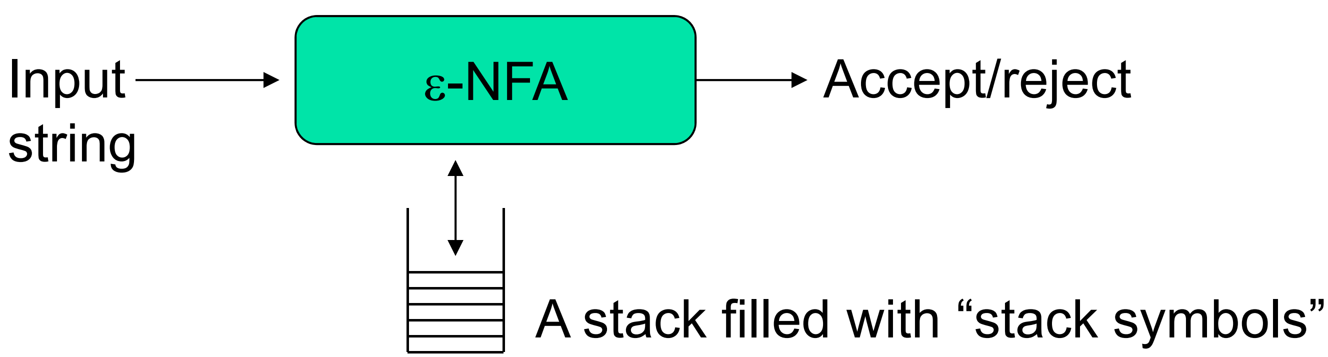
Pushdown Automata (PDA)

Reading: Chapter 6

PDA - the automata for CFLs

- What is?
  - FA to Reg Lang, PDA is to CFL
- PDA == [ε -NFA + "a stack"]
- Why a stack?



Pushdown Automata - Definition

- A PDA P := (Q,∑,Γ, δ,q_0,Z_0,F):
  - Q: states of the ε-NFA
  - ∑: input alphabet
  - Γ : stack symbols
  - δ: transition function
  - q_0: start state
  - Z_0: Initial stack top symbol
  - F: Final/accepting states

old state *input symb.* *Stack top* *new state(s)* *new Stack top(s)*

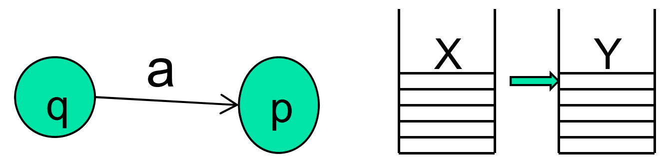
$\delta : Q \times \Sigma \times \Gamma \Rightarrow Q \times \Gamma$

δ : The Transition Function

δ(q,a,X) = {(p,Y), …}

1. state transition from q to p
2. a is the next input symbol
3. X is the current stack *top* symbol
4. Y is the replacement for X; it is in Γ^* (a string of stack symbols)
   i. Set Y = ε for: Pop(X)
   ii. If Y=X: stack top is unchanged
   iii. If $Y=Z_1Z_2\ldots Z_k$: X is popped and is replaced by Y in reverse order (i.e., Z_1 will be the new stack top)

Non-determinism



	Y = ?	Action
i)	Y=ε	Pop(X)
ii)	Y=X	Pop(X) Push(X)
iii)	$Y=Z_1Z_2..Z_k$	Pop(X) Push(Z_k) Push(Z_{k-1}) … Push(Z_2) Push(Z_1)

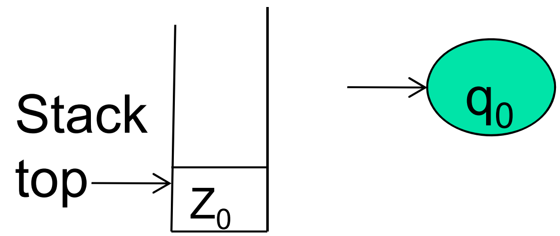
Example

Let $L_{wwr} = \{ww^R \mid w \text{ is in } (0+1)^*\}$

- CFG for L_{wwr} : $S ==> 0S0 \mid 1S1 \mid \varepsilon$
- PDA for L_{wwr} :
- $P := (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

  $= (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$

Initial state of the PDA:



PDA for L_{wwr}

1. $\delta(q_0,0, Z_0)=\{(q_0,0Z_0)\}$
2. $\delta(q_0,1, Z_0)=\{(q_0,1Z_0)\}$

First symbol push on stack

3. $\delta(q_0,0, 0)=\{(q_0,00)\}$
4. $\delta(q_0,0, 1)=\{(q_0,01)\}$
5. $\delta(q_0,1, 0)=\{(q_0,10)\}$
6. $\delta(q_0,1, 1)=\{(q_0,11)\}$

Grow the stack by pushing new symbols on top of old (w-part)

7. $\delta(q_0, \varepsilon, 0)=\{(q_1, 0)\}$
8. $\delta(q_0, \varepsilon, 1)=\{(q_1, 1)\}$
9. $\delta(q_0, \varepsilon, Z_0)=\{(q_1, Z_0)\}$

Switch to popping mode, nondeterministically (boundary between w and w^R)

10. $\delta(q_1,0, 0)=\{(q_1, \varepsilon)\}$
11. $\delta(q_1,1, 1)=\{(q_1, \varepsilon)\}$

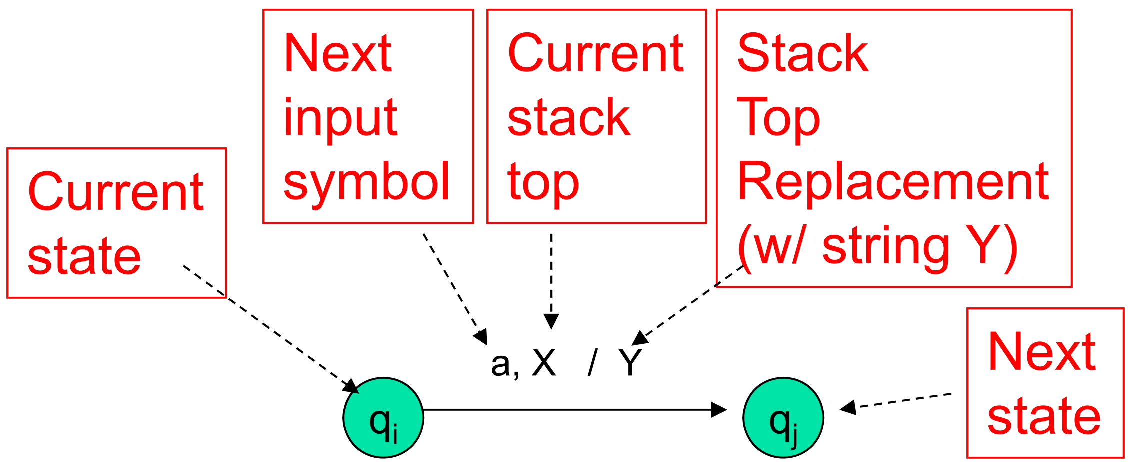
Shrink the stack by popping matching symbols (w^R-part)

12. $\delta(q_1, \varepsilon, Z_0)=\{(q_2, Z_0)\}$

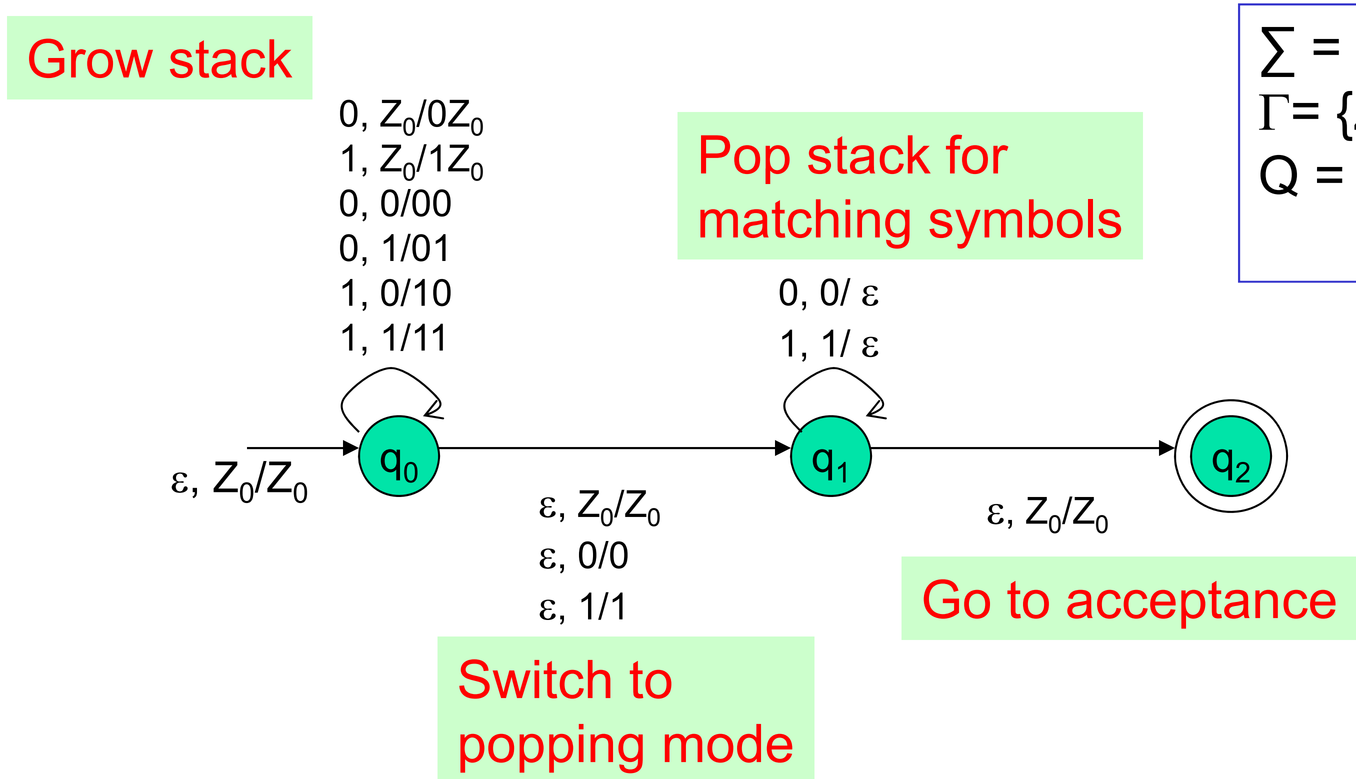
Enter acceptance state

PDA as a state diagram

$\delta(q_i, a, X) = \{(q_j, Y)\}$



PDA for L_{wwr}: Transition Diagram

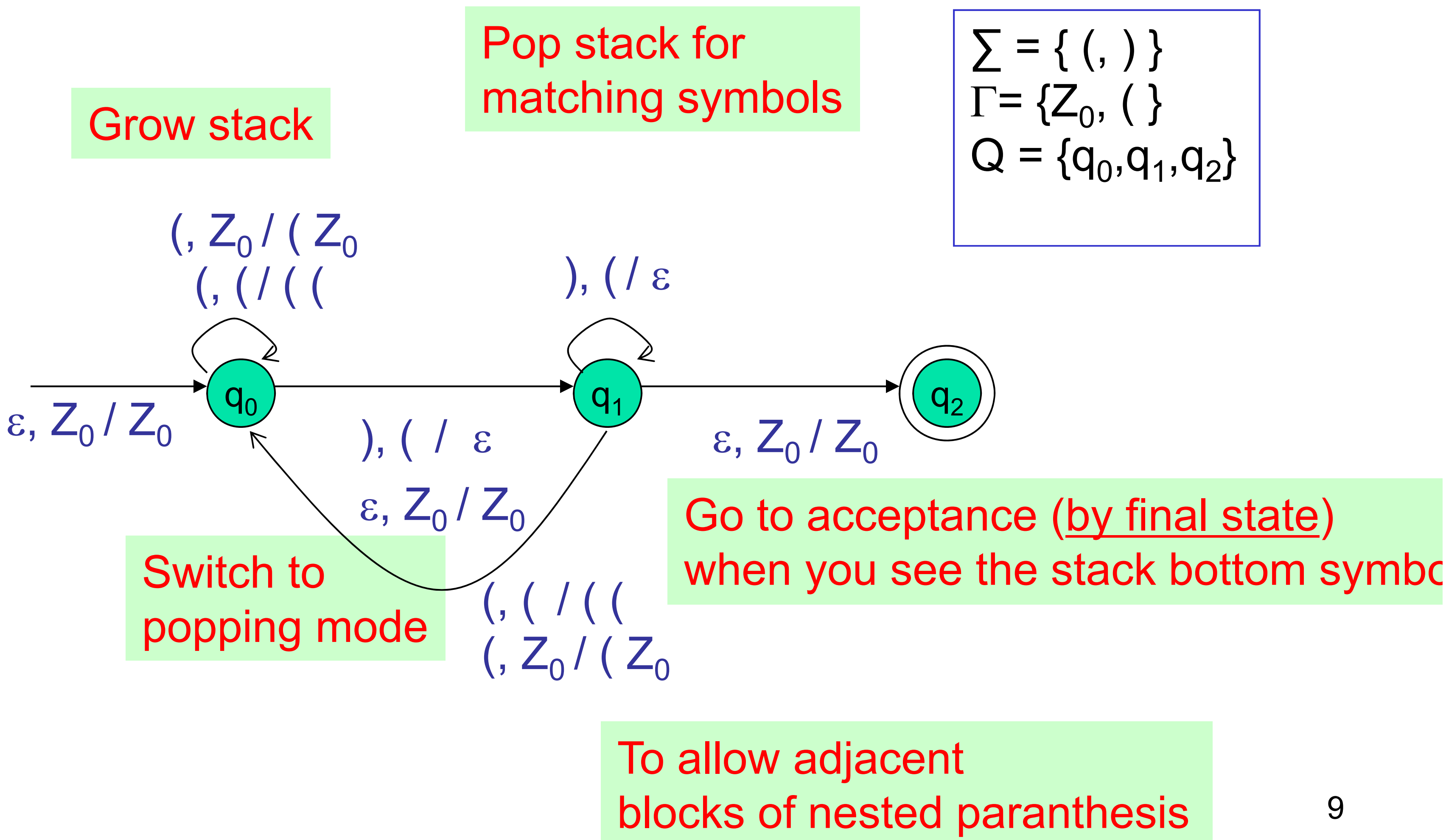


$\Sigma = \{0, 1\}$
$\Gamma = \{Z_0, 0, 1\}$
$Q = \{q_0, q_1, q_2\}$

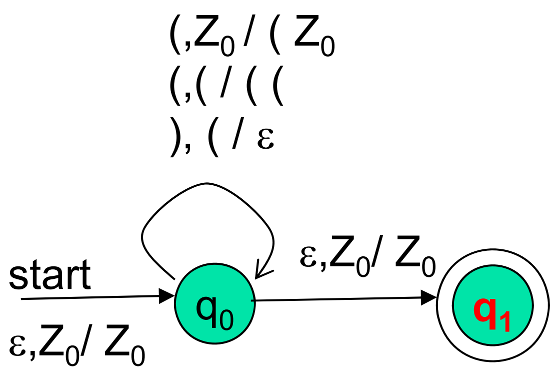
This would be a non-deterministic PDA

Example 2: language of balanced paranthesis

$\Sigma = \{ (,) \}$
$\Gamma = \{Z_0, (\}$
$Q = \{q_0, q_1, q_2\}$



Example 2: language of balanced paranthesis (another design)



$\Sigma = \{ (,) \}$
$\Gamma = \{Z_0, (\}$
$Q = \{q_0, q_1\}$

PDA's Instantaneous Description (ID)

A PDA has a configuration at any given instance:

(q,w,y)

- q - current state
- w - remainder of the input (i.e., unconsumed part)
- y - current stack contents as a string from top to bottom of stack

---

If δ(q,a, X)={(p, A)} is a transition, then the following are also true:

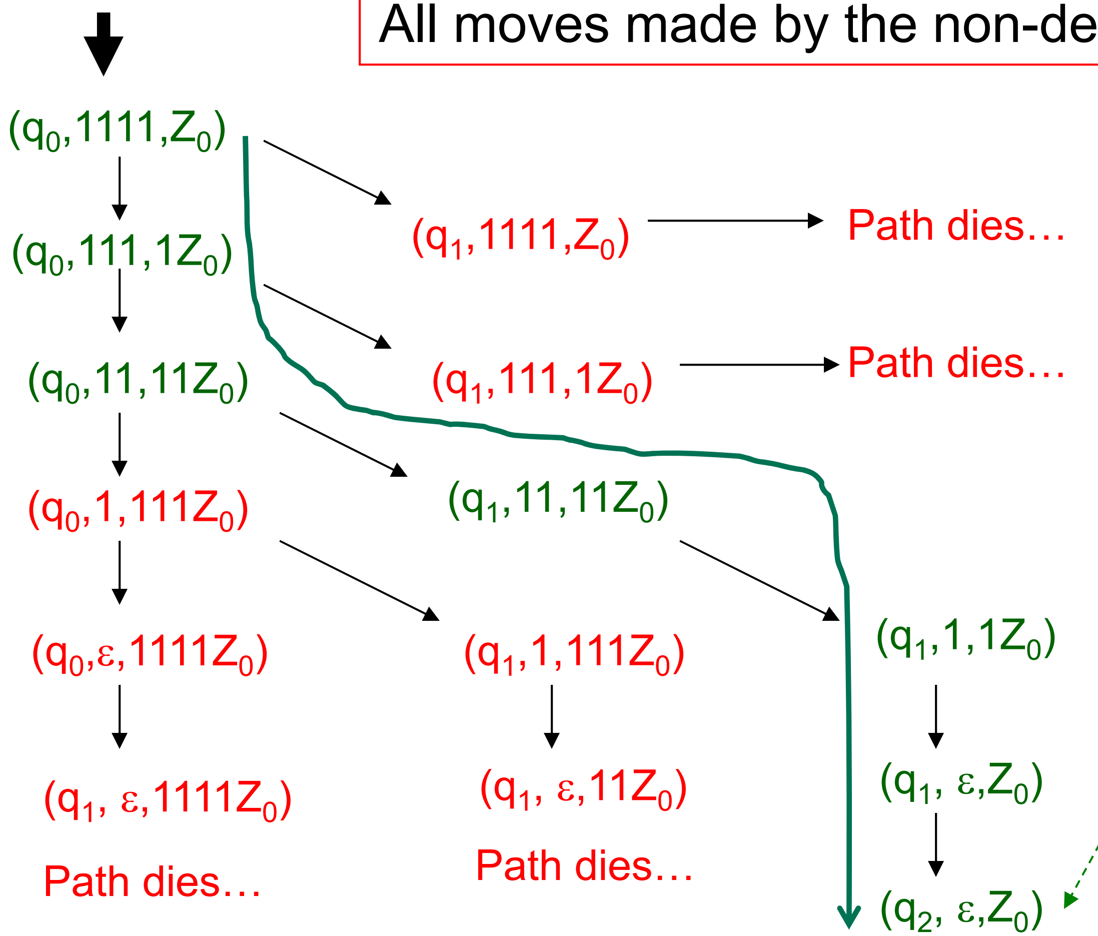
- (q, a, X) |--- (p,ε,A)
- (q, aw, XB) |--- (p,w,AB)

|--- sign is called a "turnstile notation" and represents one move

|---* sign represents a sequence of moves

How does the PDA for L_{wwr} work on input "1111"?

All moves made by the non-deterministic PDA



Acceptance by final state:

= *empty input* AND *final state*

Principles about IDs

- Theorem 1: If for a PDA, (q, x, A) |---* (p, y, B), then for any string $w \in \Sigma^*$ and $\gamma \in \Gamma^*$, it is also true that:
  - (q, x w, A γ) |---* (p, y w, B γ)

- Theorem 2: If for a PDA, (q, x w, A) |---* (p, y w, B), then it is also true that:
  - (q, x, A) |---* (p, y, B)

There are two types of PDAs that one can design: those that accept by final state or by empty stack

Acceptance by…

- ***PDAs that accept by final state:***
  - For a PDA P, the language accepted by P, denoted by L(P) by *final state*, is:
    - $\{w \mid (q_0,w,Z_0) \mid\!\text{---}^* (q,\varepsilon, A)\}$, s.t., $q \in F$

Checklist:
- input exhausted?
- in a final state?

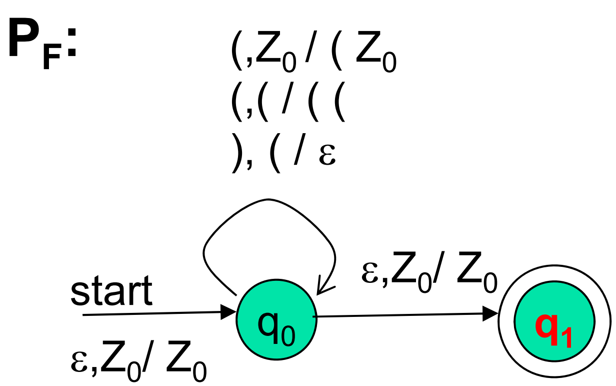
- ***PDAs that accept by empty stack:***
  - For a PDA P, the language accepted by P, denoted by N(P) by *empty stack*, is:
    - $\{w \mid (q_0,w,Z_0) \mid\!\text{---}^* (q, \varepsilon, \varepsilon)\}$, for any $q \in Q$.

Q) Does a PDA that accepts by empty stack need any final state specified in the design?

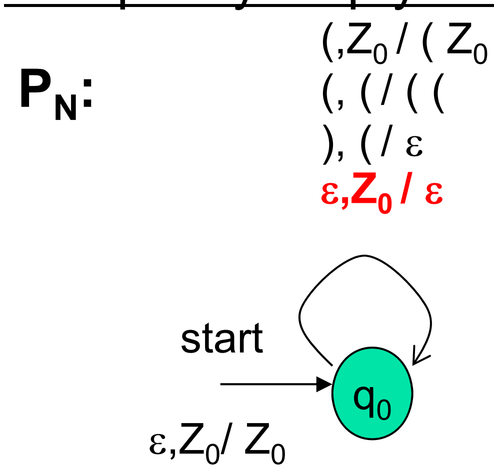
Checklist:
- input exhausted?
- is the stack empty?

Example: L of balanced parenthesis

PDA that accepts by final state



An equivalent PDA that accepts by empty stack



How will these two PDAs work on the input: ((( )) ( )) ( )

PDA for L_{wwr}: Proof of correctness

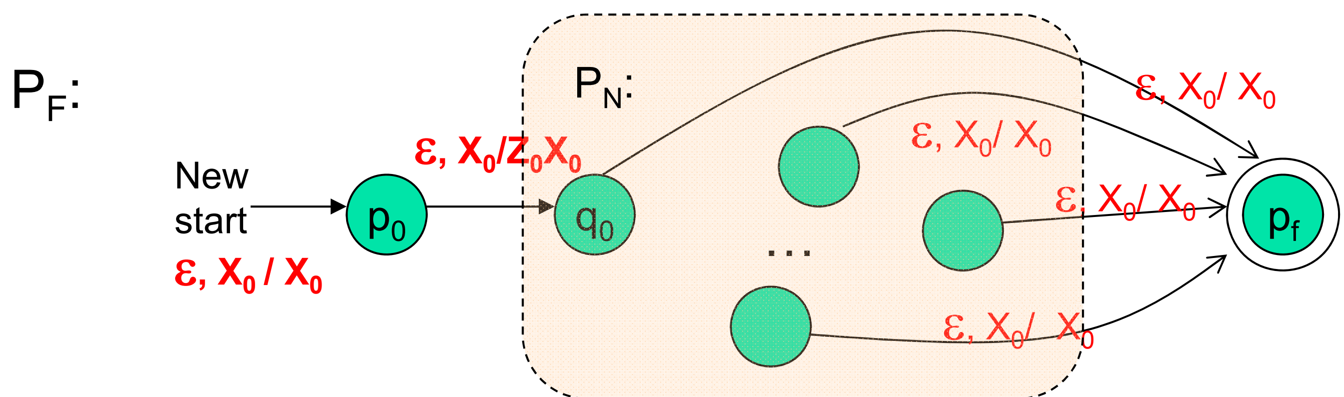
- Theorem: The PDA for L_{wwr} accepts a string x by final state if and only if x is of the form ww^R.
- Proof:
  - *(if-part)* If the string is of the form ww^R then there exists a sequence of IDs that leads to a final state: (q_0,ww^R,Z_0) |---* (q_0,w^R,wZ_0) |---* (q_1,w^R,wZ_0) |---* (q_1, ε,Z_0) |---* $(\mathbf{q_2}, \varepsilon,Z_0)$
  - *(only-if part)*
    - Proof by induction on |x|

PDAs accepting by final state and empty stack are equivalent

- P_F <= PDA accepting by final state
  - $P_F = (Q_F, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$
- P_N <= PDA accepting by empty stack
  - $P_N = (Q_N, \Sigma, \Gamma, \delta_N, q_0, Z_0)$
- Theorem:
  - (P_N==> P_F) For every P_N, there exists a P_F s.t. $L(P_F)=L(P_N)$
  - (P_F==> P_N) For every P_F, there exists a P_N s.t. $L(P_F)=L(P_N)$

P_N==> P_F construction

- Whenever P_N's stack becomes empty, make P_F go to a final state without consuming any addition symbol
- To detect empty stack in P_N: P_F pushes a new stack symbol X_0 (not in Γ of P_N) initially before simultating P_N



$P_F = (Q_N \cup \{p_0,p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$

Example: Matching parenthesis “(” “)”

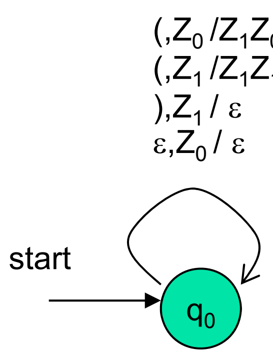
P_N: $(\{q_0\}, \{(,)\}, \{Z_0,Z_1\}, \delta_N, q_0, Z_0)$

δ_N:

$\delta_N(q_0,(,Z_0) = \{ (q_0,Z_1Z_0) \}$
$\delta_N(q_0,(,Z_1) = \{ (q_0, Z_1Z_1) \}$

$\delta_N(q_0,),Z_1) = \{ (q_0, \varepsilon) \}$

$\delta_N(q_0, \varepsilon,Z_0) = \{ (q_0, \varepsilon) \}$

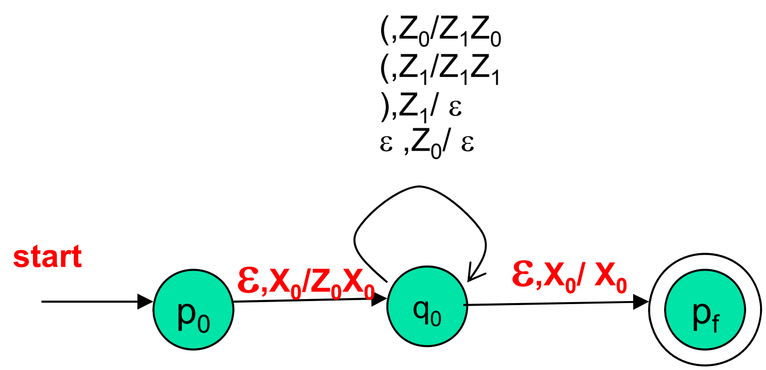


Accept by empty stack

P_f: $(\{p_0,q_0,p_f\}, \{(,)\}, \{X_0,Z_0,Z_1\}, \delta_f, p_0, X_0, p_f)$

δ_f:

$\delta_f(p_0, \varepsilon,X_0) = \{ (q_0,Z_0) \}$
$\delta_f(q_0,(,Z_0) = \{ (q_0,Z_1 Z_0) \}$
$\delta_f(q_0,(,Z_1) = \{ (q_0, Z_1Z_1) \}$
$\delta_f(q_0,),Z_1) = \{ (q_0, \varepsilon) \}$
$\delta_f(q_0, \varepsilon,Z_0) = \{ (q_0, \varepsilon) \}$
$\delta_f(p_0, \varepsilon,X_0) = \{ (p_f, X_0) \}$



Accept by final state

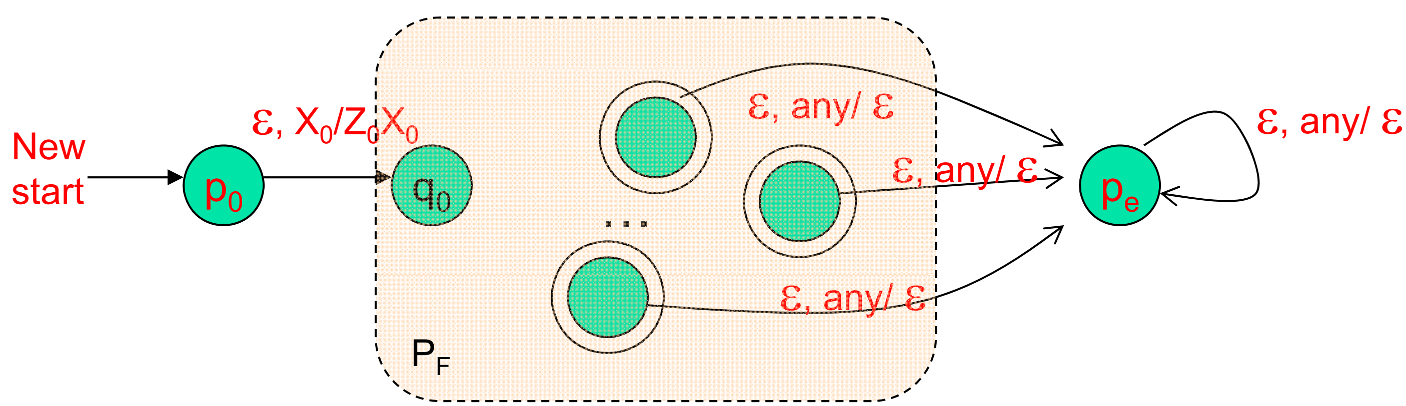
P_F ==> P_N construction

- <u>Main idea:</u>
  - Whenever P_F reaches a final state, just make an ε-transition into a new end state, clear out the stack and accept
  - Danger: What if P_F design is such that it clears the stack midway *without* entering a final state?

    ➔ to address this, add a new start symbol X_0 (not in Γ of P_F)

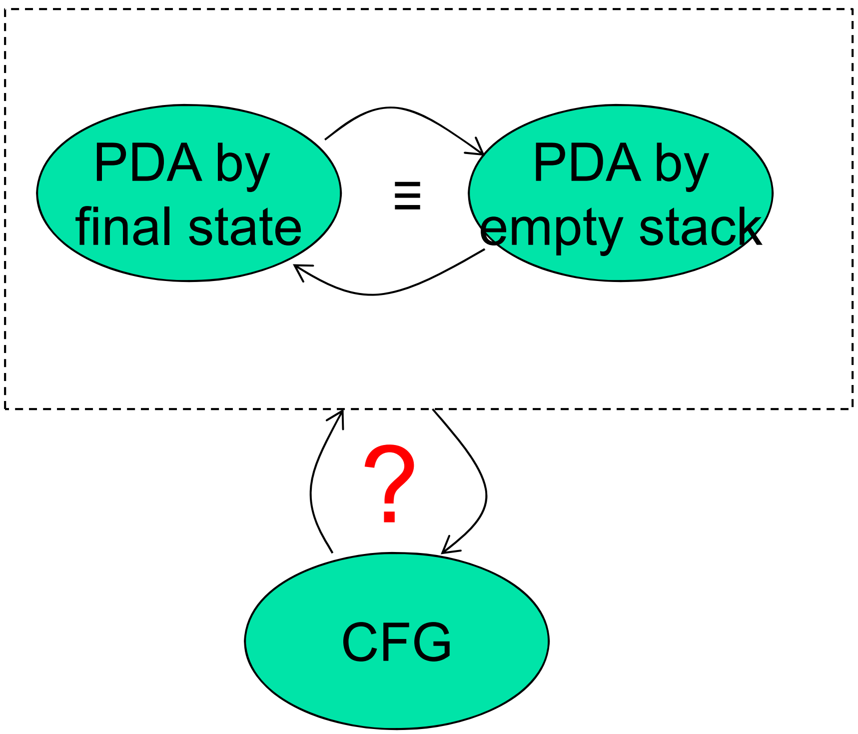
$P_N = (Q \cup \{p_0,p_e\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$

P_N:



Equivalence of PDAs and CFGs

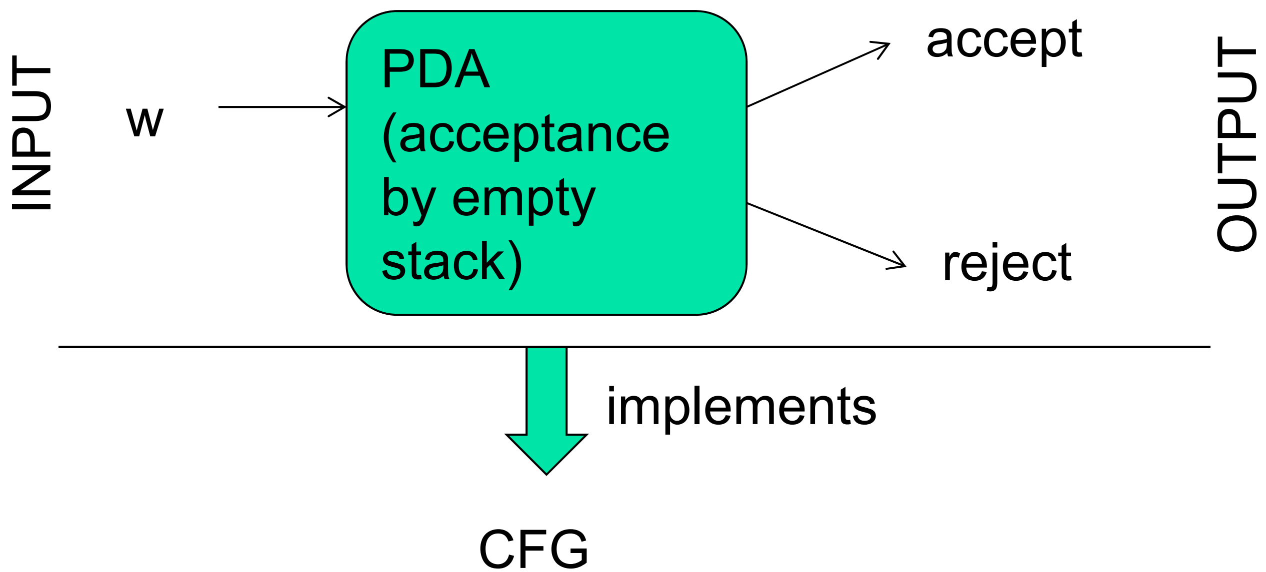
CFGs == PDAs ==> CFLs



This is same as: “implementing a CFG using a PDA”

Converting CFG to PDA

Main idea: The PDA simulates the leftmost derivation on a given w, and upon consuming it fully it either arrives at acceptance (by empty stack) or non-acceptance.



This is same as: “implementing a CFG using a PDA”

Converting a CFG into a PDA

Main idea: The PDA simulates the leftmost derivation on a given w, and upon consuming it fully it either arrives at acceptance (by empty stack) or non-acceptance.

Steps:

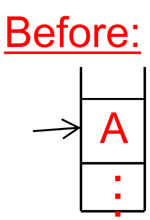
1. Push the right hand side of the production onto the stack, with leftmost symbol at the stack top
2. If stack top is the leftmost variable, then replace it by all its productions (each possible substitution will represent a *distinct* path taken by the non-deterministic PDA)
3. If stack top has a terminal symbol, and if it matches with the next symbol in the input string, then pop it

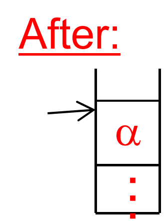
State is inconsequential (only one state is needed)

Formal construction of PDA from CFG

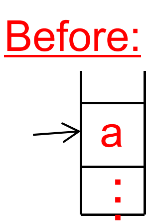
Note: Initial stack symbol (S) same as the start variable in the grammar

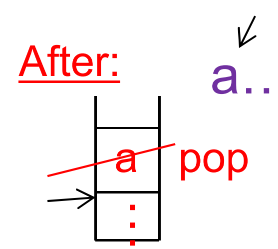
- Given: G= (V,T,P,S)
- Output: P_N = ({q}, T, V U T, δ, q, S)
- δ:





  - For all A ∈ V , add the following transition(s) in the PDA:
    - $\delta(q, \varepsilon, A) = \{ (q, \alpha) \mid \text{"}A \Longrightarrow \alpha\text{"} \in P\}$

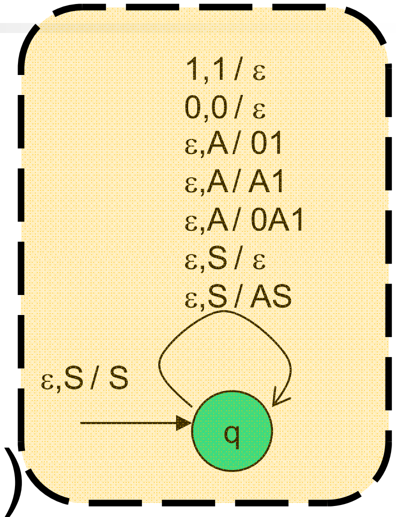




  - For all *a* ∈ T, add the following transition(s) in the PDA:
    - $\delta(q,a,a)= \{ (q, \varepsilon) \}$

Example: CFG to PDA

- G = ({S,A}, {0,1}, P, S)
- P:
  - S ==> AS | ε
  - A ==> 0A1 | A1 | 01
- PDA = ({q}, {0,1}, {0,1,A,S}, δ, q, S)
- δ:
  - δ(q, ε , S) = { (q, AS), (q, ε)}
  - δ(q, ε , A) = { (q,0A1), (q,A1), (q,01) }
  - δ(q, 0, 0) = { (q, ε) }
  - δ(q, 1, 1) = { (q, ε) }



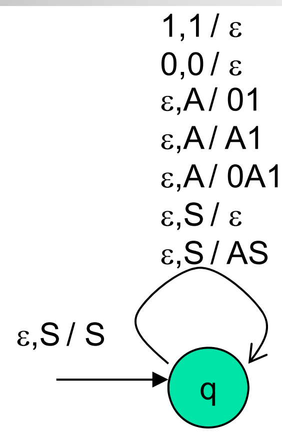
How will this new PDA work?

Lets simulate string <u>0011</u>

Simulating string 0011 on the new PDA …

PDA (δ):

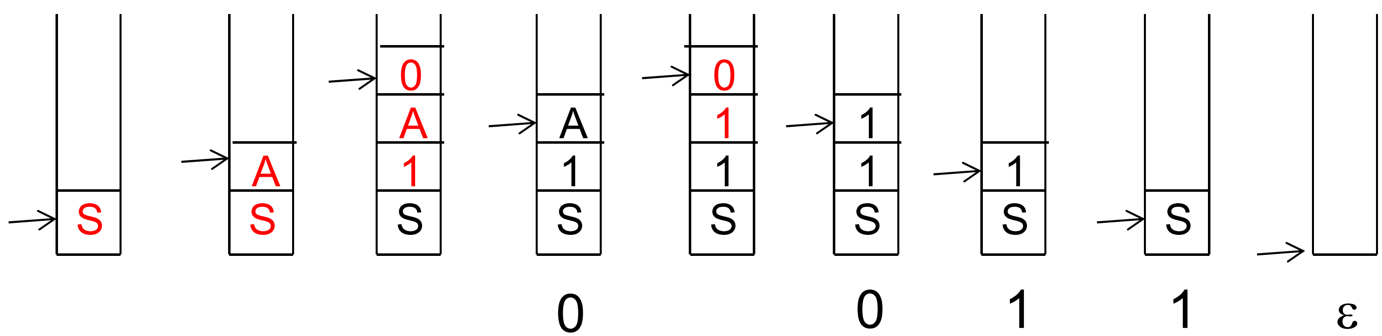
δ(q, ε , S) = { (q, AS), (q, ε)}
δ(q, ε , A) = { (q,0A1), (q,A1), (q,01) }
δ(q, 0, 0) = { (q, ε) }
δ(q, 1, 1) = { (q, ε) }



Leftmost deriv.:

S => AS
=> 0A1S
=> 0011S
=> 0011

Stack moves (shows only the successful path):



S =>AS =>0A1S =>0011S => 0011

Accept by empty stack

Proof of correctness for CFG ==> PDA construction

- Claim: A string is accepted by G iff it is accepted (by empty stack) by the PDA
- Proof:
  - (*only-if part*)
    - Prove by induction on the number of derivation steps
  - (*if part*)
    - If $(q, wx, S) \mid\text{--}^{*} (q,x,B)$ then $S \Rightarrow^{*}_{lm} wB$

Converting a PDA into a CFG

- <u>Main idea:</u> Reverse engineer the productions from transitions

If $\delta(q,a,Z) \Rightarrow (p, Y_1Y_2Y_3\ldots Y_k)$:

1. State is changed from q to p;
2. Terminal *a* is consumed;
3. Stack top symbol Z is popped and replaced with a sequence of k variables.

- <u>Action:</u> Create a grammar variable called "[qZp]" which includes the following production:
  - $[qZp] \Rightarrow a[pY_1q_1]\ [q_1Y_2q_2]\ [q_2Y_3q_3]\ldots\ [q_{k-1}Y_kq_k]$

- Proof discussion (in the book)

Example: Bracket matching

- To avoid confusion, we will use *b*="(" and *e*=")"

P_N: ($\{q_0\}$, {b,e}, $\{Z_0,Z_1\}$, δ, q_0, Z_0)

1. $\delta(q_0,b,Z_0) = \{ (q_0,Z_1Z_0) \}$
2. $\delta(q_0,b,Z_1) = \{ (q_0,Z_1Z_1) \}$
3. $\delta(q_0,e,Z_1) = \{ (q_0, \varepsilon) \}$
4. $\delta(q_0, \varepsilon, Z_0) = \{ (q_0, \varepsilon) \}$

0. $S \Rightarrow [q_0Z_0q_0]$
1. $[q_0Z_0q_0] \Rightarrow b\ [q_0Z_1q_0]\ [q_0Z_0q_0]$
2. $[q_0Z_1q_0] \Rightarrow b\ [q_0Z_1q_0]\ [q_0Z_1q_0]$
3. $[q_0Z_1q_0] \Rightarrow e$
4. $[q_0Z_0q_0] \Rightarrow \varepsilon$

Let $A=[q_0Z_0q_0]$
Let $B=[q_0Z_1q_0]$

0. S => A
1. A => b B A
2. B => b B B
3. B => e
4. A => ε

Simplifying,

0. S => b B S | ε
1. B => b B B | e

If you were to directly write a CFG:

S => b S e S | ε

Two ways to build a CFG

- Build a PDA → Construct CFG from PDA (indirect)
- Derive CFG directly (direct)

Similarly… Two ways to build a PDA

- Derive a CFG → Construct PDA from CFG (indirect)
- Design a PDA directly (direct)

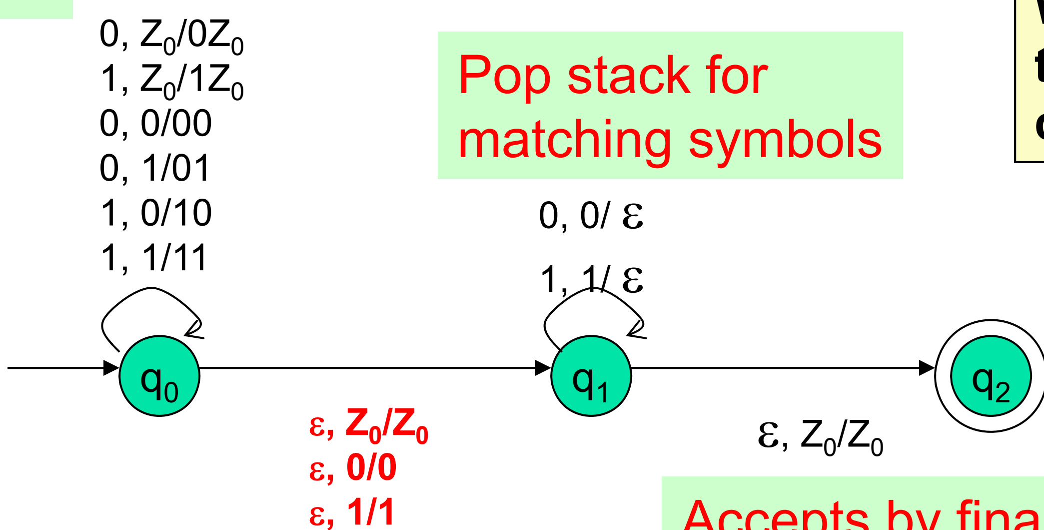
Deterministic PDAs

This PDA for L_{wwr} is non-deterministic

Grow stack

Pop stack for matching symbols

Why does it have to be non-deterministic?



Accepts by final state

Switch to popping mode

To remove guessing, impose the user to insert c in the middle

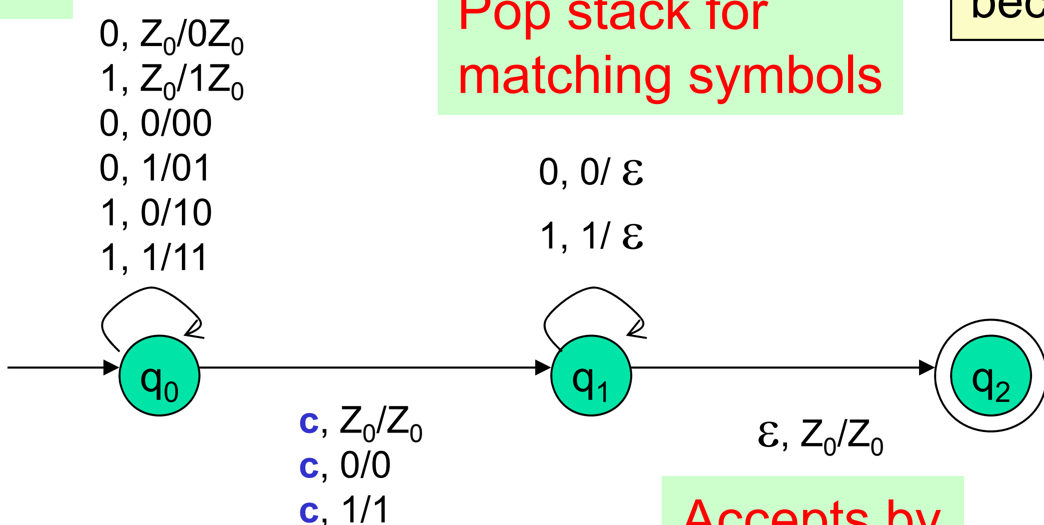
Example shows that: Nondeterministic PDAs ≠ D-PDAs

D-PDA for L_{wcwr} = {wcw^R | c is some special symbol not in w}

Note:
- all transitions have become deterministic

Grow stack

Pop stack for matching symbols



Switch to popping mode

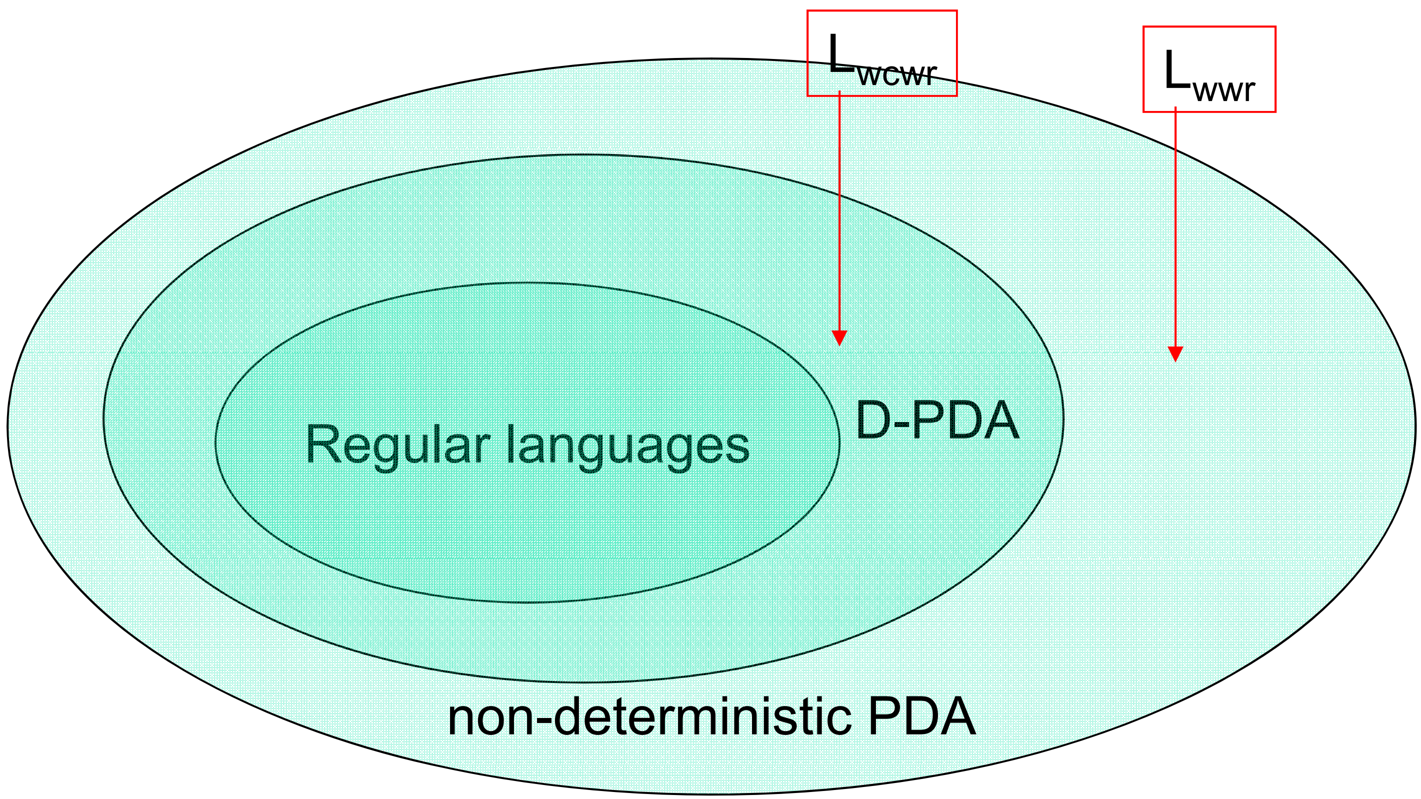
Accepts by final state

Deterministic PDA: Definition

- A PDA is *deterministic* if and only if:
  1. δ(q,a,X) has *at most one* member for any $a \in \sum \cup \{\varepsilon\}$

➔ If δ(q,a,X) is non-empty for some $a \in \sum$, then δ(q, ε,X) must be empty.

PDA vs DPDA vs Regular languages



Summary

- PDAs for CFLs and CFGs
  - Non-deterministic
  - Deterministic
- PDA acceptance types
  1. By final state
  2. By empty stack
- PDA
  - IDs, Transition diagram
- Equivalence of CFG and PDA
  - CFG => PDA construction
  - PDA => CFG construction