0	1
< 0 , 1 >	0	0
< 0 , 0 >	1	0
< 1 , 0 >	x	x
< 1 , 1 >	x	x

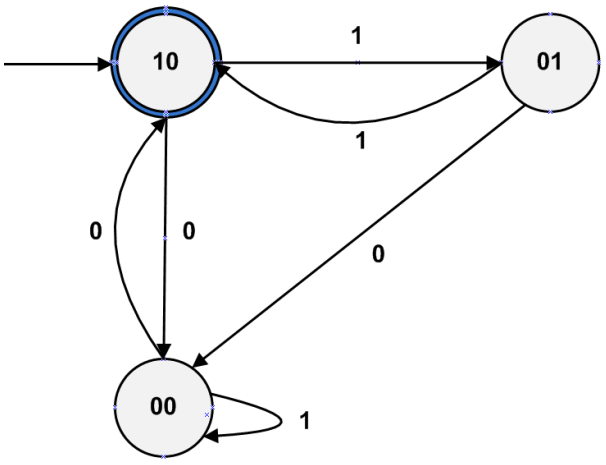
(c) Seventh OT

Figure 7: Next Three Intermediate Observation Tables

After the construction of first hypothesis, now the learner needs its validity (i.e., whether the inferred model is correct or not). For this, it sends an equivalence query (EQ) to an equivalence oracle (teacher). The teacher answers "No" with a counterexample $\langle 1, 1\rangle$, which is accepted by D but rejected by the hypothesis. Now, the learner finds the witness suffix $WS(\langle 1, 1\rangle)$ of the counterexample string. As the longest prefix of the counterexample which is present in $P = SUS.\Sigma$ is $\langle 1\rangle$, so the value of $WS(\langle 1, 1\rangle)$ is $\langle 1\rangle$. Once we have witness suffix, then we add all its suffixes $\{\langle 1\rangle, \langle\rangle\}$ into *E* column. As the suffix $\langle\rangle$ is already present in *E*, so we only add $\langle 1\rangle$ into *E*, as shown in Figure 7a. The table shown in 7a is not complete, so to complete it the learner poses new *MQs* (i.e., $\langle\rangle \cdot \langle 1\rangle, \langle 0\rangle \cdot \langle 1\rangle, \langle 1\rangle \cdot \langle 1\rangle, \langle 0, 1\rangle \cdot \langle 1\rangle, \langle 0, 0\rangle \cdot \langle 1\rangle$) and inserts its results $(0, 0, 1, 0, 0)$ in the observation table, as shown in Figure 7b. This table is not *closed* due to the presence of row indexed by row index string $\langle 1\rangle$ (present in the lower-part of OT). So, the learner moves the row $Row(\langle 1\rangle)$ from $S.\Sigma$ to S, and $\forall\ a \in \Sigma$, adds $(\langle 1\rangle \cdot\ a)$ to $S.\Sigma$ part (lower-part) of OT, as shown in Figure 7c. Again, this table is not *complete*, and to complete it, the learner asks *MQs* for strings $\langle 1, 0\rangle \cdot \langle\rangle$, $\langle 1, 0\rangle \cdot \langle 1\rangle$, $\langle 1, 1\rangle \cdot \langle\rangle$ and $\langle 1, 1\rangle \cdot \langle 1\rangle$, and inserts the results $(0, 0, 1, 0)$ to $S.\Sigma$ of OT, as shown in Figure 8a.

	< >	< 1 >
< >	1	0
< 0 >	0	0
< 1 >	0	1
< 0 , 1 >	0	0
< 0 , 0 >	1	0
< 1 , 0 >	0	0
< 1 , 1 >	1	0

(a) Eighth OT
(Closed and Consistent)



(b) Second Hypothesis

Figure 8: Second Closed and Consistent OT with Second Hypothesis

	< >	< 1 >	< 1 , 0 >
< >	1	0	X
< 0 >	0	0	X
< 1 >	0	1	X
< 0 , 1 >	0	0	X
< 0 , 0 >	1	0	X
< 1 , 0 >	0	0	X
< 1 , 1 >	1	0	X

(a) Ninth OT

	< >	< 1 >	< 1 , 0 >
< >	1	0	0
< 0 >	0	0	0
< 1 >	0	1	0
< 0 , 1 >	0	0	1
< 0 , 0 >	1	0	0
< 1 , 0 >	0	0	1
< 1 , 1 >	1	0	0

(b) Tenth OT

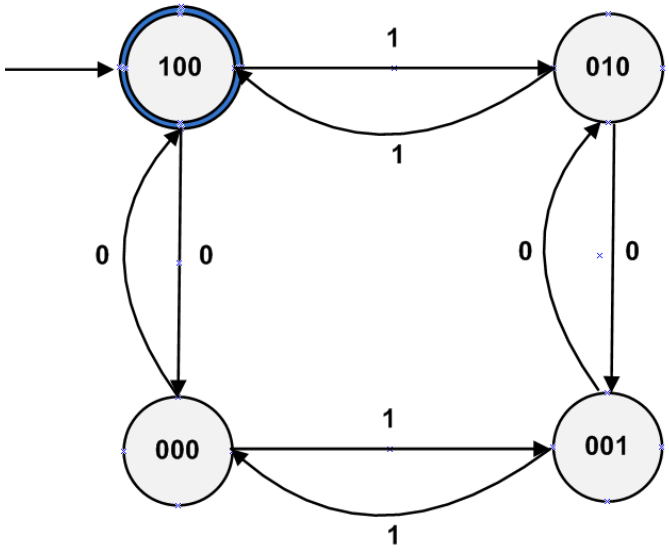
Figure 9: Intermediate OTs After Second Hypothesis

The observation table shown in Figure 8a is closed and consistent. Using this table, the learner constructs the second hypothesis, as shown in Figure 8b. Now, the learner poses second equivalence query to the teacher to check whether the learned model accepts the same language *L(D)*. The teacher again answers "No" and provides a counterexample string $\langle 0, 1, 1, 0 \rangle$ which is accepted by the hypothesis and rejected by the DFA *(D)*.

Next, the learner analyzes the counterexample string $\langle 0, 1, 1, 0 \rangle$ to find witness suffix. As the longest prefix of the counterexample present in $P = SUS.\Sigma$ is $\langle 0, 1 \rangle$, so the value of $WS(\langle 0, 1, 1, 0 \rangle)$ is $\langle 1, 0 \rangle$ which is added into *E*, as shown in Figure 9a.

	< >	< 1 >	< 1 , 0 >
< >	1	0	0
< 0 >	0	0	0
< 1 >	0	1	0
< 0 , 1 >	0	0	1
< 0 , 0 >	1	0	0
< 1 , 0 >	0	0	1
< 1 , 1 >	1	0	0
< 0, 1 , 0 >	0	1	0
< 0, 1 , 1 >	0	0	0

(a) Final OT.
(Closed and Consistent)



(b) Final Learned Model

Figure 10: Final Observation Table and Learned Model

The table shown in 9a, is not complete, so to complete it the learner poses new *MQs* and inserts its results in observation table, as shown in Figure 9b. This table is not *closed* due to the presence of row indexed by row index string $\langle 0, 1 \rangle$ (present in the lower-part of OT). So, the learner moves the row $Row(\langle 0, 1 \rangle)$ from $S.\Sigma$ to S, and $\forall\ a \in \Sigma$, adds $(\langle 0, 1 \rangle \cdot\ a)$ to $S.\Sigma$ part (lower-part) of OT, and the completed observation table has been shown in

Figure 10a. This table is closed and consistent. From this table, the learner constructs a final hypothesis and asks an equivalence query to the teacher for its validity. This time the teacher returns the learned model (no counterexample), which means that the learned model accepts the same language as the one shown in Figure 4. Thus, the learner (L*) successfully learns the unknown DFA and terminates the learning experiment.

In this example, we have explained active model learning using L* learner to infer the model in the form of a DFA. For learning the model of the system in the form of a Mealy machine, we refer the interested readers to [11]. Similarly, for other running examples, we refer the interested readers to [8,12-20]. These running examples employ L* and its variants as learners and demonstrate the operations (such as, posing of *MQs* and *EQs*, handling of counter-examples, saving responses to data structures, etc.) of active model learning process.

References

[1] Angluin D. Learning Regular Sets from Queries and Counterexamples[J]. Information and Computation, 1987, 75(2): 87–106.

[2] Gold E.M. Complexity of Automaton Identification from given Data[J]. Information and Control, 1978.

[3] Kearns M.J., Vazirani U. An Introduction to Computational Learning Theory[M]. An Introduction to Computational Learning Theory, 2018.

[4] Isberner M., Steffen B., Howar F. Learnlib Tutorial: An Open-Source Java Library for Active Automata Learning[A]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)[C].,2015.

[5] Isberner M., Howar F., Steffen B. The Open-Source LearnLib: A Framework for Active Automata Learning[A]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)[C].,2015.

[6] Malte Isberner. Foundations of Active Automata Learning: An Algorithmic Perspective[D]. 2015.

[7] Maier D. Review of "Introduction to Automata Theory, Languages and Computation" by John E. Hopcroft and Jeffrey D. Ullman. Addison-Wesley 1979.[J]. ACM SIGACT News, 1980.

[8] Hao Xiao. Automatic Model Learning and Its Applications in Malware Detection[D]. Nanyang Technological University, 2017.

[9] Hagerer A., Hungar H., Niese O., et al. Model Generation by Moderated Regular Extrapolation[A]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)[C].,2002.

[10] Rivest R.L., Schapire R.E. Inference of Finite Automata Using Homing Sequences[J]. Information and Computation, 1993.

[11] Shahbaz M., Groz R. Inferring Mealy Machines[A]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)[C].,2009.

[12] Steffen B., Howar F., Merten M. Introduction to Active Automata Learning from a Practical Perspective[A]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)[C].,2011.

[13] Vaandrager F. Model Learning[J]. Communications of the ACM, 2017.

[14] Aarts F. Tomte: Bridging the Gap between Active Learning and Real-World Systems[D]. Radboud University Nijmegen, 2004.

[15] Fiterau-Brostean P. Active Model Learning for the Analysis of Network Protocols[D]. Radbound University, Nijmegen., 2018.

[16] Irfan M.N. Analysis and Optimization of Software Model Inferenence Algorithms[D]. Universite De Grenoble, 2012.

[17] Czerny M.X. Learning-Based Software Testing: Evaluation of Angluin's L* Algorithm and Adaptations in Practice[D]. Karlsrushe Institute Technology, 2014.

[18] Henrix M. Performance Improvement in Automata Learning[D]. Radboud University, Nijmegen, 2015.

[19] Uijen J. Learning Models of Communication Protocols Using Abstraction Techniques[D]. Radbound University,Nijmegen, 2009.

[20] Aarts F. Inference and Abstraction of Communication Protocols[D]. Uppsala University, 2009.