Deterministic Finite Automata

30 January 2020
Vote for office hours:

https://forms.gle/qnHNpNpaKov9eJYVz8
Assignment 1

Posted today
Due on Tuesday
Corrections due a week from today
Previously:

- Central questions of the course
- Finite automata as a model of computation

Today:

- Formal definition of a *deterministic finite automaton* (DFA)
- Practice designing DFAs to recognize languages
- Discuss the regular languages
Defining DFAs
A small problem
A small problem
A small problem

\[
\begin{array}{c}
\text{start} \\
q_0 \\
q_2 \\
q_1
\end{array}
\]

0 1 1 0
A small problem
A small problem

\begin{center}
\begin{tikzpicture}
\node[circle, draw, fill=green!30] (q_0) at (0,0) {$q_0$};
\node[circle, draw] (q_1) at (1,0) {$q_1$};
\node[circle, draw] (q_2) at (0,-1) {$q_2$};
\draw[->] (q_0) -- ++(-90:1cm) node[below] {$0$} -- (q_2);
\draw[->] (q_0) -- ++(90:1cm) node[below] {$0$} -- (q_1);
\draw[->] (q_1) -- ++(180:1cm) node[below] {$1$} -- (q_2);
\draw[->] (q_0) -- (q_1) node[above] {$0$};
\draw[->] (q_2) -- (q_1) node[below] {$1$};
\node[anchor=east] at (-1.5,0) {$start$};
\end{tikzpicture}
\end{center}

"0 1 1 0"
A small problem
A small problem

\[ \begin{array}{c}
q_2 \\
\rightarrow 0 \\
\rightarrow 1 \\
\rightarrow 0 \\
\rightarrow q_1
\end{array} \]

\[ \begin{array}{c}
q_0 \rightarrow 0 \\
\rightarrow 0 \\
\rightarrow 1 \\
\rightarrow q_1
\end{array} \]
A small problem

![Diagram of a finite automaton with states q0, q1, q2 and transitions for inputs 0 and 1. States q0, q1 and q2 are connected with transitions labeled 0 and 1. The diagram also shows a transition from start to q0.]

0 1 1 0


A small problem

```
0 1 1 0
```

```
\begin{tikzpicture}
\node[state] (q0) at (0,0) {$q_0$};
\node[state,fill=green] (q2) at (-1,1) {$q_2$};
\node[state,fill=blue] (q1) at (1,1) {$q_1$};
\draw[->] (q0) edge [loop above] node {$0$} (q0);
\draw[->] (q0) edge [loop below] node {$0$} (q0);
\draw[->] (q0) edge [bend right] node {$1$} (q1);
\draw[->] (q2) edge [bend left] node {$0$} (q0);
\draw[->] (q2) edge [bend right] node {$1$} (q1);
\draw[->] (q1) edge [bend right] node {$1$} (q2);
\end{tikzpicture}
```
A small problem

I KNEW EXACTLY WHAT TO DO

BUT IN A MORE REAL SENSE, I HAD NO IDEA WHAT TO DO
The need for formalism

In order to reason about the limits of what finite automata can and cannot do, we need to formally specify their behavior in all cases.

What happens if there is no transition out of a state on some input?

What is there are multiple transitions out of a state on some input?
A *deterministic finite automaton* (DFA) is defined relative to some alphabet $\Sigma$.

For each state in the DFA, there must be *exactly one* transition defined for each symbol in $\Sigma$.

This is the “deterministic” part!

There is a unique start state.

There are zero or more accepting states.
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over $\{0, 1\}$?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \(\{0, 1\}\)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \( \{0, 1\}\)?
Formal definition of a deterministic finite automaton (DFA)

A DFA is represented as a five-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$: Finite set of states.
- $\Sigma$: Finite set of input symbols (the alphabet).
- $\delta$: A transition function $Q \times \Sigma \rightarrow Q$.
- $q_0 \in Q$: One state is the start state (or initial state).
- $F \subseteq Q$: Set of zero or more final states (or accepting states).
Transition function $\delta$

Takes two arguments: a state and an input symbol.

$\delta(q, a)$ = the state the DFA goes to when it is in state $q$ and reads input symbol $a$.

Since $\delta$ is a total function; there is always a next state.
If there’s no transition you want, you must add a “dead state”.
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$

2. $\delta(r_i, w_{i+1}) = r_{i+1}$
   for $i = 0, \ldots, n-1$

3. $r_n \in F$
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$. $M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$  
   (*it begins at the start state*)

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$

3. $r_n \in F$
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$ \hspace{1cm} \text{it begins at the start state}

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ \hspace{1cm} \text{each transition in the sequence is allowed by the transition function for the corresponding input symbol}

for $i = 0, \ldots, n-1$

3. $r_n \in F$
Formal description of how finite automata compute

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w$ be a string, $w_1w_2\ldots w_n$, where each $w_i \in \Sigma$.

$M$ accepts $w$ if a sequence of states $r_0, r_1, \ldots, r_n$ exists where each $r_i \in Q$ and

1. $r_0 = q_0$  
   \textit{it begins at the start state}

2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$  
   \textit{each transition in the sequence is allowed by the transition function for the corresponding input symbol}

3. $r_n \in F$  
   \textit{it ends in an accept state}
Designing DFAs
Example

Consider the language

$$L = \{ w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring} \}$$

How can we design a DFA to recognize $L$?
\( L = \{w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring}\} \)
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring} \} \]
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring}\} \]
\[ L = \{w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring}\} \]
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring}\} \]
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring} \} \]
$L = \{w \in \{a, b\}^* \mid w \text{ contains } bb \text{ as a substring}\}$
Example

Consider the language

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \]

We’re using the symbol \( a \) as a placeholder for any character that isn’t a star or slash (including spaces) to keep things simple.
Just like when you’re programming, it helps to come up with a set of test cases you want to accept and reject.

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \]

Accept

/*a*/
//**/  
/****/  
/*aaa*aaa*/  
/*a/a*/

Reject

/***
//***a/**aa*/
aaa//**/aa
//**
//***a
//aaaa
\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment}\} \]
Design strategy for DFAs

At each point in its execution, the DFA can only remember what state it’s in.

Therefore, build each state to correspond to some piece of information that you need to remember.

Each state acts as an indicator of what you’ve already seen, sufficient to let you decide what to do next.

There can only be finitely many states, so the DFA can only remember finitely many things.
Creating a finite automaton

If you think of the states as “memory”, then the memory of a finite automaton is limited to the number of states.

The consequence of a finite number of states is, if $|w| >$ number of states, some state must be repeated in the execution of the FA over $w$. 
If the machine can’t remember all the symbols it has seen so far in an input string, it has to change state based on other information, e.g.,

\[ L_1 = \text{the set of all strings with an odd number of 1s over the alphabet } \{0, 1\} \]

Don’t need to remember exactly how many 1s have been seen – just whether we’ve read an even or odd number.
Creating a finite automaton

Start by putting yourself in the place of the FA that has to make every transition choice based on a single character because it can’t look ahead or rewind.

1. Define the meaning of the states
2. Determine the transition function
3. Label the start and final states
4. Test your FA on example inputs
Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser