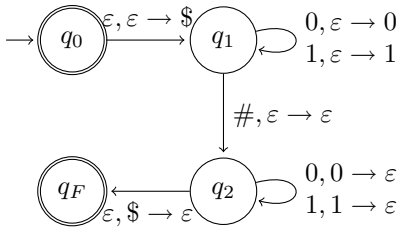
Notes for Wednesday, May 12th

(Question from the slides: why is the stack alphabet different from Σ? Because it allows us to do more: think of the stack as scratch paper)

Example: $L = \{w\#w^k | w \in \{0,1\}^*\}$ (examples of strings in L: #, 10#01, 110#011, ...)

PDA M for L :



In this PDA, q_1 is the state for "pushing" w and state q_2 is the state for "popping" w and matching it to w^R.

A formal description of $M : (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

$Q = \{q_0, q_1, q_2, q_F\}$

$\Sigma = \{0, 1, \#\}$

$\Gamma = \{0, 1, \$\}$

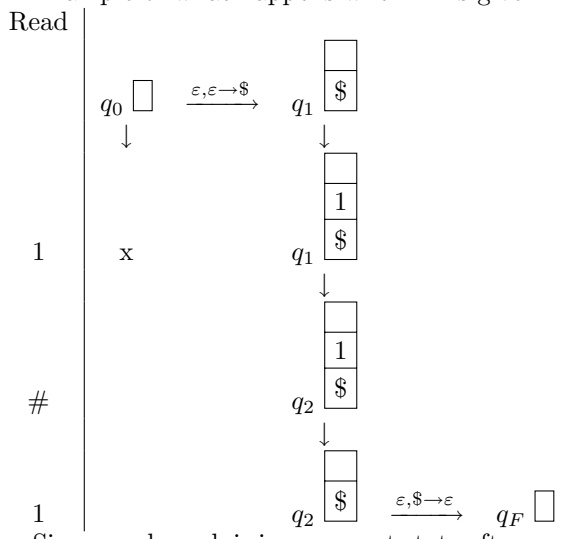
$F = \{q_F\}$

$q_0 = q_0$

and δ can be described by a table:

	0				1				#				ε			
δ	0	1	\$	ε	0	1	\$	ε	0	1	\$	ε	0	1	\$	ε
q_0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_1, \$)\}$
q_1	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_1, 0)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_1, 1)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_2, \varepsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_2	$\{(q_2, \varepsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_2, \varepsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_F, \varepsilon)\}$	$\emptyset$
q_F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

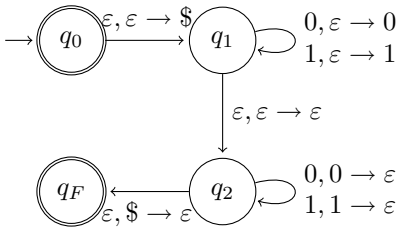
Example of what happens when M is given input 1#1 :



Since one branch is in an accept state after reading the input string, the machine accepts the input string.

Note that the machine branches off into two separate states (and thus separate stacks). We can think of these as separate processes with their own memory. Also, the rules don't require to have an empty stack to accept, but our constructions will usually make that happen.

Another example:
$L_1 = \{ww^R | w \in \{0,1\}^*\}$.
M_1 accepting L_1 :



This machine is almost exactly like M above, except for the transition between states 1 and 2. Essentially, we are constantly checking if the "second half" of the input is the reverse of the first half. Next time, we'll see how exactly the machine behaves on some input.