Lecture 3: More on DFAs

27 January 2009

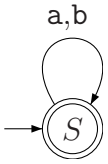
This lecture continues with material from section 1.1 of Sipser.

1 JFLAP demo

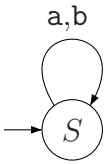
Go to `http://www.jflap.org`. Run the applet ("Try applet" near the bottom of the menu on the lefthand side). Construct some small DFA and run a few concrete examples through it.

2 Some special DFAs

For $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, consider the following DFA that accepts Σ^*:

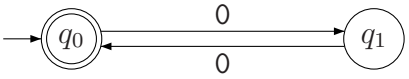


The DFA that accepts nothing, is just



3 Formal definition of a DFA

Consider the following automata, that we saw in the previous lecture:



We saw last class that the following components are needed to specify a DFA:

(i) a (finite) alphabet

(ii) a (finite) set of states

(iii) which state is the start state?

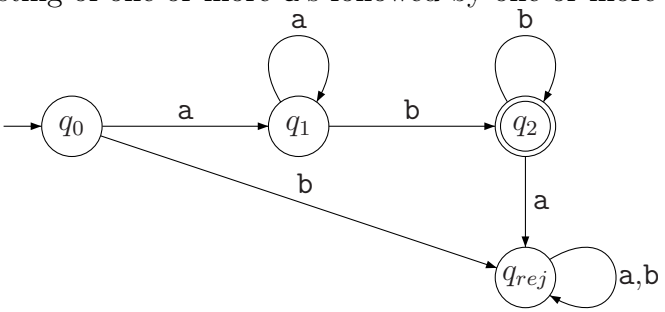
(iv) which states are the final states?

(v) what is the transition from each state, on each input character?

Formally, a ***deterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q: A finite set (the set of ***states***).
- Σ: A finite set (the ***alphabet***)
- $\delta : Q \times \Sigma \to Q$ is the ***transition function***.
- q_0: The ***start*** state (belongs to Q).
- F: The set of ***accepting*** (or ***final***) states, where $F \subseteq Q$.

For example, let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and consider the following DFA M, whose language $L(M)$ contains strings consisting of one or more a's followed by one or more b's.



Then $M = (Q, \Sigma, \delta, q_0, F)$, $Q = \{q_0, q_1, q_2, q_{rej}\}$, and $F = \{q_2\}$. The transition function δ is defined by

δ	a	b
q_0	q_1	q_{rej}
q_1	q_1	q_2
q_2	q_{rej}	q_2
q_{rej}	q_{rej}	q_{rej}

We can also define δ using a formula

$$\delta(q_0, a) = q_1$$

$$\delta(q_1, a) = q_1$$

$$\delta(q_1, b) = q_2$$

$$\delta(q_2, b) = q_2$$

$$\delta(q, t) = q_{rej} \text{ for all other values of } q \text{ and } t.$$

Tables and state diagrams are most useful for small automata. Formulas are helpful for summarizing a group of transitions that fit a common pattern. They are also helpful for describing algorithms that modify automatas.

4 Formal definition of acceptance

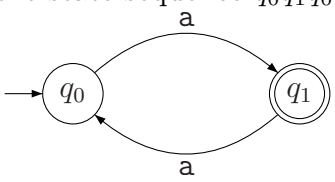
We've also seen informally how to run a DFA. Let us turn that into a formal definition. Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a given DFA and $w = w_1 w_2 \dots w_k \in \Sigma^*$ is the input string. Then M ***accepts*** w iff there exists a sequence of states $r_0, r_1, \dots r_k$ in Q, such that

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \dots, k-1$.
3. $r_k \in F$.

The ***language recognized*** by M, denoted by $L(M)$, is the set $\left\{ w \;\middle|\; M \text{ accepts } w \right\}$.

For example, when our automaton above accepts the string `aabb`, it uses the state sequence $q_0 q_1 q_1 q_2 q_2$. (Draw a picture of the transitions.) That is $r_0 = q_0$, $r_1 = q_1$, $r_2 = q_1$, $r_3 = q_2$, and $r_4 = q_2$.

Note that the states do not have to occur in numerical order in this sequence, e.g. the following DFA accepts `aaa` using the state sequence $q_0 q_1 q_0 q_1$.



A language (i.e. set of strings) is ***regular*** if it is recognized by some DFA.

5 Closure properties

Consider the set of odd integers. If we multiply two odd integers, the answer is always odd. So the set of odd integers is said to be ***closed*** under multiplication. But it is not closed under addition. For example, $3 + 5 = 8$ which is not odd.

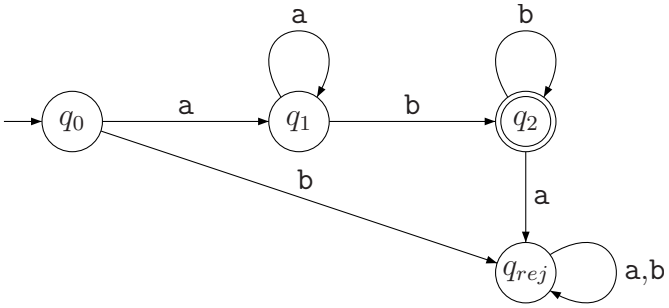
To talk about closure, you need two sets: a larger universe U and a smaller set $X \subseteq U$. The universe is often supposed to be understood from context. Suppose you have a function F that maps values in U to values in U. Then X is ***closed under*** f if F applied to values from X always produces an output value that is also in X.

For automata theory, U is usually the set of all languages and X contains languages recognized by some specific sort of machine, e.g. regular languages.

5.1 Closure under complement of regular languages

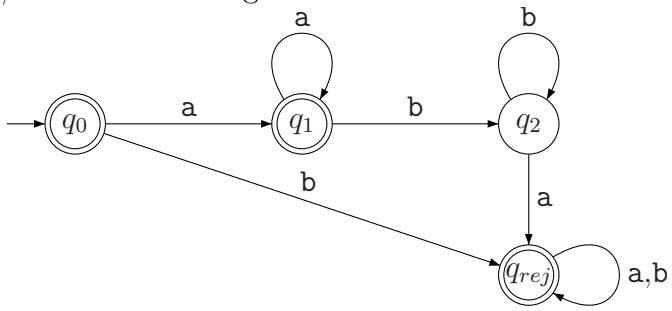
Here we are interested in the question of whether the regular languages are closed under set complement. (The complement language keeps the same alphabet.) That is, if we have a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting some language L, can we construct a new DFA M' accepting $\overline{L} = \Sigma^* \setminus L$?

Consider the automata M from above, where L is the set of all strings of at least one `a` followed by at least one `b`.



The complement language $\overline{L}$ contains the empty string, strings in which some b's precede some a's, and strings that contain only a's or only b's.

Our new DFA M' should accept exactly those strings that M rejects. So we can make M' by swapping final/non-final markings on the states:

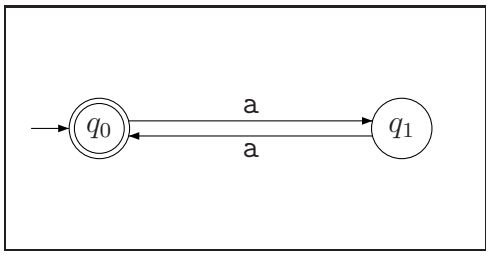


Formally, $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$.

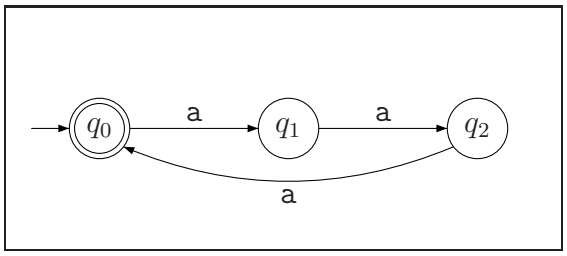
6 Closure under intersection

We saw in previous lecture an automatas that accepts strings of even length, or that their length is a product of 3. Here are their automatas:

M_1:

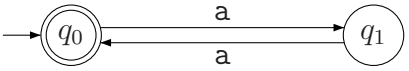


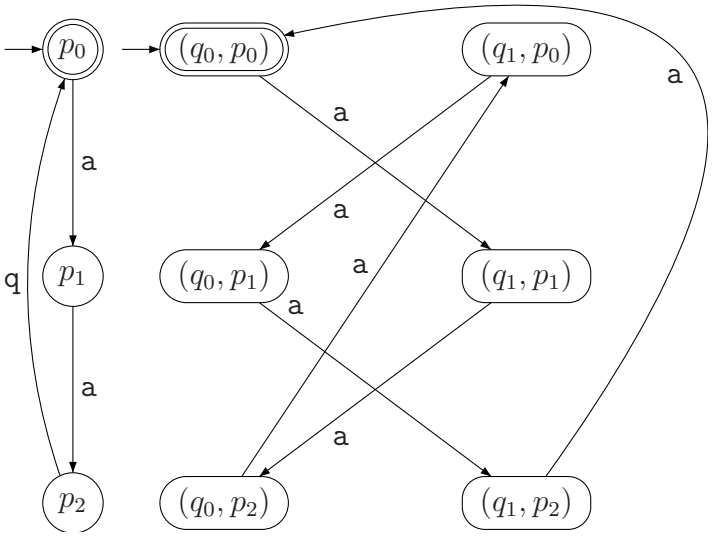
M_2:



Assume, that we would like to build an automata that accepts the language which is the intersection of the language of both automatas. That is, we would like to accept the language $L(M_1) \cap L(M_2)$. How do we build an automata for that?

The idea is to build a product automata of both automatas. See the following for an example.





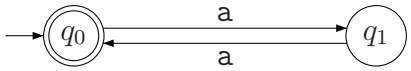
Given two automatas $M = (Q, \Sigma, \delta, q_0, F)$ and $M' = (Q', \Sigma', \delta', q_0', F')$, their ***product automata*** is the automata formed by the product of the states. Thus, a state in the resulting automata $N = M \times M'$ is a pair (q, q'), where $q \in Q$ and $q' \in Q'$.

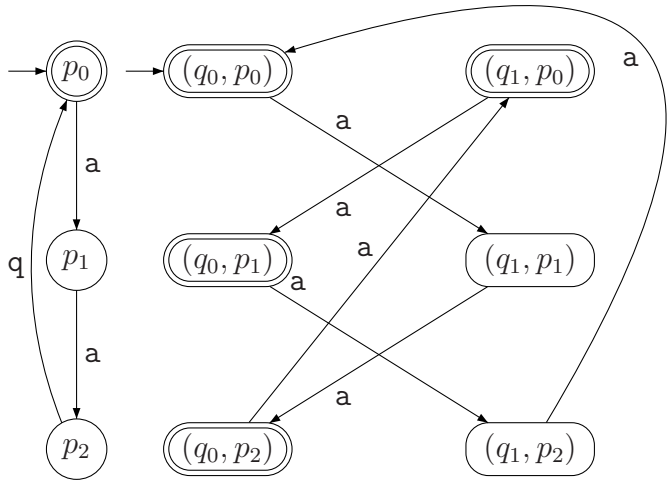
The key invariant of the product automata is that after reading a word w, its in the state (q, q'), where, q is that state that M is at after reading w, and q' is the state that M' is in after reading w.

As such, the intersection language $L(M) \cap L(M')$ is recognized by the product automata, where we set the pairs $(q, q') \in Q(N)$ to be an accepting state for N, if $q \in F$ and $q' \in F'$.

Similarly, the automata accepting the union $L(M) \cup L(M')$ is created from the product automata, by setting the accepting states to be all pairs (q, q'), such that either $q \in F$ or $q' \in F'$.

As such, the automata accepting the union language $L(M_1) \cup L(M_2)$ is the following.

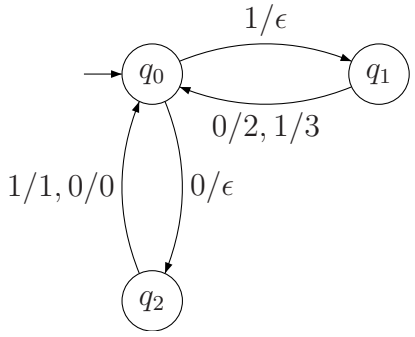




7 (Optional) Finite-state Transducers

In many applications, transitions also perform actions. E.g. a transition reading `WANNATALK` from the network might also call some `C` code that opens a new HTTP connection.

Finite-state transducers are a simple case of this. An FST is like a DFA but each transition optionally writes an output symbol. These can be used to translate strings from one alphabet to another. For example, the following FST translates binary numbers into base-4 numbers. E.g. 011110 becomes 132. We'll assume that FSTs don't accept or reject strings, just translate them.



So, formally, an FST is a 5-tuple $(Q, \Sigma, \Gamma, \delta, q_0)$, where

- Q is a finite set (the states).
- Σ and Γ are finite sets (the input and output alphabets).
- $\delta : Q \times \Sigma \to Q \times \Gamma_\epsilon$ is the transition function.
- q_0 is the start state

Notation: $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$.
The transition table for our example FST might look like the following.

δ	0	1
q_0	(q_1, ϵ)	(q_2, ϵ)
q_1	$(q_0, 0)$	$(q_0, 1)$
q_2	$(q_0, 2)$	$(q_0, 3)$