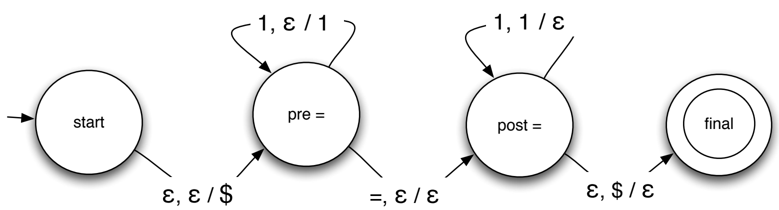
CS 361 Meeting 15 — 03/13/20

Announcements

1. Our midterm will be a 24-hour take-home open-book/notes. It will occur when we start our on-line meetings in April.
2. Homework assignment 5 is due today, but you can take a 1 week extension with no penalty.

Pushdown Automata

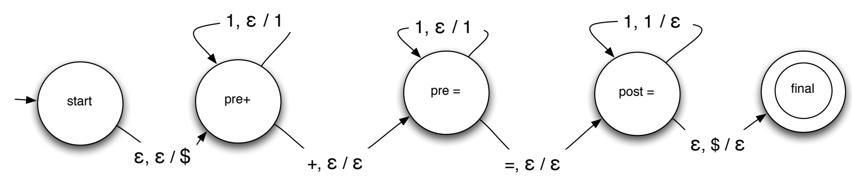
1. Last time, I introduced a new model of computation called the pushdown automaton. It processes inputs sequentially while making state transitions, but it can store data in a stack as it reads the input and its transitions can depend on the symbol on top of the stack.
2. At the end of class we were exploring a simple example of a PDA that recognized the language $\{1^n = 1^n | n \geq 0\}$.
   - The diagram below provides an informal description of a pushdown automaton that recognizes this language.



   - This machine has one feature that is somewhat an artifact of the way Sipser chooses to describe PDAs — namely, his formalism provides no way for the machine to sense if its stack is empty. This will lead most of our machines to include a start state with just one transition that puts a recognizable symbol at the bottom of the stack before processing any input.
   - Also note that this trick depends on epsilon-transitions. In particular, for now, all PDAs are non-deterministic.
   - Let's trace through the steps this machine takes processing the input 11=11.
3. To help solidify your understanding of this informal introduction to pushdown automata, I would like you to design a machine (or at least a modification of another machine) working with a fellow student, so...
   - Think about how to construct a PDA that recognizes

$$L_{add} = \{1^i + 1^j = 1^{i+j} \mid i, j \geq 0\}$$

   The answer to this question should look something like:



   The first two states to the right of the start state push as many 1s on the stack as there are before the = sign and make sure that there is exactly one + sign. The last state makes sure that the number of 1s after the equal sign matches those seen before.

PDAs Formally

1. A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

   Q is a finite set of states,

   Σ is a finite input alphabet,

   Γ is a finite stack alphabet,

   $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function, and

   $F \subset Q$ is the set of final or accepting states.

Click here to view the slides for this class

2. We say that a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts a string $w = w_1 w_1 ... w_n, w_i \in \Sigma_\epsilon$ if $\exists q_1, ... q_n \in Q$ and $s_0, s_1, ... s_n \in \Gamma^*$ such that:

   - $s_0 = \epsilon$
   - $\forall i, 1 \le i \le n, \exists h_i, p_i \in \Gamma_\epsilon$ and $t_i \in \Gamma*$ such that $s_{i-1} = h_i t_i, s_i = p_i t_i$ and $(q_i, p_i) \in \delta(q_{i-1}, w_i, h_i)$
   - $s_n \in F$

Thinking Nondeterministically

1. When we studied finite automata, we started with the deterministic model and then moved on to consider nondeterminism later. With pushdown automata, we have started right away using nondeterminism (at least in the form of epsilon transitions).

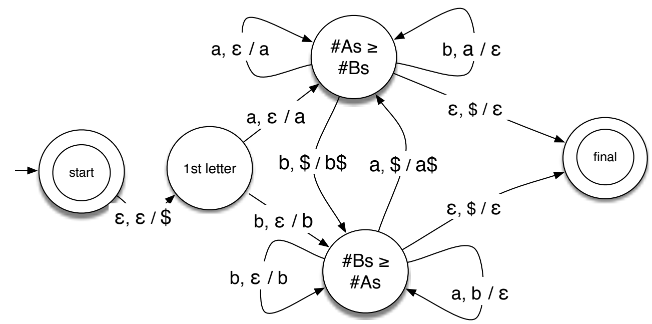
2. To reinforce the power of nondeterminism in this model, I want to explore a few solutions to the problem of building a pushdown automaton for the language

$$L_{eq\_occur} = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ contains as many a's as b's}\}$$

3. First, it should be clear that this cannot be a regular language since if we intersect L_{eq_occurs} with the language of the regular expression a^*b^* we get $\{a^n b^n \mid n \ge 0\}$ which is clearly not regular.

4. So, we should expect to have to use the stack in some way to keep track of how many letters of each type we have seen. Any suggestions?

5. If you are stumped, consider this machine and see if you can explain how it works intuitively:[1]

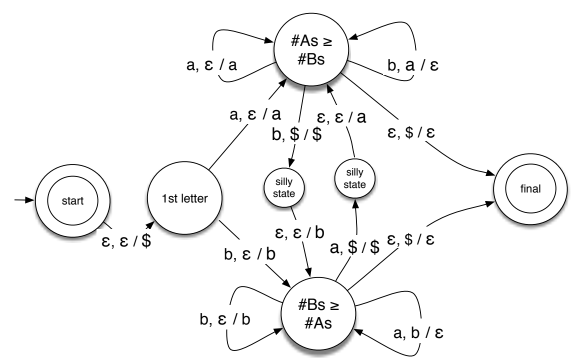


6. We cannot use one stack to simultaneously keep track of two separate counters — one for a's and one for b's. What this machine does instead is use the stack to keep track of how many more a's than b's have been seen when more a's have been seen. However, when more b's than a's have been seen it instead uses the stack to keep track of how many more b's than a's.

7. What I really want you to notice is the peculiar way this machine uses nondeterminism.

   - The only nondeterministic transitions in the machine are the ϵ transitions leaving "start" and leading to "final".

[1] The transitions between the two $\ge$ states in this version of the machine for $L_{eq-occurs}$ violate Sipser's formalism by pushing two symbols (a or b and \$) at the same time. We view these transitions as a shorthand for a series of transitions like those shown in the version of the same machine shown below:



- They both involve the peculiar fact that there is no explicit way to test for empty stack or end of input in Sipser's model of a pushdown automaton.
- The first ϵ-transition puts a marker at the bottom of the stack so that other states can tell when it is (near) empty.
- The transitions to "final" reflect the fact that there is no deterministic way to make a transition only when the machine reaches end of input.
- To accept an input, a PDA must reach a final state at a point where all of the input has been consumed.
  - Because of the way this machine uses the stack to balance the number of as and bs, it will be in either the "#As $\geq$ #Bs" state or the "#Bs $\geq$ #As" state when the input runs out.
  - Neither of these states can be final because we should not accept if any a's or b's are left in the stack.
  - The ϵ transitions from the $\geq$ states to "final" let the machine guess that it is at the end of the input whenever the stack is emtpy. If it guesses correctly, it will be able to accept the input. An incorrect guess will leave it in a final state with no way to consume the remaining input so that branch of nondeterminism will just expire.

8. Then, the cupcakes arrived!!!