# CS 3313
# Foundations of Computing:

# Pushdown Automata

http://gw-cs3313-2021.github.io

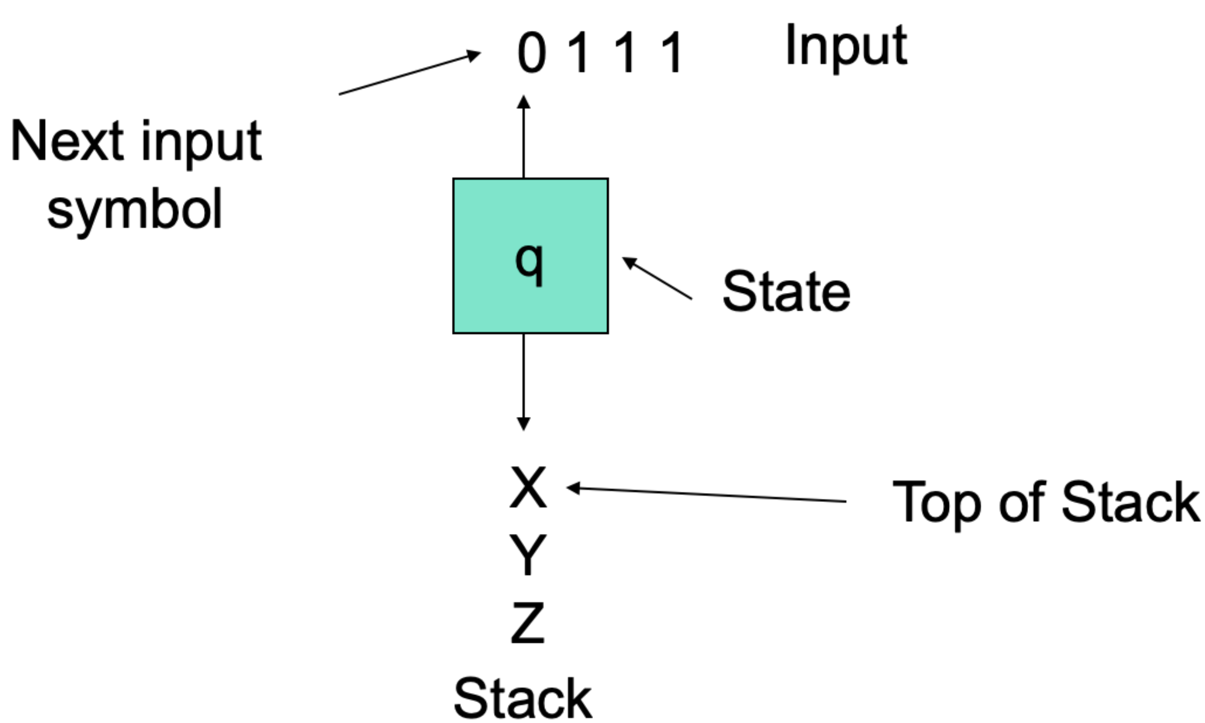
# Formal Definition: Pushdown Automaton (PDA)

- A NFA with a stack storage is a Pushdown Automaton (PDA).
  - PDA is an automaton equivalent to the CFG in language-defining power.
    - Proof/Conversion algorithm covered in lecture today
  - Only the nondeterministic PDA defines all the CFL's.
  - Deterministic PDAs accept a subset of CFLs
    - Most programming languages have deterministic PDA's.

- Next: quick review of definitions and examples, including using JFLAP to simulate and test

# Intuition: PDA

- Think of an λ-NFA with the additional power that it can manipulate a stack.
- Its moves are determined by:
  1. The current state (of its "NFA"),
  2. The current input symbol (or ϵ), and
  3. The current symbol on top of its stack.



# Intuition: PDA – (2)

- Being nondeterministic, the PDA can have a choice of next moves.
- In each choice, the PDA can:
  1. Change state, and also
  2. Replace the top symbol on the stack by a sequence of zero or more symbols.
     - Zero symbols = “pop.”
     - Many symbols = sequence of “pushes.”

# PDA Formalism

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is described by:
  1. A finite set of *states* Q (same as before).
  2. An *input alphabet* $\Sigma$ (same as before).
  3. A *stack alphabet* $\Gamma$ (typically assume $\Gamma$ disjoint from $\Sigma$).
  4. A *transition function* $\delta$
     - $\delta: (Q \times (\Sigma \cup \lambda) \times \Gamma) \rightarrow 2^{(Q \times \Gamma^*)}$ (subset of $Q \times \Gamma^*$)
  5. A *start state* $q_0$, in Q (same as before).
  6. A *start symbol* $Z_0$, in $\Gamma$ (to indicate bottom of stack).
  7. A set of *final states* $F \subseteq Q$

# Pushdown Automaton: Definitions

- There is a specific stack alphabet Γ
  - You could always make it equal to Σ
  - Better to keep it separate but can have a 1-1 mapping
    - Ex: Σ ={a,b} Γ= {X,Y} where X corresponds to a and Y to b.
- PDA by default is non-deterministic
  - δ(q,a,x) has a number of choices of (p,y) where p is a state and y is a stack symbol
  - A deterministic PDA is known as a DPDA (less powerful than PDA)
  - λ-transitions are allowed as the default
- Can also push/pop λ onto stack = push/pop nothing
- Can define a transition graph for a pda
  - each edge is labeled with the input symbol, the stack top, and the string that replaces the top of the stack
  - But cumbersome to model as a graph….so use Parse trees formalism

# Actions of the PDA

- $\delta$ takes three arguments: state, input, top of stack
- $\delta(q, a, Z)$ is a set of possible moves of the form $(p, \alpha)$.
  - p is a state; $\alpha$ is a string of stack symbols.
- If $\delta(q, a, Z)$ contains $(p, \alpha)$ among its actions, then PDA in state q, with *a* as input, and Z on top of the stack:
  1. Changes the state to p.
  2. Remove *a* from the front of the input (but *a* may be $\lambda$).
  3. Replace Z on the top of the stack by $\alpha$.
     - Pop Z and Push $\alpha$

- Note: (3) above implies that you always pop from TOS
  therefore to push onto TOS, you have to push the original TOS followed by the new stack symbol

# Instantaneous Descriptions

- current configuration of the PDA specified by ***instantaneous description*** (ID)
- An ID is a triple $(q, w, \alpha)$, where:
  1. q is the current state.
  2. w is the remaining input.
  3. $\alpha$ is the stack contents, top at the left.
- In one "move" (step), a PDA goes from one ID to another
  - A move is denoted by the symbol $\vdash$ ( "yields")
- Formally: $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$ for any w and $\alpha$,

  if $\delta(q, a, X)$ contains $(p, \beta)$.
- Extend $\vdash$ to $\vdash^*$, meaning "zero or more moves," by:
  - Basis: $I \vdash^* I$.
  - Induction: If $I \vdash^* J$ and $J \vdash K$, then $I \vdash^* K$.

# Language of a PDA

- The common way to define the language of a PDA is by *final state*.
  - set of strings that cause the PDA to halt in a final state, after starting in $q_0$ with an empty stack denoted by $Z_0$ start stack symbol at TOS.
    - The final contents of the stack are irrelevant
  - Since PDA is nondeterministic, the string is accepted if any of the computations cause it to halt in a final state
- If M is a PDA, then L(M) is the set of strings w such that

$$(q_0, w, Z_0) \vdash^* (f, \lambda, \alpha) \text{ for final state } f \text{ and any } \alpha \in \Gamma^*$$

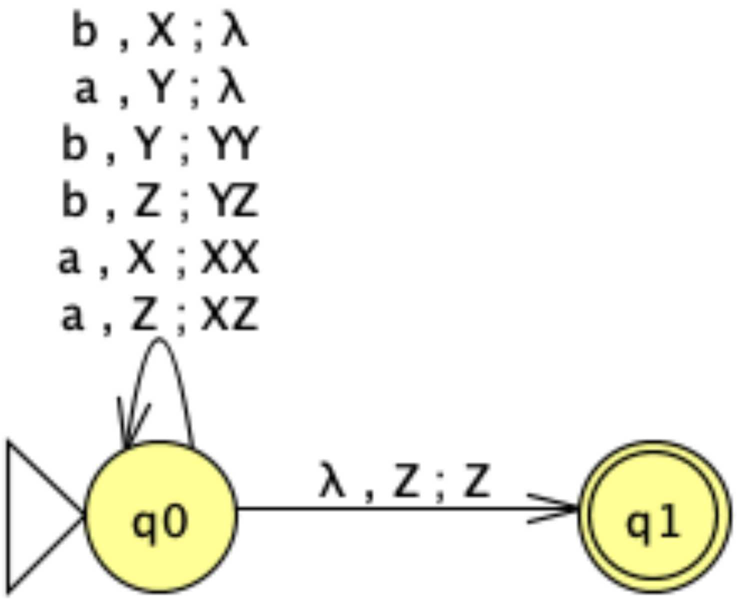
- acceptance of a language by a PDA by *empty stack.*
- If M is a PDA, then N(M) is the set of strings w such that

$$(q_0, w, Z_0) \vdash^* (q, \lambda, \lambda) \text{ for any state } q.$$

# Example 1: L = { w | w has equal number of a's and b's}

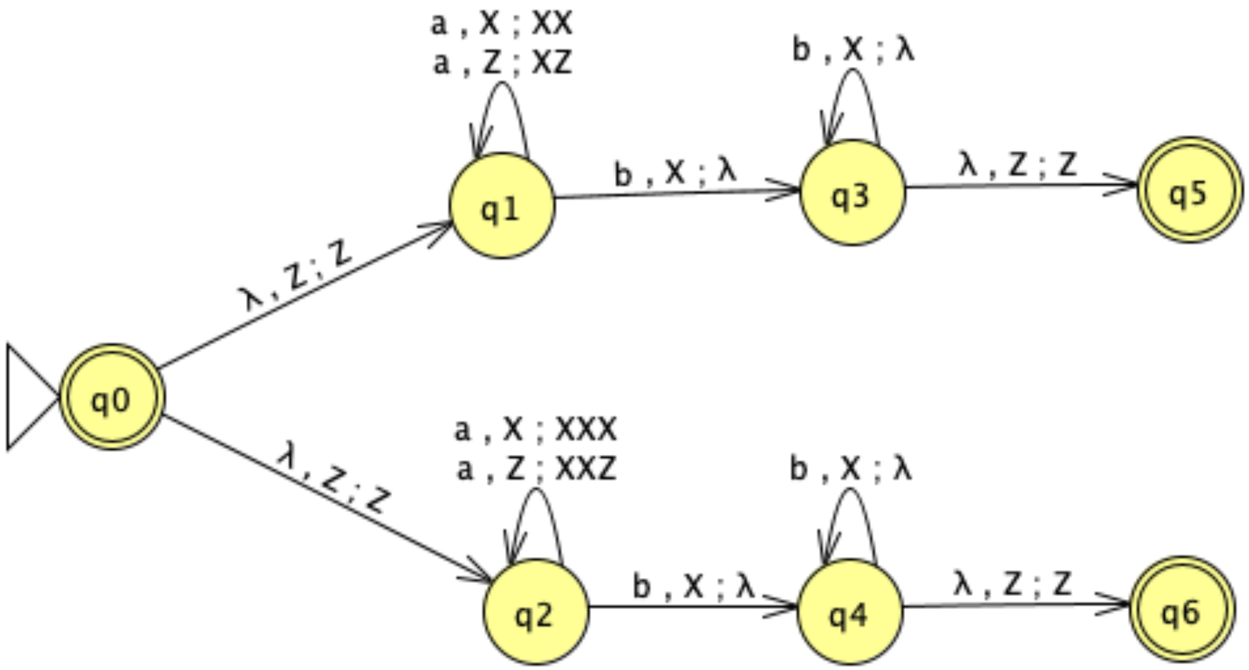
- Logic:
  - Stack symbols X for input a, Y for input b
  - For every a in the input, there is a b that cancels it out
  - Push X if you read an a, with $Z_0$ or X at TOS
  - Push Y if you read a b, with $Z_0$ or Y at TOS
  - If you read a and Y is TOS, then "pop" (cancel a with a previous b)
  - If you read b and X is TOS, then "pop" (cancel b with a previous a)

  - If you have an equal number then at the end of input you will have
    - Empty string to read on input
    - Start stack on TOS
    - $(q, \lambda, Z_0)$ is the ID


b , X ; λ
a , Y ; λ
b , Y ; YY
b , Z ; YZ
a , X ; XX
a , Z ; XZ
λ , Z ; Z
q0
q1

# Example 2: $L = \{ a^m b^n \mid n=m \text{ or } n=2m \}$

- Strings of the form aaabbb or aaabbbbbb
- Use non-determinism to design the PDA:
  - PDA1 accepts n=m or
  - PDA2 accepts n=2m
  - Start PDA and jump non-deterministically on empty string input to PDA1 or PDA2
- PDA1 starts in $q_1$
  - For every a in input, push X. For every b, pop one X
- PDA2 starts in $q_2$
  - For every a in input, push 2 X's. For every b, pop one X
- If either PDA1 or PDA2 accept then PDA M accepts.
- Observation: turns out there is no Deterministic PDA for this language – can prove it later.


a , X ; XX
a , Z ; XZ
b , X ; λ
b , X ; λ
λ , Z ; Z
λ , Z ; Z
q0
q1
q3
q5
a , X ; XXX
a , Z ; XXZ
b , X ; λ
λ , Z ; Z
λ , Z ; Z
q2
q4
q6

# Example 3: $L = \{ w w^R \mid w \text{ in } \{a,b\}^* \}$

- Recognizing $wcw^R$ was "easy"
  - Keep pushing input until you read c
  - After reading c, compare input with TOS
  - The location of c gives you the 'midpoint' (the middle of the string)
- In $ww^R$ there is no *c* in input to denote midpoint
- So "guess" non-deterministically
  - If it is the midpoint then input you read = last input you read (i.e. TOS)
  - After midpoint, "match" input with TOS (read a, X is TOS; read b with Y as TOS)…until you hit end of string and $Z_0$ as TOS
- Ex: abbbba
  - a ↑ bbbba can this be the midpoint?
  - ab↑bbba can this be the midpoint?
  - abb ↑ bba can this be midpoint ?
  - abbb↑ ba can this be midpoint ?

# Example/Exercise 3: $ww^R$

- Build the PDA for $ww^R$