CMPSCI 250: Introduction to Computation

Lecture 20: Deterministic and Nondeterministic Finite Automata
David Mix Barrington
16 April 2013

Deterministic and Nondeterministic Finite Automata

- Deterministic Finite Automata
- Formal Definition of DFA’s
- Examples of DFA’s
- Characterizing the Strings For Each State
- Nondeterministic Finite Automata
- Interpretations of Nondeterminism
- The Model of λ-NFA’s

Deterministic Finite Automata

- We now turn to **finite-state machines**, a model of computation that captures the idea of reading a file of text with a fixed limit on the memory we can use to remember what we have seen.
- In particular, the memory used must be **constant**, independent of the length of the file. (We called this “O(1)” in CMPSCI 187.) We ensure this by requiring our machine to have a **finite state set**, so that at any time during the computation all that it knows is which state it is in.
- The **initial state** is fixed. When the machine sees a new letter, it changes to a new state based on a fixed **transition function**. When it finishes the string, it gives a yes or no answer based on whether it is in a **final state**.
- Because the new state depends only on the old state and the letter seen, the computation is **deterministic** and the machine is called a **deterministic finite automaton** or **DFA**.

Where We Are Going

- A DFA **decides** a language -- it says yes or no after reading any string over its alphabet, and its language is the set of strings for which it says yes.

- The **Myhill-Nerode Theorem** will give us a way to take an arbitrary language and determine whether there is a DFA that decides it. We'll define a particular equivalence relation on strings, based only on the language. If this relation has a finite set of equivalence classes, there is a DFA for the language, and there is a **minimal DFA** with as many states as there are classes. We'll see how to compute the minimal DFA from any DFA for the language.

- As we've mentioned, there is a DFA for a language if and only if the language is **regular** (is the language denoted by some regular expression). We'll prove this important result, called **Kleene's Theorem**, over several lectures. Our proofs will show us how to convert a DFA to a regular expression and vice versa.

Formal Definition of DFA’s

- Formally a DFA is defined by its **state set** S, its **initial state** $i \in S$, its **final state set** $F \subseteq S$, its **input alphabet** Σ, and its **transition function** δ from $(S \times \Sigma)$ to S.

- We usually represent DFA’s by diagrams (labeled directed multigraphs) with a node for each state, a special mark for the initial state, a double circle on each final state, and an arrow labeled “a” from node p to node q whenever $\delta(p, a) = q$.

- The **behavior function** of a particular DFA is a function called δ^* from $(S \times \Sigma^*)$ to S, such that $\delta(p, w)$ is the state of the DFA after it starts in state p and reads the string w. Formally, we say that $\delta(p, \lambda) = p$ and that $\delta^*(p, wa) = \delta(\delta^*(p, w), a)$.

- The **language of a DFA** is defined to be the set of strings w such that $\delta^*(i, w)$ is a final state. For a DFA M, we call this language L(M).

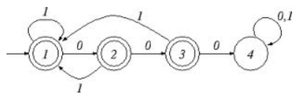
An Example of a DFA

- This DFA has four states, 1, 2, 3, and 4. Its input alphabet is {0, 1}. Its start state is 1, and its final state set is {1, 2, 3}.

- Examples of strings in the language of this DFA are λ (because 1 is final), 0 (because 2 is final), and 00100 (because 3 is final).

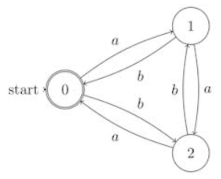
- Any string with three 0's in a row is not in the language of this DFA, because 000 takes you from anywhere to the nonfinal "death state" 4.

- In fact the only way to get to 4 is by three 0's in a row, so the language of this DFA is the *complement* of the regular language Σ*000Σ*.



iClicker Question #1: DFA Diagrams

- Which statement about the pictured DFA is **false**?
- (a) The alphabet of the DFA is {a, b}.
- (b) The start state is not a final state.
- (c) The string bbaa is in the language of the DFA.
- (d) The string aaabb is not in the language of the DFA.



Examples of DFA's

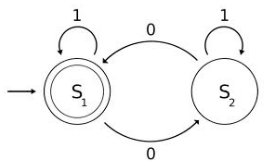
- One of the simplest possible DFA's decides the language of binary strings with an odd number of ones. It has two states E and O, representing whether the machine has seen an even or odd number of ones so far. The initial state is E, and the final state set is {O}. The transition function has $\delta(E, 0) = E$, $\delta(E, 1) = O$, $\delta(O, 0) = O$, and $\delta(O, 1) = E$.

- We can build a four-state DFA for the language EE from Discussion #11 tomorrow. Its states are EE, EO, OE, and OO, where for example $\delta^*(EE, w) = EO$ if w has an even number of a's and an odd number of b's. The initial state is EE and the final state set is {EE}. An a changes the first letter of the state, a b the second.

- Another four-state DFA can decide whether the next to last letter of a binary string w exists and is 1. The state set is {00, 01, 10, 11} and the state after reading w represents the last two letters seen. The initial state is 00 and the final state set is {10, 11}.

iClicker Question #2: Language of a DFA

- What is the language of the pictured DFA?
- (a) all strings with an even number of 1's
- (b) all strings of even length
- (c) all strings with an odd number of 0's
- (d) all strings with an even number of 0's



DFA’s in Pseudo-Java

- We consider the input to be given like a file, with a method to give the next letter and one to tell when the input is done. We relabel the state set and the alphabet to be `{0,..., states - 1}` and `{0,..., letters - 1}` respectively.

```
public class DFA {
   natural states; natural letters; natural start;
   boolean [ ] isFinal = new boolean[states];
   natural [ ] [ ] delta = new natural [states] [letters];
   natural getNext( ) {code omitted}
   boolean inputDone( ) {code omitted}

   boolean decide ( )
   {// returns whether input string is in language of DFA
      natural current = start;
      while (!inputDone( ))
         current = delta [current] [getNext( )];
      return isFinal [current];}}
```

Characterizing Strings With Given Behavior

- How do we *prove* that a particular DFA has a particular language?
- With the even-odd DFA, we can say that $\delta^*(E, w) = E$ if w has an even number of ones, and $\delta^*(E, w) = O$ if it has an odd number of ones.
- Letting $P(w)$ be the entire statement in the bullet above, we can prove $\forall w{:}P(w)$ by induction on all binary strings. $P(\lambda)$ says that $\delta^*(E, \lambda) = E$, because λ has no ones and 0 is even, and $\delta^*(E, \lambda) = E$ is true by definition of δ^*. Now assume that $P(w)$ is true, so that $\delta^*(E, w)$ is E if w has an even number of ones and O otherwise. Then $w0$ has the same number of ones as w, so $\delta^*(E, w0)$ should be the same state as $\delta^*(E, w)$. And $w1$ has one more one than w, so $\delta^*(E, w1)$ should be the other state from $\delta^*(E, w)$. In each of the four cases, the new state is the one given by the δ function of the DFA.

Another Characterization Example

- The language No-aba is the set of strings that never have an aba substring. We can build a DFA M for No-aba with state set {1, 2, 3, 4}, start state 1, final state set {1, 2, 3}, and transition function $\delta(1, a) = 2$, $\delta(1, b) = 1$, $\delta(2, a) = 2$, $\delta(2, b) = 3$, $\delta(3, a) = 4$, $\delta(3, b) = 1$, and $\delta(4, a) = \delta(4, b) = 4$. (We call 4 a **death state**.) We can see that an aba will take us from any state to 4.

- Let L_1 be the set of strings that have no aba and don't end in a or ab. Let L_2 be the set of strings that don't have an aba and end in a. L_3 is the set of strings that don't have an aba and end in ab, and L_4 is the set that have aba.

- We can make eight checks, one for each value of δ. If $\delta(i, x) = j$, we check that any string in L_i, followed by the letter x, yields a string in L_j. Then we can do an inductive proof, where our statement P(w) is the *entire* statement in the bullet above: "For all states i, $\delta^*(1, w) = i$ if and only if $w \in L_i$" where each L_i is as defined. This proves that w is in L(M) if and only if w is in No-aba.

Nondeterministic Finite Automata

- DFA's are **deterministic** in that the same input always leads to the same output. Some algorithms are not deterministic because they are randomized, but here we will consider "algorithms" that are not deterministic because they are **underdefined** -- given a single input, more than one output is possible.

- We had an example of such an algorithm with our generic search, which didn't say which element came off the open list when we needed a new one.

- Formally, a **nondeterministic finite automaton** or **NFA** has an alphabet, state set, start state, and final state just like a DFA. But instead of the transition function δ, it has a **transition relation** $\Delta \subseteq Q \times \Sigma \times Q$. If $(p, a, q) \in \Delta$, the NFA *may* move to state q if it sees the letter a while in state p. We draw an NFA like a DFA, with an a-arrow from p to q whenever $(p, a, q) \in \Delta$. The NFA no longer has the rule that there must be exactly one arrow for each letter out of each state -- there may be more than one, or none.

The Language of an NFA

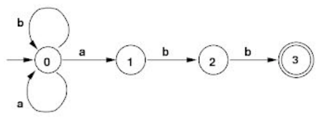
- We can no longer say what the NFA *will* do when reading a string, only what it *might* do. The **language of an NFA** N is defined to be the set {w: w *might* be accepted by N}. More formally, we define a relation $\Delta^* \subseteq Q \times \Sigma^* \times Q$ so that the triple (p, w, q) is in Δ^* if and only if N *might* go from p to q while reading w. Then $w \in L(N) \leftrightarrow (i, w, f) \in \Delta^*$ for some final state $f \in F$.

- Consider the NFA N with state set {i, p, q}, start state i, final state set {i}, alphabet {a, b, c}, and Δ = {(i, a, i), (i, a, p), (p, b, i), (i, b, q), (q, c, i)}. This is nondeterministic because there are two a-moves out of i, and several situations with no move at all. Here L(N) is the regular language (a + ab+ bc)*, because any path from i to itself must consist of pieces labeled a, ab, or bc.

- It is not immediately clear how, for a larger NFA, we could determine whether a particular string was in L(N). Our method will be to turn N into a DFA.

iClicker Question #3: The Language of an NFA

- What is the language of this NFA?
- (a) Exactly those strings ending in “abb”.
- (b) The set {abb}.
- (c) Exactly those strings containing “abb” as a substring.
- (d) The language is empty.



Interpretations of Nondeterminism

- Because we can't speak clearly of "what happens when we run N on w", we need other ways to think of the action of an NFA.

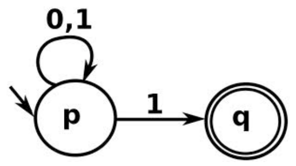
- In our proofs, we will just replace "$w \in L(N)$" by "$\exists f: (i, w, f) \in \Delta^*$" and argue about the possible w-paths in the graph of N.

- We can think of N as being **randomized**, so that whenever it has a choice of moves it selects one of them uniformly at random. (This essentially makes N a **Markov process**, as studied in CMPSCI 240.) Then we could speak of the *probability* that N accepts w, and $w \in L(N)$ if and only if this probability is greater than 0.

- We can think of the action of N on w as a **one-player game** where White, who want N to accept w, chooses each move from the set of legal options. Then White has a winning strategy for this game if and only if $w \in L(N)$.

iClicker Question #4: A Randomized NFA

- Consider running the pictured NFA *randomly* on the string "101". When there is one arrow available we take it, when there are two we flip a coin, and when there are none we die. **What is the probability** that we will accept this string?
- (a) 0
- (b) 1/4
- (c) 1/2
- (d) 3/4



The Model of λ-NFA's

- The main reason to use NFA's is that they are easier to design in many situations when we have some other definition of the language. Often we will find it convenient to give the NFA the option to jump from one state to another *without reading a letter*.

- A **λ-move** is a transition (p, λ, q) that allows a **λ-NFA** to do just that. We need to redefine the type of Δ, so that it is a subset of $Q \times (\Sigma \cup \{\lambda\}) \times Q$. In the diagram, this transition is represented by an arrow from p to q labeled with λ.

- Formally Δ^* is now more complicated to define. We say that $(p, \lambda, q) \in \Delta^*$ if there is a *path* of λ-moves from p to q. Then we define $\Delta^*(p, wa, q)$ to be true if and only if there exist states r, s, and t such that (p, w, r), (r, λ, s) and (t, λ, q) are all in Δ^*, and (s, a, t) is in Δ. What this means is that $\Delta^*(p, w, q)$ is true if and only if there exists a path from p to q such that the *letters* on the path, read in order, spell out w. There may be any number of λ-moves in the path as well. (Thus we don't even know how long the path from p to q might be.)