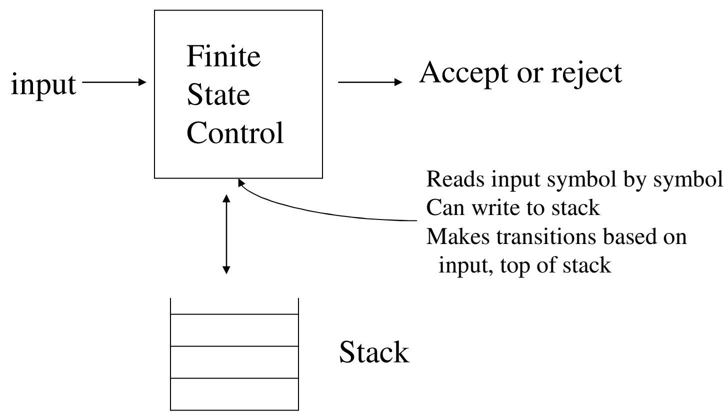
Pushdown Automata

Pushdown Automata (PDA)

- Just as a DFA is a way to implement a regular expression, a pushdown automata is a way to implement a context free grammar
  - PDA equivalent in power to a CFG
  - Can choose the representation most useful to our particular problem
- Essentially identical to a regular automata except for the addition of a stack
  - Stack is of infinite size
  - Stack allows us to recognize some of the non-regular languages

PDA

- Can visualize a PDA with the schematic



Implementing a PDA

- In one transition the PDA may do the following:
  - Consume the input symbol. If ε is the input symbol, then no input is consumed.
  - Go to a new state, which may be the same as the previous state.
  - Replace the symbol at the top of the stack by any string.
    - If this string is ε then this is a pop of the stack
    - The string might be the same as the current stack top (does nothing)
    - Replace with a new string (pop and push)
    - Replace with multiple symbols (multiple pushes)

Informal PDA Example

- Consider the language $L = \{0^n1^n \mid n \geq 0\}$.
  - We showed this is not regular
  - A finite automaton is unable to recognize this language because it cannot store an arbitrarily large number of values in its finite memory.
- A PDA is able to recognize this language!
  - Can use its stack to store the number of 0's it has seen.
    - As each 0 is read, push it onto the stack
    - As soon as 1's are read, pop a 0 off the stack
    - If reading the input is finished exactly when the stack is empty, accept the input else reject the input

PDA and Determinism

- The description of the previous PDA was deterministic
- However, in general the PDA is nondeterministic.
- This feature is crucial because, unlike finite automata, nondeterminism adds power to the capability that a PDA would have if they were only allowed to be deterministic.
  - i.e. A non-deterministic PDA can represent languages that a deterministic PDA cannot

Informal Non-Deterministic Example

- $L = \{ ww^R \mid w \text{ is in } (0+1)^* \}$
  - i.e. the language of even length palindromes
- Informal PDA description
  - Start in state q0 that represents the state where we haven't yet seen the reversed part of the string. While in state q0 we read each input symbol and push them on the stack.
  - At any time, assume we have seen the middle; i.e. "fork" off a new branch that assumes we have seen the end of w. We signify this choice by spontaneously going to state q1. This behaves just like a nondeterministic finite automaton
    - We'll continue in both the forked-branch and the original branch. One of these branches may die, but as long as one of them reaches a final state we accept the input.
  - In state q1 compare input symbols with the top of the stack. If match, pop the stack and proceed. If no match, then the branch of the automaton dies.
  - If we empty the stack then we have seen ww^R and can proceed to a final accepting state.

Formal Definition of a PDA

- $P = (Q, \Sigma, \Gamma, \delta, q_0, \$, F)$
  - Q = finite set of states, like the finite automaton
  - Σ = finite set of input symbols, the alphabet
  - Γ = finite stack alphabet, components we are allowed to push on the stack
  - q_0 = start state
  - \$ = start symbol. Initially, the PDA's stack consists of one instance of this start symbol and nothing else. We can use it to indicate the bottom of the stack. We can place \$ explicitly on the stack initially, or assume it is there already.
  - F = Set of final accepting states.

PDA Transition Function

- δ = transition function, which takes the triple: $\delta(q, a, X)$ where
  - q = state in Q
  - a = input symbol in Σ
  - X = stack symbol in Γ
- The output of δ is the finite **set** of pairs (p, γ) where p is a new state and γ is a new string of stack symbols that replaces X at the top of the stack.
  - If $\gamma = \varepsilon$ then we pop the stack
  - if $\gamma = X$ the stack is unchanged
  - if $\gamma = YZ$ then X is replaced by Z and Y is pushed on the stack. Note the new stack top is to the left end.
  - If $X = \varepsilon$ then we push on γ

Formal PDA Example

- Here is a formal description of the PDA that recognizes $L = \{0^n 1^n \mid n \geq 0\}$.
  - $Q = \{ q_1, q_2, q_3, q_4 \}$
  - $\Sigma = \{0, 1\}$
  - $\Gamma = \{0, \$\}$
  - F = {q1, q4}
- And δ is described by the table below

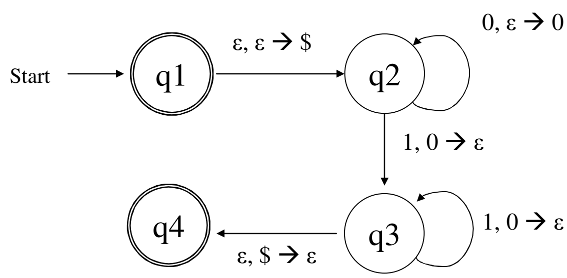
Input:	0			1			ε		
Top Stack:	0	$	ε	0	$	ε	0	$	ε
q1									q2,$
q2			q2,0	q3,ε					
q3				q3,ε				q4,ε	
q4									

Graphical Format

- Uses the format

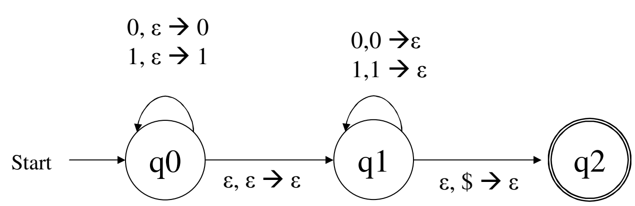
  Input-Symbol, Top-of-Stack → String-to-replace-top-of-stack

  Any of these may be empty!



Example 2

- Here is the graphical description of the PDA that accepts the language
  - L = { ww^R | w is in (0+1)* }
  - Stays in state q0 when we are reading w, saving the input symbol. Every time we "guess" that we have reached the end of w and are beginning wR by going to q1 on an epsilon-transition.
  - In state q1 we pop off each 0 or 1 we encounter that matches the input. Any other input will "die" for this branch of the PDA. If we ever reach the bottom of the stack, we go to an accepting state.



Requires $ initially on stack

Moves of a PDA

- To describe the process of taking a transition, we can adopt a notation similar to δ like we used for DFA's. In this case we use the "turnstile" symbol $\vdash$ which is used as:

  $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$

- In other words, we took a transition such that we went from state q to p, we consumed input symbol a, and we replaced the top of the stack X with some new string α.

- We can extend the move symbol to taking many moves:
  $\vdash^*$ represents zero or more moves of the PDA.

Language of a PDA

- The PDA consumes input and accepts input when it ends in a final state. We can describe this as:

  $L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)\}$ where $q \in F$

- That is, from the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

Alternate Definition for L(PDA)

- It turns out we can also describe a language of a PDA by ending up with an empty stack with no further input

  $N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$
  where q is any state.
- That is, we arrive at a state such that P can consume the entire input and at the same time empty its stack.
- It turns out that we can show the classes of languages that are L(P) for some PDA P is equivalent to the class of languages that are N(P) for some PDA P.
- This class is also exactly the context-free languages. See the text for the proof.

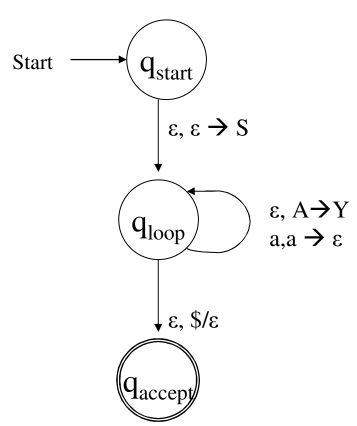
Equivalence of PDA and CFG

- A context-free grammar and pushdown automata are equivalent in power.
- Theorem: Given a CFG grammar G, then some pushdown automata P recognizes L(G).
  - To prove this, we must show that we can take any CFG and express it as a PDA. Then we must take a PDA and show we can construct an equivalent CFG.
  - We'll show the CFG→PDA process, but skip the PDA→CFG (more details in the textbook, as indicated in last slide).

Proof for CFG to PDA

- Proof idea:
  - The PDA P will work by accepting its input w, if G generates that input, by determining whether there is a derivation for w.
  - Design rules for P such that the transitions match the production rules in the grammar
    - But the PDA can access only the top symbol on the stack and that might be a terminal
      - But if our PDA makes transitions only for variables, we we won’t know to do
    - To get around this problem, we’ll use the non-determinism of the PDA to match terminal symbols on the stack with symbols in the input string before the first variable
      - This “chomps” any leading terminals until we can process a variable

State Diagram for an Arbitrary CFG



S is the start symbol

For rule A→Y, Y is a string

For terminal symbol a

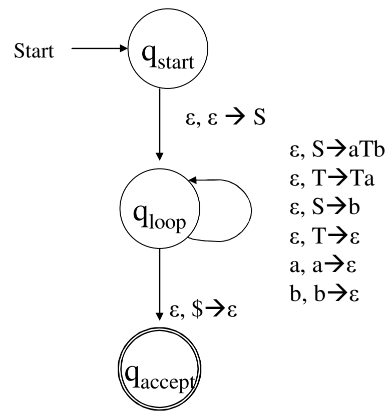
$ initially the top of the stack

Example

- Consider the grammar

  S→ aTb | b

  T → Ta | ε



Given the string "aab" derived by

$S \Rightarrow aTb \Rightarrow aTab \Rightarrow aab$

We have the corresponding moves:

$(q_s, aab, \$) \vdash (q_{loop}, aab, S\$) \vdash$
$(q_{loop}, aab, aTb\$) \vdash (q_{loop}, ab, Tb\$) \vdash$
$(q_{loop}, ab, Tab\$) \vdash (q_{loop}, ab, ab\$) \vdash$
$(q_{loop}, b, b\$) \vdash (q_{loop}, \varepsilon, \$) \vdash$
$(q_{accept}, \varepsilon, \varepsilon)$

Proof for PDA to CFG

- Harder direction; not as easy to "program" a CFG as it is a PDA

Deterministic PDA

- A DPDA is simply a pushdown automata without non-determinism.
  - i.e. no epsilon transitions or transitions to multiple states on same input
  - Only one state at a time
- DPDA not as powerful a non-deterministic PDA
  - This machine accepts a class of languages somewhere between regular languages and context-free languages.
  - For this reason, the DPDA is often skipped as a topic
  - In practice the DPDA can be useful since determinism is much easier to implement.
    - Parsers in programs such as YACC are actually implemented using a DPDA.

Context-Free Languages and Regular Languages

- Every regular language is context free.
  - How can we argue this from what we know about PDA's and context free languages?