CMPSCI 250: Introduction to Computation

Lecture #22: From λ-NFA’s to NFA’s to DFA’s
David Mix Barrington
22 April 2013

λ-NFA's to NFA's to DFA's

- Reviewing the Three Models and Kleene's Theorem
- The Subset Construction: NFA's to DFA's
- Applying the Construction to No-aba
- The Validity of the Subset Construction
- The Construction to Kill λ-Moves
- A Three-State Example
- The Validity of the Construction

Overview: Kleene’s Theorem

- We are discussing two classes of languages (sets of strings over a given alphabet). The **regular languages** are the ones that are denoted by **regular expressions**. The **recognizable languages** are the ones that can be decided by **deterministic finite automata (DFA’s)**.

- **Kleene’s Theorem** says that these two classes are *the same* (and thus both are usually called “regular languages”). We’ll do part of the proof of this theorem today and part in the next lecture on Thursday. The proof will consist of algorithms to take an arbitrary regular expression and produce an equivalent DFA, or vice versa.

- Last lecture we introduced two variants of the DFA that will make our proof easier. When we convert a regular expression to a finite automaton, our task is easier if we allow the automaton to be **nondeterministic** and to have **λ-moves**. Today we’ll see how to turn these variant machines into ordinary DFA’s.

Our Three Automaton Models

- Each of our different types of **finite-state machines** has a **state set**, a **start state**, and a set of **final states**. Given any input string w, there are zero or more possible **paths** through the machine -- sequences of **transitions** whose labels are the letters of w, in order.

- In a **DFA**, there is exactly one transition out of each state for each letter in the alphabet. Thus there is a *function* δ from $Q \times \Sigma$ to Q, such that $\delta(q, a)$ is the state to which the machine *must* move if it sees an a when in state q. Given a string w, there is one w-path from the start state, and the string is in the language of the DFA if that path goes to a final state.

- In an **NFA**, all we know is that there is a set of transitions, each an element of the set $Q \times \Sigma \times Q$. If we are in state q and see an a, there may be zero, one, or more than one states r such that (q, a, r) is a transition. The machine may take any of these transitions, and thus given an entire string w there may be zero, one or more than one path through the machine with labels forming w. The string w is in the **language of the machine** if there is at least one path.

λ-NFA's

- In a λ-NFA, there may also be transitions labeled with the empty string λ. If there is a transition (q, λ, r), this means that the machine may move from state q to state r without reading any letter at all. Thus if the machine is in state q and the next letter is a, then along with whatever letter moves (q, a, s) may exist, it has the option of taking a λ-move without reading the a.

- For w to be in the language of a λ-NFA, there must be a path from a start state to some final state, whose labels consist of the letters of w in order, along with any number of λ-moves between those letters.

- Suppose w is a string of n letters. With a DFA, we test whether w is in its language by just tracing out the *single* path labeled by w. With an NFA, we can test this by checking *all possible* paths of length n. With a λ-NFA, there is no obvious limit on how long a path we might have to check to be sure that there isn't any path from the start state to a final state.

The Subset Construction: NFA's to DFA's

- Later in this we'll see how to convert λ-NFA's to ordinary NFA's. Now, though, we will convert ordinary NFA's to DFA's using the **Subset Construction**. Given an NFA N with state set Q, we will build a DFA D whose states will be *sets of states* of N -- formally, D's state set is the **power set** of Q.

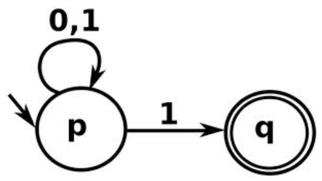
- Here's an example of an NFA N for the language (0 + 01)*, with two states i and p, start state i, final state set {i}, and transitions (i, 0, i), (i, 0, p), and (p, 1, i).

- At the start of its run, N must be in state i. If the first letter is 0, then it might be in either state i or p after reading the 0. If the first letter is 1, there is no run of N that reads that letter.

- Our DFA D has states ∅, {i}, {p}, and {i, p}. Its start state is {i}, its final states are {i} and {i, p}, and we have δ({i}, 0) = {i, p}, δ({i}, 1) = ∅, δ({i, p}, 0) = {i, p}, δ({i, p}, 1) = {i}, δ({p}, 0) = ∅, δ({p}, 1) = {0}, and δ(∅, a) = ∅ for both letters.

iClicker Question #1: Converting an NFA to a DFA

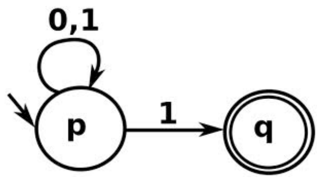
- Let N be the pictured NFA. Let D be the DFA made from N by the Subset Construction, with δ its transition function. Which statement about δ below is **correct**?



- (a) δ({p}, 0) = ∅ and δ({p}, 1) = {q}
- (b) δ({p}, 0) = {p} and δ({p}, 1) = {q}
- (c) δ({p}, 0) = ∅ and δ({p}, 1) = {p, q}
- (d) δ({p}, 0) = {p} and δ({p}, 1) = {p, q}

iClicker Question #2: Finishing The Conversion

- Since δ({p}, 1) = {p, q}, we need to determine the transitions out of the state {p, q}. Which of these four statements about δ is **correct**?
- (a) δ({p, q}, 0) = {p} and δ({p, q}, 1) = ∅
- (b) δ({p, q}, 0) = ∅ and δ({p, q}, 1) = ∅
- (c) δ({p, q}, 0) = {p} and δ({p, q}, 1) = {p, q}
- (d) δ({p, q}, 0) = ∅ and δ({p, q}, 1) = {p, q}



Details of the Construction

- The general construction works just like this example. The start state of D is {i}, where i is the start state of N. The final state set of D is the set of all states of D that *contain* final states of N, since we want D to accept if N *can* accept.

- In general, we need to define δ(S, a) where S is a state of D, meaning that S is a set of states of N. S represents the possible places N *might* be before reading the a. The set T = δ(S, a) will be the set of all states q such that the transition (s, a, q) is in Δ for *some* s ∈ S. In the graph, we take the set of destinations of all the a-arrows that start from a state of S.

- The most common mistake in computing δ comes when one of the states in S has no a-arrows out of it. Students often think that ∅ must now be part of δ(S, a). But in fact δ(S, a) is the *union* of the sets {q: Δ(s, a, q)} for each s ∈ S. So the empty set is part of the result, but doesn't show up in the description of the result because unioning in ∅ is the identity operation on sets.

Applying the Construction to No-aba

- The language Yes-aba has an easy regular expression $\Sigma^*aba\Sigma^*$. From this expression we can build an NFA N with state set {1, 2, 3, 4}, start state 1, final state set {4}, and Δ = {(1, a, 1), (1, b, 1), (1, a, 2), (2, b, 3), (3, a, 4), (4, a, 4), (4, b, 4)}. But what if we want a machine for No-aba? Switching the final and non-final states of N will not do -- can you see why?

- 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}. 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}. Finally, with δ({1, 4}, a) = {1, 2, 4} and δ({1, 4}, b) = {1, 4}, we are done -- the other states are unreachable.

- Clearly if we minimized this DFA, the three final states would merge into one. This gives us our familiar four-state DFA for Yes-aba, from which we can get one for No-aba.

The 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). 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 don't generate unreachable states, D could turn out to be much smaller.

The Construction to Kill λ-Moves

- 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 λ ∉ 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**.

- 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 λ ∈ L(M), we add the start state i to the final state set of N.

iClicker Question #3: Transitive Closure of λ-Moves

- Suppose that a λ-NFA has four states p, q, r, s, and has exactly three λ-moves: (p, λ, q), (r, λ, p), and (r, λ, s). What new λ-moves must we add to make the λ-moves transitively closed, without changing the language of the λ-NFA?
- (a) none
- (b) (q, λ, p), (p, λ, r), and (s, λ, r)
- (c) (r, λ, q) only
- (d) (r, λ, q), (s, λ, p), and (s, λ, q)

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).
- 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.
- 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 δ({p}, a) = {p, q, r} (and in fact δ takes any nonempty set and a to {p, q, r}), and δ({p}, b) = ∅. We then compute δ({p, q, r}, b) = {p, r} and δ({p, r}) = {p, r}. We have completed the Subset Construction with only four of the possible eight states being reachable.

- 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.

iClicker Question #4: New Letter Moves From Old

- Suppose that our λ-NFA has state set {p, q, r, s} and only the two λ-moves (p, λ, q) and (r, λ, s). (The transitive closure operation adds no new λ-moves.) Suppose further that (q, a, r) is a letter-move in the λ-NFA. **What set of letter moves** do we add to our ordinary NFA because of this letter-move?

- (a) none

- (b) (q, a, r) only

- (c) (p, a, r), (p, a, s), (q, a, r), and (q, a, s)

- (d) all sixteen possible moves (x, a, y) for any x and y in {p, q, r, s}

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 Δ_M*(i, w, q) is exactly the set of states r such that Δ_N*(i, w, r). But we can't do this for the empty string, because there might be more than one state of M reachable on λ, but in an ordinary NFA the only λ-path from i goes to i itself. This is why we altered the final state set of N.

- 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, and make sure as well that our change to the final state set does not affect the membership of any nonempty strings.

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 $\exists r:\exists s:\exists t: \Delta^*(i, w, r) \wedge \Delta^*(r, \lambda, s) \wedge \Delta(s, a, t) \wedge \Delta^*(t, \lambda, q)$.

- $\Gamma(s, \lambda, t)$ means just $s = t$, and $\Gamma^*(i, wa, q)$ is defined to be $\exists z: \Gamma^*(i, w, z) \wedge \Gamma(z, a, q)$, and $\Gamma(z, a, q)$ is defined to be $\exists r:\exists t: \Delta^*(z, \lambda, r) \wedge \Delta(r, a, t) \wedge \Delta^*(t, \lambda, q)$.

- For our base case we compute both $\Delta^*(i, a, q)$ and $\Gamma^*(i, a, q)$ and find them equal.

- For the inductive case we assume that $\Delta^*(i, w, q) \leftrightarrow \Gamma^*(i, w, q)$ and use the definitions above to prove that $\Delta^*(i, wa, r) \leftrightarrow \Gamma^*(i, wa, r)$.

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 $L(M) = L(N)$.

- Now suppose that $\lambda \in L(M)$, so that by our last step $F_N = F_M \cup \{i\}$. 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, $w \in L(N)$. 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.