Lecture 3: DFA and Regular Languages

built on 2021/01/11 at 08:45:28

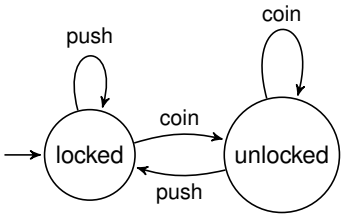
The study of computation theory often begins with the "big" question: *what is a computer?* Though everyone seems to have an idea of what a computer is, building a precise mathematical model that describes it seems to be an unsurmountable task. Modern real-world computers are simply too complicated. To make our lives more manageable, we often end up with *computational models*, which may be accurate in some ways—and not so in many others.

Today we're going to explore one of the simplest computational models—the finite-state machine. This is a puny machine, with so little capability, but as you will see, it can do surprisingly many useful things.

1 Turnstile Logic

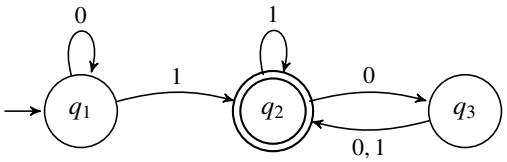
Often used to control access to subways, a turnstile is a gate with rotating arms around waist height. Those who have seen it know they are in one of the two states: locked or unlocked. It's often initially locked to prevent entry. But once you put in a token/coin, it is unlocked, allowing entry. It then goes back to being locked after a person has pushed through. From the perspective of the turnstile, there are only two actions a customer can do: give it a token/coin or push through.

How would you model such a gadget?



2 A Simple Finite Automaton

The diagram below depicts a finite automaton called M_1:



What you're seeing is the *state diagram* of M_1. It has 3 states, denoted by circles; they are q_1, q_2, and q_3. There is a special state called the *starting state*, which can be identified by the arrow pointing into it from nowhere. An *accepting state* is marked by a double circle. The arrows connecting states denote *transitions*.

To see how this works, consider what happens when the automaton M_1 receives an input string 1101. It consumes the symbol one by one and at the end, produces an output: either *accept* or *reject*—pretty much, a *yes* vs. a *no*. The processing works as follows:

- Start in state q_1 as the processing begins in the starting state;
- Receive 1, so it follows the transition arrow from q_1 to q_2;
- Receive 1, so it follows the transition arrow from q_2 to q_2;
- Receive 0, so it follows the transition arrow from q_2 to q_3;
- Receive 1, so it follows the transition arrow from q_3 to q_2; and
- Output *accept* because M_1 is in an accepting state at the end of the input.

Notice, however, that if the input were 110, the machine M_1 would reject it as the final state isn't an accepting state; it will be in q_3 at the end of the input.

You can experiment with this machine a bit more. It accepts, for example, $1, 01, 11$. It rejects $0, 10$, for example. How can we describe all the strings that M_1 accept?

2.1 Formality

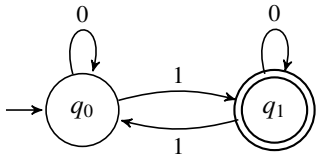
Let's now define finite automata more formally. A finite automaton, as we just saw, has several "moving parts." It is made up of states and rules for going from one state to another that depend on the symbol being fed to the machine. These symbols too come from an alphabet that the machine works on. It has a starting state—and also accepting states. These elements are combined into a 5-tuple as follows:

Definition 2.1 (Deterministic Finite Automaton) *A* deterministic finite automaton *(DFA) is a* 5*-tuple* $(Q, \Sigma, \delta, q_0, F)$*, where*

1. Q *is a finite set called the* states*;*
2. Σ *is the alphabet set;*
3. $\delta : Q \times \Sigma \to Q$ *is the* transition function*;*
4. $q_0 \in Q$ *is the* starting state*; and*
5. $F \subseteq Q$ *is the set of* accepting states*.*

The *transition function* δ deserves more explanation. Mechanically, the function takes two arguments: a state and a symbol. Indeed, it takes a state and a symbol, and returns another state. In words, the transition function encodes the rules of moving between states: if the machine is at state q and receives a symbol x, then the transition function δ tells the machine to transition to $\delta(q, x)$ next.

Example: As a concrete example, consider the following DFA:



It is formally defined as the following machine: Let $M = (Q, \Sigma, q_0, F)$ where

- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \to Q$ is specified as follows:

δ	0	1
q_0	q_0	q_1
q_1	q_1	q_0

- $q_0 = q_0$
- $F = \{q_1\}$.

How do these pieces work together? The only thing a DFA maintains is which state it is at. At the beginning, it starts out in state q_0. For each symbol that arrives, if it is in state q and receives x, then it moves to state $q' = \delta(q, x)$. Once the string is completely consumed, the DFA accepts it if the current state is one of the accepting states—and rejects this string otherwise.

More formally, consider a string $w = w_1 w_2 w_3 \ldots w_n \in \Sigma^*$. This is a string of length n and each w_i is a symbol in the alphabet Σ. A DFA M *accepts* w, or M *recognize* w, if and only if the following code returns true:

- Let $r_0 = q_0$
- For $i = 0, 1, \ldots, n-1$: $r_{i+1} = \delta(r_i, w_{i+1})$
- Return whether $r_n \in F$.

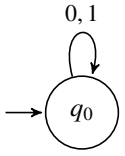
Further Observations: We could generalize the transition function to accept a string instead of one character at a time, hence defining $\delta^*(q, w)$ to return the state this machine will be in once it consumes w starting from the state q. Several things follow directly:

- A machine accepts the string w if and only if $\delta^*(q_0, w) \in F$.
- If $x = yz$ (i.e., y concatenated with z), then $\delta^*(q, x) = \delta^*(\delta^*(q, y), z)$. Among other things, this shows that a DFA has no memory beyond where it currently is. So, if there are two strings r and s that lead to the same DFA state (i.e., $\delta^*(q_0, r) = \delta^*(q_0, s)$), you know with certainty that if you give both instances the same string t after that, they'll end up in the same state: $\delta^*(q_0, rt) = \delta^*(q_0, st)$. This could be useful in arguing about what a DFA must look like in order for it to function correctly.

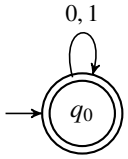
Definition 2.2 *If A is the set of all strings that machine M accepts, we say that A is the* language of machine *M and write $L(M) = A$. We also say M* recognizes *A, or M* accepts *A to mean the same thing. Notice that a machine may accept many strings, but it always recognizes one language.*

Challenge: For the machine in the example just above, can you prove that $L(M) = \{w : w \text{ has an odd number of 1s}\}$?.

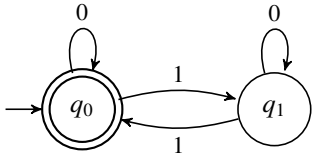
2.2 What do these recognize?

M_1: 

A: Nothing

M_2: 

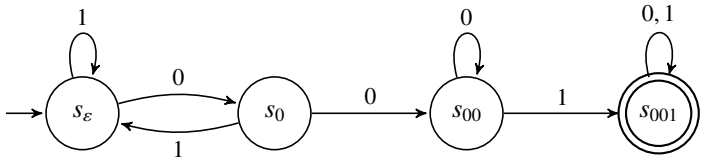
A: All binary strings

M_3: 

A: All binary strings that have an even number of 1s.

2.3 Short Exercise

Can you build a DFA that accepts all and only those strings that contain 001 (consecutively)?



Meaning of States. Most often, each state has a describable meaning. Figuring this out helps design the DFA and ensure that the DFA works correctly as intended. As an example, in the above DFA, we can say that

- s_ε represents the state where there isn't a match with the pattern.
- s_0 represents the state where we've matched with `0`.
- s_{00} represents the state where we've matched with `00`. Notice that it makes sense that seeing a 0 would transition us from s_0 (matching `0`) to s_{00} (matching 00). But if a 1 is instead seen at s_0, we have no match and hence are sent to s_ε.
- s_{001} represents the state where we've matched with `001`. This is also why it's an accepting state.

3 Regular Languages

Definition 3.1 *A language is called a* regular language *if some DFA recognizes it.*

Therefore, to prove that a language is regular, we just have to show a DFA that recognizes it, proving formally that the DFA does indeed recognize this language.

3.1 Union/Interesection

Theorem 3.2 (Union Theorem) *Given two languages L_1 and L_2, define the union of L_1 and L_2 as*

$$L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}.$$

If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ is also a regular language.

Let's brainstorm a little bit. To prove that $L_1 \cup L_2$ is regular, we just have to find a DFA that recognizes it. Now we know that there are DFAs for L_1 and L_2 because they are regular. Say they are M_1 and M_2. Can we find a way to create a new DFA from the DFAs for L_1 and L_2?

The basic idea is to run M_1 and M_2 in parallel. The new machine will accept if and only if M_1 or M_2 accepts. *Proof:* Let $M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ be a DFA for L_1 and $M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ be a DFA for L_2. We'll construct a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $L = L_1 \cup L_2$. To simulate running M_1 and M_2 in parallel, our new states will be made up all possible combinations that M_1 and M_2 can be, so let

$$\begin{aligned} Q &= Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \\ q_0 &= (q_0^1, q_0^2). \end{aligned}$$

If the simulated M_1 is at q_1, upon seeing a symbol σ, it will move to $\delta_1(q_1, \sigma)$. Similarly, if the simulated M_2 is at q_2, upon seeing a symbol σ, it will move to $\delta_2(q_2, \sigma)$. Hence, the new transition function should look as follows:

$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)).$$

For M to accept if and only if M_1 or M_2 accepts, we use

$$F = \{(q_1, q_2) \mid q_1 \in F_1 \vee q_2 \in F_2\}.$$

■

Theorem 3.3 (Intersection Theorem) *Given two languages L_1 and L_2, define the intersection of L_1 and L_2 as*

$$L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}.$$

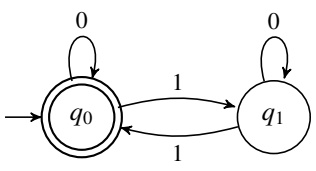
If L_1 and L_2 are regular languages, then $L_1 \cap L_2$ is also a regular language.

Exercise: Prove this theorem.

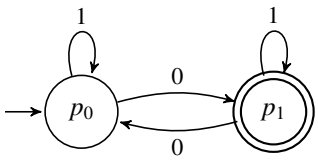
Union/Interesection In Action: Can you come up with the following DFAs:

- $L(M_1)$ consists of all binary strings that have an even number of 1s.
- $L(M_2)$ consists of all binary strings that have an odd number of 0s.

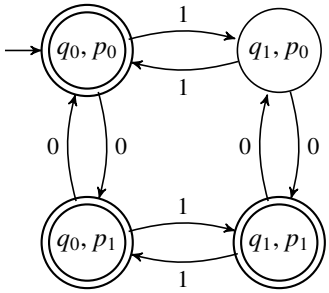
Here's M_1:



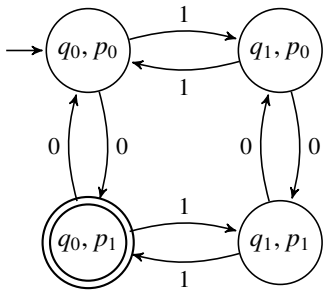
and here's M_2:



Union: The resulting DFA is supposed to accept a string with either an even number of 1s or an odd number of 0s (or both).



Intersection: The resulting DFA is supposed accept a string that has an even number of 1s and an odd number of 0s.



4 Practice Makes Perfect

1. Consider $\Sigma = \{1, 2, 3\}$. Let $L = \{x \in \Sigma^* \mid 1, 2, \text{ and } 3 \text{ appear in } x \text{ in that order but no necessarily consecutively}\}$. Prove that L is regular.
2. Prove that the language $L_2 = \{x \in \{0, 1\}^* \mid x \text{ starts and ends with the same bit}\}$ is regular by giving a DFA.