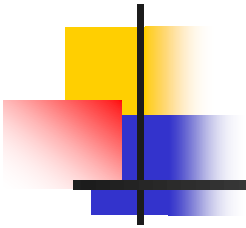
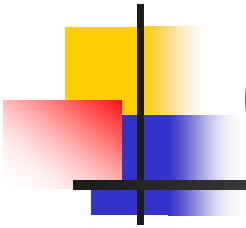


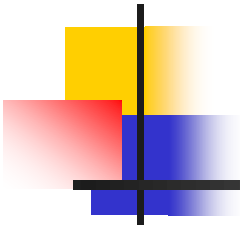
Main Topics

- Background on Regular Languages
  - Reg. lang. closure properties + DFA minimization
- CFGs
- PDAs
- CFLs & pumping lemma
- CFG simplification & normal forms

Regular Languages (Background)

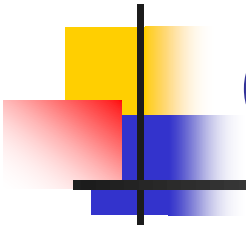


- Building DFA, NFA, ε-NFA
- Building regular expressions
- Closure property results of regular languages
- Which languages cannot be regular and why?
  - Property
  - Pumping lemma



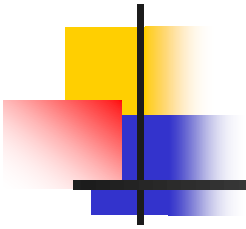
CFGs

- G=(V,T,P,S)
- Derivation, recursive inference, parse trees
  - Their equivalence
- Leftmost & rightmost derivation
  - Their equivalence
  - Generate from parse tree
- Regular languages vs. CFLs
  - Right-linear & left-linear grammars



CFGs

- Designing CFGs (tips & techniques):
  - Making your own start symbol for combining grammars
    - Eg., $S_{new} \Rightarrow S_1 \mid S_2$ (or) $S_{new} \Rightarrow S_1\ S_2$
  - Matching symbols & nested structures: (e.g., S => a **S** b | …)
  - Replicating nested structures side by side: (e.g., S => a S b **S**)
  - Use variables for specific purposes (similar to states)
  - To go to an "acceptance" from a variable
    - ==> end the recursive substitution by making it generate terminals directly
    - A => w
  - Conversely, to *not* go to acceptance from a variable, have recursion (loop back to same variable either directly or indirectly)



Proving CFGs are correct

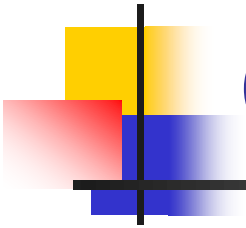
- You will use induction either on
  - Input string length
  - Derivation length

To show: “IF a string is of a particular form (e.g., balanced paranthesis), THEN it will be generated by G

- Use induction on string length

To show: “IF a string is generated by L(G), THEN it is of a particular form (e.g., balanced paranthesis)”

- Use induction on derivation length

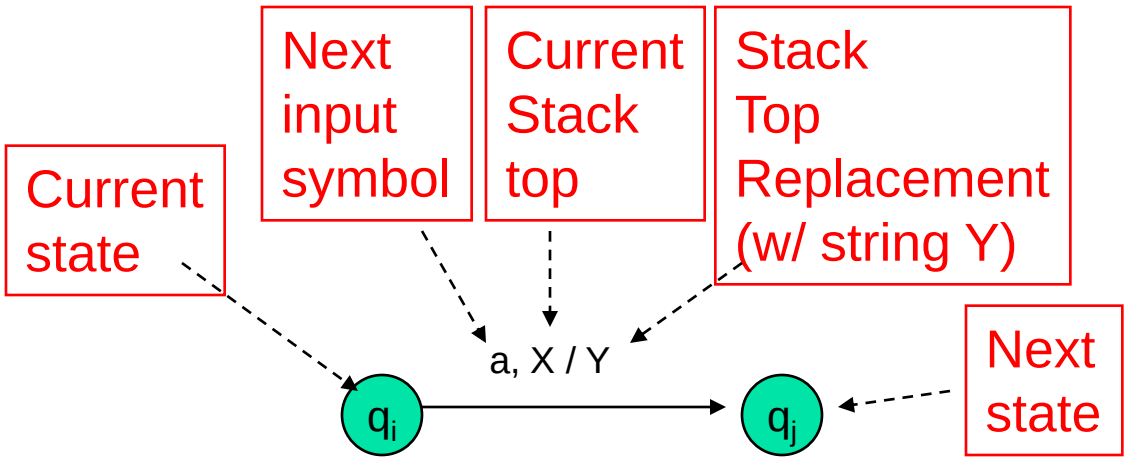


CFGs & ambiguity

- Ambiguity of CFGs
  - To show that a CFG is ambiguous, given one input string in the language which has more than one parse tree
    - (or equivalenty, >1 leftmost/rightmost derivation)
  - Finding one example is sufficient
- A CFL is *inherently ambiguous* if all grammars for that language are going to be ambiguous
- Converting ambiguous CFGs to non-ambiguous CFGs
  - Not possible for inherently ambiguous CFLs
  - For unambiguous CFLs, use ambiguity resolving techniques (e.g., precedence)

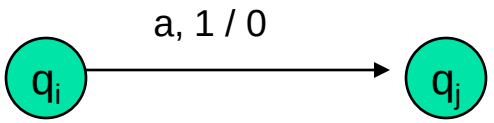
PDAs

- PDA ==> ε-NFA + “a stack”
- P = (Q,$\sum$,Γ, δ,q_0,Z_0,F)
- δ(q,a,X) = {(p,Y), …}
- ID : (q, aw, XB) |--- (p,w,AB)
- State diagram way to show the design of PDAs

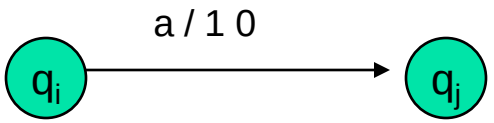


PDA - common mistakes

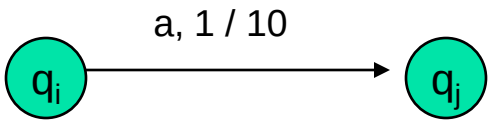
- Transition notation
  - Goal: ***push*** symbol 0 on top of the current stack top symbol 1



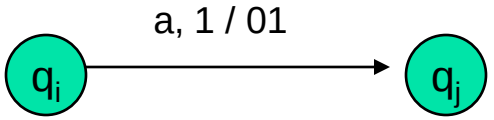
X Why?



X Why?



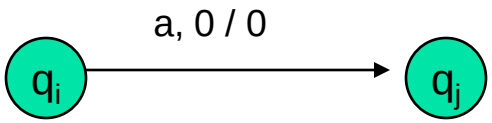
X Why?



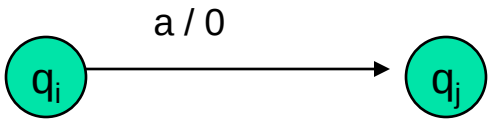
✓

PDA - common mistakes…

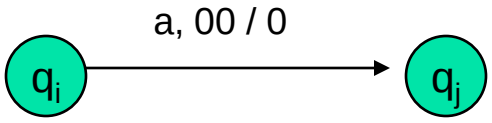
- Transition notation
  - Goal: ***pop*** stack top if stack top is 0



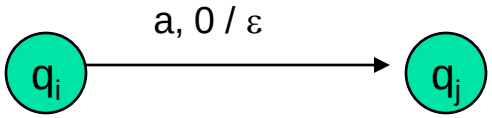
X Why?



X Why?

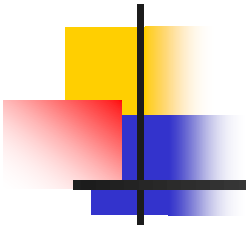


X Why?



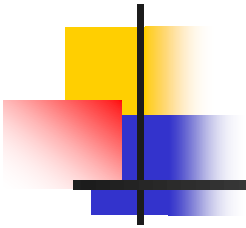
✓

Remember:
you can *push* multiple symbols in one step, but can *pop* only one symbol at a time



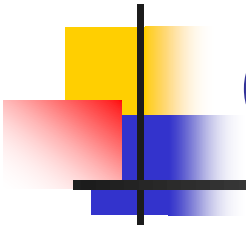
Design tips for PDAs

- Take advantage of the two types of PDAs
  - Acceptance by empty stack
    - If no more input and stack becomes empty
  - Acceptance by final state
    - If no more input and end in final state
- Convert one form to another
- Assign state for specific purposes
- Push to “remember” and Pop to “tally”
- Introducing your own stack symbols may help
- Take advantage of non-determinism



PDA design restrictions

- Feel free to design an empty stack PDA or final state PDA unless otherwise explicitly specified
  - This is meant for design convenience
- But if I ask you design a specific type of PDA in the question, then show a direct construction
  - i.e., do not convert one to another
- Same applies for PDA vs. CFG
  - i.e., If I ask you to design a PDA, then give a direct construction (do not convert from CFG)
  - Same for CFG



Conversion procedures

- Be familiar with:
  - CFG => PDA conversion
  - PDA empty stack => PDA final state
  - PDA final state => PDA empty stack

Follow the algorithms described in class.
if you come up with an ad hoc way that works for that example but not necessarily for others, then that could lead to reduction of points

CFG Simplification

1. Eliminate ε-productions: A => ε
   - ==> substitute for A (with & without)
   - Find nullable symbols first and substitute next
2. Eliminate unit productions: A=> B
   - ==> substitute for B directly in A
   - Find unit pairs and then go production by production
3. Eliminate useless symbols
   - Retain only reachable and generating symbols
   - Order: first generating test, and then reachability test

- Order is important : steps (1) => (2) => (3)

Chomsky Normal Form

- All productions of the form:
  - A => BC or A=> a
- Grammar does not contain:
  - Useless symbols, unit and ε-productions
- Converting CFG (without S=>*ε) into CNF
  - Introduce new variables that collectively represent a sequence of other variables & terminals
  - New variables for each terminal
- CNF ==> Parse tree size
  - If the length of the longest path in the parse tree is n, then $|w| \leq 2^{n-1}$.

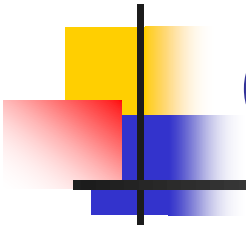
Pumping Lemma for CFLs

- Then there exists a constant n, s.t.,
  - if z is any string in L s.t. $|z| \geq n$, then we can write $z=uvwxy$, subject to the following <u>conditions:</u>
    1. $|vwx| \leq n$
    2. $vx \neq \varepsilon$
    3. For all $k \geq 0$, uv^kwx^ky is in L

Using Pumping Lemmas for CFLs

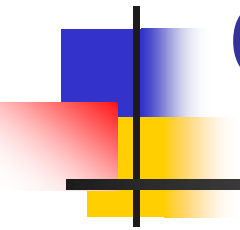
- Steps:
  1. Let n be the P/L constant
  2. Pick a word z in the language s.t. $|z| \geq n$
     - (choice critical - any arbitrary choice may not work)
  3. z=uvwxy
  4. First, argue that because of conditions (1) & (2), the portions covered by vwx on the main string z will have to satisfy some properties
  5. Next, argue that by pumping up or down you will get a new string from z that is not in L

Refer to the exercises done in class as examples

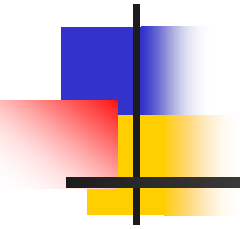


Closure Properties for CFL

- CFLs are closed under:
  - Union
  - Concatenation
  - Kleene closure operator
  - Substitution
  - Homomorphism, inverse homomorphism
- CFLs are *not* closed under:
  - Intersection
  - Difference
  - Complementation



Good luck !!

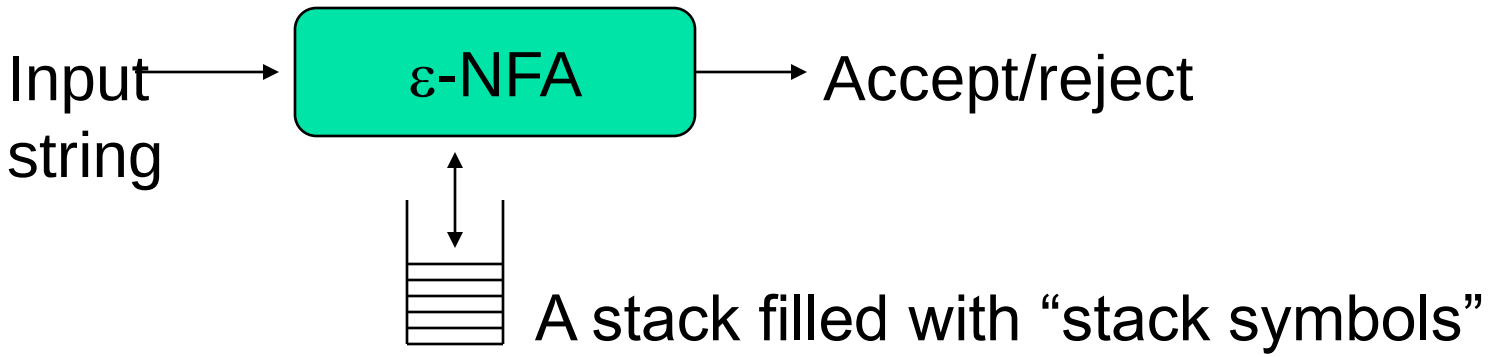


Pushdown Automata (PDA)

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)*

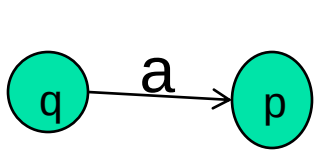
$$\delta : Q \times \Sigma \times \Gamma \Rightarrow Q \times \Gamma$$

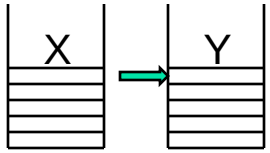
δ : The Transition Function

δ(q,a,X) = {(p,Y), …}

Non-determinism

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)





	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

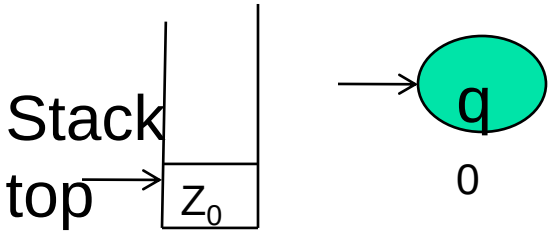
Let $L_{wwr} = \{ww^R \mid w \text{ is in } (0+1)^*\}$

- CFG for L_{wwr} : S==> 0S0 | 1S1 | ε
- PDA for L_{wwr} :
- P := (Q,$\sum$, Γ, δ,q_0,Z_0,F)

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

PDA for L_{wwr}

Initial state of the PDA:



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

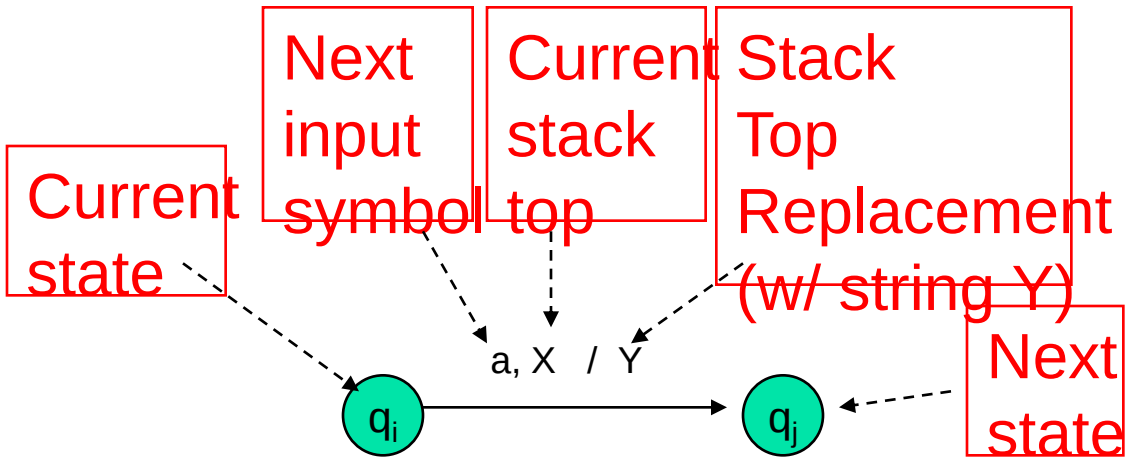
Shrink the stack by popping matching symbols (w^R-part)

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

Enter acceptance state

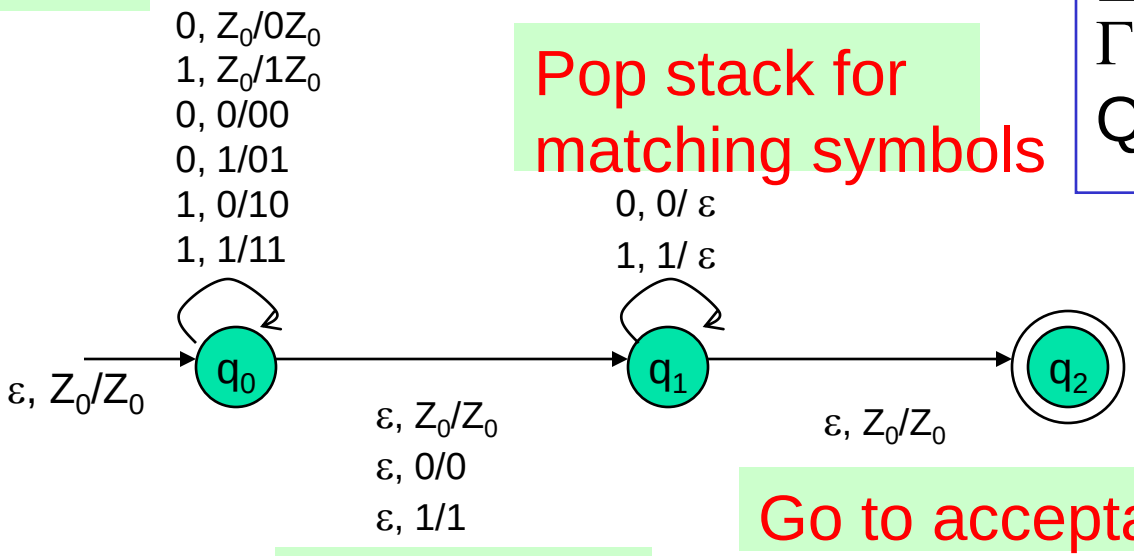
PDA as a state diagram

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



PDA for L_{wwr}: Transition Diagram

Grow stack



Pop stack for matching symbols

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

Switch to popping mode

Go to acceptance

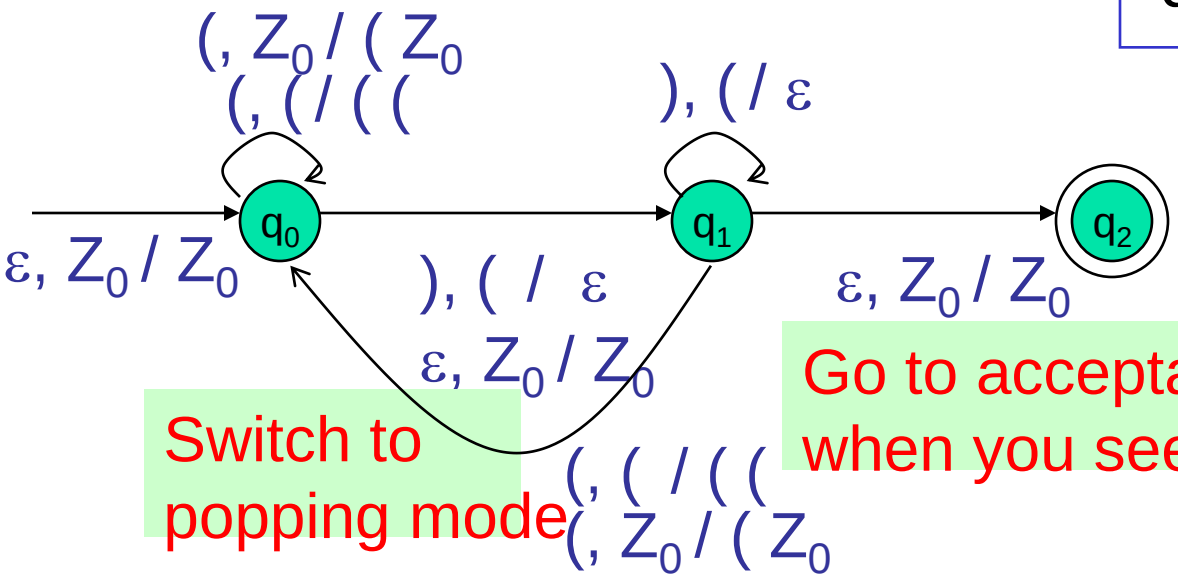
This would be a non-deterministic PDA

Example 2: language of balanced paranthesis

Pop stack for matching symbols

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

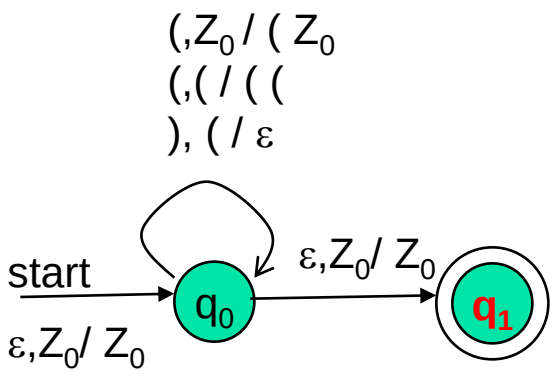
Grow stack



Go to acceptance (by final state when you see the stack bottom

To allow adjacent blocks of nested paranthesis

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

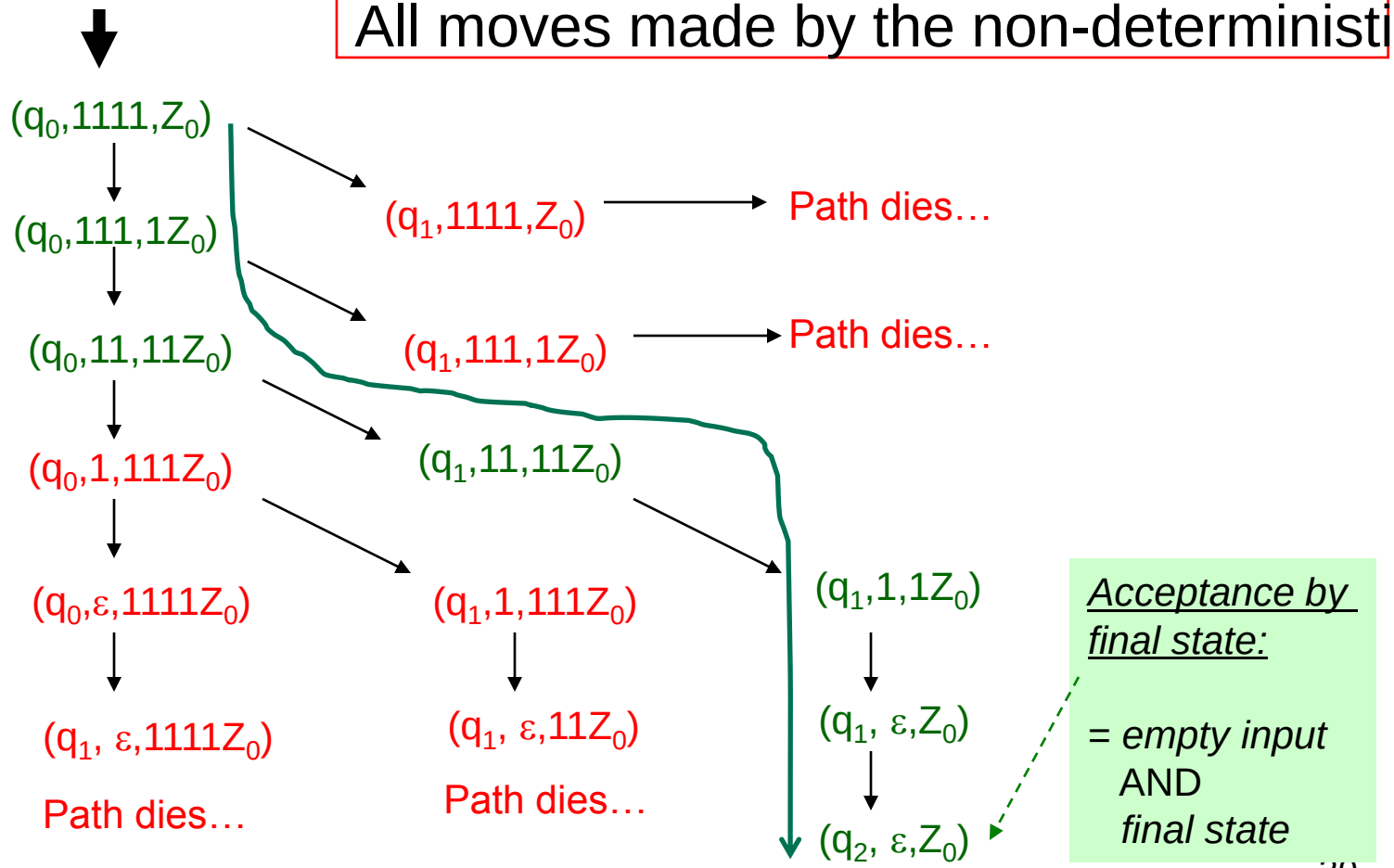
- $(q, a, X) \vdash (p,\varepsilon,A)$
- $(q, aw, XB) \vdash (p,w,AB)$

|--- 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 P



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:
    - $\{w \mid (q_0,w,Z_0) \vdash^* (q,\varepsilon, A)\}$, 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:
    - $\{w \mid (q_0,w,Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$, for any $q \in Q$.

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

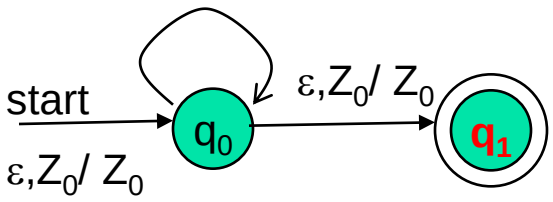
Checklist:
- input exhausted?
- is the stack empty?

Example: L of balanced parenthesis

PDA that accepts by final state

P_F:

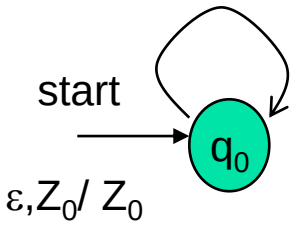
(,Z_0 / (Z_0
(,(/ ((
), (/ ε



An equivalent PDA that accepts by empty stack

P_N:

(,Z_0 / (Z_0
(, (/ ((
), (/ ε
ε,Z_0 / ε



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

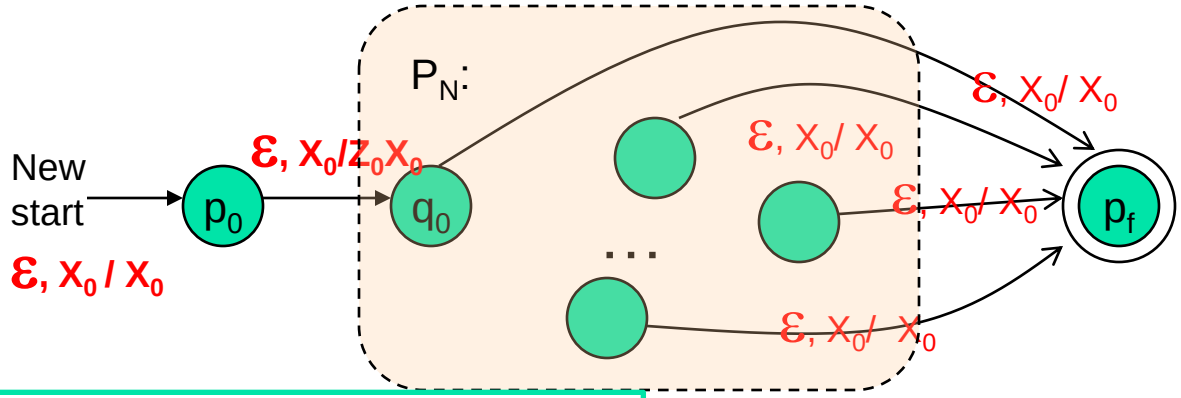
PDAs accepting by final state and empty stack are equivalent

- 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:



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

Example: Matching parenthesis "(" ")"

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

δ_N:

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

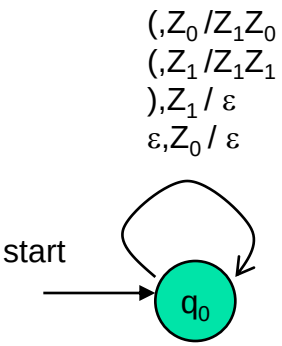
$\delta_N(q_0,),Z_1) = \{ (q_0, \varepsilon) \}$

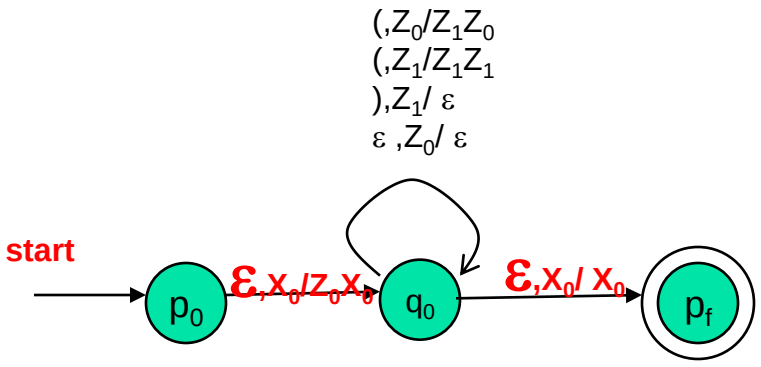
$\delta_N(q_0, \varepsilon,Z_0) = \{ (q_0, \varepsilon) \}$

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 empty stack

Accept by final state

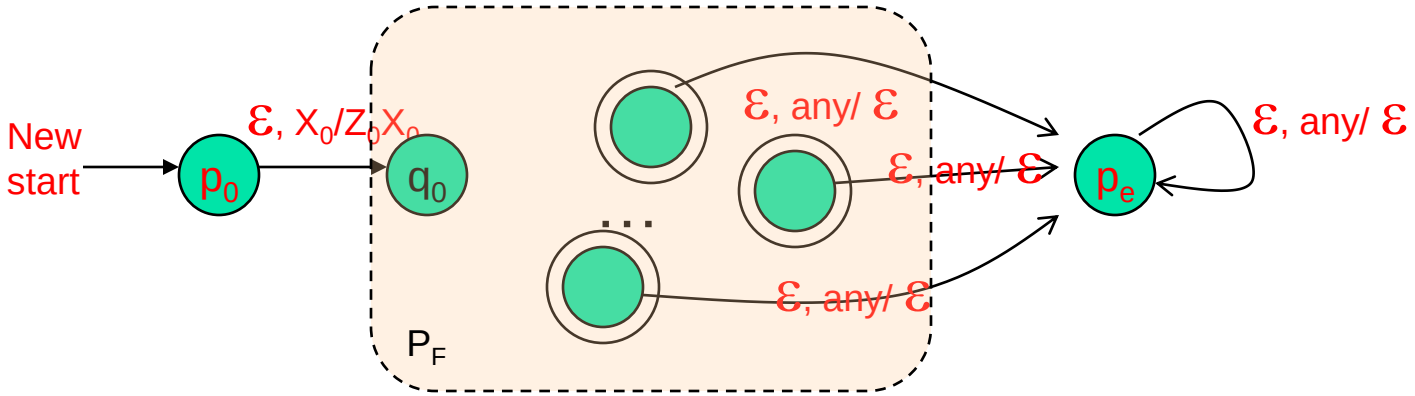
P_F==> P_N construction

- Main idea:
  - 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?

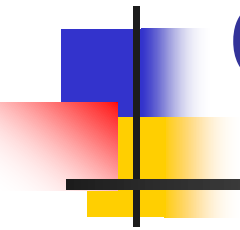
    ➔ to address this, add a new start symbol X_0 (not in Γ of P_F)

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

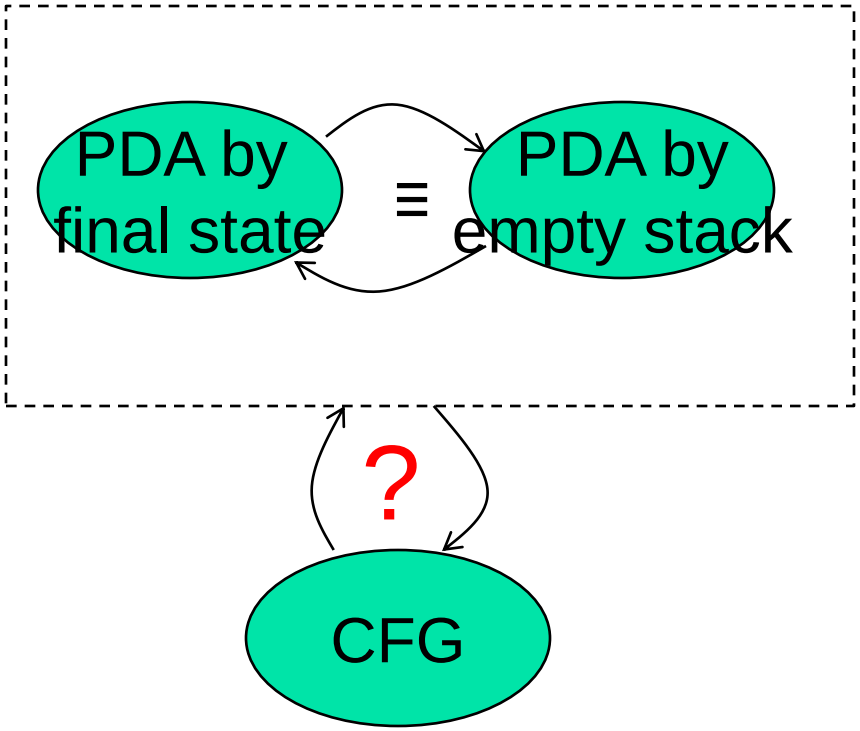
P_N:



Equivalence of PDAs and CFGs



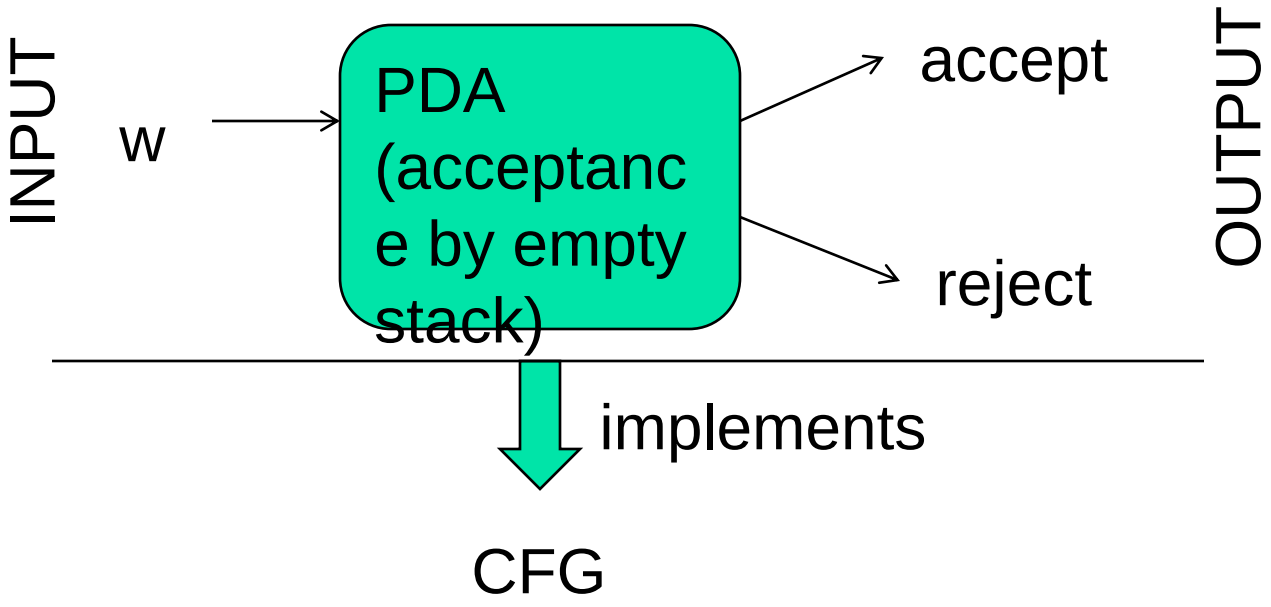
CFGs == PDAs ==> CFLs



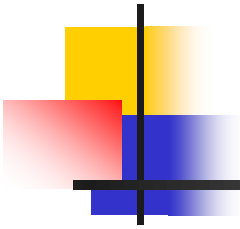
This is same as: “implementing a CFG using a PDA”

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.



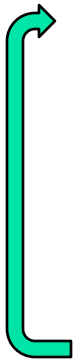
This is same as: “implementing a CFG using a PDA”



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

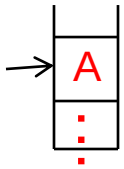
State is inconsequential (only one state is needed)

Formal construction of PDA from CFG

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

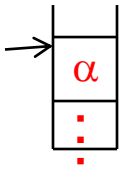
- Given: G= (V,T,P,S)
- Output: $P_N = (\{q\}, T, V \cup T, \delta, q, S)$
- δ:

Before:

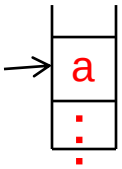


- For all $A \in V$, add the following transition(s) in the PDA:
  - $\delta(q, \varepsilon, A) = \{ (q, \alpha) \mid \text{“}A ==> \alpha\text{”} \in P\}$

After:

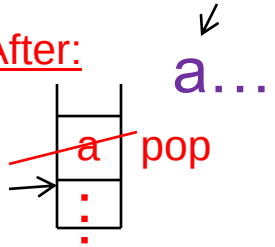


Before:



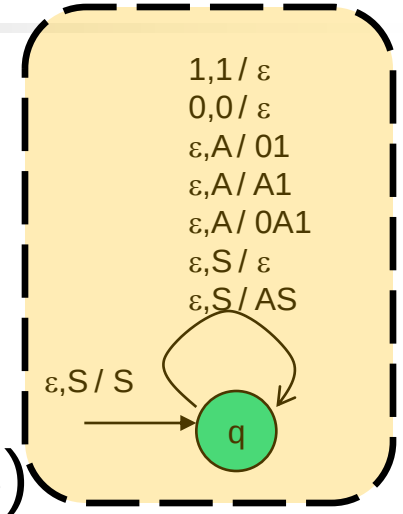
- For all *a* $\in$ T, add the following transition(s) in the PDA:
  - $\delta(q, a, a) = \{ (q, \varepsilon) \}$

After:



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) }



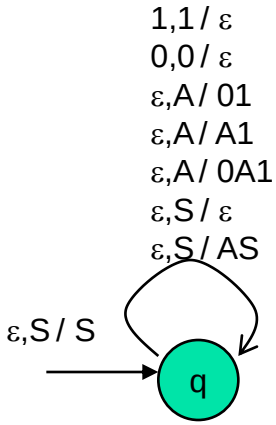
How will this new PDA work?

Lets simulate string 0011

Simulating string 0011 on the new PDA …

PDA (δ):

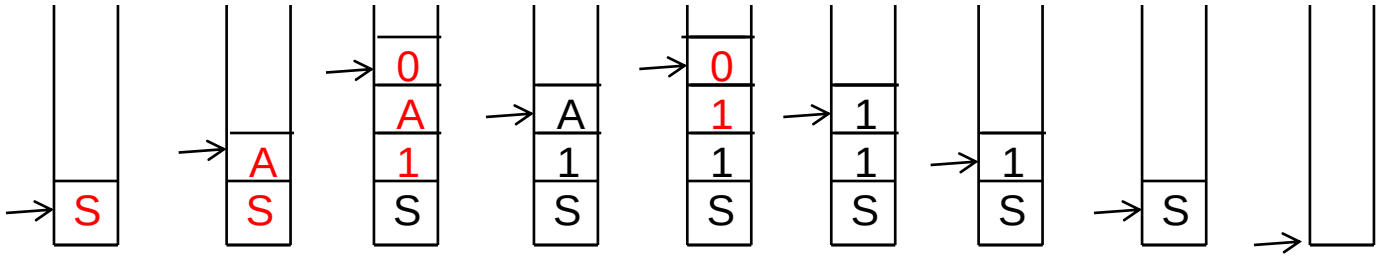
δ(q, ε , S) = { (q, AS), (q, ε)}
δ(q, ε , A) = { (q,0A1), (q,A1), (q,01) }
δ(q, 0, 0) = { (q, ε) }
δ(q, 1, 1) = { (q, ε) }



Leftmost deriv.

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

Stack moves (shows only the successful path):



Accept by empty stack

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

Converting a PDA into a CFG

- Main idea: 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.

- Action: 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. $S \Rightarrow A$
1. $A \Rightarrow b\ B\ A$
2. $B \Rightarrow b\ B\ B$
3. $B \Rightarrow e$
4. $A \Rightarrow \varepsilon$

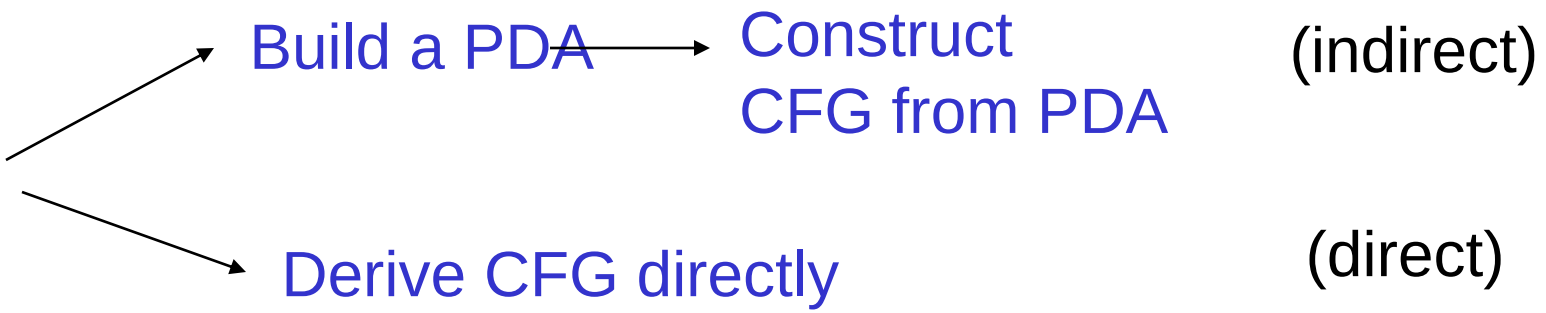
Simplifying,

0. $S \Rightarrow b\ B\ S \mid \varepsilon$
1. $B \Rightarrow b\ B\ B \mid e$

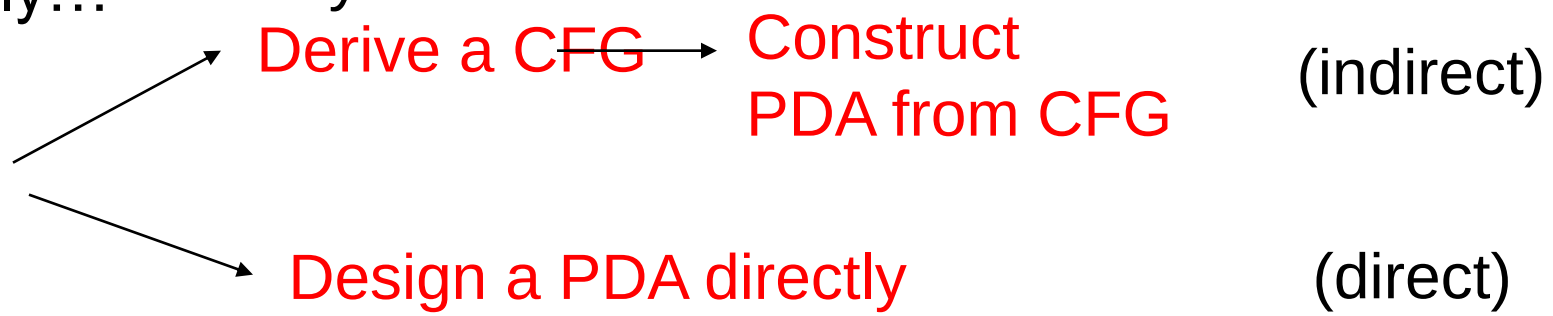
If you were to directly write a CFG:

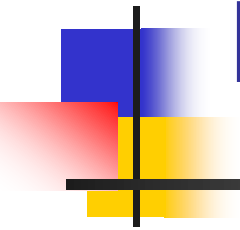
$S \Rightarrow b\ S\ e\ S \mid \varepsilon$

Two ways to build a CFG



Similarly… Two ways to build a PDA

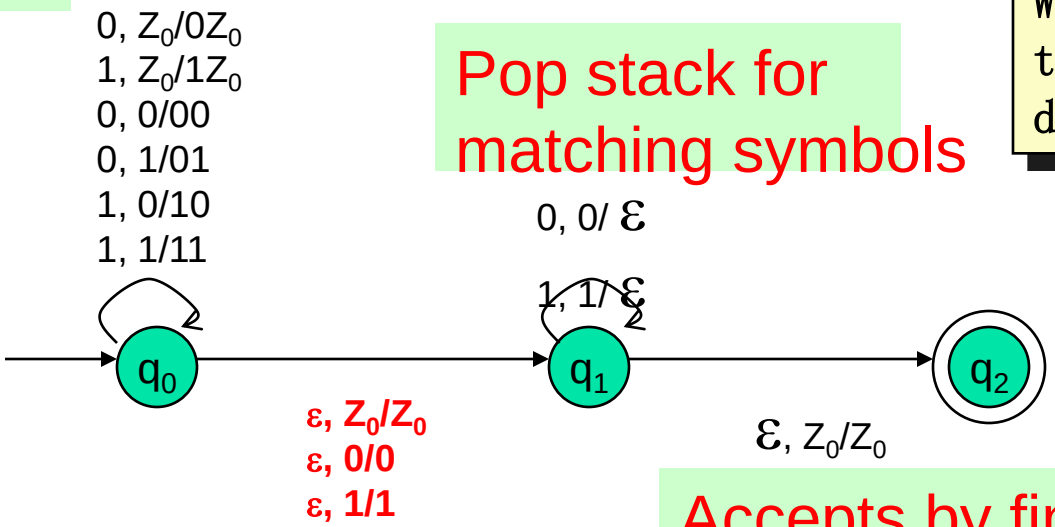




Deterministic PDAs

This PDA for L_{wwr} is non-deterministic

Grow stack



Pop stack for matching symbols

Why does it have to be non-deterministic?

Switch to popping mode

Accepts by final state

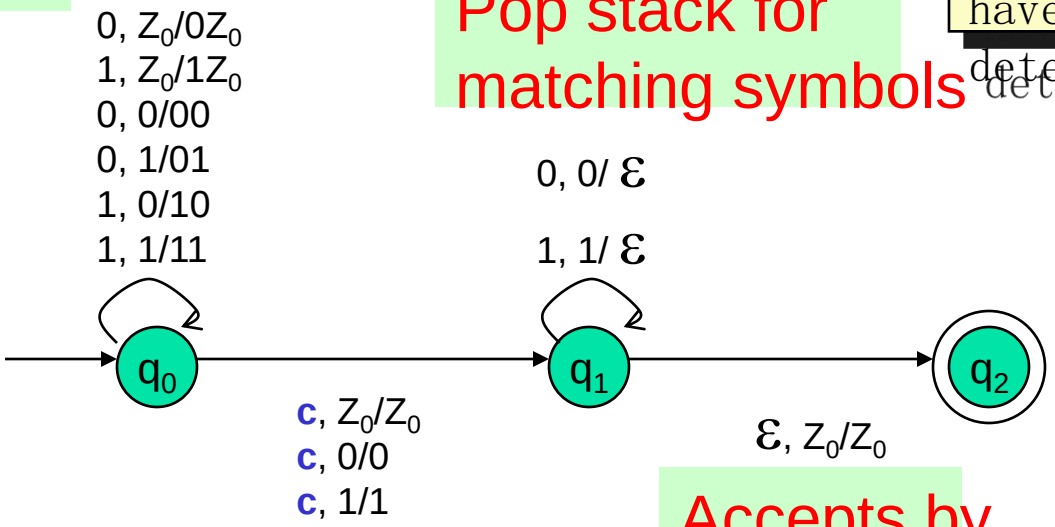
To remove guessing, impose the user to insert

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

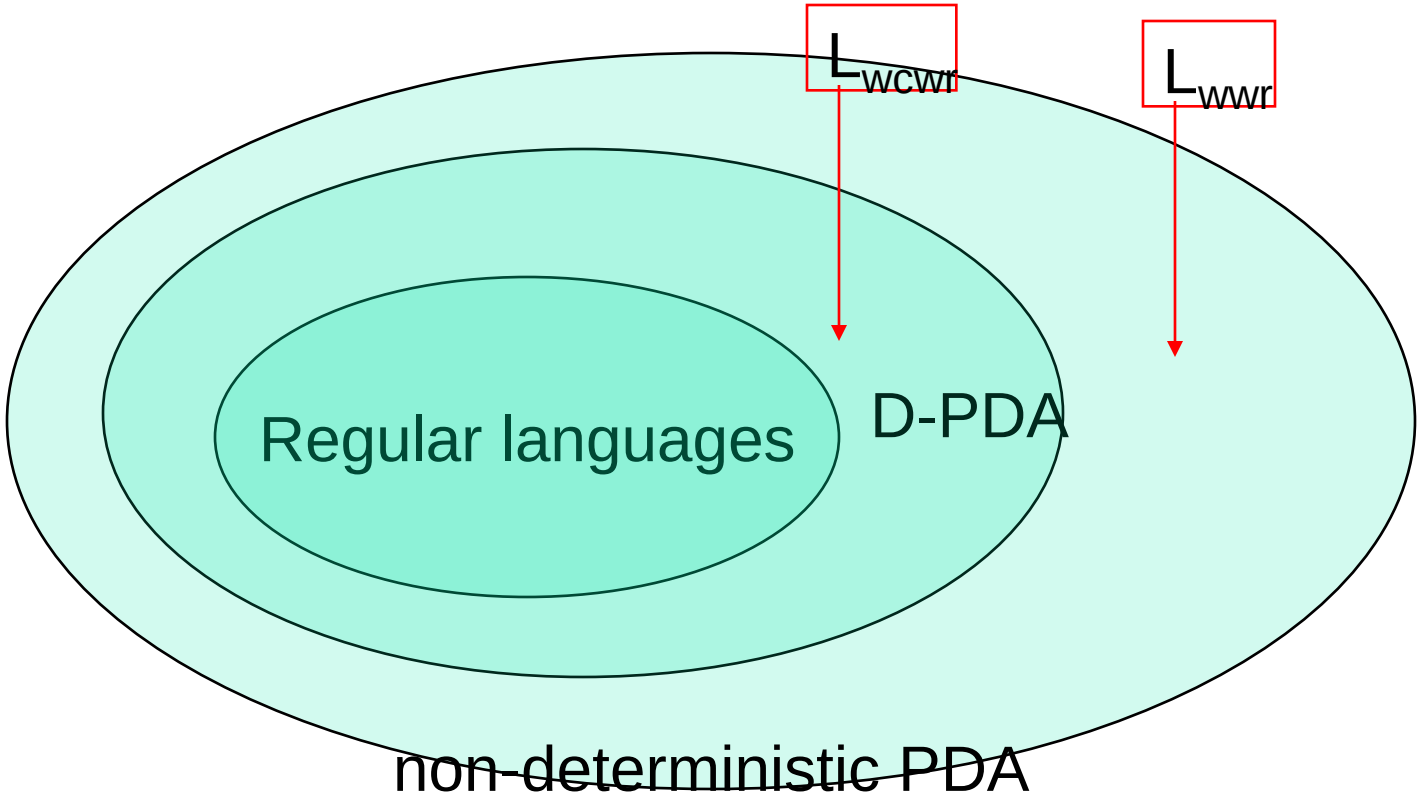
Accepts by final state

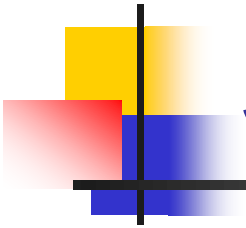
Deterministic PDA: Definition

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

➔ If $\delta(q,a,X)$ is non-empty for some $a \in \sum$, then $\delta(q, \varepsilon, X)$ 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