CS 373: Theory of Computation
Sariel Har-Peled and Madhusudan Parthasarathy

Lecture 6: Closure properties

February 5, 2009

This lecture covers the last part of section 1.2 of Sipser (pp. 58–63), part of 1.3 (pp. 66–69), and also closure under string reversal and homomorphism.

1 Overview

We defined a language to be ***regular*** if it is recognized by some DFA. The agenda for the new few lectures is to show that three different ways of defining languages, that is NFAs, DFAs, and regexes, and in fact all equivalent; that is, they all define regular languages. We will show this equivalence, as follows.

$$\begin{tikzcd}
\text{DFA} \arrow[rr, bend left, "\text{next next lecture}"] \arrow[dr, bend left, "\text{trivial}"] & & \text{Regex} \arrow[dl, bend left, "\text{today}"] \\
& \text{NFA} \arrow[ul, bend left, "\text{next lecture}"] &
\end{tikzcd}$$

One of the main properties of languages we are interested in are closure properties, and the fact that regular languages are closed under union, intersection, complement, concatenation, and star (and also under homomorphism).

However, closure operations are easier to show in one model than the other. For example, for DFAs showing that they are closed under union, intersection, complement are easy. But showing closure of DFA under concatenation and $*$ is hard.

Here is a table that lists the closure property and how hard it is to show it in the various models of regular languages.

Model					
Property	$\cap$ intersection	$\cup$ union	$\overline{L}$ complement	$\circ$ concatenation	$*$ star
DFA	Easy (done)	Easy (done)	Easy (done)	Hard	Hard
NFA	Doable (hw?)	Easy: Lemma **??**	Hard	Easy: Lemma **??**	Easy: Lemma **??**
regex	Hard	Easy	Hard	Easy	Easy

Recall what it means for regular languages to be closed under an operation op. If L_1 and L_2 are regular, then L_1 op L_2 is regular. That is, if we have an NFA recognizing L_1 and an NFA recognizing L_2, we can construct an NFA recognizing L_1 op L_2.

The extra power of NFAs makes it easy to prove closure properties for NFAs. When we know all DFAs, NFAs, and regexes are equivalent, these closure results then apply to all three representations. Namely, they would imply that regular languages have these closure properties.

2 Closure under string reversal for NFAs

Consider a word w, we denote by w^R the ***reversed*** word. It is just w in the characters in reverse order. For example, for $w =$ `barbados`, we have $w^R =$ `sodabrab`. For a language L, the ***reverse*** language is

$$L^R = \left\{ w^R \,\middle|\, w \in L \right\}.$$

We would like to claim that if L is regular, then so is L^R. Formally, we need to be a little bit more careful, since we still did not show that a language being regular implies that it is recognized by an NFA.

Claim 2.1 *If L is recognized by an NFA, then there is an NFA that recognizes L^R.*

Proof: Let M be an NFA recognizing L. We need to construct an NFA N recognizing L^R.

The idea is to reverse the arrows in the NFA $M = (Q, \Sigma, \delta, q_0, F)$, and swap final and initial states. There is a bug in applying this idea in a naive fashion. Indeed, there is only one initial state but multiple final states.

To overcome this, let us modify M to have a single final state q_S, connected to old ones with epsilon transitions. Thus, the modified NFA accepting L, is

$$M' = \left(Q \cup \{q_S\}, \Sigma, \delta', q_0, \{q_S\}\right),$$

where q_S is the only accepting state for M. Note, that δ' is identical to δ, except that

$$\forall q \in F \quad \delta'(q, \epsilon) = q_S. \tag{1}$$

Note, that $L(M) = L(M') = L$.

As such, q_S will become the start state of the "reversed" NFA.

Now, the new "reversed" NFA N, for the language L^R, is

$$N = \left(Q \cup \{q_S\}, \Sigma, \delta'', q_S, \{q_0\}\right).$$

Here, the transition function δ'' is defined as

(i) $\delta''(q_0, t) = \emptyset$ for every $t \in \Sigma$.

(ii) $\delta''(q, t) = \left\{ r \in Q \,\middle|\, q \in \delta'(r, t) \right\}$, for every $q \in Q \cup \{q_S\}$, $t \in \Sigma_\epsilon$.

(iii) $\delta''(q_S, \epsilon) = F$ (the reversal of Eq. (**??**)). [1]

Now, we need to prove formally that if $w \in L(M)$ then $w^R \in L(N)$, but this is easy induction, and we omit it. ■

Note, that this will not work for a DFA. First, we can not force a DFA to have a single final state. Second, a state may have two incoming transitions on the same character, resulting in non-determinism when reversed.

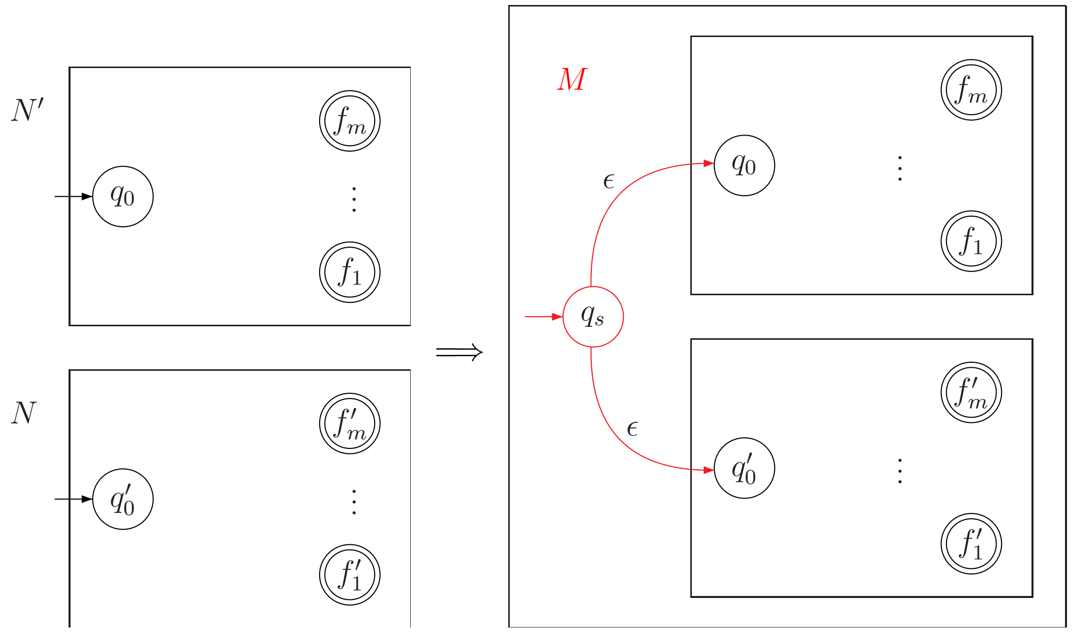
3 Closure of NFAs under regular operations

We consider the ***regular operations*** to be union, concatenation, and the star operator.

Advice to instructor: Do the following constructions via pictures, and give detailed tuple notation for only one of them. **!!!**

3.1 NFA closure under union

Given two NFAs, say N and N, we would like to build an NFA for the language $L(N) \cup L(N)$. The idea is to create a new initial state q_s and connect it with an ϵ-transition to the two initial states of N and N. Visually, the resulting NFA M looks as follows.



Formally, we are given two NFAs $N = (Q, \Sigma, \delta, q_0, F)$ and $N' = (Q', \Sigma, \delta', q'_0, F')$, where $Q \cap Q' = \emptyset$ and the new state q_s is not in Q or Q'. The new NFA M is

$$M = (Q \cup Q' \cup \{q_s\}, \Sigma, \delta_M, s_q, F \cup F'),$$

[1]This can be omitted, since it is implied by the (ii) rule.

where

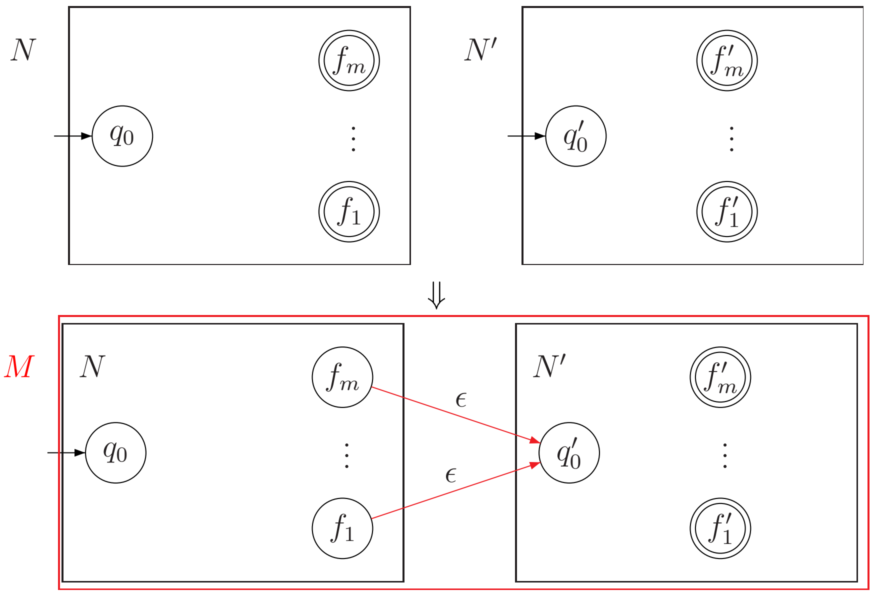
$$\delta_M(q,c) = \begin{cases} \delta(q,c) & q \in Q, c \in \Sigma_\epsilon \\ \delta'(q,c) & q \in Q', c \in \Sigma_\epsilon \\ \{q_0, q_0'\} & q = q_s, c = \epsilon \\ \emptyset & q = q_s, c \neq \epsilon. \end{cases}$$

We thus showed the following.

Lemma 3.1 *Given two NFAs N and N' one can construct an NFA M, such that $L(M) = L(N) \cup L(N')$.*

3.2 NFA closure under concatenation

Given two NFAs N and N', we would like to construct an NFA for the concatenated language $L(N) \circ L(N') = \left\{ xy \,\middle|\, x \in L(N) \text{ and } y \in L(N') \right\}$. The idea is to concatenate the two automatas, by connecting the final states of the first automata, by ϵ-transitions, into the start state of the second NFA. We also make the accepting states of N not-accepting. The idea is that in the resulting NFA M, given input w, it "guesses" how to break it into two strings $x \in L(N)$ and $y \in L(N')$, so that $w = xy$. Now, there exists an execution trace for N accepting x, then we can jump into the starting state of N' and then use the execution trace accepting y, to reach an accepting state of the new NFA M. Here is how visually the resulting automata looks like.



Formally, we are given two NFAs $N = (Q, \Sigma, \delta, q_0, F)$ and $N' = (Q', \Sigma, \delta', q_0', F')$, where $Q \cap Q' = \emptyset$. The new automata is

$$M = (Q \cup Q', \Sigma, \delta_M, q_0, F'),$$

where

$$\delta_M(q,c) = \begin{cases} \delta(q,\epsilon) \cup \{q'_0\} & q \in F, c = \epsilon \\ \delta(q,c) & q \in F, c \neq \epsilon \\ \delta(q,c) & q \in Q \setminus F, c \in \Sigma_\epsilon \\ \delta'(q,c) & q \in Q', c \in \Sigma_\epsilon. \end{cases}$$

Lemma 3.2 *Given two NFAs N and N' one can construct an NFA M, such that $L(M) = L(N) \circ L(N') = L(N)L(N')$.*

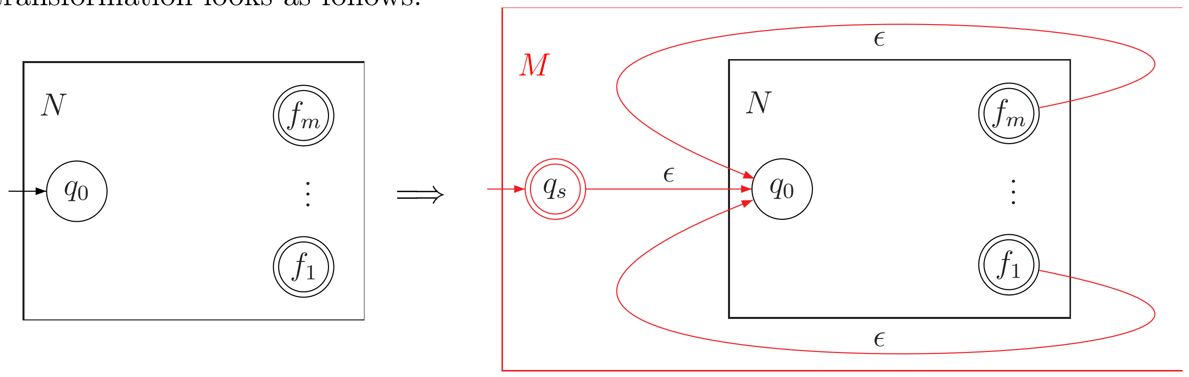
Proof: The construction is described above, and the proof of the correctness (of the construction) is easy and sketched above, so we skip it. You might want to verify that you know how to fill in the details for this proof (wink, wink). ■

3.3 NFA closure under the (Kleene) star

We are given a NFA N, and we would like to build an NFA for the Kleene star language

$$(L(N))^* = \left\{ w_1 w_2 \ldots w_k \;\middle|\; w_1, \ldots, w_k \in L(N), k \geq 0 \right\}.$$

The idea is to connect the final states of N back to the initial state using ϵ-transitions, so that it can loop back after recognizing a word of $L(N)$. As such, in the ith loop, during the execution, the new NFA M recognized the word w_i. Naturally, the NFA needs to guess when to jump back to the start state of N. One minor technicality, is that $\epsilon \in (L(N))^*$, but it might not be in $L(N)$. To overcome this, we introduce a new start state q_s (which is accepting), and its connected by (you guessed it) an ϵ-transition to the initial state of N. This way, $\epsilon \in L(M)$, and as such it recognized the required language. Visually, the transformation looks as follows.



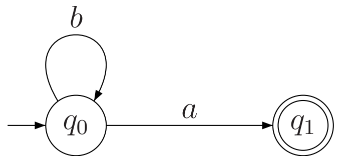
Formally, we are given the NFA $N = (Q, \Sigma, \delta, q_0, F)$, where $q_s \notin Q$. The new NFA is

$$M = \Big(Q \cup \{q_s\},\ \Sigma,\ \delta_M,\ q_s,\ F \cup \{q_s\} \Big),$$

where

$$\delta_M(q,c) = \begin{cases} \delta(q,\epsilon) \cup \{q_0\} & q \in F, c = \epsilon \\ \delta(q,\epsilon) & q \in F, c \neq \epsilon \\ \delta(q,c) & q \in Q \setminus F \\ \{q_0\} & q = q_0, c = \epsilon \\ \emptyset & q = q_0, c \neq \epsilon. \end{cases}$$

Why the extra state? The construction for star needs some explanation. We add arcs from final states back to initial state to do the loop. But then we need to ensure that ϵ is accepted. It's tempting to just make the initial state final, but this doesn't work for examples like the following. So we need to add a new initial state to handle ϵ.



Notice that it also works to send the loopback arcs to the new initial state rather than to the old initial state.

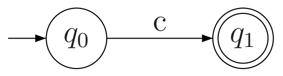
Lemma 3.3 *Given an* NFA N*, one can construct an* NFA M *that accepts the language* $(L(N))^*$.

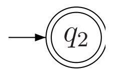
3.4 Translating regular expressions into NFAs

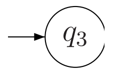
Lemma 3.4 *For every regular expression* R *over alphabet* Σ*, there is a* NFA N_R *such that* $L(R) = L(N_R)$.

Proof: The proof is by induction on the structure of R (can be interpreted as induction over the number of operators in R)

The base of the induction is when R contains no operator (i.e., the number operators in R is zero), then R must be one of the following:

(i) If $R = c$, where $c \in \Sigma$, then the corresponding NFA is  .

(ii) If $R = \epsilon$ then the corresponding NFA is  .

(iii) If $R = \emptyset$ then the corresponding NFA is  .

As for induction step, assume that we proved the claim for all expressions having at most $k - 1$ operators, and R has k operators in it. We consider if R can be written in any of the following forms:

(i) $R = R_1 + R_2$. By the induction hypothesis, there exists two NFAs N_1 and N_2 such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$. By Lemma **??**, there exists an NFA M that recognizes the union; that is $L(M) = L(N_1) \cup L(N_2) = L(R_1) \cup L(R_2) = L(R)$.

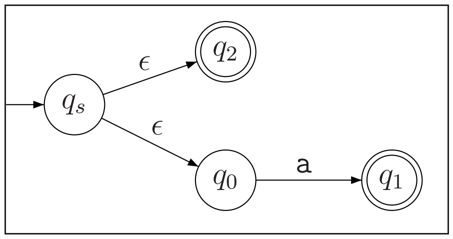
(ii) $R = R_1 \circ R_2 \equiv R_1 R_2$. By the induction hypothesis, there exists two NFAs N_1 and N_2 such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$. By Lemma **??**, there exists an NFA M that recognizes the concatenated language; that is, $L(M) = L(N_1) \circ L(N_2) = L(R_1) \circ L(R_2) = L(R)$.

(iii) $R = (R_1)^*$. By the induction hypothesis, there exists a NFA N_1, such that $L(N_1) = L(R_1)$. By Lemma **??**, there exists an NFA M that recognizes the star language; that is, $L(M) = (L(N_1))^* = (L(R_1))^* = L(R)$.

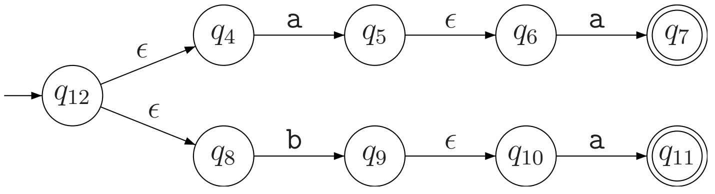
This completes the proof of the lemma, since we showed for all possible regular expressions with k operators how to build a NFA for them. ■

3.4.1 Example: From regular expression into NFA

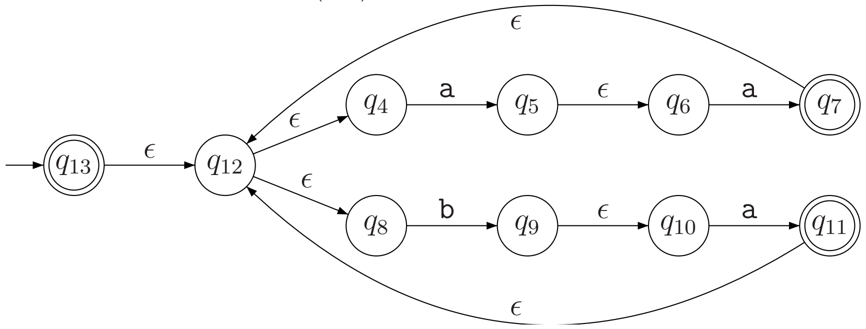
Consider the regular expression $R = (\mathtt{a} + \epsilon)(\mathtt{aa} + \mathtt{ba})^*$. We have that $R = R_1 \circ R_2$, where $R_1 = \mathtt{a} + \epsilon$ and $R_2 = (\mathtt{aa} + \mathtt{ba})^*$. Let use first build an NFA for $R_1 = \mathtt{a} + \epsilon$. The NFA for ϵ is $\begin{tikzcd} {} \arrow[r] & q_2 \end{tikzcd}$. and for $\mathtt{a}$ is $\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, "\mathtt{a}"] & q_1 \end{tikzcd}$. By Lemma ??, the NFA for their union, and thus of R_1, is



Now, $R_2 = (R_3)^*$, where $R_3 = \mathtt{aa} + \mathtt{ba}$. The NFA for $\mathtt{a}$ is $\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, "\mathtt{a}"] & q_1 \end{tikzcd}$, and as such the NFA for $\mathtt{aa}$ is $\begin{tikzcd} {} \arrow[r] & q_4 \arrow[r, "\mathtt{a}"] & q_5 \arrow[r, "\epsilon"] & q_6 \arrow[r, "\mathtt{a}"] & q_7 \end{tikzcd}$, by Lemma ??. Similarly, the NFA for $\mathtt{ba}$ is $\begin{tikzcd} {} \arrow[r] & q_8 \arrow[r, "\mathtt{b}"] & q_9 \arrow[r, "\epsilon"] & q_{10} \arrow[r, "\mathtt{a}"] & q_{11} \end{tikzcd}$. As such, by Lemma ??, the NFA for $R_3 = \mathtt{aa} + \mathtt{ba}$ is



By Lemma ??, the NFA for $R_2 = (R_3)^*$ is



Now, $R = R_1 R_2 = R_1 \circ R_2$, and by Lemma ??, the NFA for R is depicted in Figure ??.

Note, that the resulting NFA is by no way the simplest and more elegant NFA for this language (far from it), but rather the NFA we get by following our construction carefully.

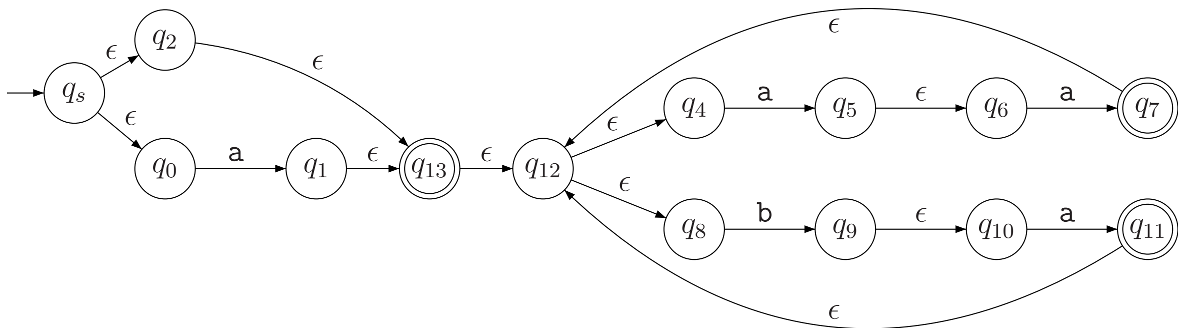


Figure 1: The NFA constructed for the regular expression $R = (\mathtt{a} + \epsilon)(\mathtt{aa} + \mathtt{ba})^*$.