Deterministic
Finite Automata

Lecture 3
10 September 2019
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

Soon:

Assignment 1
Office hours
Defining DFAs
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 \( \{0, 1\} \)?
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).
- \(\delta\): \(Q \times \Sigma \rightarrow Q\): A transition function.
- \(q_0 \in Q\): 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. $r_0 = q_0$  \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}
   \hspace{0.5cm} \text{for } i = 0, \ldots, n-1
3. $r_n \in F$  \hspace{1cm} \text{it ends in an accept state}
Designing DFAs
Example

Consider the language

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

How can we design a DFA to recognize \( L \)?
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 11 as a substring} \} \]
\( L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \)
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/aaa*/

Reject
- /**
- /**/a/*aaa*/
- aaa/**/aa
- /*
- /**a
- /**aa
- //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 $|w| > \text{number of states}$, 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
Problem

Build an automaton to recognize the set of strings that end with “ing”.
Problem

Build an automaton to recognize the set of strings that start and end with the same symbol.
Formal DFAs revisited
Tabular DFAs

Another way we can write down a DFA is as a transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>
Tabular DFAs

Another way we can write down a DFA is as a transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>→q₀</td>
<td>→q₁</td>
</tr>
<tr>
<td>q₀</td>
<td>q₀</td>
<td>q₁</td>
</tr>
<tr>
<td>q₁</td>
<td>q₀</td>
<td>q₂</td>
</tr>
<tr>
<td>*</td>
<td>q₂</td>
<td>q₂</td>
</tr>
</tbody>
</table>

By marking the start state with → and accepting states with *, the transition table that defines δ also specifies the entire DFA.
Tabular DFAs illustrate suggest how easy it is to implement a DFA in software.

```python
transition_table = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q0", "1": "q2"},
    "q2": {"0": "q2", "1": "q2"}
}

accept_states = ["q2"]

def run_dfa(word):
    state = "q0"
    for char in word:
        state = transition_table[state][char]
    return state in accept_states
```
Extension of $\delta$ to paths

Intuitively, a finite automaton accepts a string $w = a_1a_2...a_n$ iff there is a path in the transition diagram that:

1. Begins at the start state
2. Ends at an accepting state
3. Has sequence of labels $a_1, a_2, ..., a_n$. 
Formally, we extend transition function $\delta$ to $\hat{\delta}(q, w)$ where $w$ can be any string of input symbols:

**Basis:** $\hat{\delta}(q, \varepsilon) = q$

I.e., on no input, the FA doesn’t go anywhere

**Induction:** $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$, where $w$ is a string and $a$ is a single symbol.

I.e., see where the FA goes on $w$, then look for the transition on the last symbol from that state.

$\hat{\delta}$ represents paths.

That is, if $w = a_1a_2…a_n$ and $\delta(p_i, a_i) = p_{i+1}$ for all $i = 0, 1, …, n-1$, then $\hat{\delta}(p_0, w) = p_n$. 
Acceptance of strings

A finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ accepts string $w$ if
\[ \hat{\delta}(q_0, w) \in F \]

Language of a finite automaton

A finite automaton $M$ recognizes the language
\[ L(M) = \{ w \mid \hat{\delta}(q_0, w) \in F \} \]
Aside: type errors

A major source of confusion when dealing with automata – or mathematics in general – is making “type errors”.

**Example:** Don't confuse $M$, a DFA, i.e., a program, with $L(M)$, which is of type “set of strings”.

**Example:** The start state $q_0$ is of type “state”, but the accepting states $F$ is of type “set of states”.
Regular languages
Regular languages

DEFINITION. A language $L$ is called a regular language iff there exists a DFA $D$ such that $L(D) = L$, i.e., there is a DFA that recognizes it.

If $L$ is a language and $L(D) = L$, we say that $D$ recognizes the language $L$. 

The complement of a language

Given a language $L \subseteq \Sigma^*$, the complement of that language (denoted $\bar{L}$) is the language of all strings in $\Sigma^*$ that aren’t in $L$.

Formally:

$$\bar{L} = \Sigma^* - L$$
Complementing regular languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

\[ \bar{L} = \{ w \in \{0, 1\}^* \mid w \text{ does not contain } 11 \text{ as a substring} \} \]
\[ L = \{w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment}\} \]
\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Closure properties

THEOREM. If $L$ is a regular language, then $\overline{L}$ is also a regular language.

As a result, we say that the regular languages are closed under complementation.

Are the nonregular languages closed under complementation? Why or why not?
In addition to complement, the regular languages are closed under the regular operations:

Union (\(\cup\)):

\[ A \cup B = \{w \mid w \in A \text{ or } w \in B\} \]

Concatenation (\(\circ\)):

\[ A \circ B = AB = \{xy \mid x \in A, y \in B\} \]

Kleene star (\(\ast\)):

\[ A^* = \{w \mid w = x_1x_2\ldots x_k, \ k \geq 0, \ x_i \in A\} \]
Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser