CS/Math 240: Introduction to Discrete Mathematics 4/14/2011

Lecture 21 : Finite State Machines

Instructor: Dieter van Melkebeek Scribe: Dalibor Zelený

DRAFT

Last time we started discussing applications of graphs in computer science. We phrased the register allocation problem in terms of graph coloring. Today we discuss finite state machines, a simple model of a computer that can be viewed as a graph.

21.1 Finite State Machines

A *finite state machine* consists of a *finite control*, which we view as a set of *states* the machine can be in. The machine runs in steps. In each step, it receives an input signal. Upon its receipt, the machine produces an output signal and changes its state according to some rules.

One way of describing the functionality of a finite state machine is to draw a graph. Each state of the machine corresponds to one vertex of the graph. We draw an arrow pointing to the state in which the machine starts after being "powered on". We use edges to describe the *transitions* between states. For each transition, we list the inputs that cause it, and also the outputs that are produced when this transition is made. For example, in Figure 21.1, there is an edge from state 5 to state 15. The edge is labeled $10;0$, which means that this transition occurs when the machine receives the input 10 in state 5. The machine then goes to state 15 and outputs 0.

Example 21.1: Consider a vending machine. For simplicity, suppose it only sells one item, priced at 30 cents, and doesn't give any change. It accepts 5-cent, 10-cent, and 25-cent coins.

The states of the machine represent how much money has been put in since the machine first started, or since the last item disposal, whichever came last. The machine outputs 1 (i.e., dispenses an item) if it has received enough money to dispense an item, and outputs 0 (i.e., does not dispense an item) in all other cases. When it dispenses an item, it goes back to the state that indicates no money has been put in. All other state changes are shown in Figure 21.1.

⊠

Before we define a finite state machine more formally, let's see another example. This one brings us closer to the notion of a finite state automaton which we will be discussing for the rest of the lecture.

Example 21.2: Consider a machine that receives one of the letters A through Z in each step. It outputs 1 if the word NANO appears in the sequence of letters given to the machine.

We design the machine so that its states indicate how much of the word NANO it has recently seen. We label the states 0 through 4. State i indicates that the last i symbols received were the first i symbols from the word NANO.

If the machine is in state i for $i \in \{0,1,2,3\}$, it goes to state $i+1$ if the next letter given to it as input is the $(i+1)$st letter of the word NANO.

What if the machine receives an input that does not continue spelling the word NANO? Then the machine must go back to an earlier state, but not necessarily the starting state. For example, if the last three inputs received were NAN, the machine is in state 3. If the next input is A, the last two letters received are also the first two letters of the word NANO, so the machine goes to state 2.

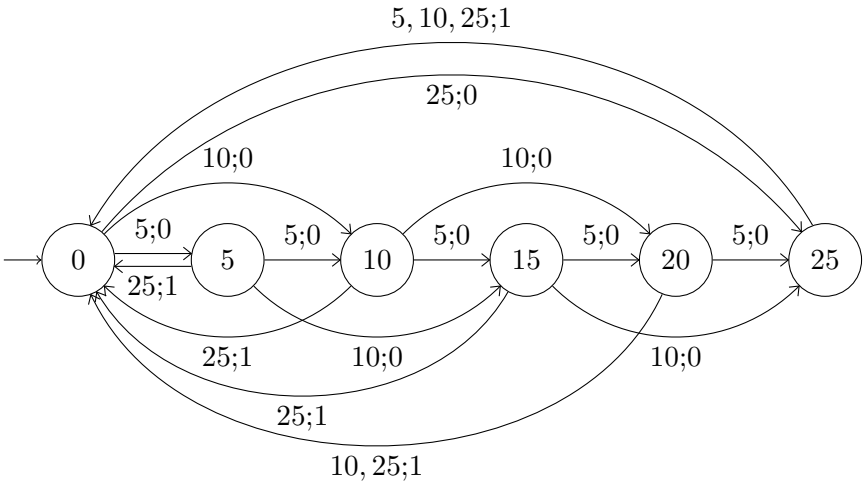

Figure 21.1: The transition diagram for the vending machine.

Going to any other state could cause the machine to produce incorrect output. For example, if the next two letters after A were NO, the machine would not recognize that the word NANO appeared in the input if it went to state 0 instead of state 2 after seeing A.

Finally, once the machine finds the word NANO, it stays in state 4 and keeps outputting 1 to indicate that it has seen the word. We show all the transitions in Figure 21.2.

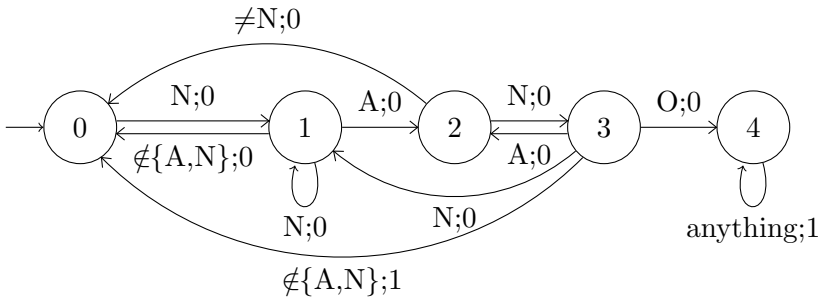

Figure 21.2: A finite state automaton that decides whether a sequence of inputs contains the word NANO in it.

⊠

21.2 Finite State Automata

The finite state machine from Example 21.2 has two special properties. First, there are only two output symbols, 0 and 1, which we can interpret as "no" and "yes" (or "reject" and "accept"), respectively. Second, the output symbol only depends on the state to which the machine transitions, and does not depend on the input. Any finite state machine that satisfies these two properties is called a *finite state automaton*.

When can simplify the drawing of a finite state automaton. Since there are only two outputs, and the output only depends on the destination of the transition, we don't show outputs on the edge labels, and, instead, draw the vertices representing states where the output is "yes" differently

than the vertices representing states where the output is "no". We use two circles for the former, and one circle for the latter. We show the result of making there changes to Figure 21.2 in Figure 21.3.

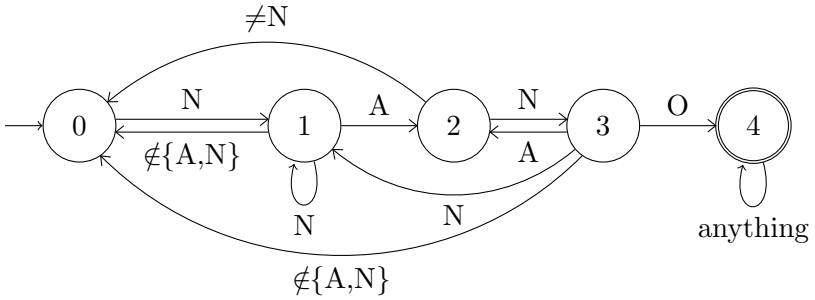

Figure 21.3: A finite state automaton that decides whether a sequence of inputs contains the word NANO in it.

21.2.1 Formal Definition

A finite state automaton receives inputs from some finite set of symbols. We call this finite set an *alphabet*. The alphabet for Example 21.1 was the set $\{5, 10, 25\}$, and the alphabet for Example 21.2 was the set $\{\text{A},\ldots,\text{Z}\}$.

Definition 21.1. *A* finite state automaton *is a* 5*-tuple* (S, Σ, ν, s_0, A)*, where*

- S *is a finite set of* states*.*
- Σ *is a finite set of symbols called the* alphabet*.*
- $\nu : S \times \Sigma \to S$ *is the* transition function*. The inputs to this function are the current state and the last input symbol. The function value* $\nu(s, x)$ *is the state the automaton goes to from state* s *after reading symbol* x*.*
- *The automaton starts in the* start state $s_0 \in S$*.*
- *The set of* accepting states A *indicates which states cause the automaton to output "yes".*

Example 21.3: We show what the five parts of the tuple $M_1 = (S, \Sigma, \nu, s_0, A)$ are for the finite state automaton in Figure 21.4.

- $S = \{s_0, s_1\}$.
- $\Sigma = \{0, 1\}$.
- $\nu(s_0, 0) = s_0$, $\nu(s_0, 1) = s_1$, $\nu(s_1, 0) = s_1$, $\nu(s_1, 1) = s_0$.
- The start state is s_0.
- $A = \{1\}$.

⊠

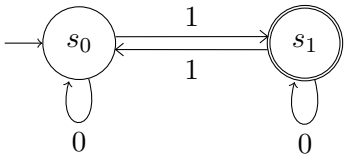

Figure 21.4: The automaton M_1 used in Example 21.3.

21.2.2 Strings and Languages

In each step, a finite state automaton processes some symbol from the alphabet. The input sequence could be infinite, thus causing the finite state automaton to run forever. For most computer programs, this is a situation we want to avoid, so let's focus on the case where the input sequences have finite length.

Definition 21.2. *A finite sequence of symbols from an alphabet Σ is called a* string *over Σ. An empty sequence of symbols is called an* empty string*, and is usually denoted by λ or ϵ. If x is a string over Σ, $|x|$ denotes the* length *of the string, that is, the length of the sequence of symbols x stands for.*

For example, the length of the empty string is $|\epsilon| = 0$. The length of the string "NANO" from Example 21.2 is 4.

We can further group strings into sets called languages.

Definition 21.3. *A* language *over alphabet Σ is a set of strings over Σ.*

Suppose you have a program in some text file. You can view this program as a string. Some strings represent valid programs, and some do not. It is the job of the compiler to distinguish between the former and the latter. For the former, it should say the program is valid, and for the latter, it should generate some error message. The set of all strings that represent valid programs is called a programming language. This is what motivates the definition above.

With every automaton, we associate the language of all strings on which it outputs "yes".

Definition 21.4. *Given a finite automaton M, the set*

$$L(M) = \{x \mid \text{when } M \text{ is run on the string } x \text{ starting from the start state, the final state is in } A\}$$

is called the language decided by M.

Using this definition, we could say that the language decided by the compiler is the language of valid programs in some programming language. However, we need a stronger computational model than a finite state automaton to implement a compiler.

Let's now look at a language that can be decided by a finite state automaton.

Example 21.4: Let's find the language decided by the finite state automaton M_1 from Example 21.3. We start by finding some short strings in $L(M_1)$.

The empty string ϵ is not in the language because M_1 on input ϵ enters the state s_0, and terminates immediately because there are no more input symbols to read. Thus, M_1 ends in a non-accepting state on this input.

The string 0 is also not in the language. The machine M_1 starts in state s_0, and $\nu(s_0, 0) = s_0$, so M_1 stays in s_0 after reading the first and only input symbol. It outputs "no" after that.

The string 1 is in the language because the transition M_1 makes from the start state after reading 1 is to state s_1, which is an accepting state. It outputs "yes" after that.

The strings 01 and 10 are in the language. The sequences of states after reading the input symbols one by one are s_0, s_0, s_1 and s_0, s_1, s_1, respectively. The strings 00 and 11 are not in the language.

The strings $001, 010, 100, 111$ are all in $L(M_1)$, and no other strings of length 3 are.

We observe that M_1 changes states if and only if the next input symbol is 1. Therefore, the first time it gets to the accepting state s_1 is after seeing the first 1. It leaves this state after seeing the next 1, comes back after seeing another 1 after that, and so on. In other words, M_1 changes states if and only if it reads a 1. Because the start state is $s_0 \neq s_1$, $M-1$ ends in state s_1 if and only if it changes states an odd number of times, that is, if and only if the input contains an odd number of ones.

We now turn the intuitive explanation from the previous paragraph in a formal proof. We can think of it as proving the equality of two sets. One set is $L(M_1)$, and the other set is the set of strings with an odd number of ones in them. We do not pursue this direction, and use invariants instead.

First we introduce some notation that extends the transition function ν. Let x be a string over Σ. Then $\nu(s, x)$ is the state of the automaton after reading all symbols in x, assuming the automaton was in state s before processing the first symbol in x. Notice that if x is a single symbol from the alphabet, this coincides with the definition of the transition function.

We have $\nu(s, \epsilon) = \epsilon$ for any state s. If x is not the empty string, it looks like ya where y is a string of length $|x| - 1$ and $a \in \Sigma$. After processing y, the automaton is in state $\nu(s, y)$. The next input symbol is a, and M_1 goes to state $\nu(\nu(s, y), a)$ after processing it. Thus, we get a recursive definition of the form $\nu(s, ya) = \nu(\nu(s, y), a)$.

Now we prove the following invariant by induction on the number of steps: "M_1 is in state s_1 after n steps if and only if the number of ones in the first n symbols of x is odd".

We saw earlier that $\nu(s_0, \epsilon) = s_0$. Since ϵ has an even number of ones in it, this proves the base case.

Now suppose our invariant holds after the first n steps, and consider the $(n+1)$st step. The first $n+1$ symbols in the input have the form ya where $|y| = n$ and $a \in \Sigma$. There are two cases to consider.

Case 1: M_1 is in state s_1 after processing y. This happens if and only if y has an odd number of ones in it by the induction hypothesis. Now if $a = 1$, the state changes to s_0, and the additional 1 makes the number of ones in the first $n+1$ symbols even. If $a = 0$, M_1 stays in state s_1, and the number of ones in the first $n+1$ symbols stays odd. Thus, the invariant holds after $n+1$ steps in this case.

Case 2: M_1 is in state s_0 after processing y. This happens if and only if y has an even number of ones in it by the induction hypothesis. Now if $a = 1$, the state changes to s_1, and the additional 1 makes the number of ones in the first $n+1$ symbols odd. If $a = 0$, M_1 stays in state s_0, and the number of ones in the first $n+1$ symbols stays even. Thus, the invariant holds after $n+1$ steps in this case too.

Then consider the situation after the entire input string x is processed, i.e., after $|x|$ steps. By the invariant we just proved, M_1 is in state s_1 if and only if the input string contains an odd number of ones. Since s_1 is the only accepting state, this implies that M_1 outputs "yes" on input x if and only if x contains an odd number of ones. ⊠

21.2.3 Finite State Automata as a Model of Computation

We can view a finite state automaton $M = (S, \Sigma, \nu, s_0, A)$ as a simple computer consisting of the following parts.

- A finite control that knows the the tuple M and stores the automaton's current state.
- A tape that has the input written on it.
- A tape head that is positioned above the tape and can read the symbol underneath it.

When the automaton starts, the finite control sets the current state to the start state s_0 and positions the tape head above the first symbol on the tape.

In each step, the automaton reads the symbol under its tape head. The finite control looks at that symbol and at the current state, changes states according to the transition function, and moves the tape head above the next symbol on the tape.

The finite state automaton looks like a very simple computing device. We show the setup in Figure 21.5.

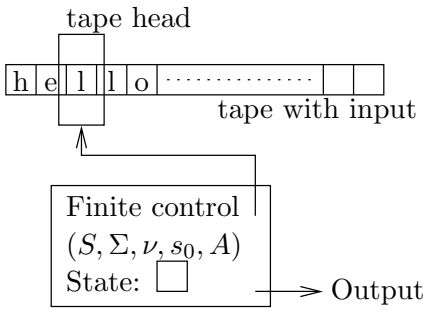

Figure 21.5: Components of a finite state automaton.

21.2.4 Designing Finite State Automata

We mentioned in Lecture 15 that one often uses invariants to design algorithms. This is also true when designing finite state automata for deciding a given language L. In particular, invariants can describe all situations in which an automaton is in a particular state. Let's demonstrate this technique on an example.

Example 21.5: We design a finite state automaton M that decides the language consisting of all strings over the alphabet $\{0, 1\}$ that start and end with the same symbol. For example, the strings 101 and 0111010 are in the language, and the strings 110 and 01010 are not in the language.

We create a start state s. We make it so that the only way for the computation to end in this state is if the input is the empty string. Let's agree that the empty string starts and ends with the same symbol, so s is an accepting state.

Our automaton M must know what the first symbol in the input was because otherwise it will have no way of telling whether the last symbol read so far is the same as the first symbol. Thus, we create states s_0 and s_1, which indicate that the first symbol in the string was 0 and 1, respectively.

Next, we need to decide on the logic for each of the two situations mentioned in the previous paragraph. There cannot be any transitions from s_0 to s_1 or from s_1 to s_0 because such transition would lose information about the first symbol in the input.

Let's focus on the situation when the first symbol was zero first. Suppose M is in state s_0 before it reads the last symbol in the input. If this symbol is 0, M should transition to some accepting state, and should transition to a non-accepting state otherwise. We could use the state s_0 as that accepting state. This state must be accepting anyway because the string 0 starts and ends with zero, and M ends in state s_0 after processing this string. So we make s_0 an accepting state and add a transition from state s_0 to state s_0 on input 0. Now on input 1 in state s_0, M needs to transition away from s_0 to some non-accepting state because if this 1 were the last input in the string, staying in s_0 would cause M to accept incorrectly. So we add a state s_{01}. This state is reached if the first symbol in the string was 0, and the last symbol read was 1. We also redefine the meaning of the state s_0 to "the first and the last symbol read were 0". To complete this part of the picture, there is a transition from s_{01} to s_0 on input 0, and from s_{01} to s_{01} on input 1.

Likewise, we make the state s_1 accepting and redefine its meaning to "the first and the last symbol read were both 1". We also add the state s_{10} which indicates that the first symbol in the string was a 1 and the last symbol read was a zero. We add transitions similar to the case when the first input symbol was zero.

We show M in Figure 21.6. ⊠

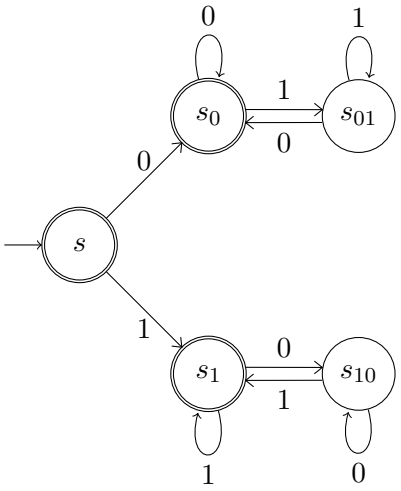

Figure 21.6: The automaton M for the language over $\{0, 1\}$ consisting of strings that start and end with the same symbol.

Exercise 21.1: For next time, think about automata that accept binary representations of multiples of k for various values of k.