1 Finite Automata (1.1, Continued)

This continues from the notes from Monday, January 3.

1.1 Formal Definition of Computation

To review, let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each $w_i \in \Sigma$. Then, we say that M accepts w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$: The machine starts in the start state.
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$: The machine goes from state to state according to the transition function.
3. $r_n \in F$: The machine accepts its input if it ends up in an accept state.

In particular, we say that M recognizes language A if $A = \{w \mid M \text{ accepts } w\}$.

Definition 1.1: Regular Language

A language is called a **regular language** if some finite automaton recognizes it.

1.1.1 Example 1: State Machine

Recall, for example, our state machine B_3 in the previous lecture notes. If w was the string:

```
10 RESET 22 RESET 012
```

Then, M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is:

$$q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$$

In particular:

1. The machine starts in the start state as expected.
2. The machine goes from state to state as expected.
3. The machine ends at the accept state.

1.2 Designing Finite Automata

A helpful approach when designing various types of automata is:

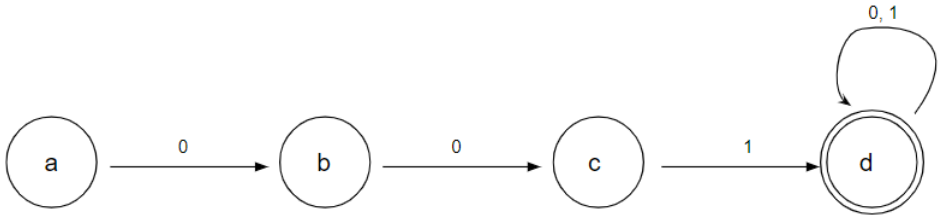
> *Put yourself in the place of the machine you are trying to design and then see how you would go about performing the machine's task.*

Suppose you are given some language and want to design a finite automaton that recognizes it. Given some input string, your goal is to determine if it is a member of the language the automaton is supposed to recognize. However, you can only see each symbol one at a time; after each symbol, you need to decide whether the string seen is in the language. The hardest part is that you need to figure out what you need to remember about the string as you are reading it. Remember: you only have a finite number of states, which means finite memory (hence, *finite* automata).

1.2.1 Example 1: Designing a Finite Automaton

Given $\Sigma = \{0, 1\}$, suppose we need to create a finite automaton E_2 that recognizes the regular language of all strings that contain 001 as a substring. For example, 001, 1001, 0010, 111111001111101 are all in the language; however, 0000 and 11 are not.

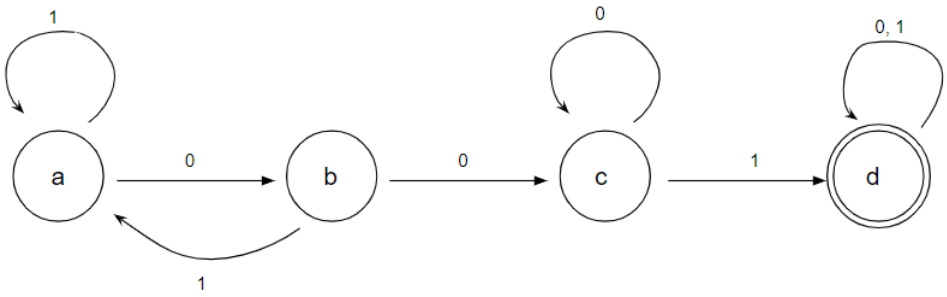
Well, the first thing we can do is create the set of states that will result in an ACCEPT state. This is as simple as:



Here, it's clear that a string like 001 will result in an ACCEPT state. Now, we need to account for any other strings. In particular, we need to account for several different possibilities:

- We haven't seen any symbols associated with the pattern (e.g. we start with 1s, or we saw a 0 and then a 1).
- We just saw 0.
- We just saw 00.
- We have seen the pattern 001.

This gives us the automaton:



1.3 The Regular Operations

In arithmetic, the basic objects are numbers and the tools are operations for manipulating them (e.g. + or ×). In the theory of computation, the objects are languages and the tools include operations designed for manipulating them. We call these **regular operations**.

Definition 1.2

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- **Union:** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation:** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- **Star:** $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Remarks:

- The union operation simply takes all strings in both A and B and puts them together into one language.
- The concatenation operation attaches a string from A in front of a string from B in *all possible ways to get the strings in the new language.*
- The star operation attaches any number of strings in A together to get a string in the new language. Note that *any number* includes 0, so the empty string ϵ is always in A^*.

1.3.1 Example 1: String Manipulation

Suppose $\Sigma = \{\texttt{a}, \texttt{b}, \dots, \texttt{z}\}$ is the standard 26 letters. Define the two languages to be:

$$A = \{\texttt{good}, \texttt{bad}\}$$

$$B = \{\texttt{boy}, \texttt{girl}\}$$

Then:

- $A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\}$
- $A \circ B = \{\texttt{goodboy}, \texttt{goodgirl}, \texttt{badboy}, \texttt{badgirl}\}$
- $A^* = \{\epsilon, \texttt{good}, \texttt{bad}, \texttt{goodgood}, \texttt{goodbad}, \texttt{badgood}, \texttt{badbad}, \texttt{goodgoodgood}, \dots\}$

1.4 Properties of Regular Operations

Theorem 1.1

The class of regular languages is closed under the union operation.

Remark: In other words, if A_1 and A_2 are regular language, so is $A_1 \cup A_2$.

Proof. The proof is left for another day. □

Theorem 1.2

The class of regular languages is closed under the concatenation operation.

Remark: In other words, if A_1 and A_2 are regular language, then so is $A_1 \circ A_2$.

Proof. The proof is left for another day. □

1.5 Justifying DFAs

To prove that the DFA that we build, M, actually recognizes the language L, we ask the following questions:

1. Is every string accepted by M in L?
2. Is every string from L accepted by M?[1]

A string is accepted by a DFA when:

$$L(M) = \{w \mid \delta^*(q_0, w) \in F\}$$

Where δ^* is defined by:

$$\delta^*(q, w) = \begin{cases} q & w = \epsilon \\ \delta(q, c) & w = c, c \in \Sigma \\ \delta^*(\delta(q, c), w') & w \subset w', c \in \Sigma, w' \in \Sigma^* \end{cases}$$

[1] The contrapositive version is: Is every string rejected by M not in L?