Finite State Machines

No warm up today.
Announcements

Final will go from Saturday Dec. 12 at noon to Thursday Dec. 17 at noon.

Practice materials will go up over the weekend (if not earlier).
For now, final homeworks from Spring 20 and Winter 20 are on their respective webpages.

Collaboration rules same as the midterm.

Did I miss an email from you? Please send another!
Last Two Weeks

What computers can and can’t do... Given any finite amount of time.

We’ll start with the simplest model of computer – finite state machines.
Our machine is going to get a string as input. It will read one character at a time and update “its state.” At every step, the machine thinks of itself as in one of the (finite number) vertices. When it reads the character it follows the arrow labeled with that character to its next state.

Start at the “start state” (unlabeled incoming arrow). After you’ve read the last character, accept the string if and only if you’re in a “final state” (double circle).
Let’s see an example

Input string:

```
011
```

reject this string
"this is not in the language of the machine"

```
1010
```
Let’s see an example

Input string:

011

1010
Let’s see an example

Input string:

011

1010
Let’s see an example

Input string:

011

1010

Start

Zero

One
Let’s see an example

Input string:

011

1010
Let’s see an example

Input string:

011

1010

accept this string

```
start
```

```
zero
```

```
one
```
Deterministic Finite Automata

Some more requirements:

Every machine is defined with respect to an alphabet $\Sigma$
Every state has exactly one outgoing edge for every character in $\Sigma$.

There is exactly one start state; can have as many accept states (aka final states) as you want – including none.
Deterministic Finite Automata

Can also represent transitions with a table.

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Deterministic Finite Automata

What is the language of this DFA?
I.e. the set of all strings it accepts?

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Deterministic Finite Automata

- If the string has 111, then you’ll end up in $s_3$ and never leave.
- If you end with a 0 you’re back in $s_0$ which also accepts.
- And... $\varepsilon$

$[(0 \cup 1)^* 111(0 \cup 1)^*] \cup [(0 \cup 1)^*]$

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Design some DFAs

Let $\Sigma = \{0,1,2\}$

$M_1$ should recognize “strings with an even number of 2’s.”

What do you need to remember?

$M_2$ should recognize “strings where the sum of the digits is congruent to 0 ($mod$ 3).”

Need to remember: sum so far $\%$ 3
Let $\Sigma = \{0,1,2\}$

$M_1$ should recognize “strings with an even number of 2’s.”

$M_2$ should recognize “strings where the sum of the digits is congruent to 0 (mod 3)”
Designing DFAs notes

DFAs can’t “count arbitrarily high”

For example, we could not make a DFA that remembers the overall sum of all the digits (not taken % 3) and then

That would have infinitely many states! We’re only allowed a finite number.
Strings over \(\{0,1,2\}\) w/ even number of 2’s and \(\text{sum}\%3 = 0\)

\[ S = \{s_0, s_1, s_2\} \]
\[ T = \{t_0, t_1, t_2\} \]
\[ S \times T \]

"Cross product construction!"
Strings over \(\{0,1,2\} \) w/ even number of 2's and sum%3=0
Strings over \(\{0,1,2\}\) with even number of 2’s and sum \(\% 3 = 0\)
Strings over \( \{0,1,2\} \) with even number of 2’s and \( \text{sum} \% 3 = 0 \)

Changed notation – final states with bold outlines.
Strings over \(\{0,1,2\}\) w/ even number of 2’s \textbf{OR} sum\%3=0

Change the and to or – don’t need to change states or transitions...
Strings over \{0,1,2\} w/ even number of 2’s OR \text{sum}\%3=0

Change the and to or – don’t need to change states or transitions...
Just which states are accept.
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the start
The set of binary strings with a 1 in the 3rd position from the start
The set of binary strings with a 1 in the 3rd position from the end

What do we need to remember?

We can’t know what string was third from the end until we have read the last character.

So we’ll need to keep track of “the character that was 3 ago” in case this was the end of the string.

But if it’s not...we’ll need the character 2 ago, to update what the character 3 ago becomes. Same with the last character.
3 bit shift register  “Remember the last three bits”
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the end
The set of binary strings with a 1 in the 3rd position from the end
The beginning versus the end
From the beginning was “easier” than “from the end”

At least in the sense that we needed fewer states.

That might be surprising since a java program wouldn’t be much different for those two.

Not being able to access the full input at once limits your abilities somewhat and makes some jobs harder than others.
What language does this machine recognize?
What language does this machine recognize?

#1s even

#1s odd
What language does this machine recognize?

#0s even

#0s odd
What language does this machine recognize?

#0s is congruent to #1s (mod 2)

Wait...there’s an easier way to describe that....
What language does this machine recognize?

That’s all binary strings of even length.
Takeaways

The first DFA might not be the simplest.

Try to think of other descriptions – you might realize you can keep track of fewer things than you thought.

Boy...it’d be nice if we could know that we have the smallest possible DFA for a given language...
DFA Minimization

We can know!

Fun fact: there is a unique minimum DFA for every language (up to renaming the states)

High level idea – final states and non-final states must be different.
Otherwise, hope that states can be the same, and iteratively separate when they have to go to different spots.

In a normal quarter, we’d cover it in detail. But...we ran out of time.
Optional slides – won’t be required in HW or final but you might find it useful/interesting for your own learning.
Next Time

Some (historic and modern) applications of DFAs
There are some languages DFAs can’t recognize (say, $\{0^k 1^k | k \geq 0\}$)
What if we give the DFAs a little more power...try to get them to do more things.