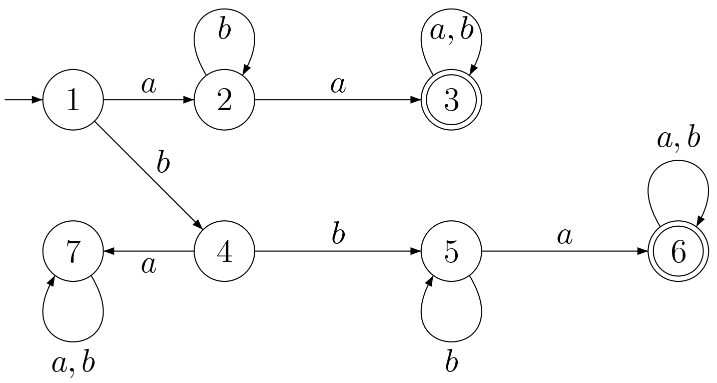
CS 273, Lecture 10
DFA minimization

14 February 2008

In this lecture, we will see that every language has a unique minimal DFA. We will see this fact from two perspectives. First, we'll see a practical algorithm for minimizing a DFA. Second, we will see a theoretical analysis of the situation.

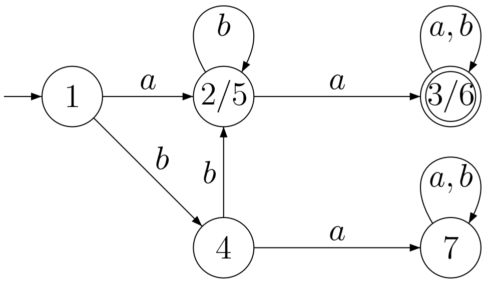
1 On the number of states of DFA

1.1 Starting a DFA from different states



Consider the DFA on the right. It has a particular defined start state. However, we could start it from any of its states. If the original DFA was named M, define M_q to be the DFA with its start state changed to state q. Then the language L_q, is the one accepted if you start at q.

For example, in this picture, L_3 is $(a+b)^*$, and L_6 is the same. Also, L_2 and L_5 are both $b^*a(a+b)^*$. Finally, L_7 is $\emptyset$.



Suppose that $L_q = L_r$, for two states q and r. Then once we get to q or r, the DFA is going to do the same thing from then on (i.e., its going to accept or reject *exactly* the same strings).

So these two states can be merged. In particular, in the above automata, we can merge 2 and 5 and the states 3 and 6. We can the new automata, depicted on the right.

1.2 Suffix Languages

Let Σ be some alphabet.

Definition 1.1 Let $L \subseteq \Sigma^*$ be any language.

The ***suffix language*** of L with respect to a word $x \in \Sigma^*$ is defined as

$$\mathsf{suffix}(L, x) = \left\{ y \;\middle|\; x\,y \in L \right\}.$$

In words, $\mathsf{suffix}(L, x)$ is the language made out of all the words, such that if we append x to them as a prefix, we get a word in L.

The ***class of suffix languages*** of L is

$$\mathcal{C}(L) = \left\{ \mathit{Suffix}(L, x) \;\middle|\; x \in \Sigma^* \right\}.$$

Example 1.2 For example, if $L = 0^*1^*$, then:

- $\mathsf{suffix}(L, \epsilon) = 0^*1^* = L$
- $\mathsf{suffix}(L, 0) = 0^*1^* = L$
- $\mathsf{suffix}(L, 0^i) = 0^*1^* = L$, for any $i \in \mathbb{N}$
- $\mathsf{suffix}(L, 1) = 1^*$
- $\mathsf{suffix}(L, 1^i) = 1^*$, for any $i \geq 1$

Hence there are only two suffix languages for $L = 0^*1^*$: 0^*1^* and 1^*. So $\mathcal{C}(L) = \{0^*1^*, 1^*\}$.

1.2.1 Regular languages have few suffix languages

Now, consider a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting some language L. Let $x \in \Sigma^*$, and let M reach the state q on reading x. The suffix language $\mathsf{suffix}(L, x)$ is precisely the set of strings w, such that xw is in L. But this is exactly the same as L_q. That is, $\mathsf{suffix}(L, x) = L_q$, where q is the state reached by M on reading x. Hence the suffix languages of a regular language accepted by a DFA are precisely those languages L_q, where $q \in Q$.

Notice that the definition of suffix languages is more general, because it can also be applied to non-regular languages.

Lemma 1.3 *For a regular language L, the number of different suffix languages is has is bounded; that is $\mathcal{C}(L)$ is bounded by a constant (that depends on L).*

Proof: Consider the DFA M that accepts L. For any string x, the suffix language $\mathsf{suffix}(L, x)$ is just the languages associated with L_q, where q is the state M is in after reading x.

Indeed, the suffix language $\mathsf{suffix}(L, x)$ is the set of strings w such that $xw \in L$. Since the DFA reaches q on x, it is clear that the suffix language of x is precisely the language accepted by M starting from the state q, which is L_q. Hence, for every $x \in \Sigma^*$, $\mathsf{suffix}(L, x) = L_q$, where q is the state the automaton reaches on x.

As such, any suffix language of L is realizable as the language of a state of M. Since the number of states of M is some constant k, it follows that the number of suffix languages of L is bounded by k. ■

1.2.2 The suffix languages of a non-regular language

Consider the language $L = \left\{ \mathsf{a}^n \mathsf{b}^n \;\middle|\; n \in \mathbb{N} \right\}$. The suffix language of L for a^i is

$$\mathsf{suffix}(L, \mathsf{a}^i) = \left\{ \mathsf{a}^{n-i} \mathsf{b}^n \;\middle|\; n \in \mathbb{N} \right\}.$$

Note, that $\mathsf{b}^i \in \mathsf{suffix}(L, a^i)$, but this is the only string made out of b that is in this language. As such, for any i, j, where i and j are different, the suffix language of L with respect to a^i is different from that of L with respect to a^j (i.e. $\mathsf{suffix}(L, x) \neq \mathsf{suffix}(L, y)$). Hence L has infinitely many suffix languages, and hence is not regular, by the theorem we showed.

Let us summarize what we had seen so far:

- Any state of a DFA for a (regular) language L is associated with a suffix language of L.
- If two states are associated with the same suffix language, that we can merge them into a single state.
- At least one non-regular language $\left\{ \mathsf{a}^n \mathsf{b}^n \;\middle|\; n \in \mathbb{N} \right\}$ has infinite number of suffix languages.

It is thus natural to conjecture that the number of suffix languages of a language, is a good indicator of how many states an automata for this language would require. And this is indeed true, as the following section testifies.

2 Regular Languages and Suffix Languages

We can now state a characterization of regular languages in term of suffix languages.

Theorem 2.1 (Myhill-Nerode theorem) *A language $L \subseteq \Sigma^*$ is regular if and only if the number of suffix languages of L is finite (i.e. $\mathcal{C}(L)$ is finite).*

Moreover, if $\mathcal{C}(L)$ contains exactly k languages, we can build a DFA for L that has k states; also, any *DFA accepting L must have k states.*

The full Myhill-Nerode theorem also shows that all minimal DFAs for L are isomorphic, i.e. have identical transitions as well as the same number of states. But we won't do that part.

Proof: If L is regular, then $\mathcal{C}(L)$ is a finite set by Lemma 1.3.

Second, let us show that if $\mathcal{C}(L)$ is finite, then L is regular. Let the suffix languages of L be $\mathcal{C}(L) = \{\mathsf{suffix}(L, x_1), \mathsf{suffix}(L, x_2), \ldots, \mathsf{suffix}(L, x_k)\}$. Note that for any $y \in \Sigma^*$, $\mathsf{suffix}(L, y) = \mathsf{suffix}(L, x_j)$, for some $j \in \{1, \ldots, k\}$.

Moreover, the following property holds:

$$(*) \text{ if } x, y \in \Sigma^* \text{ and } \mathsf{suffix}(L, x) = \mathsf{suffix}(L, y) \\ \implies \quad \mathsf{suffix}(L, x\mathsf{a}) = \mathsf{suffix}(L, y\mathsf{a}) \quad \text{for any } \mathsf{a} \in \Sigma.$$

We will construct a DFA whose states are the various suffix languages of L; hence we will have k states in the DFA. Moreover, the DFA will be designed such that after reading y, the DFA will end up in the state $\mathsf{suffix}(L, y)$.

The DFA is $M = (Q, \Sigma, q_0, \delta, F)$ where

- $Q = \{\mathsf{suffix}(L, x_1), \mathsf{suffix}(L, x_2), \ldots, \mathsf{suffix}(L, x_k)\}$
- $q_0 = \mathsf{suffix}(L, \epsilon)$
- $F = \{\mathsf{suffix}(L, x) \mid \epsilon \in \mathsf{suffix}(L, x)\}$
- $\delta(\mathsf{suffix}(L, x), \mathtt{a}) = \mathsf{suffix}(L, x\mathtt{a})$ for every $\mathtt{a} \in \Sigma$.

The transition function δ is well-defined because of property (*) above.

We can now prove, by induction on the length of x, that after reading x, the DFA reaches the state $\mathsf{suffix}(L, x)$. Since $x \in L$ iff $\epsilon \in \mathsf{suffix}(L, x)$, it follows by the definition of F that the DFA accepts precisely the language L. Hence, the language L is regular.

Let the number of suffix languages of L be k; notice that the DFA above has k states. Now, let us prove that *any* DFA for L must have at least k states.

For any two $x, y \in \Sigma^*$ such that $\mathsf{suffix}(L, x) \neq \mathsf{suffix}(L, y)$, since there is some w such that $xw \in L \iff yw \notin L$, it follows that any DFA accepting L must go to *different* states after reading x and after reading y (since from one of them it must accept w while from the other it must reject w). Hence, it is easy to see that any DFA must have at least k states. This finishes the proof.

We can further argue that any DFA for L that has k states must be *identical* to the DFA we created above. This is a bit more involved notationally, and is proved by showing a $1-1$ correspondence between the two DFAs and arguing they must be connected the same way. We omit this part of the theorem and proof. ■

2.1 Examples

Let us explain the theorem we just proved using examples.

2.1.1 The suffix languages of a non-regular language

Consider the language $L \subseteq \{\mathtt{a}, \mathtt{b}\}^*$:

$$L = \left\{ w \;\middle|\; w \text{ has an odd number of } \mathtt{a}\text{'s} \right\}.$$

The suffix language of $x \in \Sigma^*$, where x has an even number of $\mathtt{a}$'s is:

$$\mathsf{suffix}(L, x) = \left\{ w \;\middle|\; w \text{ has an odd number of } \mathtt{a}\text{'s} \right\} = L.$$

The suffix language of $x \in \Sigma^*$, where x has an odd number of a's is:

$$\mathsf{suffix}(L, x) = \left\{ w \;\middle|\; w \text{ has an even number of } \mathtt{a}\text{'s} \right\}.$$

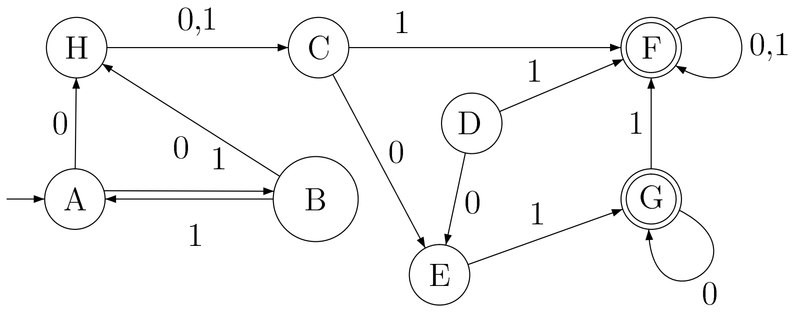


Figure 1: The automata to be minimized.

Hence there are only two distinct suffix languages for L. By the theorem, we know L must be regular and the minimal DFA for L has two states. Going with the construction of the DFA mentioned in the proof of the theorem, we see that we have two states, $q_0 = \mathsf{suffix}(L, \epsilon)$ and $q_1 = \mathsf{suffix}(L, a)$. The transitions are as follows:

- From $q_0 = \mathsf{suffix}(L, \epsilon)$, on a we go to $\mathsf{suffix}(L, a)$, which is the state q_1.
- From $q_0 = \mathsf{suffix}(L, \epsilon)$, on b we go to $\mathsf{suffix}(L, b)$, which is same as $\mathsf{suffix}(L, \epsilon)$, i.e. the state q_0.
- From $q_1 = \mathsf{suffix}(L, a)$, on a we go to $\mathsf{suffix}(L, aa)$, which is same as $\mathsf{suffix}(L, \epsilon)$, i.e. the state q_0.
- From $q_1 = \mathsf{suffix}(L, a)$, on b we go to $\mathsf{suffix}(L, ab)$, which is same as $\mathsf{suffix}(L, a)$, i.e. the state q_1.

The initial state is $\mathsf{suffix}(L, \epsilon)$ which is the state q_0, and the final states are those states $\mathsf{suffix}(L, x)$ that have ϵ in them, which is the set $\{q_1\}$.

We hence have a DFA for L, and in fact this is the minimal automaton accepting L.

3 Minimization algorithm

The above discussion leaves us with a way to decide what is the minimum number of states of a DFA that accepts a language, but it is not clear how to turn this into an algorithm (in particular, we do not have an efficient way to compute suffix languages of a language).

The idea is to work directly on a given DFA and compute a minimum DFA from it. So consider the DFA of Figure 1. It is more complex than it needs to be.

The DFA minimization algorithm first removes any states which are not reachable from the start state, because they obviously aren't contributing anything to what the DFA accepts. (D in this example.) It then marks which of the remaining states are distinct. States not marked as distinct can then be merged, to create a simpler DFA.

3.1 Idea of algorithm

Suppose the given DFA is $M = (Q, \Sigma, \delta, q_0, F)$.

We know by the above discussion that two states p and q are distinct if their two languages are different. Namely, there is some word w that belongs to L_p but w is not in L_q (or vice versa). It is not clear however how to detect when two states have different suffix languages. The idea is to start with the "easy" case, and then propagate the information.

As such, p and q are ***distinct*** if there exists w, such that $\delta(p, w)$ is an accept state and $\delta(q, w)$ is not an accept state. In particular, for $w = \epsilon$, we have that p and q are distinct if $p \in F$ and $q \notin F$, or vice versa.

for $w = c_1 c_2 \ldots c_m$, we have that p and q are ***distinct*** if

$$\delta\Big(\delta(p, c_1)\,, c_2 c_3 \ldots c_m\Big) \in F \text{ and } \delta\Big(\delta(q, c_1)\,, c_2 c_3 \ldots c_m\Big) \notin F,$$

or vice versa.

In particular, this implies that if p and q are distinct because of word w of length m, then $\delta(p, c_1)$ and $\delta(q, c_1)$ are distinct because of a word $w' = c_2 \ldots c_m$ of length $m - 1$.

Thus, its easy to compute the pairs of states distinct because of empty words, and if we computed all the states distinct because of words of length $m - 1$, we can "propagate" this information for pairs of states distinct by states of length m.

3.2 The algorithm

The algorithm for marking distinct states follows the above (recursive) definition. Create a table Distinct with an entry for each pair of states. Table cells are initially blank.

```
(1) For every pair of states $(p, q)$
        If p is final and q is not, or vice versa,
            Set Distinct$(p, q)$ to be $\epsilon$.
(2) Loop until there is no change in the table contents
        For each pair of states $(p, q)$ and each character a in the alphabet:
            if Distinct$(p, q)$ is empty and Distinct$(\delta(p, a),\ \delta(q, a))$ is not empty
                Set Distinct$(p, q)$ to be a.
(3) Two states $p$ and $q$ are distinct iff Distinct$(p, q)$ is not empty.
```

3.2.1 Example of how the algorithm works

The following is the execution of the algorithm on the DFA of Figure 1.

After step (1):

b							
c							
d							
e							
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h						ϵ	ϵ
	a	b	c	d	e	f	g

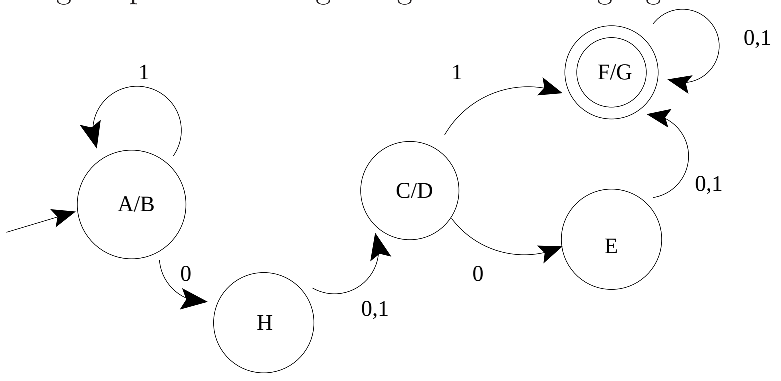
After one iteration of step (2):

b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h			1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

After the second iteration of step (2):

b							
c	1	1					
d	1	1					
e	0	0	0	0			
f	ϵ	ϵ	ϵ	ϵ	ϵ		
g	ϵ	ϵ	ϵ	ϵ	ϵ		
h	1	1	1	1	0	ϵ	ϵ
	a	b	c	d	e	f	g

Third iteration of step (2) makes no changes to the table, so we halt. The cells (a, b), (c, d) and (f, g) are still empty, so these pairs of states are not distinct. Merging them produces the following simpler DFA recognizing the same language.



1
A/B
0
H
0,1
C/D
1
0
E
F/G
0,1
0,1