Lecture 5

Nondeterministic Finite Automata

Nondeterminism

Nondeterminism is an important abstraction in computer science. It refers to situations in which the next state of a computation is not uniquely determined by the current state. Nondeterminism arises in real life when there is incomplete information about the state or when there are external forces at work that can affect the course of a computation. For example, the behavior of a process in a distributed system might depend on messages from other processes that arrive at unpredictable times with unpredictable contents.

Nondeterminism is also important in the design of efficient algorithms. There are many instances of important combinatorial problems with efficient nondeterministic solutions but no known efficient deterministic solution. The famous $P = NP$ problem—whether all problems solvable in nondeterministic polynomial time can be solved in deterministic polynomial time—is a major open problem in computer science and arguably one of the most important open problems in all of mathematics.

In nondeterministic situations, we may not know how a computation will evolve, but we may have some idea of the range of possibilities. This is modeled formally by allowing automata to have multiple-valued transition functions.

In this lecture and the next, we will show how nondeterminism is incorporated naturally in the context of finite automata. One might think that adding nondeterminism might increase expressive power, but in fact for finite automata it does not: in terms of the sets accepted, nondeterministic finite automata are no more powerful than deterministic ones. In other words, for every nondeterministic finite automaton, there is a deterministic one accepting the same set. However, nondeterministic machines may be exponentially more succinct.

Nondeterministic Finite Automata

A *nondeterministic finite automaton* (NFA) is one for which the next state is not necessarily uniquely determined by the current state and input symbol. In a deterministic automaton, there is exactly one start state and exactly one transition out of each state for each symbol in Σ. In a nondeterministic automaton, there may be one, more than one, or zero. The set of *possible* next states that the automaton may move to from a particular state q in response to a particular input symbol a is part of the specification of the automaton, but there is no mechanism for deciding which one will actually be taken. Formally, we won't be able to represent this with a function $\delta : Q \times \Sigma \to Q$ anymore; we will have to use something more general. Also, a nondeterministic automaton may have many start states and may start in any one of them.

Informally, a nondeterministic automaton is said to *accept* its input x if it is possible to start in some start state and scan x, moving according to the transition rules and making choices along the way whenever the next state is not uniquely determined, such that when the end of x is reached, the machine is in an accept state. Because the start state is not determined and because of the choices along the way, there may be several possible paths through the automaton in response to the input x; some may lead to accept states while others may lead to reject states. The automaton is said to *accept* x if *at least one* computation path on input x starting from *at least one* start state leads to an accept state. The automaton is said to *reject* x if *no* computation path on input x from *any* start state leads to an accept state. Another way of saying this is that x is accepted iff there exists a path with label x from some start state to some accept state. Again, there is no mechanism for determining which state to start in or which of the possible next moves to take in response to an input symbol.

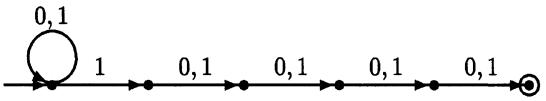
It is helpful to think about this process in terms of *guessing* and *verifying*. On a given input, imagine the automaton *guessing* a successful computation or proof that the input is a "yes" instance of the decision problem, then *verifying* that its guess was indeed correct.

For example, consider the set

$$A = \{x \in \{0,1\}^* \mid \text{the fifth symbol from the right is } 1\}.$$

Thus $11010010 \in A$ but $11000010 \notin A$.

Here is a six-state nondeterministic automaton accepting A:



There is only one start state, namely the leftmost, and only one accept state, namely the rightmost. The automaton is not deterministic, because there are two transitions from the leftmost state labeled 1 (one back to itself and one to the second state) and no transitions from the rightmost state. This automaton accepts the set A, because for any string x whose fifth symbol from the right is 1, *there exists* a sequence of legal transitions leading from the start state to the accept state (it moves from the first state to the second when it scans the fifth symbol from the right); and for any string x whose fifth symbol from the right is 0, there is *no possible* sequence of legal transitions leading to the accept state, no matter what choices it makes (recall that to accept, the machine must be in an accept state when the end of the input string is reached).

Intuitively, we can think of the machine in the leftmost state as *guessing*, every time it sees a 1, whether that 1 is the fifth letter from the right. It might be and it might not be—the machine doesn't know, and there is no way for it to tell at that point. If it guesses that it is not, then it goes around the loop again. If it guesses that it is, then it commits to that guess by moving to the second state, an irrevocable decision. Now it must *verify* that its guess was correct; this is the purpose of the tail of the automaton leading to the accept state. If the 1 that it guessed was fifth from the right really is fifth from the right, then the machine will be in its accept state exactly when it comes to the end of the input string, therefore it will accept the string. If not, then maybe the symbol fifth from the right is a 0, and *no* guess would have worked; or maybe the symbol fifth from the right was a 1, but the machine just guessed the wrong 1.

Note, however, that for any string $x \in A$ (that is, for any string with a 1 fifth from the right), *there is* a lucky guess that leads to acceptance; whereas for any string $x \notin A$ (that is, for any string with a 0 fifth from the right), *no* guess can possibly lead to acceptance, no matter how lucky the automaton is.

In general, to show that a nondeterministic machine accepts a set B, we must argue that for any string $x \in B$, there is a lucky sequence of guesses that leads from a start state to an accept state when the end of x is reached;

but for any string $x \notin B$, *no* sequence of guesses leads to an accept state when the end of x is reached, no matter how lucky the automaton is.

Keep in mind that this process of *guessing and verifying* is just an intuitive aid. The formal definition of nondeterministic acceptance will be given in Lecture 6.

There does exist a deterministic automaton accepting the set A, but any such automaton must have at least $2^5 = 32$ states, since a deterministic machine essentially has to remember the last five symbols seen.

The Subset Construction

We will prove a rather remarkable fact: in terms of the sets accepted, nondeterministic finite automata are no more powerful than deterministic ones. In other words, for every nondeterministic finite automaton, there is a deterministic one accepting the same set. The deterministic automaton, however, may require more states.

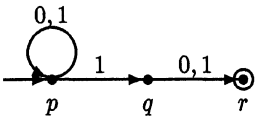
This theorem can be proved using the *subset construction.* Here is the intuitive idea; we will give a formal treatment in Lecture 6. Given a nondeterministic machine N, think of putting pebbles on the states to keep track of all the states N could possibly be in after scanning a prefix of the input. We start with pebbles on all the start states of the nondeterministic machine. Say after scanning some prefix y of the input string, we have pebbles on some set P of states, and say P is the set of all states N could possibly be in after scanning y, depending on the nondeterministic choices that N could have made so far. If input symbol b comes in, pick the pebbles up off the states of P and put a pebble down on each state reachable from a state in P under input symbol b. Let P' be the new set of states covered by pebbles. Then P' is the set of states that N could possibly be in after scanning yb.

Although for a state q of N, there may be many possible next states after scanning b, note that the set P' is uniquely determined by b and the set P. We will thus build a deterministic automaton M *whose states are these sets.* That is, a state of M will be a *set* of states of N. The start state of M will be the *set* of start states of N, indicating that we start with one pebble on each of the start states of N. A final state of M will be any set P containing a final state of N, since we want to accept x if it is possible for N to have made choices while scanning x that lead to an accept state of N.

It takes a stretch of the imagination to regard a set of states of N as a single state of M. Let's illustrate the construction with a shortened version of the example above.

Example 5.1 Consider the set

$$A = \{x \in \{0,1\}^* \mid \text{the second symbol from the right is } 1\}.$$



Label the states p, q, r from left to right, as illustrated. The states of M will be *subsets* of the set of states of N. In this example there are eight such subsets:

$$\varnothing,\ \{p\},\ \{q\},\ \{r\},\ \{p,q\},\ \{p,r\},\ \{q,r\},\ \{p,q,r\}.$$

Here is the deterministic automaton M:

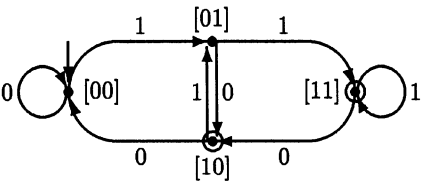
	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\rightarrow \{p\}$	$\{p\}$	$\{p,q\}$
$\{q\}$	$\{r\}$	$\{r\}$
$\{r\}F$	$\varnothing$	$\varnothing$
$\{p,q\}$	$\{p,r\}$	$\{p,q,r\}$
$\{p,r\}F$	$\{p\}$	$\{p,q\}$
$\{q,r\}F$	$\{r\}$	$\{r\}$
$\{p,q,r\}F$	$\{p,r\}$	$\{p,q,r\}$

For example, if we have pebbles on p and q (the fifth row of the table), and if we see input symbol 0 (first column), then in the next step there will be pebbles on p and r. This is because in the automaton N, p is reachable from p under input 0 and r is reachable from q under input 0, and these are the only states reachable from p and q under input 0. The accept states of M (marked F in the table) are those sets containing an accept state of N. The start state of M is $\{p\}$, the set of all start states of N.

Following 0 and 1 transitions from the start state $\{p\}$ of M, one can see that states $\{q,r\}$, $\{q\}$, $\{r\}$, $\varnothing$ of M can never be reached. These states of M are *inaccessible*, and we might as well throw them out. This leaves

	0	1
$\rightarrow \{p\}$	$\{p\}$	$\{p,q\}$
$\{p,q\}$	$\{p,r\}$	$\{p,q,r\}$
$\{p,r\}F$	$\{p\}$	$\{p,q\}$
$\{p,q,r\}F$	$\{p,r\}$	$\{p,q,r\}$

This four-state automaton is exactly the one you would have come up with if you had built a deterministic automaton directly to remember the last two bits seen and accept if the next-to-last bit is a 1:



Here the state labels $[bc]$ indicate the last two bits seen (for our purposes the null string is as good as having just seen two 0's). Note that these two automata are isomorphic (i.e., they are the same automaton up to the renaming of states):

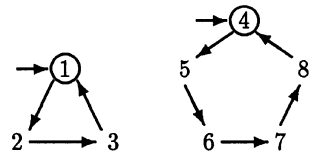
$$\begin{aligned} \{p\} &\approx [00], \\ \{p,q\} &\approx [01], \\ \{p,r\} &\approx [10], \\ \{p,q,r\} &\approx [11]. \end{aligned}$$

□

Example 5.2 Consider the set

$$\{x \in \{a\}^* \mid |x| \text{ is divisible by 3 or 5}\}. \tag{5.1}$$

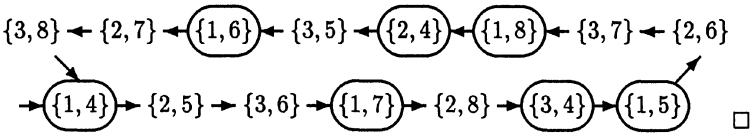
Here is an eight-state nondeterministic automaton N with two start states accepting this set (labels a on transitions are omitted since there is only one input symbol).



The only nondeterminism is in the choice of start state. The machine guesses at the outset whether to check for divisibility by 3 or 5. After that, the computation is deterministic.

Let Q be the states of N. We will build a deterministic machine M whose states are subsets of Q. There are $2^8 = 256$ of these in all, but most will be inaccessible (not reachable from the start state of M under any input). Think about moving pebbles—for this particular automaton, if you start with pebbles on the start states and move pebbles to mark all states the machine could possibly be in, you always have exactly two pebbles on N. This says that only subsets of Q with two elements will be accessible as states of M.

The subset construction gives the following deterministic automaton M with 15 accessible states:



□

In the next lecture we will give a formal definition of nondeterministic finite automata and a general account of the subset construction.