CS 373: Theory of Computation (Summer 2011)
Lucas Cook

Lecture 10: Regex are regular!

2011 June 27 at 2:00 PM

---

1 The theorem

Today we will argue that regular expressions describe exactly the regular languages. Stated formally,

Theorem 1.1 *A language L is regular if and only if there is a regular expression R such that $\mathcal{L}(R) = L$.*

Our argument will occur in two pieces. First (the $\Leftarrow$ direction) , we will prove that regular expressions have 'equivalent' NFAs using the closure properties that we already know. Second (the $\rightarrow$ direction.) , we will show a technique for translating any DFA or NFA into a regular expression and briefly argue its correctness.

1.1 From regex to NFA

Given the inductive definition for regular expressions, our proof will proceed inductively. It turns out to be quite simple, given all the knowledge we have about regular languages already.

Proof:

By induction on the structure of a regular expression R over Σ. At base, $R = \varnothing$, ε, or a for some $a \in \Sigma$. The language $\mathcal{L}(R)$ is $\varnothing$, $\{\varepsilon\}$, and $\{a\}$ in each of the respective cases. Since these languages are finite, they are all regular.

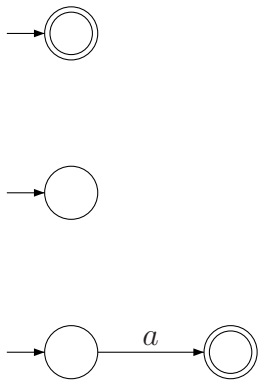
Now consider regular expressions R_1 and R_2 such that $\mathcal{L}(R_1)$ and $\mathcal{L}(R_2)$ are regular. Induction proceeds in three cases:

- For $R' = R_1 \cup R_2$, note that $\mathcal{L}(R') = \mathcal{L}(R_1 \cup R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$, which is regular since the regular languages are closed under union.
- For $R' = R_1 \circ R_2$, note that $\mathcal{L}(R') = \mathcal{L}(R_1 \circ R_2) = \mathcal{L}(R_1) \circ \mathcal{L}(R_2)$, which is regular since the regular languages are closed under concatenation.
- For $R' = R_1^*$, note that $\mathcal{L}(R') = \mathcal{L}(R_1^*) = (\mathcal{L}(R_1))^*$, which is regular since the regular languages are closed under star.

Thus all regular expressions describe/generate regular languages.

■

Another way to prove this is by drawing NFAs for each piece in the structural induction. $\mathcal{L}(\varepsilon)$, $\mathcal{L}(\varnothing)$, and $\mathcal{L}(a)$ are accepted by the following NFAs:



Then we can construct the union, concatenation, and star of any NFA by the closure property constructions that we saw in Lecture 8.

1.2 From NFA to regex

This part of the proof is not as simple. At first glance, it's not clear how regular expressions relate to automata at all! How can I extract a regular expression from the accepted languages of a complicated NFA? We need to understand what regular expressions mean in terms of a DFA/NFA.

Consider the regular expression $R = 0^+1^+ \cup (01)^*$ and a DFA M that accepts $\mathcal{L}(R)$. R describes what a certain set of strings looks like:

- either they are of the form 0^+1^+ or $(01)^*$
- in the first case, it's one or more 0s followed by one or more 1s
- in the second case, it's 01 repeating zero or more times

These strings are also the accepted strings in M, though. Each $w \in \mathcal{L}(R)$ corresponds to some sequence of states $r_0, r_1, r_2, ...r_n$ from M that ends in an accept state, or equivalently some 'accepting path' from the initial to a final state. Since $\mathcal{L}(R)$ has every accepted string, we can think of R as a compact way of representing *all* of the accepting paths in M.

1.2.1 The GNFA conversion method

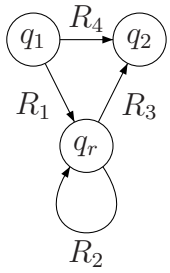
There is a technique to 'compress' every NFA into a single regular expression that represents every accepting path. The algorithm uses a new automaton type called the **Generalized Nondeterministic Finite Automaton (GNFA)**, which has the following properties:

- The transitions have regular expressions, whereas the NFA transitions have only symbols. A string w is accepted in a GNFA if there is a path of transitions with labels $R_1, R_2, R_3, ...R_m$ such that $w \in \mathcal{L}(R_1 \circ R_2 \circ R_3 \circ ... \circ R_m)$.

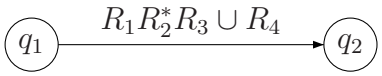
- There is a single start state with outgoing edges to all states and no incoming edges.
- There is a single final state with incoming edges from all states and no outgoing edges.
- All states besides the start and final have edges between them.
- Any unspecified edge (i.e. not drawn) has the regular expression $\varnothing$.

The compression process works as follows:

1. Given NFA N, turn N into a GNFA by:
   (a) Adding the new start state, connecting a ε transition to the start state of N.
   (b) Adding the new final state, connection a ε transition from *each* final state of N.
   (c) Adding $\varnothing$ transitions between any two states of N that have no edge.
   (d) Make each of N's final states nonfinal.
2. While: there is a state besides the new start and new final, remove one such state and update nearby transitions. Given a situation like the following, where q_r is the removed state and R_1,R_2,R_3, and R_4 are regular expressions:

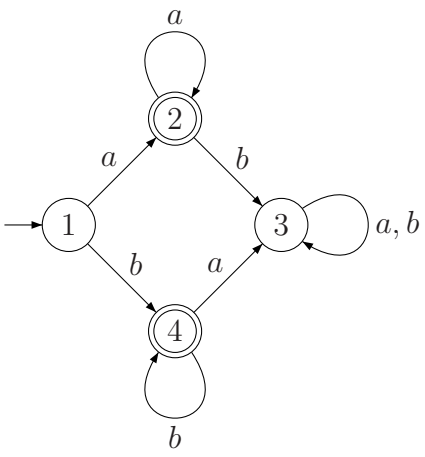


   q_r disappears and we update the edge between q_1 and q_2:

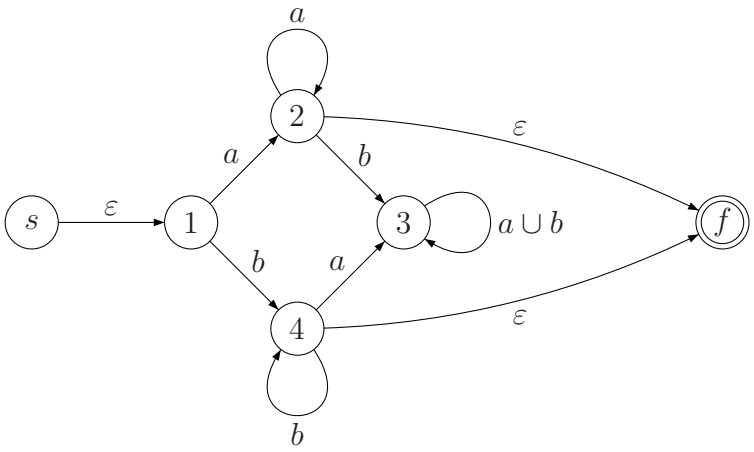


   This process needs to be done for *all* q_1 and q_2 that are directly adjacent to q_r, including the start and final. q_1 and q_2 might even be the same state, so we are updating a self loop!
3. Finally, when there is only the start and final state with one edge between them, the regular expression for $\mathcal{L}(N)$ is on the edge.

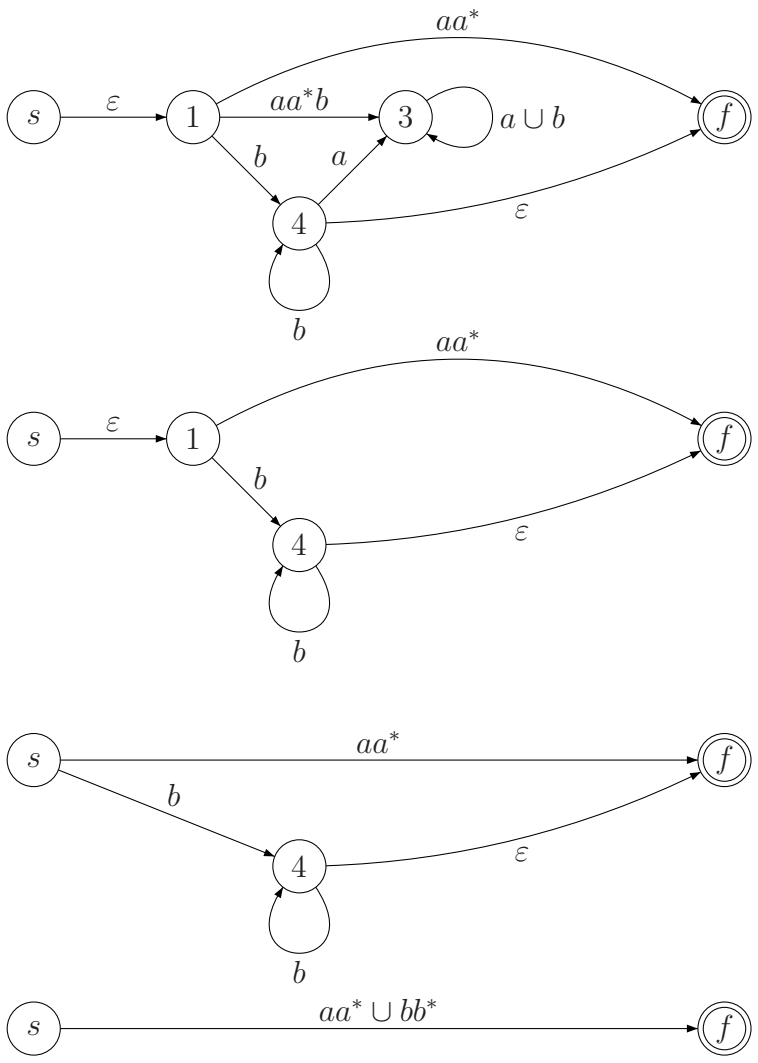
This is easier to understand by example. Starting with a DFA...



create the equivalent GNFA...



And remove states in any order until only the new start and final remain. Notice that removing state 3 does not require modifying any transitions. There are no paths through 3 between two adjacent nodes!



Now we have the regular expression $aa^* \cup bb^* = a^+ \cup b^+$ for the DFA.

1.2.2 Proof of correctness (sketch)

Without going through all the details of formalizing GNFAs, here is a rough outline of the proof that would be necessary: [1]

1. Argue every DFA/NFA M can be modified into a GNFA with the same language. (easy, just juggling notation)

2. Argue the language of any GNFA remains unchanged after removing one node. (harder)

[1] See other resources like Sipser or `http://www.cs.uiuc.edu/class/fa10/cs373/Lectures` for more details.

3. Conclude that there is one edge remaining at the end of the repeated node removal process, and it contains a regular expression R with $\mathcal{L}(R) = \mathcal{L}(M)$. (easy)

It seems clear that the first and last steps are true. You should convince yourself that the middle step is as well, since we update all the necessary paths on the appropriate transitions.

2 Closure property: Homomorphism

Using our newfound regular expression equivalence, let's do a quick closure property proof.

A *homomorphism* is a function $f : \Sigma \to \Delta^*$ from symbols in one alphabet Σ to strings over another alphabet Δ (possibly the same one). Given a string as input, it applies symbol by symbol: $f(w) = f(w_1) \circ f(w_2) \circ ... \circ f(w_n)$. For example, define:

$$f : \{a, b, c, d\} \to \{0, 1\}^*$$

$$f(a) = 0, f(b) = 1, f(c) = 00, f(d) = \varepsilon$$

Thus $f(aba) = 010$, $f(ca) = 000 = f(aaa) = f(dddaddddcdddddd)$. Homomorphisms are useful for changing between alphabets, removing characters, and doing all sorts of substitutions in strings.[2] For a language L, define $f(L) = \{f(w) \mid w \in L\}$ by applying f to each string in L.

Claim 2.1 *Given homomorphism $f : \Sigma \to \Delta^*$ and regular language L over Σ, $f(L)$ is regular.*

Proof: There is a regular expression R such that $\mathcal{L}(R) = L$. Create a new regular expression R' by replacing each occurrence of every $a \in \Sigma$ with the corresponding $f(a)$. For all strings $w_1 w_2 ... w_n \in L$ with $w_i \in \Sigma$, the modified regular expression has $f(w_i)$ in place of each w_i, making $f(w_1)f(w_2)...f(w_n) = f(w) \in \mathcal{L}(R')$. Also, $\mathcal{L}(R')$ can only contain strings that are in $f(\mathcal{L}(R))$. Thus $\mathcal{L}(R') = f(L)$ and $f(L)$ is regular. ■

[2] And for solving homework problems. There are lots of quick proofs using homomorphisms and their (more complicated) inverses.