CMPSCI 250: Introduction to Computation

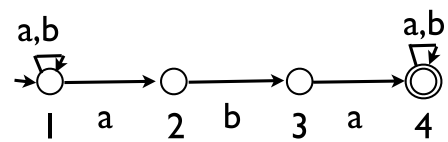
Lecture #34: Killing λ-Moves: λ-NFA's to NFA's
David Mix Barrington
16 April 2014

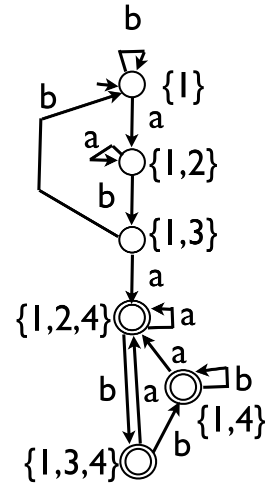
Killing λ-Moves: λ-NFA's to NFA's

- (last five slides of Lecture #33)
- Review: Kleene's Theorem Overview
- The Construction
- A Three-State Example
- Finishing the Example
- Validity of the Construction
- The Main Lemma
- The Case of Empty Strings

Applying This to No-aba

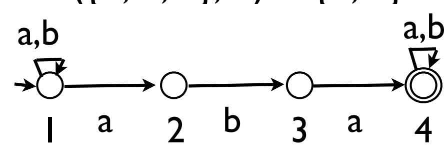
- The best way to get a DFA for No-aba is to first get one for Yes-aba.
- We begin with the start state {1} and compute δ({1}, a) = {1, 2} and δ({1}, b) = {1}. Then we compute δ({1, 2}, a) = {1, 2} and δ({1, 2}, b) = {1, 3}.

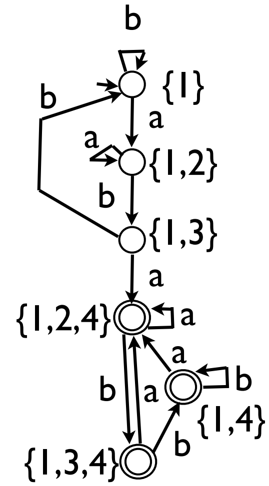




Applying This to No-aba

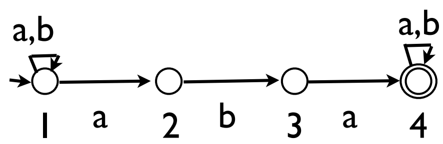
- Since {1, 3} is new, we must compute δ({1, 3}, a) = {1, 2, 4} and δ({1, 3}, b) = {1}.
- Then we get δ({1, 2, 4}, a) = {1, 2, 4} and δ({1, 2, 4}, b) = {1, 3, 4}. Not done yet!
- We have δ({1, 3, 4}, a) = {1, 2, 4} and δ({1, 3, 4}, b) = {1, 4}.

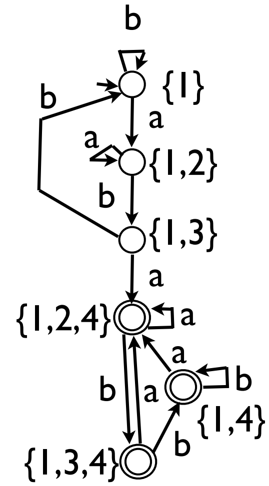




Applying This to No-aba

- Finally, with $\delta(\{1, 4\}, a) = \{1, 2, 4\}$ and $\delta(\{1, 4\}, b) = \{1, 4\}$, we're done -- we have all reachable states.
- If we minimized this DFA, the three final states would merge into one. This gives us our four-state DFA for Yes-aba, from which we can get one for No-aba.





Validity of the Construction

- How can we prove that for any NFA N, the DFA D that we construct in this way has L(D) = L(N)?
- The key property of D is that for any string w, $\delta^*(\{i\}, w)$ is exactly the set of states $\{q: \Delta^*(i, w, q)\}$ that could be reached from i on a w-path.
- We prove this property by induction -- it is clearly true for λ (though if we had λ-moves it would not be).

Validity of the Construction

- If we assume that $\delta^*(\{i\}, w) = \{q: \Delta^*(i, w, q)\}$, we can then prove $\delta^*(\{i\}, wa) = \{r: \Delta^*(i, wa, r)\}$ for an arbitrary letter a, using the inductive definition of δ^* in terms of δ, of δ in terms of Δ, and of Δ^* in terms of Δ.
- Once this is done, it is clear that $w \in L(D) \leftrightarrow$

  $\exists f: f \in \delta^*(\{i\}, w) \leftrightarrow \exists f: \Delta^*(i, w, f) \leftrightarrow w \in L(N)$.
- Note that in general D could have 2^k states when N has k states. But if we leave out unreachable states, D could be much smaller.

Review: Kleene's Theorem

- Our current project is to prove Kleene's Theorem, which says that a language has a regular expression if and only if it has a DFA.
- After Monday's lecture, we know that a language has a DFA if and only if it has an ordinary NFA, with no λ-moves.
- But when we convert regular expressions to machines, it will be much easier to have λ-moves available to us. To do this, we need to be able to convert a λ-NFA to an equivalent ordinary NFA. That is today's task.

Kleene's Theorem

- In one sense this construction is not costly -- the ordinary NFA we produce has the same number of states as the λ-NFA.
- But it is technically the most complicated construction in the Kleene's Theorem proof, and we will need a fair number of inductive arguments to prove the construction correct.

The Construction

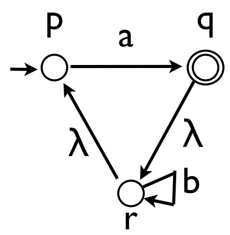
- Assume that we have a λ-NFA M, and we want to make an equivalent ordinary NFA N.
- M and N will have the same state set, start state, and input alphabet. Furthermore, if $\lambda \notin L(M)$, they also have the same final state set.
- The construction has three parts. We consider the transitions in two groups, the **letter moves** and the **λ-moves**.

The Construction

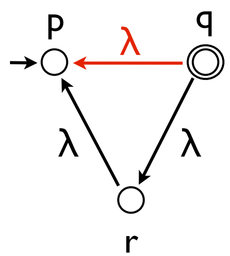
- We first add λ-moves to M until they are **transitively closed**, meaning that any λ-path has an equivalent λ-move.
- We then make the letter moves of N by finding all paths of M that read exactly one letter. We can find these by taking all three-step paths of a λ-move, a letter move, and a λ-move. (We ignore multiple copies of the same move.)
- If $\lambda \in L(M)$, we add the start state i to the final state set of N.

A Three-State Example

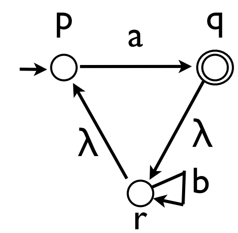
- Define a λ-NFA with state set {p, q, r}, start state p, final state set {q}, input alphabet {a, b}, and Δ = {(p, a, q), (q, λ, r), (r, λ, p), (r, b, r)}.



- There are two letter moves and two λ-moves. For the transitive closure we must add one more move (q, λ, p).



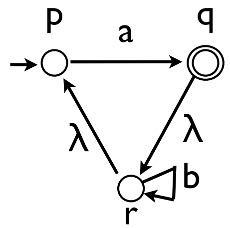
Clicker Question #1



- What is the language of this λ-NFA?
- (a) $a + (b^*a)^*$
- (b) $a(b^*a)^*$
- (c) $(a + b)^*$
- (d) $(ab^*a)^*$

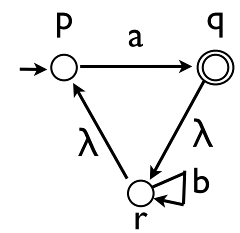
Answer #1

- What is the language of this λ-NFA?
- (a) $a + (b^*a)^*$
- *(b) $a(b^*a)^*$*
- (c) $(a + b)^*$
- (d) $(ab^*a)^*$

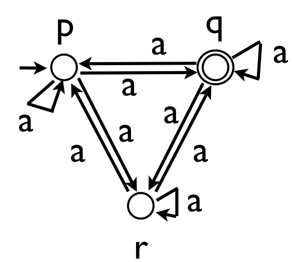


A Three-State Example

- The letter move (p, a, q) gives us a letter move from any state with a λ-move to p, to any state with a λ-move from q.

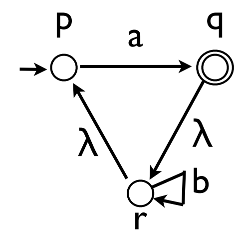


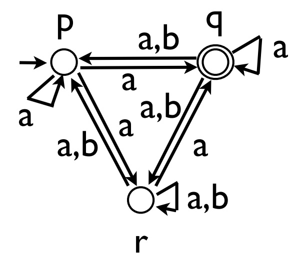
- This gives us all nine possible a-moves, since we can get from anywhere to p and from q to anywhere on λ.



A Three-State Example

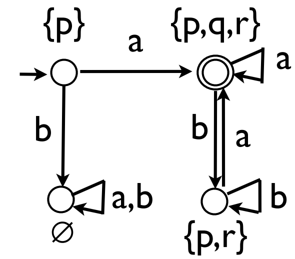
- The letter move (r, b, r) gives us letter moves from either q or r to either r or p.
- There are four such b-moves, so the ordinary NFA has 13 letter moves in all.
- Since λ ∉ L(M), we don't need to alter the final state set of the ordinary NFA.



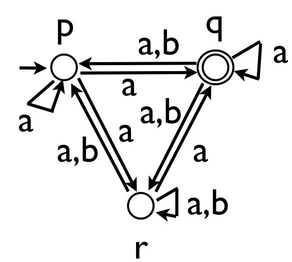


Finishing the Example

- Let's form a DFA from this NFA. The start state of the DFA is {p}. We compute $\delta(\{p\}, a) = \{p, q, r\}$ (and in fact $\delta(S, a) = \{p, q, r\}$ for any set $S \neq \varnothing$), and $\delta(\{p\}, b) = \varnothing$.

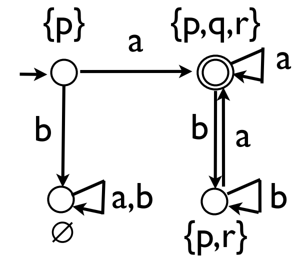


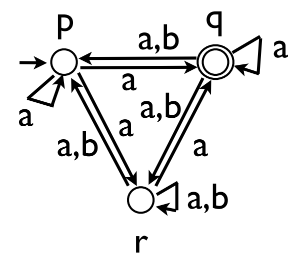
- We then compute $\delta(\{p, q, r\}, b) = \{p, r\}$ and $\delta(\{p, r\}) = \{p, r\}$. We have completed the Subset Construction with only 4 of the 8 states.



Finishing the Example

- This DFA is also the minimal DFA. We could carry out the construction, but it is perhaps easier just to show that the three non-final states are pairwise distinguishable. (Of course the single final state, {p, q, r}, is in a class by itself.)

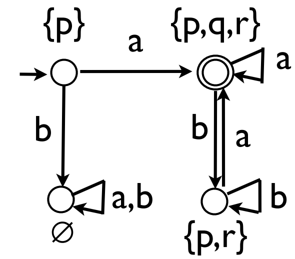


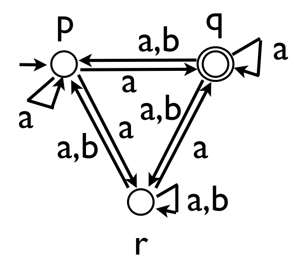


- The string a distinguishes either {p} or {p, r} from ∅, and the string b distinguishes {p} and {p, r} from each other.

Clicker Question #2

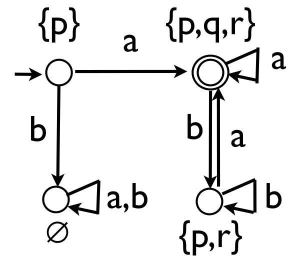
- With a DFA, it is much easier to determine what strings are *not* in the language. How many strings of length *exactly three* are not in the language?
- (a) four
- (b) six
- (c) seven
- (d) eight

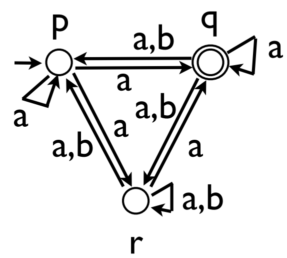




Answer #2

- With a DFA, it is much easier to determine what strings are *not* in the language. How many strings of length *exactly three* are not in the language?
- (a) four
- *(b) six (all but aaa and aba)*
- (c) seven
- (d) eight





Validity of the Construction

- Let's now assume that we have carried out this construction on a λ-NFA M to produce an ordinary NFA N -- we would like to prove that L(M) = L(N).
- We would like it to be true that for any string w, the set of states q, such that $\Delta_M^*(i, w, q)$ is true, is exactly the set of states r such that $\Delta_N^*(i, w, r)$ is true.

Validity of the Construction

- But we can't do this for the empty string λ, because there might be more than one state of M reachable on λ. In any ordinary NFA, however, the only λ-path from i goes to i itself.
- This is why we altered the final state set of N.

Validity of the Construction

- We will thus have a Lemma that these two sets are equal for any nonempty string, and we will prove this by induction on strings.
- We then have to account for empty strings. We must also make sure that our change to the final state set does not affect the membership of any nonempty strings.

Clicker Question #3

- For our Main Lemma we want to prove that for all nonempty strings w, the two machines have exactly the same Δ^* relation. How shall we do this by induction?
- (a) One base case for each a in Σ, induction $P(w) \rightarrow P(wa)$ for each a in Σ
- (b) Base case $w = \lambda$, induction $P(w) \rightarrow P(wa)$
- (c) Base cases a and $\emptyset$, induction for $+, \cdot, ^*$
- (d) Base case $P(0)$, induction $P(n) \rightarrow P(n+1)$

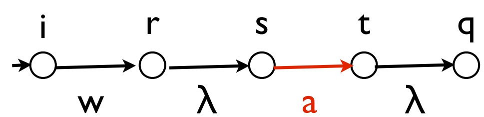
Answer #3

- For our Main Lemma we want to prove that for all nonempty strings w, the two machines have exactly the same Δ^* relation. How shall we do this by induction?
- *(a) One base case for each a in Σ, induction $P(w) \rightarrow P(wa)$ for each a in Σ*
- (b) Base case $w = \lambda$, induction $P(w) \rightarrow P(wa)$
- (c) Base cases a and $\emptyset$, induction for $+$, $\cdot$, *
- (d) Base case $P(0)$, induction $P(n) \rightarrow P(n+1)$

The Main Lemma

- To save subscripts, we will refer to the relations for M as Δ and Δ^*, and those for N as Γ and Γ^*. We are proving $\forall w: (w \neq \lambda) \rightarrow [\forall q: \Delta^*(i, w, q) \leftrightarrow \Gamma^*(i, w, q)]$.

- Remember that Δ^* with middle term λ is defined in terms of λ-paths, and that $\Delta^*(i, wa, q)$ is defined to be $\exists r: \exists s: \exists t: \Delta^*(i, w, r) \wedge \Delta^*(r, \lambda, s) \wedge \Delta(s, a, t) \wedge \Delta^*(t, \lambda, q)$.



Proving the Main Lemma

- $\Gamma(s, \lambda, t)$ means just $s = t$, and $\Gamma^*(i, wa, q)$ is defined to be $\exists z: \Gamma^*(i, w, z) \wedge \Gamma(z, a, q)$. By the definition of Γ, we know that $\Gamma(z, a, q)$ is true if and only if $\exists r: \exists t: \Delta^*(z, \lambda, r) \wedge \Delta(r, a, t) \wedge \Delta^*(t, \lambda, q)$.
- For our base case we compute both $\Delta^*(i, a, q)$ and $\Gamma^*(i, a, q)$ and find them to be equal.

Proving the Main Lemma

- For the inductive case we assume that $\Delta^*(i, w, q) \leftrightarrow \Gamma^*(i, w, q)$ and use the definitions above to prove that $\Delta^*(i, wa, r) \leftrightarrow \Gamma^*(i, wa, r)$.
- $\Delta^*(i, wa, r) \leftrightarrow \exists z{:}\exists s{:}\exists t{:}\ \Delta^*(i, w, z) \wedge \Delta^*(z, \lambda, s) \wedge \Delta(s, a, t) \wedge \Delta^*(t, \lambda, r)$
- $\Gamma^*(i, wa, r) \leftrightarrow \exists z{:}\ \Gamma^*(i, w, z) \wedge \exists s{:}\exists t{:}\ \Delta^*(z, \lambda, s) \wedge \Delta(s, a, t) \wedge \Delta^*(t, \lambda, r)$

The Case of Empty Strings

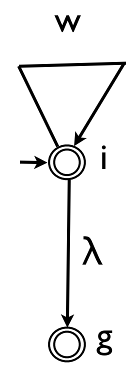
- If $\lambda \notin L(M)$, the final state sets F_M and F_N are the same, so we know from the Lemma that every nonempty string is in L(M) if and only if it is in L(N).
- All we need to do, then, is prove that λ is not in L(N). Since N has no λ-moves, we just need to show that i is not a final state. But if i were a final state, λ would be in L(M), and it isn't. So in this case L(M) = L(N).

The Case of Empty Strings

- Now suppose that $\lambda \in L(M)$, so that by the last step of our construction $F_N = F_M \cup \{i\}$.
- It's clear that λ is in $L(N)$, which is good because it is in $L(M)$.
- Now consider any non-empty string w. If $w \in L(M)$, then $\Delta^*(i, w, f)$ for some $f \in F_M$. By the Lemma, $\Gamma^*(i, w, f)$ is also true, and since $f \in F_N$ as well, $w \in L(N)$.

The Case of Empty Strings

- Finally, suppose that $w \in L(N)$, so that $\Gamma^*(i, w, f)$ for some $f \in F_N$. By the Lemma, $\Delta^*(i, w, f)$ as well. If $f \in F_M$, this tells us that $w \in L(N)$.
- But what if $f = i$? Since $\lambda \in L(M)$, we have $\Delta^*(i, \lambda, g)$ for some state $g \in F_M$. From $\Delta^*(i, w, i)$ and $\Delta^*(i, \lambda, g)$ we can derive $\Delta^*(i, w, g)$, and thus $w \in L(M)$ here as well.



a