CS 4540/CS 8803: Algorithmic Theory of Intelligence Fall 2023

Lecture 11: Angluin's algorithm for learning a Finite Automaton

Instructor: Santosh Vempala *Lecture date: 11/1*

Scribed by: Mirabel Reid, Xinyuan Cao

1 Regular expressions and Deterministic Finite Automaton (DFA)

Definition 1 (DFA). *A deterministic finite automaton $M^* = (Q, \Sigma, \delta, q_0, F)$ consists of*

- *A finite set of states Q*
- *A finite set of input symbols called the alphabet Σ*
- *a transition function $\delta : Q \times \Sigma \to Q$*
- *an initial state $q_0 \in Q$*
- *a set of accept states $F \subseteq Q$*

Let $x \in \Sigma^*$ be a string over the alphabet. The automation M^* either accepts or rejects x.

Given a regular expression, one can construct a DFA to determine whether to accept or reject strings. Denote 1 as acceptance and 0 as rejection. One can think of DFA as a machine that takes input a string and output either 0 or 1. We denote $L(M^*)$ as the set of strings that M^* accepts.

For example, let $\Sigma = \{a, b\}$. For the regular expression $a^*baba^*b^*$, we can construct the following DFA, where q_3 and q_4 (green) are the accept states and q_5 (red) is the reject state.

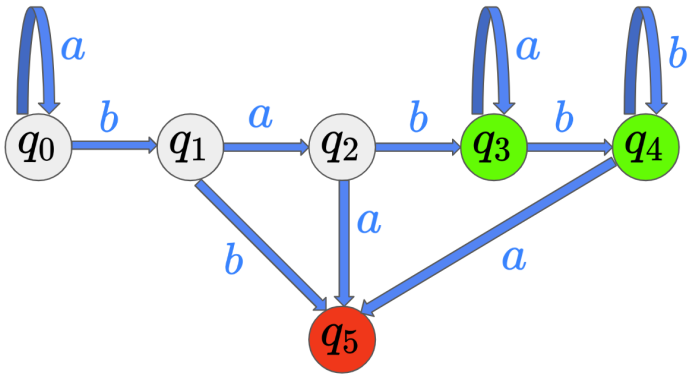


Figure 1: DFA for the regular expression $a^*baba^*b^*$.

2 Efficient Algorithm to Learn DFA

Finding the smallest DFA that accepts a given set of strings and rejects another given set is a difficult problem. A straightforward algorithm is to to enumerate all DFA's of sizes $1, 2, \ldots, N$ and check each one. This process takes time exponential in the size of the target DFA. Instead we will assume that we can make membership queries and equivalence queries. This will allow us to efficiently construct a DFA.

Definition 2 (Membership Query)**.** *For a string $x \in \Sigma^*$, one can query whether $x \in L$. The response will be either YES or NO.*

Definition 3 (Equivalence Query)**.** *For a candidate DFA M, one can query whether $M = M^*$. The response will be either YES, or NO accompanied by a string serving as a counterexample that illustrates an inconsistency between M and M^*.*

The primary goal of the algorithm is to construct a minimal DFA that is consistent with the answers to the membership and equivalence queries made to the oracle. The algorithm incrementally builds an observation table, a data structure used to record and process the information obtained from the queries. We denote the table as (S, E, T). The table has the following components.

- S: a set of strings that is prefix closed. For example, if $110 \in S$, then $1, 11 \in S$.
- E: a set of strings that is suffix closed. For example, if $110 \in S$, then $0, 10 \in S$.
- A finite function $T : (S \cup (S \cdot \Sigma)) \cdot E \to \{0, 1\}$. We use $\cdot$ to indicate the concat.

  For row s and column e, where $s \in S \cup (S \cdot \Sigma), e \in E$, $T(s \cdot e) = \begin{cases} 1 & \text{if } s \cdot e \in L \\ 0 & o.w. \end{cases}$
- For each $s \in S \cup (S \cdot \Sigma)$, let $\text{row}(s)$ be the row of s in the table.

The table records whether strings in the language (formed by concatenating elements from S and E) are accepted or rejected by the DFA. It keeps growing as the algorithm learns more about the DFA. The algorithm is described as in Algorithm 1.

Algorithm 1 Angluin's Algorithm [Angluin 1987]

After initialization step 1, the algorithm iterates step 2 to 4 until until the oracle confirms that the constructed DFA is correct.

1. **Initialization.** Start with S and E with empty set ϵ.
2. **Update.** We update the table by making membership queries for all strings in S concatenated with strings. Specifically, we do the following to keep the table closed and consistent.
   - **Closed**: For $\forall s \in S, \forall a \in \Sigma$, we have $\text{row}(s \cdot a) \in \text{row}(S)$. If not, add the string sa to S and update the table with membership queries.
   - **Consistent**: For $s_1, s_2 \in S$, if $\text{row}(s_1) = \text{row}(s_2)$, then $\forall a \in \Sigma$, $\text{row}(s_1 \cdot a) = \text{row}(s_2 \cdot a)$. If inconsistency is found, add this suffix a to E, and update T with membership queries.
3. **Build the Hypothesis DFA.**
   - Once the table is both closed and consistent, construct a DFA where each unique row in S represents a state. That is, $Q = \{\text{row}(s) : s \in S\}$.
   - The start state is represented by the row for the empty string ϵ. That is, $q_0 = \text{row}(\epsilon)$.
   - Transitions are defined by the behavior of the strings with the addition of an alphabet symbol (from s to $s \cdot a$). That is, $\delta(\text{row}(s), a) = \text{row}(s \cdot a)$.
   - Accepting states are those corresponding to rows where the empty string leads to 'accept'. $F = \{\text{row}(s) : s \in S \text{ and } T(s) = 1\}$.
4. **Equivalence Query and Refinement.**
   - Use an equivalence query to check if the constructed DFA matches the unknown DFA. If the oracle says 'yes', the algorithm concludes.
   - If the oracle responds 'no' and provides a counterexample denoted as x. Add all prefixes of x into S, and update the table correspondingly from step 2 to ensure the closeness and consistency of the table.

2.1 Analysis

By Angluin's Algorithm, we construct a DFA by the observation table (S, E, T). We denote the DFA as M. Each state represents a unique row in the observation table. We will show that M is the smallest such DFA and the algorithm is efficient.

Theorem 4. *If (S, E, T) is a closed consistent observation table, then the output DFA M is consistent with T and M is the smallest such DFA.*

The theorem is proved by the following lemmas.

Lemma 5. *Assume that (S, E, T) is closed and consistent. For the acceptor M and every $s \in S \cup (S \cdot \Sigma), \delta(q_0, s) = row(s)$.*

Proof. We prove by induction on the length of s. The base case is when the length of s is 0, i.e., $s = \epsilon$. Since $q_0 = \text{row}(\epsilon)$, the base case holds. Then we assume that for any $s \in S \cup (S \cdot \Sigma)$ of length at most k satisfies $\delta(q_0, s) = \text{row}(s)$. Let t be an element of $S \cup (S \cdot \Sigma)$ of length $k + 1$. That is $t = s \cdot a$ for some string s of length k and some $a \in \Sigma$. There are two cases. The first case is $t \in S \cdot \Sigma$. Then $s \in S$ and $a \in \Sigma$. The second case is $t \in S$. Since S is prefix-closed, we also have $s \in S$. So for both cases, $s \in S$. This implies that

$$\begin{aligned}
\delta(q_0, t) =& \delta(\delta(q_0, s), a) \\
=& \delta(\text{row}(s), a) \qquad \text{by the induction hypothesis} \\
=& \text{row}(s \cdot a) \qquad \text{by the definition of } \delta \\
=& \text{row}(t) \qquad \text{since } t = s \cdot a
\end{aligned}$$

Then by induction, we complete the proof of the first part. □

Lemma 6. *Assume that (S, E, T) is closed and consistent. Then the acceptor M is consistent with T. That is, for any $s \in S \cup (S \cdot \Sigma)$ and $e \in E$, $\delta(q_0, s \cdot e) \in F$ if and only if $T(s \cdot e) = 1$.*

The lemma is proved by induction on the length of e. We leave the proof to the readers to complete.

Lemma 7. *Assume that (S, E, T) is closed and consistent. Suppose the acceptor M has n states. If $M' = (Q', q_0', F', \delta')$ is any acceptor consistent with T that has n or fewer states, then M' is isomorphic to M.*

Lemma 5 and Lemma 6 show that M is consistent with T. Lemma 7 shows that any other acceptor that is consistent with T is either isomorphic to M or has more states. Thus we conclude the proof of Theorem 4.

Next we analyze the time complexity for the algorithm. Let n be the number of states in the DFA and m be the max length of the string of the counterexamples output by the equivalence query. Then we will show that the algorithm's running time is polynomial in m and n.

Lemma 8. *The total number of update operations (updating if the table is either not closed or not consistent) does not exceed $n - 1$.*

Proof. Suppose that the table is not closed and then $s \cdot a$ is added to S. Then by definition $\text{row}(s \cdot a)$ is different from $\text{row}(s)$ for all $s \in S$ before S is augmented. So the number of distinct values $\text{row}(s)$ is increased by at least one.

Now suppose that the table is not consistent and a string is added to E. Then the number of distinct values $\text{row}(s)$ for $s \in S$ must increase by at least one because the two previously equal values, $\text{row}(s_1)$ and $\text{row}(s_2)$ are no longer equal after E is augmented.

Initially there are one value of $\text{row}(s)$, and there cannot be more than n when the algorithm ends. Thus the total number of update operations is at most $n - 1$. □

Lemma 9. *The algorithm makes at most $n - 1$ incorrect conjectures.*

Proof. Suppose that a conjecture M is found to be incorrect by some counter example. On one hand, M is consistent with T, and thus is the one with minimal states by Theorem 4. On the other hand, the correct minimum acceptor M_U is also consistent with T and in-equivalent to M. So M_U has at least one more state than M. After seeing the counterexample for M, the algorithm will then update S and make the next conjecture M' after making the table closed and consistent. Then we know M' is also consistent with T, but in-equivalent to M. So the next conjecture M' has at least one more state than M. In other words, before termination, the algorithm make a sequence of incorrect conjectures with increasing number of states. Since the correct acceptor has n states, the number of incorrect conjecture cannot exceed $n-1$.

□

Theorem 10. *Given an unknown regular set U, algorithm 1 terminates and outputs an acceptor isomorphic to the minimum DFA that accepting U. The total running time of the algorithm is bounded by a polynomial in m and n.*

Proof. Initially E contains one element ϵ. Each time the table is not consistent, we add one element to E. By Lemma 8, the table is found to be not consistent at most $n-1$ times. So the number of strings in E is at most n. The max length of any string in E is initially zero and is increased by at most one with every string added to E. So the max length of any string in E is $n-1$.

Similarly, S contains one element ϵ in the initial step. Each time the table is not closed, we add one element to S. By Lemma 8, the number of updates by finding the table not closed does not exceed $n-1$. In addition, for each counterexample of length at most m provided by the equivalence query, we add at most m strings to S. By Lemma 9, there are at most $n-1$ counterexamples. So in the end the number of the strings in S cannot exceed $1+n-1+m(n-1) = n+m(n-1)$. The maximum length of any string in S is increased by at most one with every string added because the table is not closed. So the max length of any string in S is at most $m+n-1$.

Combining these, the maximum cardinality of the table $(S \cdot (S \cdot \Sigma)) \cdot E$ is at most

$$(|\Sigma|+1)(n+m(n-1))n = O(mn^2)$$

The max length of any string in $(S \cdot (S \cdot \Sigma)) \cdot E$ is at most

$$m+n-1+n-1+1 = m+2n-1 = O(m+n)$$

So the observation table has size $O(m^2n^2+mn^3)$.

Checking the table's closeness and consistency takes polynomial in size of the observation table, and is done at most $n-1$ times. Adding a string to S or E requires at most $O(mn)$ membership queries of string of length at most $O(m+n)$ to fill the table. Whenever the table is closed and consistent, it also takes polynomial time to build a conjucture DFA, and this is also done at most $n-1$ times. Thus the total running time of the algorithm can be bounded by a polynomial of m and n.

□

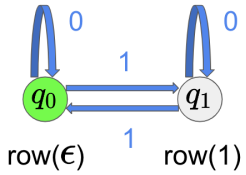
2.2 Examples

Now we provide some examples to better illustrate the algorithm. First, consider the example of a table that is not closed, as shown in Figure 2a. Since $\text{row}(\epsilon) \neq \text{row}(1)$, the table is not closed. As per step 2 of the algorithm, we need to add 1 to S and update the table using membership queries. Consequently, after updating, the table becomes as depicted in Figure 2b, which is now both closed and consistent. Following this, we construct the hypothesis DFA according to the updated table in step 3. This DFA essentially determines whether the input string contains an even number of 1s.

S \ E	ϵ
ϵ	1
0	1
1	0

(a) The table is not closed.

S \ E	ϵ
ϵ	1
1	0
0	1
1 0	0
1 1	1

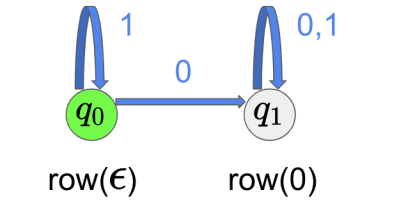


(b) After adding 1 to S and update, it is closed and consistent. We build a hypothesis DFA accordingly. This indicates whether there are even number of 1s in the string.

Figure 2: An example of the observation table with update steps.

Next we see another table that is closed and consistent, shown in Figure 3. According to the table, we build a hypothesis DFA and use the equivalence query. However, the answer is 'No' with the counter example $T(000) = 1$. In this scenario, we have to refine the table according to step 4 in the algorithm. So we add $00, 000$ to S and update the table. As shown in Figure 4a, the refined table is not consistent because $\text{row}(00) = 0$ but $\text{row}(000) = 1$. So we update the table again using step 2, and finally get a closed and consistent table. Then we build a hypothesis DFA according to the table, which is shown in Figure 4b.

S \ E	ϵ
ϵ	1
0	0
1	1
0 0	0
0 1	0



Counter example: T(000)=1

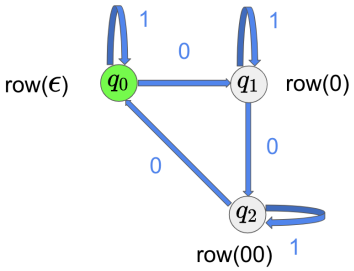
Figure 3: An example of observation table, that is not the correct one after using equivalence query.

S \ E	ϵ
ϵ	1
0	0
0 0	0
0 0 0	1
1	1
0 0	0
0 1	0
0 0 1	0
0 0 0 0	0
0 0 0 1	1

Not consistent!
row(00)=0
row(000)=1

S \ E	ϵ	0
ϵ	1	0
0	0	0
0 0	0	1
0 0 0	1	0
1	1	0
0 0	0	1
0 1	0	0
0 0 1	0	1
0 0 0 0	0	0
0 0 0 1	1	0

(a) The table after refinement is not consistent. We update the table as in step 2 and get a closed and consistent table.



(b) Build a hypothesis DFA according to the new table.

Figure 4: After refining the table in Figure 3, we get the new table, which is not consistent. So we do another update step and then build a new hypothesis DFA based on the new closed and consistent table.