CMPSCI 250: Introduction to Computation

Lecture #35: Constructing λ-NFA's From Regular Expressions
David Mix Barrington
20 April 2012

Constructing λ-NFA's From Regular Expressions

- Review: Parts of Kleene's Theorem
- Review: Induction on Regular Expressions
- A Normal Form for λ-NFA's
- The Construction
- The Star Case and the Proof
- An Example: (ab + ba)* + bb
- Taking This Example to a Minimal DFA

Review: Parts of Kleene's Theorem

- Our goal in Kleene's Theorem is to be able to convert regular expressions to DFA's and vice versa.

- In the last two lectures we've provided two of the three pieces of the transformation from regular expressions to DFA's. We defined ordinary NFA's and λ-NFA's, then presented the Subset Construction to turn ordinary NFA's to DFA's, then presented the Killing λ-Moves Construction to turn λ-NFA's to ordinary NFA's.

- Today we will see how to convert regular expressions to equivalent λ-NFA's. This will complete the steps needed to go from regular expressions to DFA's.

- In our next lecture we will finish Kleene's Theorem by presenting the State Elimination Construction to convert DFA's (or NFA's, or λ-NFA's) to regular expressions.

Review: Induction on Regular Expressions

- We want to prove that for every regular expression R, we can construct a λ-NFA N such that L(N) = L(R).

- The way to prove a proposition P(R) for all regular expressions R is to use induction on the definition of regular expressions. We must prove the two base cases, (1) P(∅) and (2) P(a) for every letter a ∈ Σ. Then we must prove the three inductive cases. If P(R) and P(S) are true, we must (3) prove P(R + S), (4) prove P(RS), and (5) prove P(R*).

- Here P(R) is "there exists N with L(N) = L(R)". As with our other inductive proofs on regular expressions, we will actually define a recursive algorithm that will take R as input and return N as output. We could code this algorithm in pseudo-Java using the class definition for `RegExp` and a new class definition for `LambdaNFA`, but we will stick with an informal description here.

A Normal Form for λ-NFA's

- Since we want to actually carry out this construction by hand on examples, we're going to make it a little more complicated than it would need to be just to prove that a valid construction exists. We'll produce λ-NFA's in a particular **normal form** -- they will satisfy three rules that will allow us to make simpler λ-NFA's in most cases.

- Rule (1) says that the λ-NFA has exactly one final state, which isn't the start state.

- Rule (2) says that no transitions go into the start state.

- Rule (3) says that no transitions go out of the start state.

- Similar rules will also show up later in the State Elimination Construction.

The Construction

- (1) For ∅, we need a λ-NFA with a start state and a final state. That's all we need -- if it has no transitions, it accepts no strings and its language is ∅.
- (2) For a, we can again have a start state i and a final state f, with a single transition (i, a, f). The rules are satisfied, and the language is {a} as it should be.
- (3) Now assume, as our IH, that we have constructed λ-NFA's M and M′ for our two regular expressions R and R′, and that M and N follow the three rules. We need to build a new λ-NFA M′′ such that L(M′′) = L(M) ∪ L(M′) = L(R + R′). M′′ will have copies of all the states of M and M′, but we will *merge* the two initial states, and *merge* the two final states.
- (4) To make M′′′ with L(M′′′) = L(M)L(M′) = L(RR′), we instead merge the final state of M with the initial state of M′, making the new state non-final.

The Star Case and the Proof

- (5) Finally we want to build a λ-NFA N such that L(N) = L(M)* = L(R*). Assume that M has start state i and final state f. N's states will be M's states plus two more, a new start state s and a new final state t. We then add four new λ-moves: (s, λ, i), (i, λ, f), (f, λ, i), and (f, λ, t). We make f now a non-final state.

- Now we want to prove by induction on all regular expressions that this construction is correct -- if N is the λ-NFA made from R, then L(N) = L(R). This is pretty obvious for the two base cases as we can check the languages of the λ-NFA's directly. So we must check the three inductive cases.

- With the two λ-NFA's connected in **parallel** in step (3), a path from the start to final state of M′′ must either pass through only states of M or only states of M′. This is because the first move must be a move of either one machine or the other, and from that point we must stay in that machine until we finish, since we can't return to the start or continue past the finish, due to the rules. The path has *either* read a string in L(M) *or* read a string in L(M′).

Finishing the Correctness Proof

- In step (4) we created M‴ by connecting M and M′ in **series**, and we must show that L(M‴) = L(M)L(M′). How could a path get from the start state of M‴ (which is the start state of M) to the final state of M‴ (which is the start state of M′)? The first transition has to be in M, then the path must stay in M until it reaches the final state of M′. The only way out of that state is into M′, where it must stay until it reaches the final state and then stops. So the path reads a string in L(M) followed by a string in L(M′), as it should.

- In step (5) we created N by adding two new states and four new λ-moves to N. First note that we can read any sequence of zero or more strings in L(M) by going to i, reading each string going from i to f, returning to i each time, then winding up in t. Furthermore, any path from s to t must consist of some combination of trips from i through M to f, and uses of the new λ-moves. So the string we read is the concatenation of zero or more strings in L(M), and thus is in L(M)*.

Some Notes on the Construction

- The construction makes use of the normal form constantly -- if we could not assume that the input λ-NFA's followed the rules, we would need to introduce new states and new λ-moves in steps (3) and (4) as well as in (5). We pay for the normal form in step (5). We need to connect the start and final states, but then to obey the rules we need to put in new start and final states.

- We only create λ-moves when we do step (5). Thus if R has few or no stars, we will get a λ-NFA with few or no λ-moves, which can be good because making an ordinary NFA is more complicated the more λ-moves there are.

- We can sometimes see ways to simplify the λ-NFA without changing the language. But we need to be careful that our simplification is correct.

- It can be shown that the number of states in the λ-NFA is about the same as the length of the regular expression. So the only big blowup is NFA's to DFA's.

An Example: (ab + ba)* + bb

- Let's see how the construction works on a fairly complicated regular expression. (There are diagrams of this example in the text.) We can think of the construction either top-down or bottom-up -- let's try bottom-up.

- The three regular expressions "ab", "ba", and "bb" each get three-state λ-NFA's, with letter moves from the start state to a middle state and from that middle state to a final state.

- The λ-NFA for "ab + ba" has four states, three each for "ab" and "ba" minus two when we merge the two start states and two final states. To get a λ-NFA for (ab + ba)* we add a new start and final state, plus four new λ-moves, to get a six-state λ-NFA with four letter moves and four λ-moves. Finally, we place this six-state machine in parallel with the three-state machine for "bb", getting a seven-state machine with six letter moves and four λ-moves.

Taking This Example to a Minimal DFA

- Killing the λ-moves in this seven-state λ-NFA gives us a seven-state ordinary NFA with state set {i, p, q, r, s, t, f}, start state i, final state set {i, f}, and fourteen transitions: (i, a, q), (i, b, r), (i, b, t), (p, a, q), (p, b, r), (q, b, p), (q, b, f), (q, b, s), (r, a, p), (r, a, s), (r, a, f), (s, a, q), (s, b, r), and (t, b, f).

- We could potentially get 128 states in our DFA, but fortunately the process stops with only seven. State {i} goes to {q} on a and {r,t} on b, state {q} goes to ∅ on a and {p,s,f} on b, state {r,t} goes to {p,s,f} on a and {f} on b, state ∅ stays at ∅ on both, state {p,s,f} goes to {q} on a and to {r} on b, state {f} goes to ∅ on both, and state {r} goes to {p,s,f} on a and to ∅ on b.

- This DFA is minimal. The three final states are {i}, {p,s,f}, and {f}. String bb separates {i} from {p,s,f}, and ab separates these two from {f}. The four non-final states are {q}, {r,t}, {r}, and ∅, and we can separate these pairwise as well.