Lecture 22: Finite state machines

\[ G: S \rightarrow SS | (S) | \varepsilon \]

- \( \forall S, \text{ if } S \text{ has matched } ( ) \Rightarrow G \text{ produces } S \).
- \( p(n) = \forall S, \text{ let } \lambda(S) = n, \text{ if } S \text{ has } ( ) \Rightarrow G \text{ produces } S \).  \\

Base Case: \( p(0) \), \( S \in \Sigma \Rightarrow S \in \Sigma \).

IH: For some \( n > 0 \) and any \( 0 \leq j \leq n \), \( p(j) \) holds.

JS: Goal \( p(n+1) \). Fix \( T \) with matched \( ( ) \).

\[ S \rightarrow SS \rightarrow (S)S \]
review: finite state machines

- States
- Transitions on inputs
- Start state and final states
- The language recognized by a machine is the set of strings that reach a final state

<table>
<thead>
<tr>
<th>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>
applications of FSMs (aka finite automata)

- Implementation of regular expression matching in programs like `grep`
- Control structures for sequential logic in digital circuits
- Algorithms for communication and cache-coherence protocols
  - Each agent runs its own FSM
- Design specifications for reactive systems
  - Components are communicating FSMs
applications of FSMs (aka finite automata)

- Formal verification of systems
  - Is an unsafe state reachable?
- Computer games
  - FSMs provide worlds to explore
  - Character AI
- Minimization algorithms for FSMs can be extended to more general models used in
  - Text prediction
  - Speech recognition
waka waka
Timeout after two maximum segment lifetimes (2*MSL)
what language does this machine recognize?
can we recognize these languages with DFAs?

- $\varepsilon$
- $\emptyset$
- $\sum^*$
- $\{ x \in \{0,1\}^*: \text{len}(x) > 1 \}$
strings over \{0, 1, 2\}^*

$M_1$: Strings with an even number of 2’s

$M_2$: Strings where the sum of digits mod 3 is 0
both: even number of 2’s and sum mod 3 = 0
DFA that accepts strings of a’s, b’s, c’s with no more than 3 a’s
FSM that accepts binary strings with a 1 three positions from the end.
“Remember the last three bits”  3 bit shift register
FSMs with output

“Tug-of-war”

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_2$</td>
<td>$s_4$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$s_3$</td>
<td>$s_5$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>$s_4$</td>
<td>$s_5$</td>
</tr>
</tbody>
</table>

[Beep]
We’re only making $5.50/hour writing regular expressions.

Let’s design a vending machine.

“He does not think like normal people, and as a result his tests are quite difficult. His lectures are amusing and get the material across, but his office hours are not always too helpful. **Beware the vending machine final.**”

**Vending spec:**
Enter 15 cents in dimes or nickels
Press S or B for a candy bar
Basic transitions on N (nickel), D (dime), B (butterfinger), S (snickers)
Adding output to states: N – Nickel, S – Snickers, B – Butterfinger
Adding additional “unexpected” transitions