CSC 320 - Lecture 07

#minimization #DFA #regular #non-regular #languages #pumping-lemma

---

Knowledge So Far

Remember. Class of Languages recognized by DFAs $=$ Class of Languages recognized by NFAs $=$ Regular Languages = Class of Language described by Regular Expressions.

Knowledge To Come

- DFA State minimization (Myhill-Nerode)
- Non-regular languages and the Pumping Lemme

If you are given a language that is finite - can you come up with a finite automaton for it? Yes or No? **YES**. (Always possible)!

Thus, given a finite language you can always create a finite automaton.

DFA State Minimization

Given a DFA.

Goal. Reduce number of states without changing language recognized.

1. Remove unreachable states (You can do this yourself with graph theory - Graph Traversal).
2. Identify/Collapse states that yield the same result (maintain determinism).

Note. Mark all the states that we do not want to collapse. And then determine the sates to join.

What Kind of States Can/Cannot Be Collapsed?

- Don't collapse accept and non-accept states.
  - If there exists a string w and states p, q such that...
    - When w is processed starting at state p M yields acceptance and
    - When w is processed starting at state q M yields non-acceptance.

- Then do not collapse p and q.

State Equivalence

Two states p, q of DFA M are **equivalent (p ~ q)** if and only if for all $w \in \Sigma^*$...

- computation of M for w starting at state p ends in accept state if and only if computation of M for w starting at state q ends in accept state.

Note. It might sound complicated, but it is fairly easy.

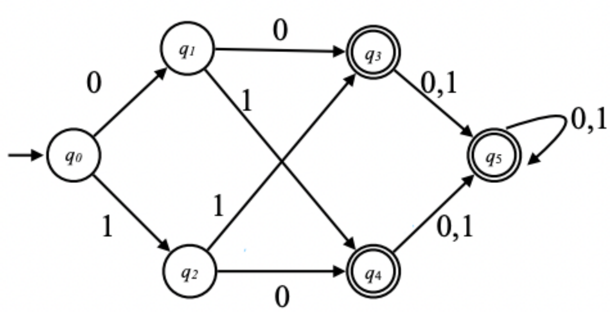
State Minimization Algorithm

Let M be a DFA with no inaccessible state.

- Write down all pairs $\{p, q\}$ of states in M (initially unmarked).
- For each unordered pair $\{p, q\}$: mark $\{p, q\}$ if $p \in F$ and $q \notin F$ (or vice versa).
- Repeat until no changes occur:
  - If there exists an unmarked pair $\{p, q\}$ such that $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$ then mark $\{p, q\}$.

Note. The algorithm produces state equivalence: $\{p, q\}$ unmarked if and only if p ~ q.

Example 01



First we write down all pairs $\{p, q\}$ of states in M (initially unmarked).

We will scan through all of them one by one.

$\{q_0, q_1\}$	$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_3, q_4\}$	$\{q_4, q_5\}$

$\{q_0, q_2\}$	$\{q_1, q_3\}$	$\{q_2, q_4\}$	$\{q_3, q_5\}$	
$\{q_0, q_3\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$		
$\{q_0, q_4\}$	$\{q_1, q_5\}$			
$\{q_0, q_5\}$				

For each pair $\{p, q\}$: mark $\{p, q\}$ if $p \in F$ and $q \notin F$

- Starting at $\{q_0, q_1\}$ we go down our table row by row and mark accordingly. We mark $\{q_0, q_3\}$ as q_3 is an accept state. The same for $\{q_0, q_4\}$ and $\{q_0, q_5\}$.
- Next we go to the following column. We mark $\{q_1, q_3\}$, $\{q_1, q_4\}$, and $\{q_1, q_5\}$.
- In column three we mark $\{q_2, q_3\}$, $\{q_2, q_4\}$, and $\{q_2, q_5\}$.
- In column four we mark none as they are both in accept states. And same with column five.

$\{q_0, q_1\}$	$\{q_1, q_2\}$	~$\{q_2, q_3\}$~	$\{q_3, q_4\}$	$\{q_4, q_5\}$
$\{q_0, q_2\}$	~$\{q_1, q_3\}$~	~$\{q_2, q_4\}$~	$\{q_3, q_5\}$	
~$\{q_0, q_3\}$~	~$\{q_1, q_4\}$~	~$\{q_2, q_5\}$~		
~$\{q_0, q_4\}$~	~$\{q_1, q_5\}$~			
~$\{q_0, q_5\}$~				

Repeat until no changes occur: If there exists an unmarked pair $\{p, q\}$ such that $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$ then mark $\{p, q\}$.

- We start with $\{q_0, q_1\}$. We have $\{\delta(q_0, 0), \delta(q_1, 0)\}$ become $\{q_1, q_3\}$ which is marked so we mark $\{q_0, q_1\}$. Next we check $\{q_0, q_2\}$. We note that $\{\delta(q_0, 0), \delta(q_2, 0)\}$ becomes $\{q_1, q_4\}$ thus we also mark.
- Next we check $\{q_1, q_2\}$ which is $\{\delta(q_1, 0), \delta(q_2, 0)\}$ and it becomes $\{q_3, q_4\}$ which is not marked, so we move on. We then check $\{\delta(q_1, 1), \delta(q_2, 1)\}$ which becomes $\{q_4, q_3\}$ which is also unmarked. Thus, we leave it unmarked.
- We then check $\{q_3, q_4\}$ which is $\{q_5\}$ in both situations and thus we leave unmarked. The same can be observed for $\{q_4, q_3\}$.
- Finally we check $\{q_4, q_5\}$ which also in both cases is $\{q_5\}$ and thus we also leave it unmarked.

We do a final pass and notice no change and thus we obtain the following.

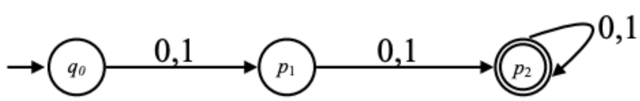
Note. We will want to program this, because it is very time consuming.

~$\{q_0, q_1\}$~	$\{q_1, q_2\}$	~$\{q_2, q_3\}$~	$\{q_3, q_4\}$	$\{q_4, q_5\}$
~$\{q_0, q_2\}$~	~$\{q_1, q_3\}$~	~$\{q_2, q_4\}$~	$\{q_3, q_5\}$	
~$\{q_0, q_3\}$~	~$\{q_1, q_4\}$~	~$\{q_2, q_5\}$~		
~$\{q_0, q_4\}$~	~$\{q_1, q_5\}$~			
~$\{q_0, q_5\}$~				

We note the following equivalences...

- $q_1 \sim q_2$
- $q_3 \sim q_4 \sim q_5$

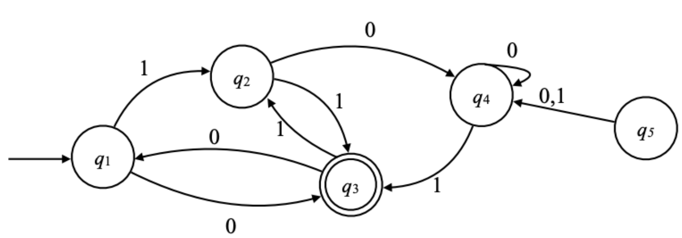
With this information we can minimize our DFA.



This looks a lot cooler than the initial graph we had, and also you may not have known that the initial graph could have became smaller at first glance.

Side Question. Could we come up with something else?

Example 02



Remove unreachable state q_5. And then write all the list of pairs.

$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_3, q_4\}$

$\{q_1, q_3\}$	$\{q_2, q_4\}$	
$\{q_1, q_4\}$		

Mark pairs of states where one state is an accept state and the other one is not.

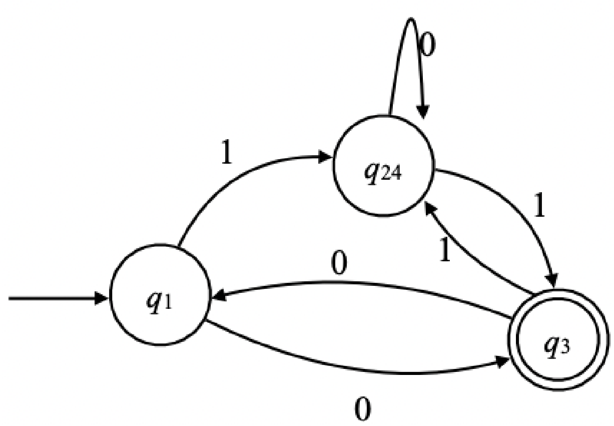
$\{q_1, q_2\}$	~$\{q_2, q_3\}$~	~$\{q_3, q_4\}$~
~$\{q_1, q_3\}$~	$\{q_2, q_4\}$	
$\{q_1, q_4\}$		

Mark pairs of states $\{q_i, q_j\}$ where for some a in $\{\delta(q_i, a), \delta(q_j, a)\}$ on state is an accept state and the other one is not.

So $\{\delta(q_1, 0), \delta(q_2, 0)\}$ is $\{q_3, q_4\}$, so we mark. Next, $\{\delta(q_1, 0), \delta(q_4, 0)\}$ is $\{q_3, q_4\}$ so we mark as well. And finally, $\{\delta(q_2, 0), \delta(q_4, 0)\}$ is $\{q_4, q_4\}$ and $\{\delta(q_2, 1), \delta(q_4, 1)\}$ is $\{q_3, q_3\}$ so we leave it unmarked.

~$\{q_1, q_2\}$~	~$\{q_2, q_3\}$~	~$\{q_3, q_4\}$~
~$\{q_1, q_3\}$~	$\{q_2, q_4\}$	
~$\{q_1, q_4\}$~		

Note that q_2 ~ q_4. And we can minimize our DFA as follows...



It is not always advantageous to make the automaton smaller. It is dependent on circumstances.

Note. You could test all of them at once, but it is better to go one by one to avoid unnecessary computation.

Regular and Non-Regular Languages

Regular Languages

Recall. The set of regular language is...

- The set of all languages recognized by a deterministic finite automata, and also it is the same as
- The set of all languages recognized by nondeterministic finite automate, as well as it is the same as
- The set of all languages described by regular expressions.

Consider the Following Language: Is L Regular?

$L = \{w \in \{0, 1\}^* | w \text{ has an equal number of occurences of 01 and 10 as substrings}\}$.

- $\varepsilon \in L$
- $0, 1 \in L$
- $w \in L$ must start and end with the same character

Regular Expression: $\varepsilon \cup 0 \cup 1 \cup 0(0 \cup 1)^*0 \cup 1(0 \cup 1)^*1$.

In the language: 010, 01100110.

Not in the language: 1001010.

Note. You have to have a finite number of states.

Consider the Following Language: Are L_1, L_2, or L_3 Regular?

- $L_1 = \{w \in \{0, 1\}^* | w \text{ has an equal number of } 0\text{s and } 1\text{s } \}$????
- $L_2 = \{0^n 1^n | n \geq 0\}$????
- $L_3 = \{0^3 1^n | n \geq 0\} \longleftarrow$ Regular: $L_3 = L(0001^*)$

Idea. L_1 would need to be able to keep track of how many 0s there are and how many 1s are allowed. It does not have unlimited memory. L_2 also has the same problem.

Non-Regular Languages

Recall. Given language L if there exists an FA M with $L(M) = L$ then L is regular.

Therefore. If a language is non-regular then no finite automaton exists that recognizes it.

A technique for proving that languages are non-regular: **Pumping Lemma**.

Pumping Lemma

If L is a regular language, then there is a natural number p (the pumping length) where:

- If s is any string in L of length at least p (i.e., $s \in L, |s| \geq p$) (at least as long), then s can be divided into $s = xyz$ (concatenation) satisfying the following...
    1. for each $i \geq 0 : xy^i z \in L$
    2. $|y| > 0$ (i.e., $y \neq \varepsilon$)
    3. $|xy| \leq p$

Note. y^i means the concatenation of i copies of string y
Note. Conditions 1, 2, 3 hold **for all** strings of length at least p in L

Up Next

- Investigate what the pumping lemma is and what it is good for.
- Prove why the pumping lemma is correct.
- Examples to show that certain languages are not regular, using the pumping lemma.
- Push down automaton. They have a little extra memory, and they have a stack.

Previous Lecture

[Lecture06]

Next Lecture

[Lecture08]