CS 4510: Automata and Complexity **Fall 2019**

Lecture 10: Learning a DFA

September 30, 2019

Lecturer: Santosh Vempala *Scribe: Xiaofu Niu*

10.1 Introduction

Suppose we have a fixed regular language L in mind. This regular language L is decided by some DFA M. Now we want somebody else to guess set of rules that define this language L. We can answer two kinds of question: given a string, is it in L? Given a DFA, is it identical to M?

This is similar to the puzzle game we did in the lecture. In the class, we can either give a challenge or make a guess of the five-word rule. In fact, those questions are defined as **membership query** and **equivalence query**. Based on the response of thoes questions, we can rebuild the DFA that defines this regular language.

Definition 10.1 ***Membership queries*** *consist of a string $w \in \Sigma^*$. The answer is either yes or no depending on whether w is in this language.*

Definition 10.2 ***Equivalence queries*** *take a hypothesis DFA. If the hypothesis DFA is identical to M, the output is yes. Otherwise the output is no and a counterexample (A counterexample is a string that is accepted by the hypothesis DFA but not by M or vise versa).*

Suppose we have a regular language $L \in \Sigma^*$ and two finite collections of strings such that $x_1, x_2, \ldots x_n \in L$ and $y_1, y_2, \ldots y_m \notin L$. Based on these information, we can use a Turing machine to find the minimal DFA for L. Let the Turing machine enumerates all possible DFAs based on number of states. Then feed two sets into this DFA. Output the first DFA that accepts all x_i and rejects all y_j. Otherwise fetch the next DFA and repeat.

The above algorithm will find the minimal DFA that defines language L. However, this algorithm might take a very long time (even exponential time) and is very inefficient. Does there exist a more efficient way to do that?

10.2 Angluin's Algorithm

Suppose we have a regular language $L \in \Sigma^*$ and a DFA M that recognizes it. In Angluin's algorithm, a "learner" can ask membership queries and equivalence queries to the "teacher" (someone who has access to L and M). Based on the returned results, it finds the minimal DFA M for a regular language L in number of steps polynomial in the number of states of the DFA M and the length of the longest counterexample.

10.2.1 Observation Table

The Angluin's Algorithm maintains a **observation table** which holds the set of candidate states of a DFA S and a set of query strings E. Both S and E are set of strings over Σ^*. S is a nonempty finite prefix-closed set(A set is prefix $\Longleftrightarrow$ every prefix of every member of the set is also in the set). E is a nonempty finite suffix-closed set(A set is prefix $\Longleftrightarrow$ every suffix of every member of the set is also in the set). Both S and E contain ϵ by definition. Columns of observation table are labelled by elements from set $S \bigcup (S \cdot \Sigma)$ (S concatenate an alphabet) while rows are labelled by elements from E.

The rule for the entries of observation table is defined as:

$$T(s,e) = \begin{cases} 1 & \text{for} \quad s \cdot e \in L \\ 0 & \text{for} \quad s \cdot e \notin L \end{cases}$$

The interpretation of $T(s,e)$ is that $T(s,e)$ is 1 if and only if s concatenated by e is a string in L.

10.2.2 Closeness and Consistency

The Angluin's Algorithm defines two properties of observation table.

Definition 10.3 *An observation table is* ***closed*** $\iff$ *for each s in S and each a in Σ, $row(s \cdot a)$ is in S.*

Definition 10.4 *An observation table is* ***consistent*** $\iff$ *if $row(s_1) = row(s_2)$, for each a in Σ, $row(s_1 \cdot a) = row(s_2 \cdot a)$.*

Angluin's Algorithm requires the observation table to maintain these two properties all the time. If at some step the observation table is non-closed, we resolve this by adding the string $s \cdot a$ into the set S. If the observation table is non-consistent, we resolve this by adding a into set E. When we have a conterexample, we also add it to S.

Given a closed and consistent observation table, we define a corresponding DFA over alphabet Σ with set of states Q, initial state q_0, set of final states F and transition function δ as followed:

- $Q = \{row(s) : s \in S\}$,
- $q_0 = row(\epsilon)$,
- $F = \{row(s) : s \in S \text{ and } T(s) = 1\}$
- $\delta(rows(s), a) = row(s \cdot a)$

For new DFA M we just constructed, each state represents a unique row in observation matrix. This DFA is a feasible DFA. Furthermore, if other DFAs that is acceptance consistent with M but is not identical to M must have more states than M.

10.3 Example

Consider the run of Angluin's Algorithm with the following language:

$$\mathrm{L} = \{s \in \{0,1\}^* : \#0's \text{ is multiple of } 3\}$$

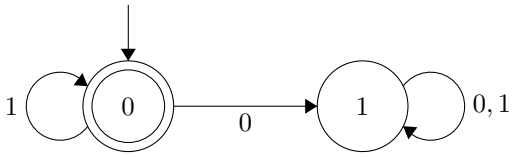
The Algluin's Algorithm begins with set $S = \{\epsilon\}$ and $E = \{\epsilon\}$. Since the empty string ϵ is in L, the entry $T(\epsilon, \epsilon) = 1$. To keep the observation table closed, we add string 0 into S with entry $T(0, \epsilon) = 0$. Given $S = \{\epsilon, 0\}$, we define the set $S \cdot \Sigma = \{1, 00, 01\}$ and add it to the rows. Thus we have a observation table looks like figure 10.1. Since now the size of E is 1, the observation table has only one column labelled by ϵ.

		E
		ϵ
S	ϵ	1
	0	0
$S \cdot \Sigma$	1	1
	00	0
	01	0

Figure 10.1: Observation table 1

This observation table is closed. Since there is only two elements in S their entries are different, the observation table is also consistent. So the next step is to construct a DFA based on it and verify if it is the correct DFA.

Since there are two distinct rows, the DFA have two states. We can label the states with values of $row(s)$ as $\{0, 1\}$. Initial state is the state at entry (ϵ, ϵ), and accepting state is the $row(s)$ whose first column is 1, in this case is state 0. Then we define transition function. Starting from state $0(row(\epsilon))$, the DFA goes to state $0(row(0))$ on input 0 and state $1(row(1))$ on input 1. Similarly, at state 1, the DFA goes to state $0(row(00)$ and $row(01))$ on both input 1 and 0.



Given the hypothetical DFA, we ask the equivalence problem: Is this the DFA that we want? The answer is obviously no. The easiest counterexample is string 000 which should be accepted but ends up at state 1. So we add 000 into S. To keep it closed, we also need to add stirng 00 into S. Now we have $S = \{\epsilon, 0, 00, 000\}$ while E keeps unchanged. The observation table is listed in figure 10.2.

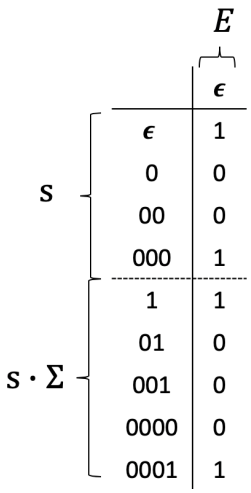


Figure 10.2: Observation table 2

Is this observation table consistent? Unfortunately, no. Take a look at row $row(0)$ and $row(00)$. $row(0) = row(00)$, but appending 0 to them breaks the rule, $row(00) \neq row(000)$. To resolve this, we add the string 0 to E and rebuild the observation table by adding a new column. Notice that entries in the second column does not indicate whether $s \in L$. Instead, it indicates whether $s \cdot 0 \in L$. The new observation table shown in figure 10.3 is closed and consistent.

		E	
		ϵ	0
S	ϵ	1	0
	0	0	0
	00	0	1
	000	1	0
$S \cdot \Sigma$	1	1	0
	01	0	0
	001	0	1
	0000	0	0
	0001	1	0

Figure 10.3: Observation table 3

Next we can construct corresponding DFA for this observation table. Starting with states Q. Although there are four rows in S, but there are only three unique rows $\{00, 01, 10\}$. So DFA has three states. 10 is starting state as well as accepting state. We define transition function in the same manner as we did in the first DFA. The resulting DFA is listed below. This is the correct DFA that recognize the language L.