Chapter 5

Pushdown Automata

Regular Languages (Review)

- Regular languages are,
  - described by regular expressions.
  - generated via regular grammars.
  - accepted by
    - deterministic finite automata DFA
    - nondeterministic finite automata NFA
  - There is an equivalence between the deterministic and nondeterministic versions.
- Every regular language RL is CFL.
- But some CFL are not regular:
  - $L = \{a^n b^n : n \geq 1\}$ has CFG.
    - $S \rightarrow aSb \mid ab$
  - The language $\{ww^R : w \in \{a, b\}^*\}$ has CFG.
    - $S \rightarrow aSa \mid bSb \mid \varepsilon$

Context-Free Languages (Review)

- CFL are generated by a context-free grammar CFG.

- A grammar $G = (NT, T, S, P)$ is CFG if all production rules have the form.
  - $A \rightarrow y$, where $A \in NT$, and $y \in (NT \cup T)^*$
  - i.e., there is a single NT on the left hand side.

- A language L is CFL iff there is a CFG G such that $L = L(G)$.

- All regular languages, and some non-regular languages, can be generated by CFGs.
  - regular languages are a proper subset of context-free languages.

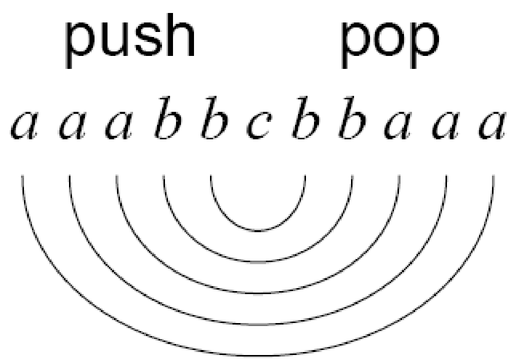
Stack Memory

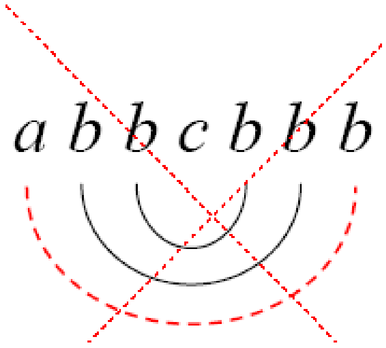
- The language $L = \{wcw^R : w \in \{a, b\}^*\}$ is CFL but not RL
  - We can not have a DFA for L.
  - Problem is **memory**, DFA's cannot remember left hand substring.

- What kind of memory do we need to be able to recognize strings in L?
  - Answer: a **stack.**

Stack Memory…

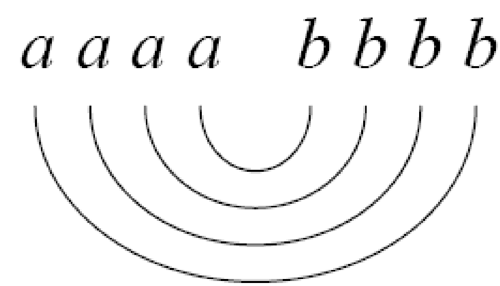
- Example: $u = aaabb\mathbf{c}bbaaa \in L.$
  - We **push** the first part of the string onto the stack and
  - after the *c* is encountered.
  - start **popping** characters off of the stack and matching them with each character.
  - if everything matches, this string $\in L.$





Stack Memory…

- We can also use a stack for counting out equal numbers of *a*’s and *b*’s.
- Example:
  - $L = \{a^n b^n : n \geq 0\}$
  - $w = aaaabbbb \in L$
  - **Push** the *a*’s onto the stack, then **pop** an *a* off and match it with each *b*.
  - If we finish processing the string successfully (and there are no more *a*’s on our stack), then the string belongs to *L*.



Nondeterministic Push-Down Automata

- A language is ***context free*** (CFL) iff some **Nondeterministic PushDown Automata** (NPDA) recognizes (accepts) it.
  - Intuition: **NPDA** = **NFA** + one **stack** for memory.
  - Stack remembers info about previous part of string.
  - E.g., to accept $a^n b^n$

- **Deterministic PushDown Automata** (DPDA) can accept **some but *not all*** of the CFLs.

- Thus, there is no longer an equivalence between the deterministic and nondeterministic versions,
  - i.e., languages recognized by DPDA are a proper subset of context-free languages.

NPDA....

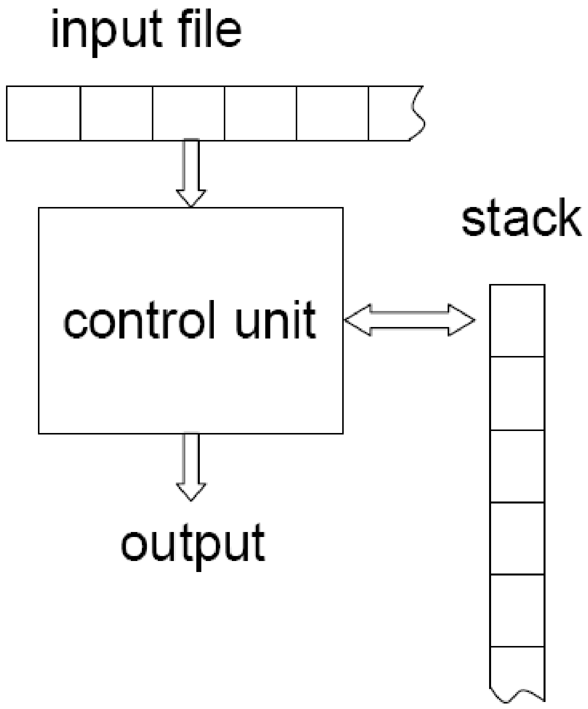
- You can *only* access the *top* element of the stack.
- To access the top element of the stack, you have to **pop** it off the stack.
  - Once the top element of the stack has been popped, if you want to save it, you need to **push** it back onto the stack.

- Symbols from the input string must be read one symbol at a time. You cannot back up.

- The current configuration (*state*, *string*, *stack*) of the NPDA includes:
  - the current *state,*
  - the remaining symbols left in the input *string*, and
  - the entire contents of the *stack.*

NPDA....

- NPDA consists of,
  - Input file, Control unit, Stack.
  - Output
    - output is yes/no.
    - indicates string belongs to given language.
- Each move
  - reads a symbol from the input.
    - ε-moves are legal.
  - pops a symbol from the stack.
    - no move is possible if stack is empty.
  - pushes a string, *right-to-left*, onto the stack.
  - move to the target state.

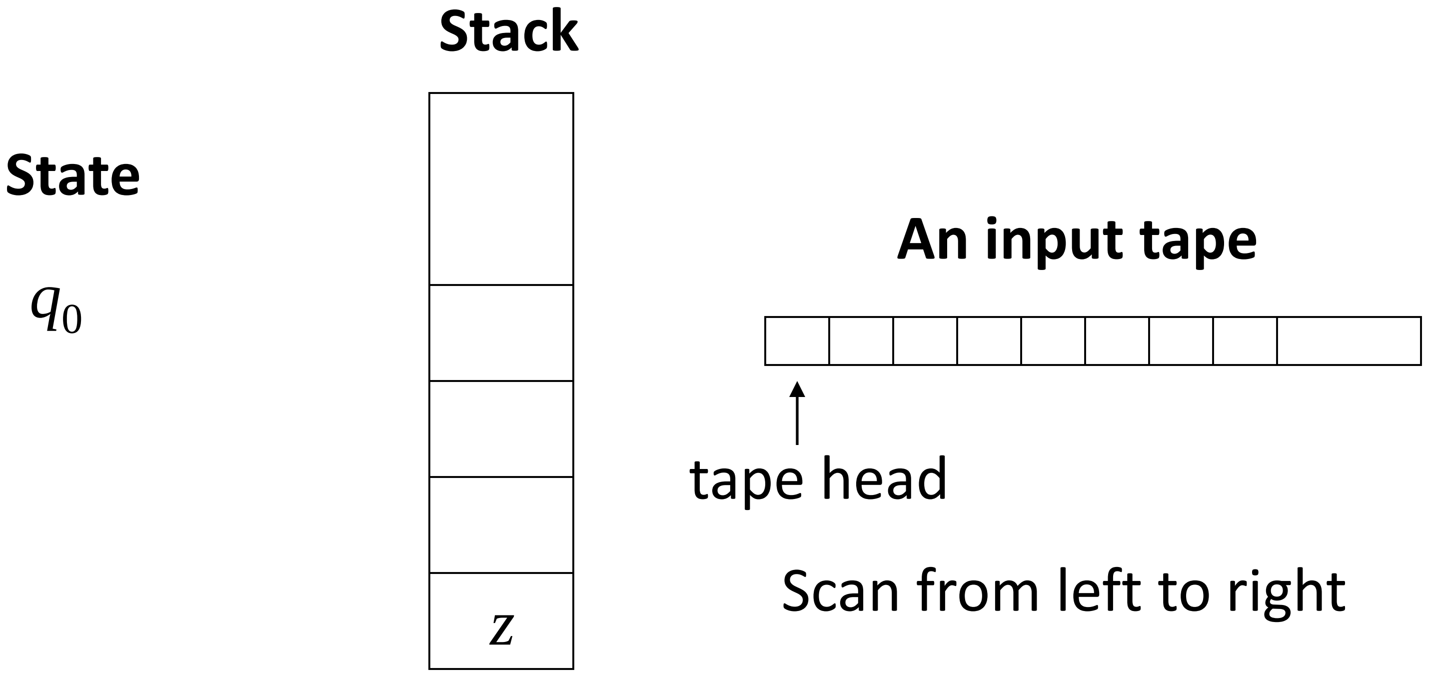


NPDA....

- A **NPDA** is a seven-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ where,
  - Q finite set of states
  - Σ finite set of input alphabet
  - Γ finite set of stack alphabet
  - δ transition function from
    - $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$
    - $\delta(q_n, \boldsymbol{a}, \boldsymbol{A}) = \{(q_m, \boldsymbol{B})\}$
  - q_0 start state $q_0 \in Q$
  - z initial stack symbol $z \in \Gamma$
  - F final states $F \subseteq Q$

NPDA....

- There are **three** things in a NPDA:



NPDA….

- The transition function deserves further explanation.
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$.
  - A 3-tuple mapped onto one or more 2-tuples.
- Transition function now depends upon **three** items:
  - current state, input symbol, and stack symbol.

stack symbol popped

next state

string of stack symbols pushed onto stack (right-to-left), i.e., the left-most symbol is at top of stack

- $\delta(q_h, a, p) = \underbrace{\{(q_i, s_n), \ldots, (q_j, s_m)\}}_{\text{nondeterministically chooses one of these pairs}}$

current state

input symbol read (includes λ)

NPDA….

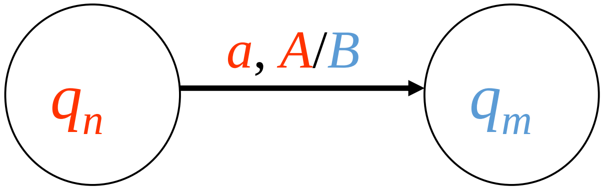
- Note that in a DFA, each transition told us that when we were in a given state and saw a specific symbol, we moved to a specified state.

- In a NPDA, we read an input symbol, but we also need to know what is on the stack before we can decide what new state to move to.

- When moving to the new state, we also need to decide what to do with the stack.

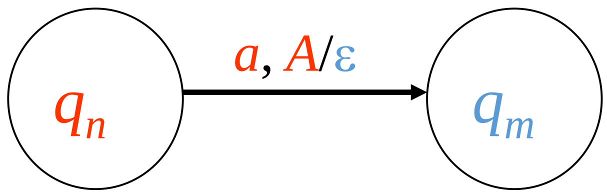
NPDA….

- What it does mean if $\delta(q_n, a, A) = (q_m, B)$?
- It means if
  - the current state is q_n
  - the current input letter is a
  - the top of the stack is A
- Then the machine should
  - change the state to q_m
  - process input letter a
  - pop A off the stack
  - push B onto the stack



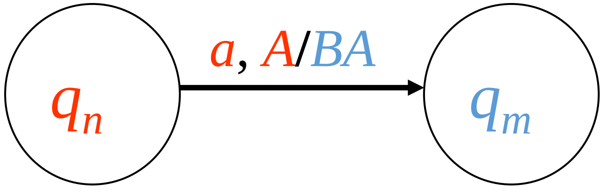
NPDA....

- What it does mean if $\delta(q_n, a, A) = (q_m, \varepsilon)$?
- It means if
  - the current state is q_n
  - the current input letter is a
  - the top of the stack is A
- Then the machine should
  - change the state to q_m
  - process input letter a
  - pop A off the stack
  - *don't push anything onto the stack*



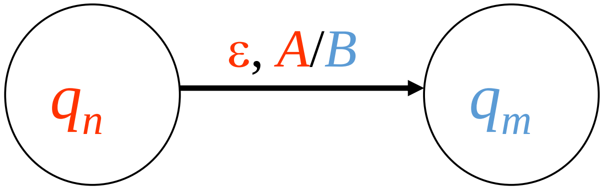
NPDA....

- What it does mean if $\delta(q_n, a, A) = (q_m, BA)$?
  - change the state to q_m
  - process input letter a
  - *don't pop anything from the stack*
  - push B onto the stack



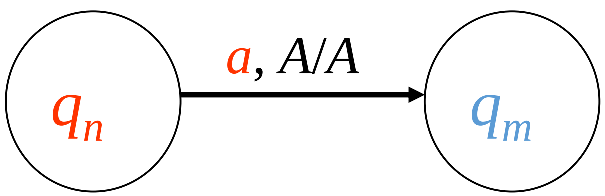
NPDA….

- What it does mean if $\delta(q_n, \varepsilon, A) = (q_m, B)$?
  - change the state to q_m
  - *don't process any input letter*
  - pop A from the stack
  - push B onto the stack



NPDA....

- What it does mean if $\delta(q_n, a, A) = (q_m, A)$?
  - change the state to q_m
  - process input letter a
  - *don't pop anything from the stack*
  - *don't push anything onto the stack*



NPDA....

- Language: $L = \{a^n b^n : n \geq 0\}$
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where
  - $Q = \{q_0, q_1, q_2, q_3\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{Z, a\}$
  - δ
  - q_0 is the start state
  - Z is the initial stack symbol
  - $F = \{q_3\}$

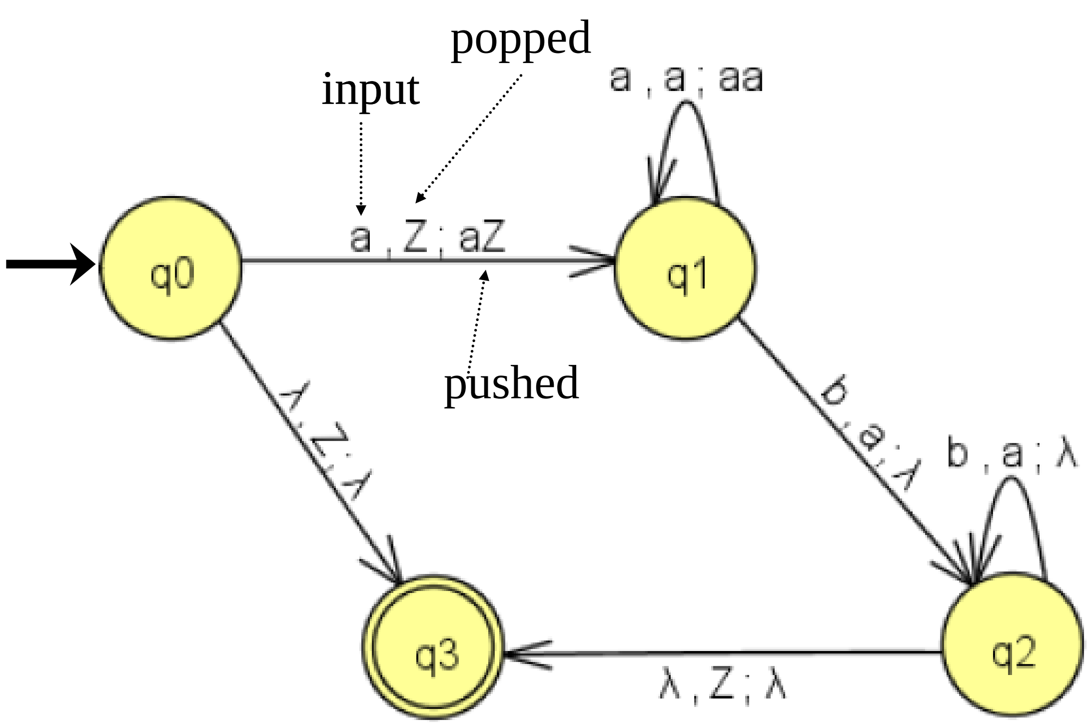
input popped pushed

$$\delta(q_0, a, Z) = \{(q_1, aZ)\}$$
$$\delta(q_0, \varepsilon, Z) = \{(q_3, \varepsilon)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_2, \varepsilon)\}$$
$$\delta(q_2, b, a) = \{(q_2, \varepsilon)\}$$
$$\delta(q_2, \varepsilon, Z) = \{(q_3, \varepsilon)\}$$

- Can be modeled with graph
  - edge triplet is (input, popped, pushed)

NPDA....

- Language: $L = \{a^n b^n : n \geq 0\}$
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where



NPDA….

- A NPDA **configuration** is represented by,
  - $[q_n, u, \alpha]$ where
    - q_n : current state
    - u : unprocessed input
    - α : current stack content

- if $\delta(q_n, a, A) = (q_m, B)$ then $[q_n, au, A\alpha] \vdash [q_m, u, B\alpha]$.

- The notation $[q_n, u, \alpha] \vdash [q_m, v, \beta]$ indicates that configuration $[q_m, v, \beta]$ is obtained from $[q_n, u, \alpha]$ by a *single* transition of the NPDA.

NPDA....

- The notation $[q_n, u, \alpha] \vdash^* [q_m, v, \beta]$ indicates that configuration $[q_m, v, \beta]$ is obtained from $[q_n, u, \alpha]$ by *zero or more* transitions of the NPDA.

- A **computation** of a NPDA is a sequence of transitions beginning with ***start state***.

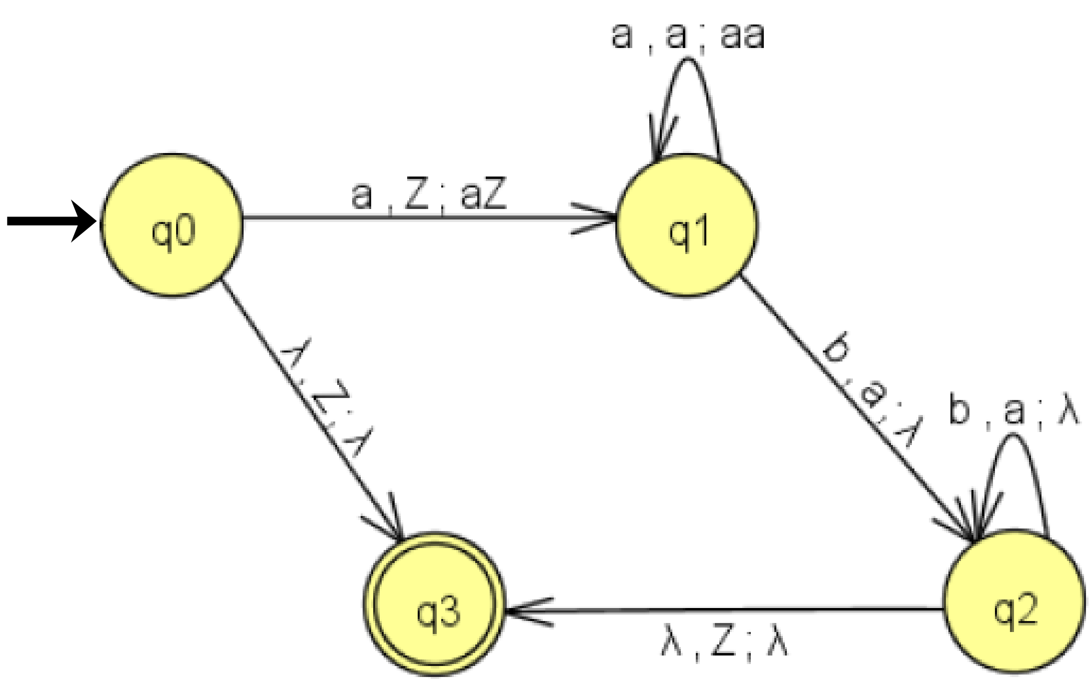
NPDA....

- The language accepted by NPDA M is
- $L(M) = \{w \in \Sigma^*$:
  - Accept when out of input at a ***final state.***
    - $[q_0, w, z] \vdash^* [q_i, \varepsilon, u]$ with $q_i \in F$
  - Accept when out of input at an ***empty stack.***
    - $[q_0, w, z] \vdash^* [q_i, \varepsilon, \varepsilon]$ q_i may not be in F
  - Accept when out of input at a ***final state*** and ***empty stack.***
    - $[q_0, w, z] \vdash^* [q_i, \varepsilon, \varepsilon]$ with $q_i \in F$

NPDA….

- Language: $L = \{a^n b^n : n \geq 0\}$
- The computation generated by the input string *aaabbb* is

$[q_0, aaabbb, Z] \vdash [q_1, aabbb, aZ] \vdash [q_1, abbb, aaZ] \vdash$
$[q_1, bbb, aaaZ] \vdash [q_2, bb, aaZ] \vdash [q_2, b, aZ] \vdash$
$[q_2, \varepsilon, Z] \vdash [q_3, \varepsilon, \varepsilon]$



state	string	stack
q_0	*aaabbb*	Z
q_1	*aabbb*	aZ
q_1	*abbb*	aaZ
q_1	*bbb*	$aaaZ$
q_2	*bb*	aaZ
q_2	*b*	aZ
q_2	λ	Z
q_3	λ	λ

PDA…

- $L=\{wcw^R \mid w\in\{a,b\}^*\}$ is CFL and accepted by NPDA $\{Q=\{q_0,q_1,q_2\},\Sigma=\{a,b,c\},q_0,\Gamma=\{A,B,Z\},Z,F=\{q_2\}\}$

$\delta(q_0, a, Z) = (q_0, AZ)$

$\delta(q_0, a, A) = (q_0, AA)$

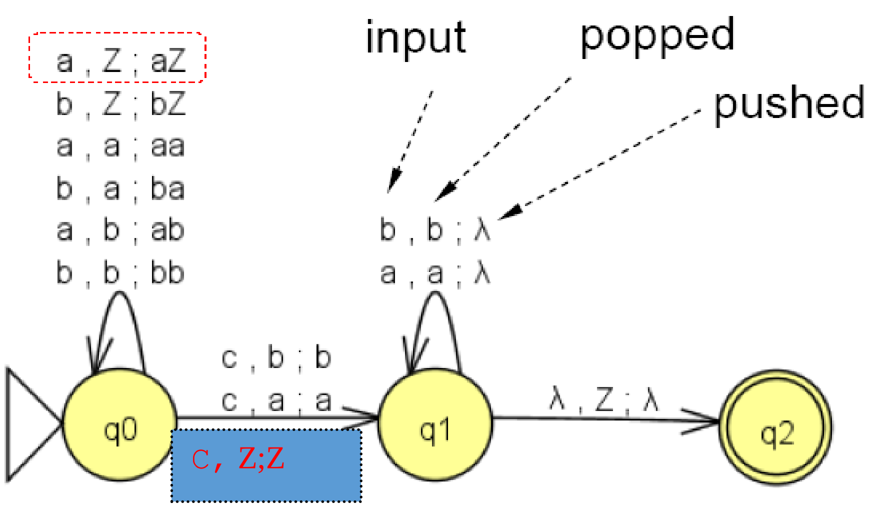
$\delta(q_0, a, B) = (q_0, AB)$

$\delta(q_0, b, Z) = (q_0, BZ)$

$\delta(q_0, b, A) = (q_0, BA)$

$\delta(q_0, b, B) = (q_0, BB)$

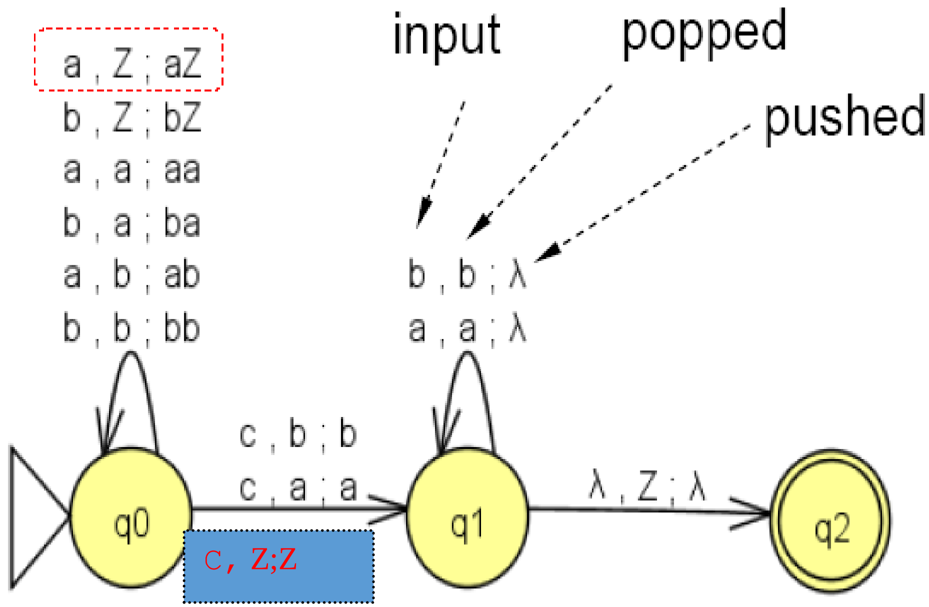
$\delta(q_0, c, Z) = (q_1, Z)$
$\delta(q_0, c, A) = (q_1, A)$
$\delta(q_0, c, B) = (q_1, B)$
$\delta(q_1, a, A) = (q_1, \varepsilon)$
$\delta(q_1, b, B) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z) = (q_2, Z)$



PDA…

- The computation generated by the input string *abcba* is

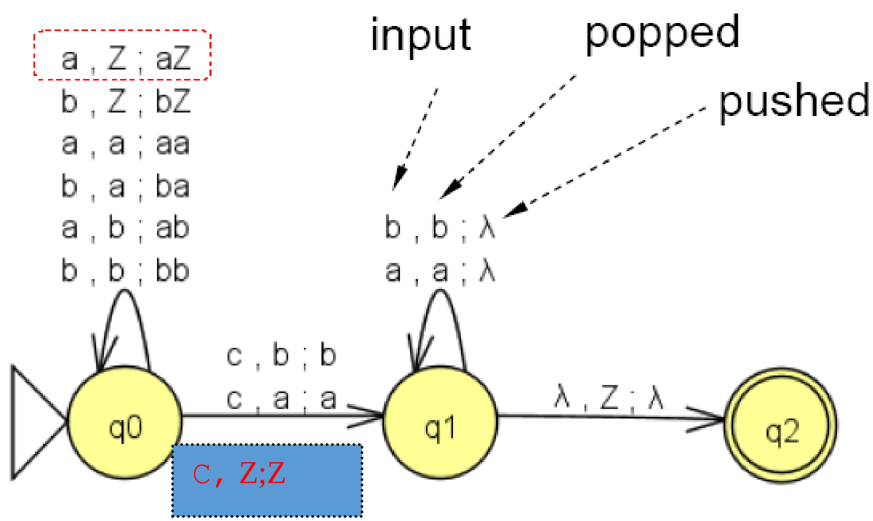
$$
\begin{array}{ll}
& [q_0, abcba, Z] \\
\vdash & [q_0, bcba, AZ] \\
\vdash & [q_0, cba, BAZ] \\
\vdash & [q_1, ba, BAZ] \\
\vdash & [q_1, a, AZ] \\
\vdash & [q_1, \varepsilon, Z] \\
\vdash & [q_2, \varepsilon, Z]
\end{array}
$$



PDA…

- Consider $w = aabcaaa$

$$
\begin{aligned}
& [q_0, aabcaaa, Z] \\
\vdash\ & [q_0, abcaaa, AZ] \\
\vdash\ & [q_0, bcaaa, AAZ] \\
\vdash\ & [q_0, caaa, BAAZ] \\
\vdash\ & [q_1, aaa, \mathbf{B}AAZ]
\end{aligned}
$$



- dead configuration, $w = aabcaaa \notin L$

NPDA…

- A deterministic pushdown accepter (which we have not yet considered) ***must have only one transition*** for any given input symbol and stack symbol.

- A nondeterministic pushdown accepter ***may have no transition or several transitions*** defined for a particular input symbol and stack symbol.

- In a npda, there may be several ***“paths***” to follow to process a given input string. Some of the paths may result in accepting the string. Other paths **may end in a non-accepting state**.

- As with an nfa, an npda magically (and correctly) “guesses” which path to follow through the machine in order to accept a string (if the string is in the language).

NPDA...

- Language: $L = \{ww^R : w \in \{a, b\}^*\}$
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where
  - $Q = \{q_0, q_1, q_2\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{Z, a, b\}$
  - δ
  - q_0 is the start state
  - Z is the initial stack symbol
  - $F = \{q_3\}$
- Explanation
  - stack a's remember input a's
  - stack b's remember input b's
  - switch states when required
  - pop a's and b's as long as they match
  - λ-move to final state if Z at top of stack

$\delta(q_0, a, Z) = \{(q_0, aZ)\}$
$\delta(q_0, b, Z) = \{(q_0, bZ)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, a) = \{(q_0, ba)\}$
$\delta(q_0, a, b) = \{(q_0, ab)\}$
$\delta(q_0, b, b) = \{(q_0, bb)\}$

remember input string

must be ε

$\delta(q_0, a, a) = \{(q_1, a)\}$
$\delta(q_0, b, b) = \{(q_1, b)\}$

found center (magically)

$\delta(q_1, a, a) = \{(q_1, \lambda)\}$
$\delta(q_1, b, b) = \{(q_1, \lambda)\}$

pop if match, else die

$\delta(q_1, \lambda, Z) = \{(q_2, \lambda)\}$

goto final if good string

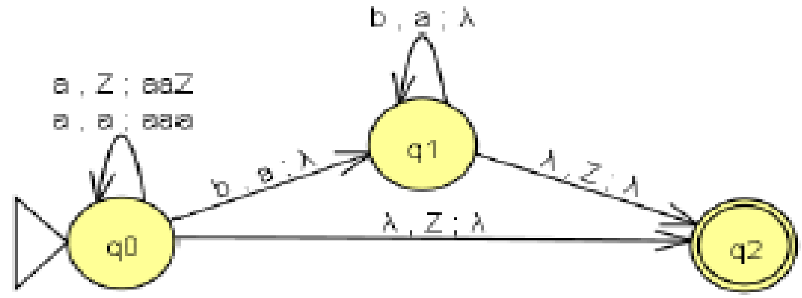
NPDA…

- Language: $L = \{a^n b^{2n} : n \geq 0\}$
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where
  - $Q = \{q_0, q_1, q_2\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{Z, a\}$
  - δ
  - q_0 is the start state
  - Z is the initial stack symbol
  - $F = \{q_2\}$
- Explanation
  - 2 stack a's for each input a
  - switch states and start popping when b appears
  - pop a's as long as they match input b's
  - λ-move to final state if Z at top of stack

$\delta(q_0, a, Z) = \{(q_0, aaZ)\}$ $\delta(q_0, a, a) = \{(q_0, aaa)\}$ $\delta(q_0, \lambda, Z) = \{(q_2, \lambda)\}$	two a's per input a also $\lambda \in L$
$\delta(q_0, b, a) = \{(q_1, \lambda)\}$ $\delta(q_1, b, a) = \{(q_1, \lambda)\}$	pop matching a's, else die
$\delta(q_1, \lambda, Z) = \{(q_2, \lambda)\}$	goto final if good string

NPDA…

- Language: $L = \{a^n b^{2n} : n \geq 0\}$



- Consider $w = aabbbb \in L$

$(q_0, aabbbb, Z) \vdash (q_0, abbbb, aaZ) \vdash (q_0, bbbb, aaaaZ) \vdash$
$(q_1, bbb, aaaZ) \vdash (q_1, bb, aaZ) \vdash (q_1, b, aZ) \vdash (q_1, \lambda, Z) \vdash (q_2, \lambda, \lambda)$

- Consider $u = aabb \notin L$

$(q_0, aabb, Z) \vdash (q_0, abb, aaZ) \vdash (q_0, bb, aaaaZ) \vdash$
$(q_1, b, aaaZ) \vdash (q_1, \lambda, aaZ)$

dead configuration, $u = aabb \notin L$

- Consider $v = \lambda \in L$

$(q_0, \lambda, Z) \vdash (q_2, \lambda, \lambda)$

PDA & CFL

- Every CFL is accepted by PDA.
  - For any CFL L, there exists a PDA M such that $L(M) = L$.
  - The reverse is true as well.

- Let G be the CFG of L such that $L(G) = L$.

- Construct a PDA M such that $L(M) = L(G) = L$.
  - M is constructed from CFG G.
  - CFG $\Rightarrow$ PDA

PDA & CFL…

- Given a context-free grammar in GNF, the basic idea is to construct a npda that does a leftmost derivation of any string in the language.
- Rules:
  - always have ε-production from start state to push *S* onto the stack.
  - push NTs on the right hand side onto the stack.
  - the single terminal on the right hand side is treated as input.
  - NT on the left hand side is the top of the stack to be popped.
  - have ε-production to accepting state if *Z* on top of stack.

PDA & CFL…

- Always start with $\delta(q_0, \varepsilon, Z) = (q_1, SZ)$
  - begin in state q_0, pop Z, move to q_1 without reading input, push SZ.
- Repeatedly apply rule.
  - If $A \to aX$ add $\delta(q_1, a, A) = (q_1, X)$
  - note always start and end in state q_1
  - begin in state q_1, pop A, move to state q_1 while reading input a, push X.
- Always end with $\delta(q_1, \varepsilon, Z) = (q_f, \varepsilon)$
  - note Z must be at top of stack.
  - begin in state q_1, pop Z, move to state q_f without reading input symbol.

PDA & CFL…

- Input Grammar in Greibach NF $G = (NT, \Sigma, P, S)$.

- Output NPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$
  - $Q = \{q_0, q_1, q_f\}$, $\Sigma = \Sigma$, $\Gamma = NT \cup \{Z\}$, $F = \{q_f\}$

- δ:
  - $\delta(q_0, \varepsilon, Z) \quad = (q_1, SZ)$ //always
  - $\delta(q_1, a, A) \quad = (q_1, w)$ if $A \to aw \in P$
  - $\delta(q_1, \varepsilon, Z) \quad = (q_f, \varepsilon)$ //always

PDA & CFL…

- Simple example:
  - CFG $G = (\{S,A\},\{a,b\}, S, \{S \to aSA \mid a, A \to aA \mid b\})$

- production transition

production	transition
(always)	$\delta(q_0, \varepsilon, Z) = \{(q_1, SZ)\}$
$S \to aSA \mid a\varepsilon$	$\delta(q_1, a, S) = \{(q_1, SA), (q_1, \varepsilon)\}$
$A \to aA$	$\delta(q_1, a, A) = \{(q_1, A)\}$
$A \to b\varepsilon$	$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$
(always)	$\delta(q_1, \varepsilon, Z) = \{(q_f, \varepsilon)\}$

PDA & CFL…

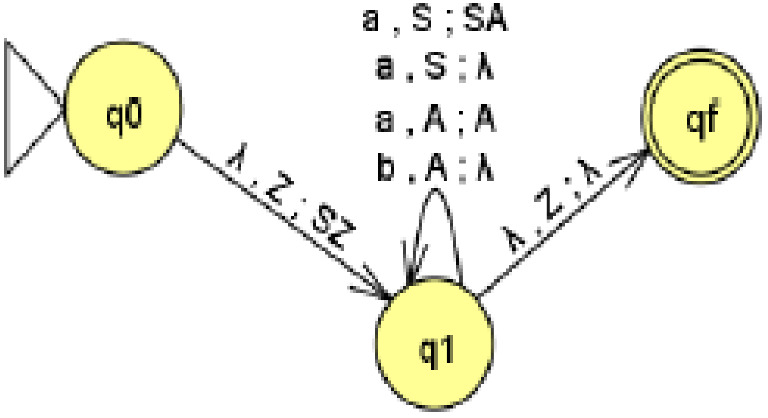
- GNF grammar

$$S \to aSA \mid a$$
$$A \to aA$$
$$A \to b$$

- Derivation of $w = aaabb$

$$S \Rightarrow aSA \Rightarrow$$
$$aaSAA \Rightarrow aaaAA \Rightarrow$$
$$aaabA \Rightarrow aaabb$$

- Equivalent npda
  - recall: (input, popped, pushed)



- Acceptance of $w = aaabb$

$$(q_0, aaabb, Z) \vdash (q_1, aaabb, SZ) \vdash$$
$$(q_1, aabb, SAZ) \vdash (q_1, abb, SAAZ) \vdash$$
$$(q_1, bb, AAZ) \vdash (q_1, b, AZ) \vdash$$
$$(q_1, \lambda, Z) \vdash (q_f, \lambda, \lambda)$$

PDA & CFL…

- Let $G_{GNF} = (V, \Sigma, S, P) = (\{S, A, B, C\}, \{a, b, c\}, S, P)$ have production rules

  $S \longrightarrow aA$

  $A \longrightarrow aABC \mid bB \mid a$

  $B \longrightarrow b$

  $C \longrightarrow c$

- Convert to a npda

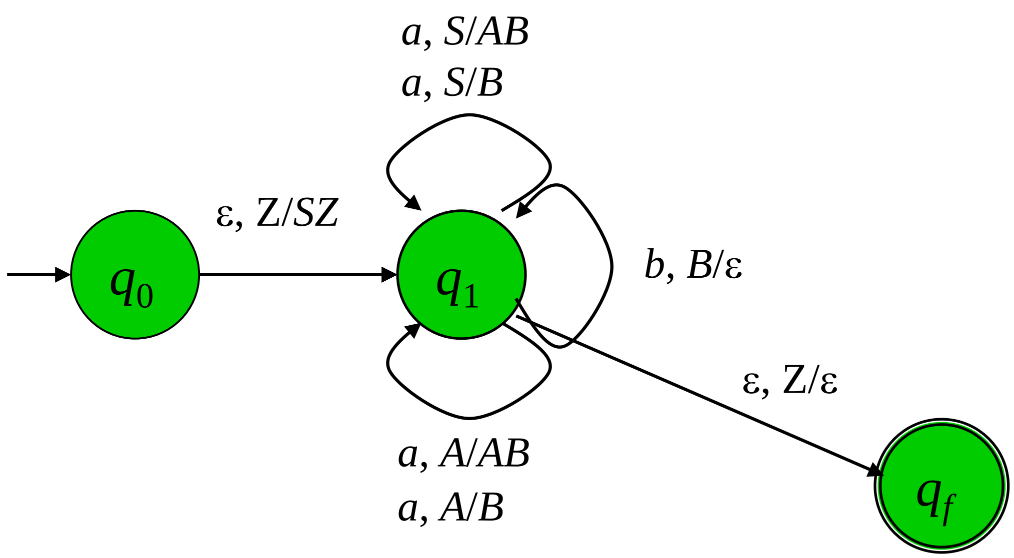
production	transitions (δ)
(always)	1. $\delta(q_0, \lambda, Z) = \{(q_1, SZ)\}$
$S \longrightarrow aA$	2. $\delta(q_1, a, S) = \{(q_1, A)\}$
$A \longrightarrow aABC \mid a$	3. $\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$
$A \longrightarrow bB$	4. $\delta(q_1, b, A) = \{(q_1, B)\}$
$B \longrightarrow b$	5. $\delta(q_1, b, B) = \{(q_1, \lambda)\}$
$C \longrightarrow c$	6. $\delta(q_1, c, C) = \{(q_1, \lambda)\}$
(always)	7. $\delta(q_1, \lambda, Z) = \{(q_f, \lambda)\}$

- Thus, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F) = (\{q_0, q_1, q_f\}, \Sigma, V \cup \{Z\}, \delta, q_0, Z, \{q_f\})$

PDA & CFL…

- Input CFG $G = \{\{S, A, B\}, \{a, b\}, S, P\}$
  - P:
    - $S \rightarrow aAB \mid aB$
    - $A \rightarrow aAB \mid aB$
    - $B \rightarrow b$
- What is NPDA?

What is $L(G)$?



PDA & CFL…

Computation of *aaabbb*

$$
\begin{array}{lll}
S & \Rightarrow & aAB \\
 & \Rightarrow & aaABB \\
 & \Rightarrow & aaaBBB \\
 & \Rightarrow & aaabBB \\
 & \Rightarrow & aaabbB \\
 & \Rightarrow & aaabbb
\end{array}
$$

$$
\begin{array}{llll}
 & [q_0, & aaabbb, & Z] \\
\vdash & [q_1, & aaabbb, & SZ] \\
\vdash & [q_1, & aabbb, & ABZ] \\
\vdash & [q_1, & abbb, & ABBZ] \\
\vdash & [q_1, & bbb, & BBBZ] \\
\vdash & [q_1, & bb, & BBZ] \\
\vdash & [q_1, & b, & BZ] \\
\vdash & [q_1, & \varepsilon, & Z] \\
\vdash & [q_f, & \varepsilon, & \varepsilon]
\end{array}
$$

On your own, draw computation trees for other strings not in the language and see that they are not accepted.

PDA & CFL…

- Let CFG $G = (\{S, A, B\}, \{a, b\}, S, P)$ where P is
  - $S \rightarrow aAA$ $A \rightarrow aB \mid bB \mid a$ $B \rightarrow c$
- Construct NPDA M:
  - $(\{q_0, q_1, q_f\}, \{a, b\}, \{S, A, B, Z\}, \delta, q_0, \{q_f\})$
- where
  - $\delta(q_0, \varepsilon, Z) = (q_1, SZ)$
  - $\delta(q_1, a, S) = (q_1, AA)$ $S \rightarrow aAA$
  - $\delta(q_1, a, A) = \{(q_1, B), (q_1, \varepsilon)\}$ $A \rightarrow aB \mid a\varepsilon$
  - $\delta(q_1, b, A) = (q_1, B)$ $A \rightarrow bB$
  - $\delta(q_1, c, B) = (q_1, \varepsilon)$ $B \rightarrow c\varepsilon$
  - $\delta(q_1, \varepsilon, Z) = (q_f, \varepsilon)$

Deterministic PDA

- A PDA is **deterministic** if its transition function satisfies **both** of the following properties.
  - For all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $X \in \Gamma$,
    - the set $\delta(q, a, X)$ has ***at most*** one element.
    - there is only one move for any input and stack combination.
  - For all $q \in Q$ and $X \in \Gamma$,
    - if $\delta(q, \varepsilon, X) \neq \{\}$, then $\delta(q, a, X) = \{\}\ \forall\ a \in \Sigma$
    - an ε-transition has no input-consuming alternatives, i.e., there cannot exist another move with stack = X from the *same state q*.

Deterministic PDA…

- A language L is a deterministic context-free language if and only if there is a DPDA that accepts L.

- Some context-free languages which are initially described in a nondeterministic way via a NPDA can also be described in a deterministic way via DPDA.

- Some context-free languages are inherently nondeterministic, e.g., $L = \{w \in (a|b)^* : w = w^R\}$ cannot be accepted by any dpda.

- Deterministic PDA (DPDA) can only represent a subset of CFL, e.g., L = $\{ww^R \mid w \in (a|b)^*\}$ cannot be represented by DPDA.

- A key point in all this is that the equivalence between deterministic and nondeterministic finite automata is not found with deterministic and nondeterministic pushdown automata.

- Unless otherwise stated, we assume that a PDA is nondeterministic.

Deterministic PDA…

$L = \{a^n \mid n \geq 0\} \cup \{a^n b^n \mid n \geq 0\}$ is CFL and accepted by a **non-deterministic** PDA M.

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{Z, A\}$, q_0, Z, $F = \{q_2\}$

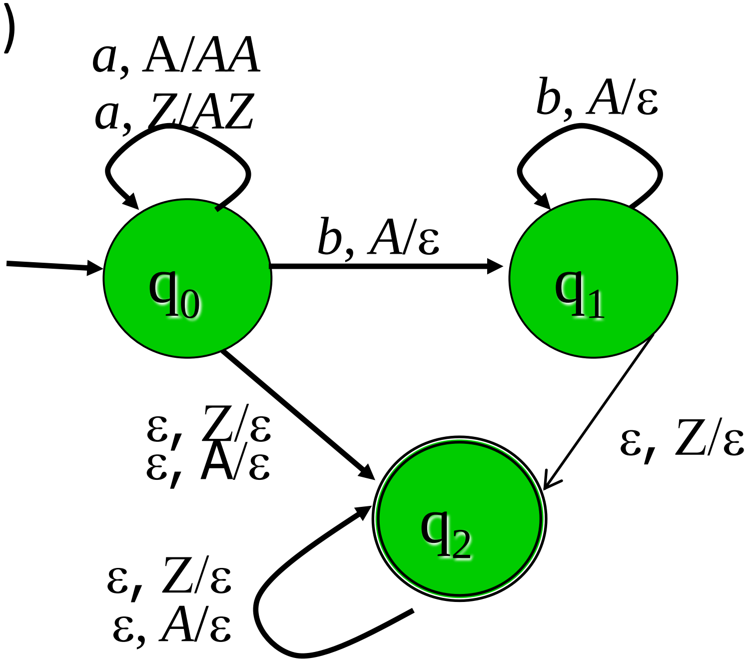
$\delta(q_0, a, Z) = (q_0, AZ)$, $\delta(q_0, b, A) = (q_1, \varepsilon)$

$\delta(q_0, \varepsilon, Z) = (q_2, \varepsilon)$ $\delta(q_1, b, A) = (q_1, \varepsilon)$

$\delta(q_0, \varepsilon, Z) = (q_2, \varepsilon)$ $\delta(q_1, \varepsilon, Z) = (q_1, \varepsilon)$

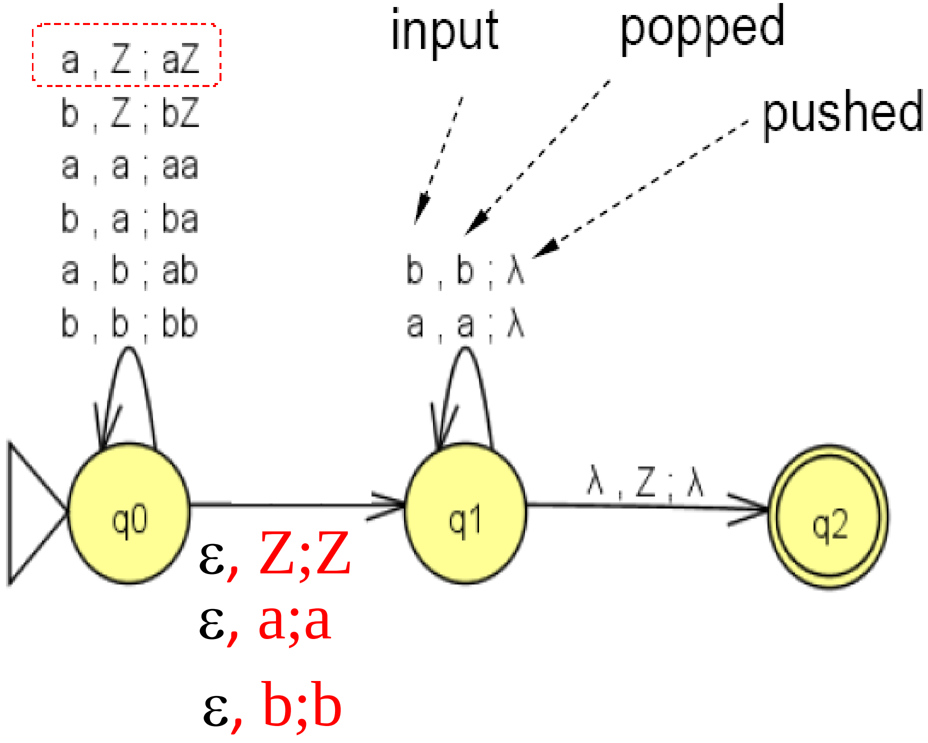
$\delta(q_2, \varepsilon, Z) = (q_2, \varepsilon)$ $\delta(q_2, \varepsilon, A) = (q_2, \varepsilon)$

$\delta(q_0, a, A) = (q_0, AA)$



Deterministic PDA…

The language of (strings over {*a*, *b*} of *even* length and spelled the same forwards and backwards) = $\{ww^R \mid w \in \{a, b\}^*\}$ is CFL and accepted by a **non-deterministic** PDA M.



Deterministic PDA…

- Language $L = \{w \in \{a, b\}^* : n_a(w) > n_b(w)\}$ accepted via a npda

$\delta(q_0, a, Z) = \{(q_0, aZ)\}$

$\delta(q_0, b, Z) = \{(q_0, bZ)\}$

$\delta(q_0, a, a) = \{(q_0, aa)\}$

$\delta(q_0, b, b) = \{(q_0, bb)\}$

$\delta(q_0, a, b) = \{(q_0, \lambda)\}$

$\delta(q_0, b, a) = \{(q_0, \lambda)\}$

$\delta(q_0, \lambda, a) = \{(q_1, a)\}$

This is a npda because these transitions violate the 2nd rule associated with dpda,

$$\delta(q, \lambda, b) \neq \emptyset \Rightarrow \forall c \in \Sigma \quad \delta(q, c, b) = \emptyset$$

- Operation of npda
  - start in state q_0, read first symbol and push it onto the stack, then…
  - if input and stack symbols match, push both symbols onto the stack
  - if input and stack symbols differ, discard both
  - when no more input, if a on top of stack, λ-move to accepting state q_1

Deterministic PDA…

- Same language $L = \{w \in \{a, b\}^* : n_a(w) > n_b(w)\}$ accepted via a dpda

$\delta(q_0, a, Z) = \{(q_1, Z)\}$

$\delta(q_0, b, Z) = \{(q_0, bZ)\}$

$\delta(q_0, a, b) = \{(q_0, \lambda)\}$

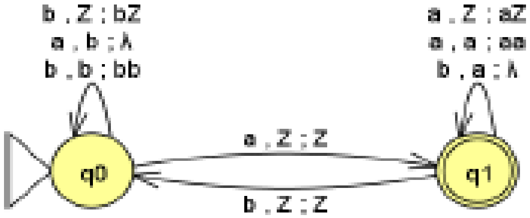
$\delta(q_0, b, b) = \{(q_0, bb)\}$

$\delta(q_1, a, Z) = \{(q_1, aZ)\}$

$\delta(q_1, b, Z) = \{(q_0, Z)\}$

$\delta(q_1, a, a) = \{(q_1, aa)\}$

$\delta(q_1, b, a) = \{(q_1, \lambda)\}$



$L = \{w \in \{a, b\}^* : n_a(w) > n_b(w)\}$ is accepted by a dpda. By definition 7.4, it is therefore a deterministic context-free language

- Operation of dpda
  - state q_0 means $n_a(w) \leq n_b(w)$
  - state q_1 means $n_a(w) > n_b(w)$
  - jump between states based on input and current top of stack
  - when input ends, halt; q_1 is accepting state

Deterministic PDA…

- 7.3.1 show $L = \{a^n b^{2n} : n \geq 0\}$ is a deterministic context-free language
- Per definition 7.4 we need to find a dpda that accepts L
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$
- $M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, Z\}, \delta, q_0, Z, \{q_0, q_3\})$

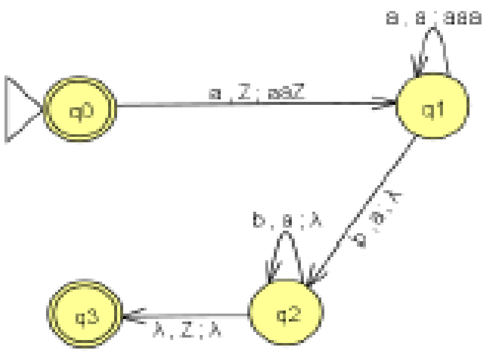
  $\delta(q_0, a, Z) = \{(q_1, aaZ)\}$

  $\delta(q_1, a, a) = \{(q_1, aaa)\}$

  $\delta(q_1, b, a) = \{(q_2, \lambda)\}$

  $\delta(q_2, b, a) = \{(q_2, \lambda)\}$

  $\delta(q_2, \lambda, Z) = \{(q_3, \lambda)\}$

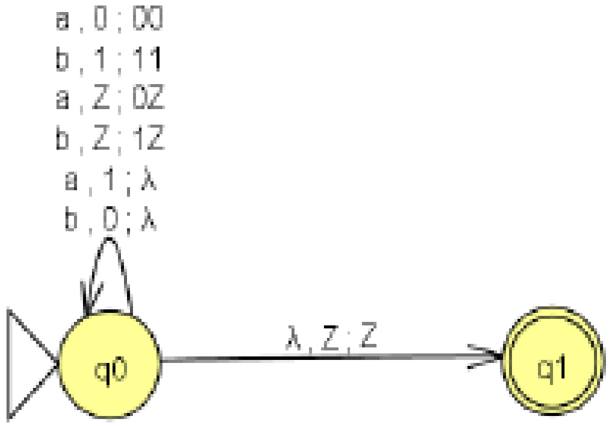


- Operation of dpda
  - state q_0 accepts λ, if input is a, push 2 a's and goto q_1
  - state q_1 pushes 2 a's for each input a , if input is b, pop a and goto q_2
  - state q_2 pops a for each b, λ-move to q_3 if Z on top and no more input
  - state q_3 accepts $a^n b^{2n} : n > 0$

Deterministic PDA…

- 7.3.5 show example 7.4 is a npda but that $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$ is a deterministic context-free language.
- Machine is npda

$$\delta(q_0, a, 0) = \{(q_0, 00)\}$$
$$\delta(q_0, b, 1) = \{(q_0, 11)\}$$
$$\delta(q_0, a, Z) = \{(q_0, 0Z)\}$$
$$\delta(q_0, b, Z) = \{(q_0, 1Z)\}$$
$$\delta(q_0, a, 1) = \{(q_0, \lambda)\}$$
$$\delta(q_0, b, 0) = \{(q_0, \lambda)\}$$
$$\delta(q_0, \lambda, Z) = \{(q_1, Z)\}$$



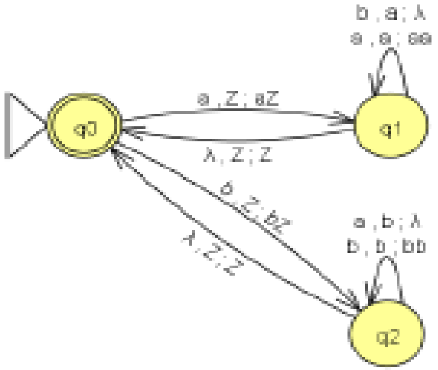
This is a npda because these transitions violate the 2nd rule associated with dpda,

$$\delta(q_0, \lambda, Z) \neq \emptyset \Rightarrow \forall c \in \Sigma \quad \delta(q_0, c, Z) = \emptyset$$

Deterministic PDA…

- 7.3.5 (continued) show that $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$ is a deterministic context-free language. $\delta(q, \lambda, b) \neq \emptyset \Rightarrow \forall c \in \Sigma \quad \delta(q, c, b) = \emptyset$
- A dpda that accepts L; $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z\}, \delta, q_0, Z, \{q_0\})$

$$\delta(q_0, a, Z) = \{(q_1, aZ)\}$$
$$\delta(q_0, b, Z) = \{(q_2, bZ)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_1, \lambda)\}$$
$$\delta(q_1, \lambda, Z) = \{(q_0, Z)\}$$
$$\delta(q_2, b, b) = \{(q_2, bb)\}$$
$$\delta(q_2, a, b) = \{(q_2, \lambda)\}$$
$$\delta(q_2, \lambda, Z) = \{(q_0, \lambda)\}$$



- Operation of dpda
  - state q_0 accepts strings in the language ($n_a(w) = n_b(w)$ including λ)
  - state q_1 adds a's and subtracts b's; on empty stack λ-move back to q_0
  - state q_2 adds b's and subtracts a's; on empty stack λ-move back to q_0