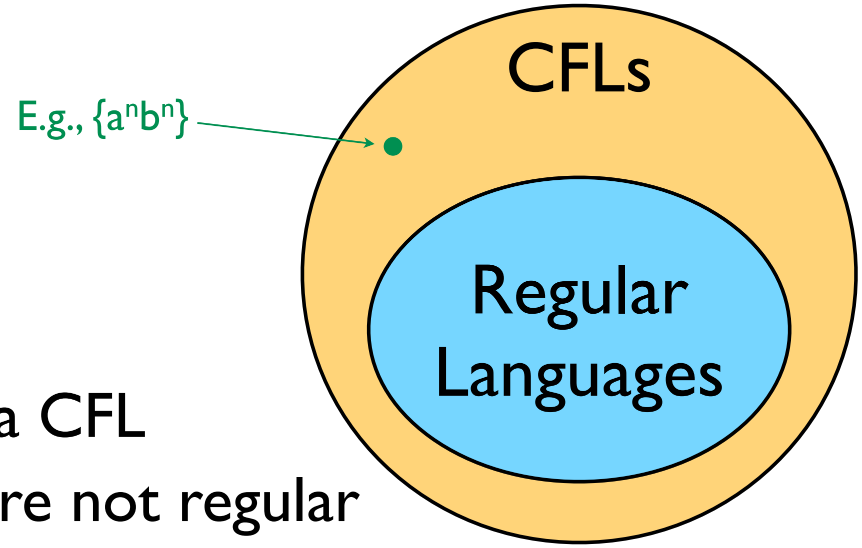
Context-free Languages and Pushdown Automata

Finite Automata vs CFLs



From earlier results:

every regular language is a CFL

but there are CFLs that are not regular

Can we extend Finite Automata to equal CFLs?
I.e., get a machine-like characterization of CFLs?

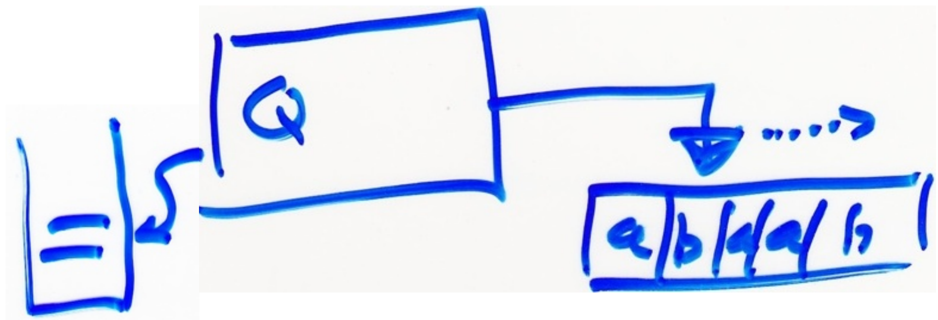
CF but not Regular $a^n b^n$, ww^R, $\#a = \#b$, ...

A key feature: recursion →

$S \to aSb \mid \varepsilon$

$S \to aSa \mid bSb \mid \varepsilon$

Recursion's twin: a pushdown stack →



Pushdown sufficient? intuitively, yes:

$a^n b^n$: push a's
pop/match b's

ww^R : push input
At middle, (Guess!)
Flip state to pop/match

Pushdown Automaton

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

Q is finite set (states)

Σ . . . . (input alphabet)

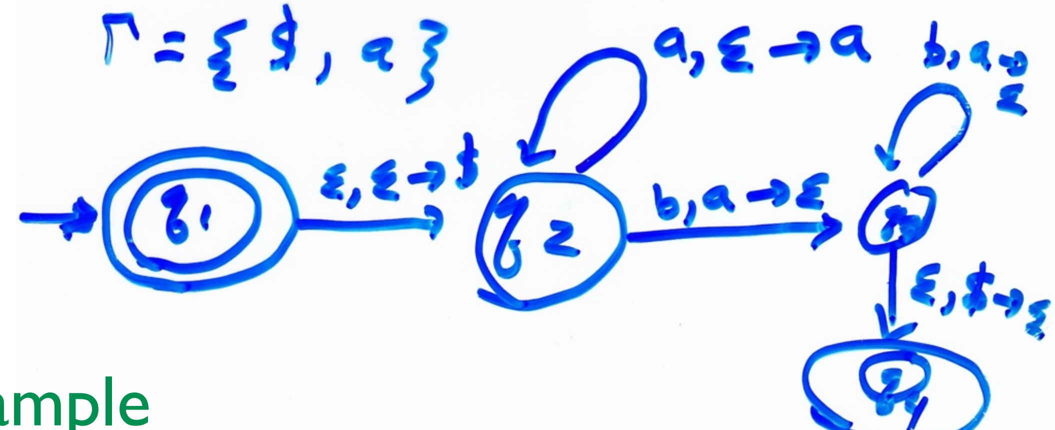
Γ . . . . (stack alphabet)

$q_0 \in Q$ start state

$F \subseteq Q$ accept states

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to 2^{Q \times \Gamma_\varepsilon}$$

$\Gamma = \{\$, a\}$



Example

M can reach state q with $\gamma \in \Gamma^*$ on its stack after reading w

if $\exists\, w_1 w_2 \cdots w_m \in \Sigma_\varepsilon$

st $w = w_1 \cdot w_2 \cdots w_m$

$\exists\, r_0\, r_1 \ldots r_m \in Q$

$\exists\, s_0, \ldots s_m \in \Gamma^*$

(1) $r_0 = q_0$,

(2) $s_0 = \varepsilon$

(3) $\forall\, i = 0 .. m-1$

$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$

for some $a, b \in \Gamma_\varepsilon$, $t \in \Gamma^*$

with $s_i = at$, $s_{i+1} = bt$

(4) $\gamma = s_m$

M accepts w if $r_m \in F$

$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$

Alternate way to define this:

A PDA *Configuration* (stack top on left):

$\langle$state, stack, input$\rangle$

A PDA *Move*:

$\langle p, at, wx \rangle \vdash \langle q, bt, x \rangle$

if $\exists$ p,q $\in$ Q, a $\in \Gamma \cup \{\varepsilon\}$, t $\in \Gamma^*$, w $\in \Sigma \cup \{\varepsilon\}$, x $\in \Sigma^*$ s.t. $(q,b) \in \delta(p,w,a)$

Multiple moves:

$\vdash^k$: exactly k steps

$\vdash^*$: 0 or more steps

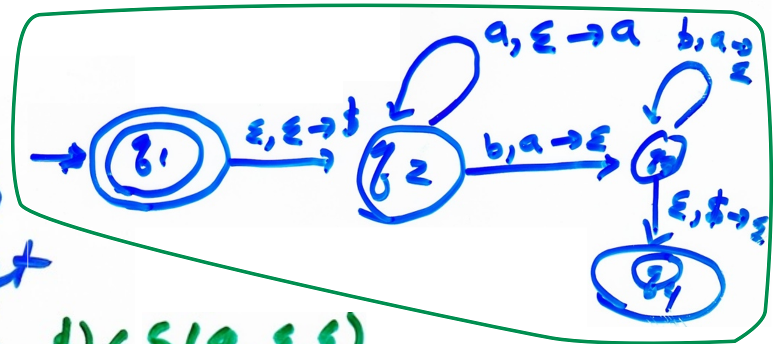
M *can reach q with $\gamma \in \Gamma^*$ on its stack after reading $w \in \Sigma^*$* if

$\langle q_0, \varepsilon, w \rangle \vdash^* \langle q, \gamma, \varepsilon \rangle$

M accepts w if above, and $q \in F$

$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$

Example: A computation of M above on input $w = aabb$



State	Stack	remaining input	Which move
$r_0 = q_1$	$s_0 = \varepsilon$	$aabb$	$(q_2, \$) \in \delta(q_1, \varepsilon, \varepsilon)$
$r_1 = q_2$	$s_1 = \$$	$aabb$	$(q_2, a) \in \delta(q_2, a, \varepsilon)$
$r_2 = q_2$	$s_2 = \$a$	abb	" "
$r_3 = q_2$	$s_3 = \$aa$	bb	$(q_3, \varepsilon) \in \delta(q_2, b, a)$
$r_4 = q_3$	$s_4 = \$a$	b	$(q_3, \varepsilon) \in \delta(q_3, b, a)$
$r_5 = q_3$	$s_5 = \$$	ε	$(q_4, \varepsilon) \in \delta(q_3, \varepsilon, \$)$
$r_6 = q_4$	$s_6 = \varepsilon$	ε	

top of stack @ right

Which move

E.g., "M can reach q_3 with \$ on its stack after reading a^2b^2", and "M can reach q_4 with ε on stack reading a^2b^2" and "M accepts a^2b^2".

Every CFL is accepted by some PDA

Every regular language is accepted by some PDA (basically, just ignore the stack...)

Above examples show that PDAs are sufficiently powerful to accept some context-free but non-regular languages, too

In fact, they can accept *every* CFL:

Proof 1: the book's "top down" parser (next)

Proof 2: "bottom up," (aka "shift-reduce") parser (later)

PDAs accept all CFLs

"Top-Down"

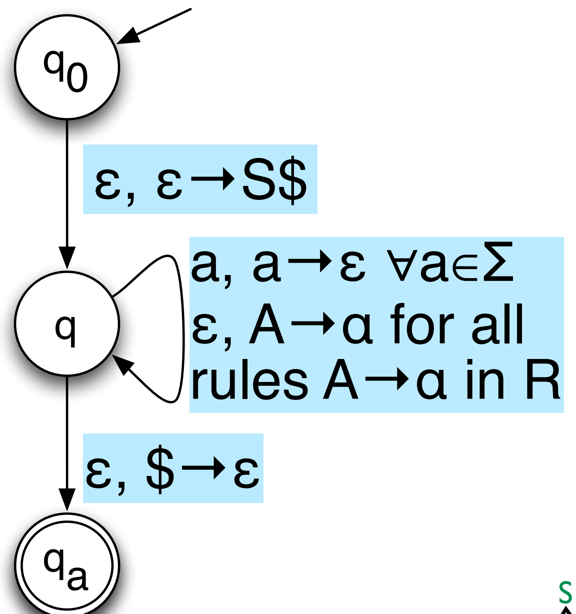
For any CFG $G=(V, \Sigma, R, S)$, build PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

$Q = \{q_0, q, q_{accept}\}$

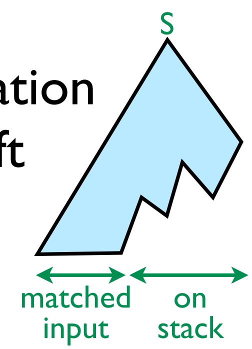
$\Gamma = V \cup \Sigma \cup \{\$\}$ $(\$ \notin V \cup \Sigma)$

$F = \{q_{accept}\}$, and

δ is defined by the diagram



Idea: on input w, M nondeterministically picks a leftmost derivation of w from S. Stack holds intermediate strings in derivation (left end at top); letters in Σ on top of stack matched against input.



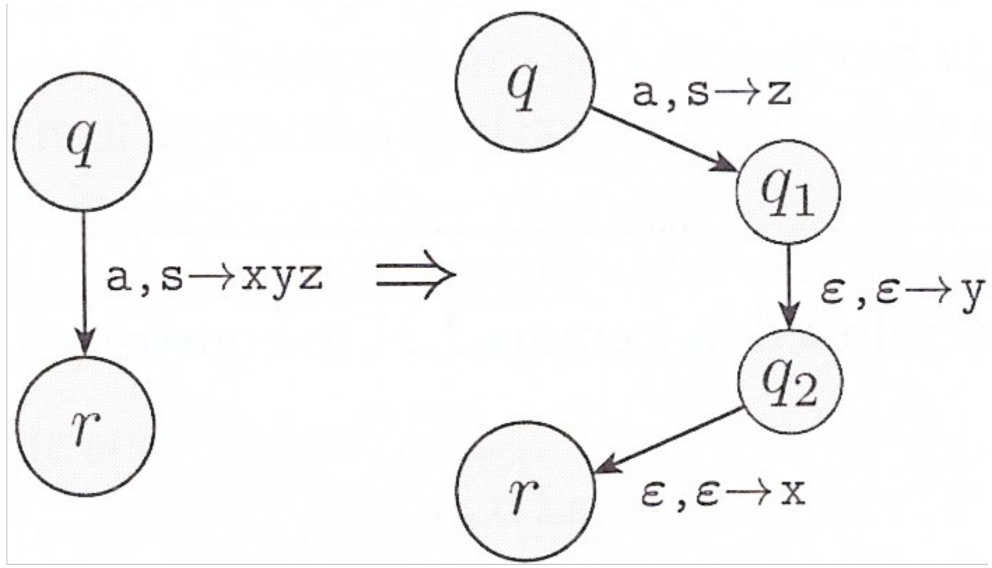
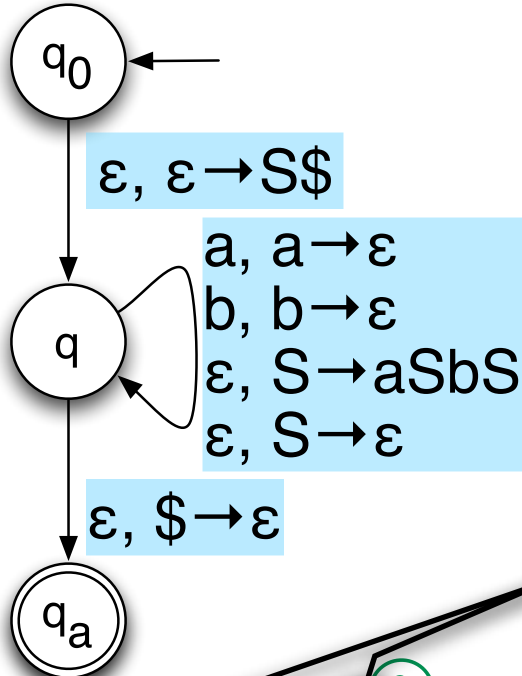


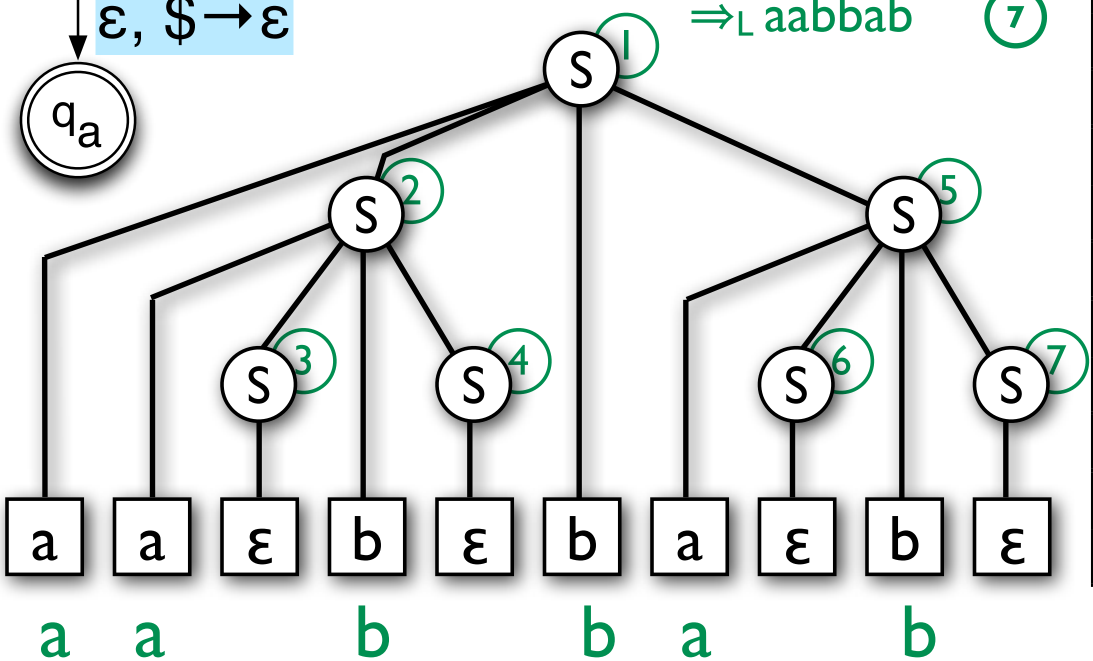
FIGURE 2.23
Implementing the shorthand $(r, xyz) \in \delta(q, a, s)$

Top-Down Parser

$S \to aSbS \mid \varepsilon$



$S \Rightarrow_L aSbS$ (1)
$\Rightarrow_L aaSbSbS$ (2)
$\Rightarrow_L aabSbS$ (3)
$\Rightarrow_L aabbS$ (4)
$\Rightarrow_L aabbaSbS$ (5)
$\Rightarrow_L aabbabS$ (6)
$\Rightarrow_L aabbab$ (7)



top ⟷ bot

State	Stack	Input	Deriv
q_0	ε	aabbab	
q	S$	aabbab	1
q	aSbS$	aabbab	
q	SbS$	abbab	2
q	aSbSbS$	abbab	
q	SbSbS$	bbab	3
q	bSbS$	bbab	
q	SbS$	bab	4
q	bS$	bab	
q	S$	ab	5
q	aSbS$	ab	
q	SbS$	b	6
q	bS$	b	
q	S$	ε	7
q	$	ε	
q_{accept}	ε	ε	

Input accepted

PDAs accept all CFLs

"Bottom-Up" / "Shift-Reduce"

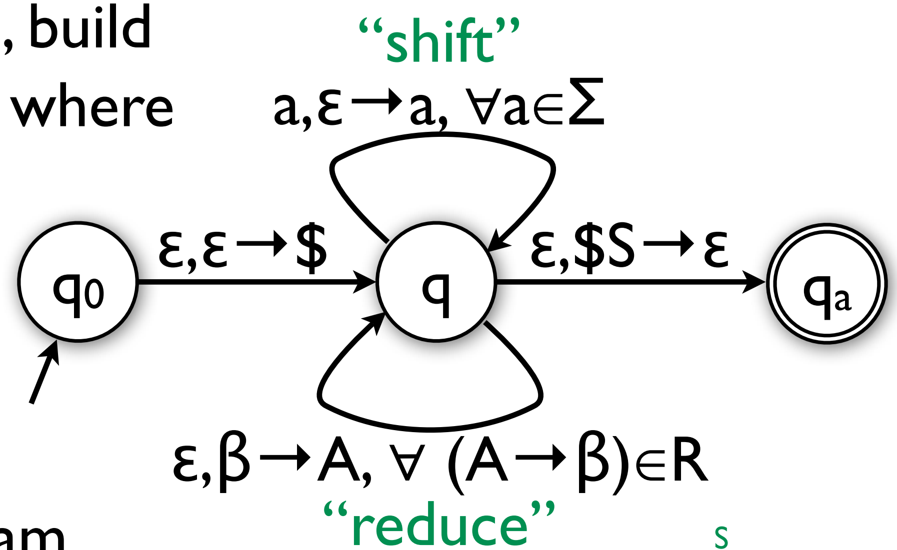
For any CFG $G=(V, \Sigma, R, S)$, build PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

$Q = \{q_0, q, q_{accept}\}$

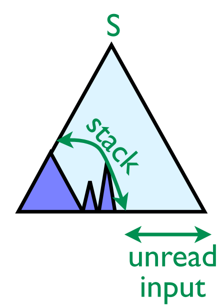
$\Gamma = V \cup \Sigma \cup \{\$\}$ ($\$ \notin V \cup \Sigma$)

$F = \{q_{accept}\}$, and

δ is defined by the diagram

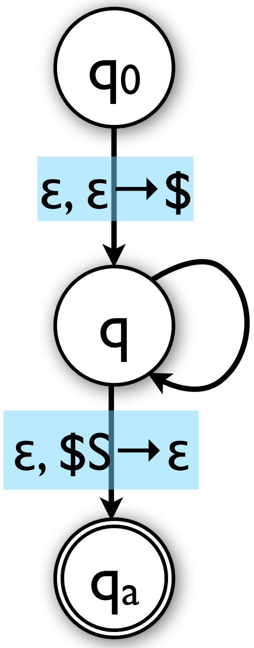


Idea: on input w, M nondeterministically picks a *right*most derivation *backwards*, from w to S. Shift input onto stack or "reduce" top few symbols at each step.



S→aSbS|ε

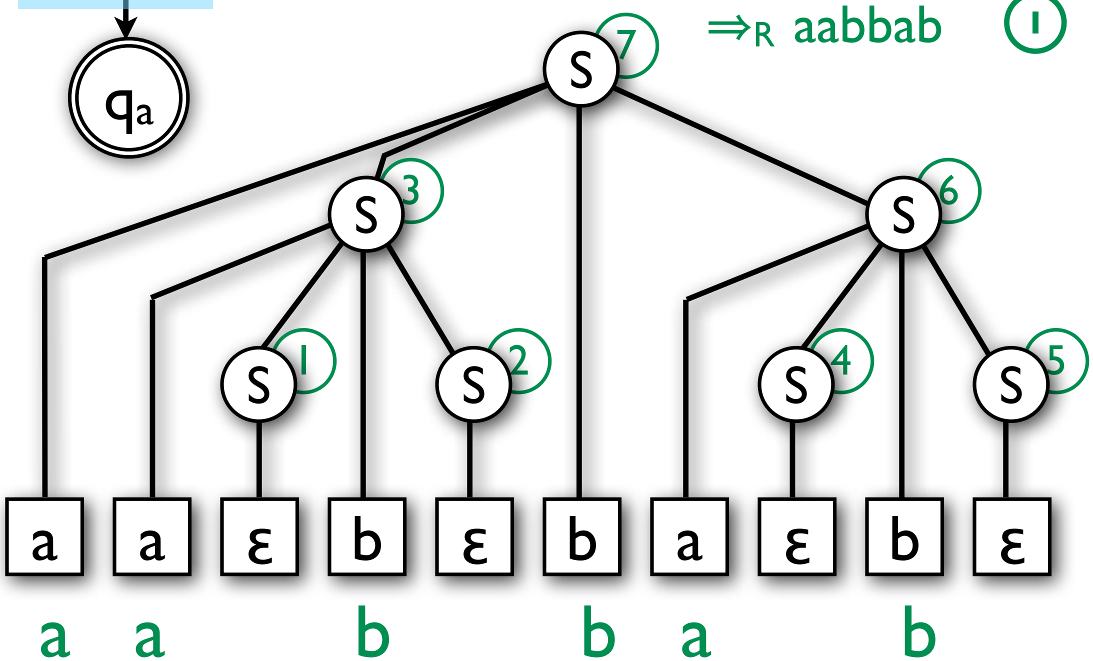
Shift-Reduce Parser



shift:
- a, ε→a
- b, ε→b

reduce:
- ε, ε→S
- ε, aSbS→S

S ⇒$_R$ aSbS 7
⇒$_R$ aSbaSbS 6
⇒$_R$ aSbaSb 5
⇒$_R$ aSbab 4
⇒$_R$ aaSbSbab 3
⇒$_R$ aaSbbab 2
⇒$_R$ aabbab 1



bot ⟷ top

State	kcatS	Input	Deriv
q_0	ε	aabbab	
q	$	aabbab	
q	$a	abbab	
q	$aa	bbab	
q	$aaS	bbab	1
q	$aaSb	bab	
q	$aaSbS	bab	2
q	$aS	bab	3
q	$aSb	ab	
q	$aSba	b	
q	$aSbaS	b	4
q	$aSbaSb	ε	
q	$aSbaSbS	ε	5
q	$aSbS	ε	6
q	$S	ε	7
q_{accept}	ε	ε	

Input accepted

Shift-Reduce (Ex 2)

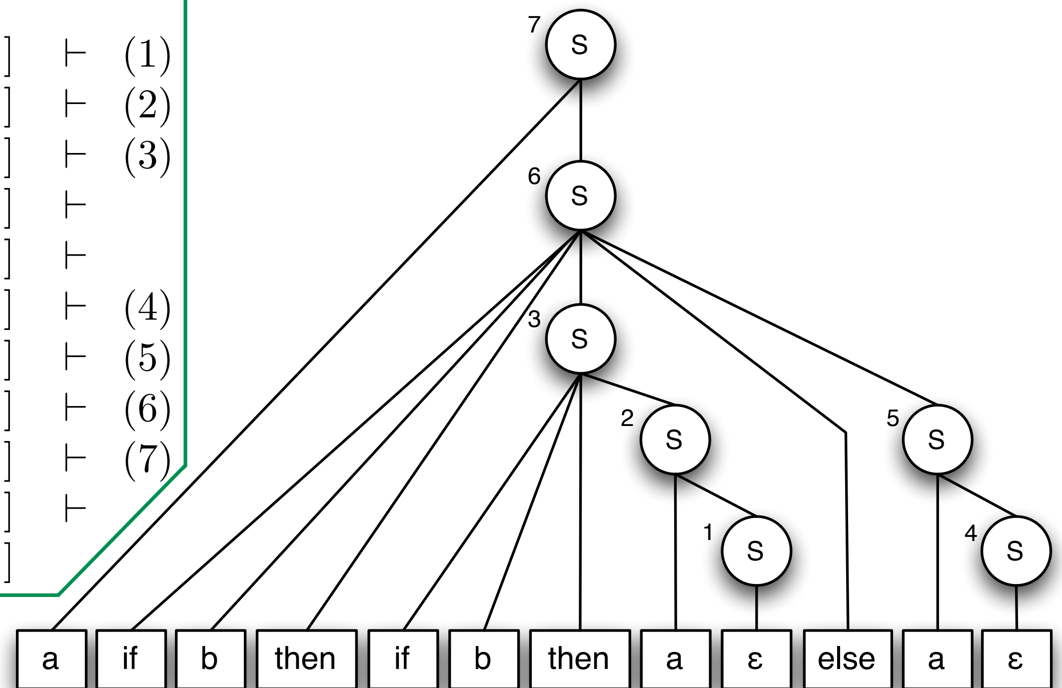
$$
\begin{aligned}
S &\rightarrow \text{a } S \\
S &\rightarrow \text{if b then } S \\
S &\rightarrow \text{if b then } S \text{ else } S \\
S &\rightarrow \varepsilon
\end{aligned}
$$

For all $\gamma \in (V \cup \Sigma)^*$ and all $w \in \Sigma^*$,

$\gamma \Rightarrow_R^k w$ if and only if $[q, \epsilon, w] \vdash^{k+|w|} [q, \gamma, \epsilon]$

[q_0,	ε,	a if b then if b then a else a]	$\vdash$	
[q,	\$,	a if b then if b then a else a]	$\vdash$	
[q,	\$ a,	if b then if b then a else a]	$\vdash$	
[q,	\$ a if,	b then if b then a else a]	$\vdash$	
	⋮ 6 more shifts			
[q,	\$ a if b then if b then a,	else a]	$\vdash$	(1)
[q,	\$ a if b then if b then a S,	else a]	$\vdash$	(2)
[q,	\$ a if b then if b then S,	else a]	$\vdash$	(3)
[q,	\$ a if b then S,	else a]	$\vdash$	
[q,	\$ a if b then S else,	a]	$\vdash$	
[q,	\$ a if b then S else a,	ε]	$\vdash$	(4)
[q,	\$ a if b then S else a S,	ε]	$\vdash$	(5)
[q,	\$ a if b then S else S,	ε]	$\vdash$	(6)
[q,	\$ a S,	ε]	$\vdash$	(7)
[q,	\$ S,	ε]	$\vdash$	
[q_a,	ε,	ε]		

$$
\begin{aligned}
S &\Rightarrow_R \text{a } S && (7)\\
&\Rightarrow_R \text{a if b then } S \text{ else } S && (6)\\
&\Rightarrow_R \text{a if b then } S \text{ else a } S && (5)\\
&\Rightarrow_R \text{a if b then } S \text{ else a} && (4)\\
&\Rightarrow_R \text{a if b then if b then } S \text{ else a} && (3)\\
&\Rightarrow_R \text{a if b then if b then a } S \text{ else a} && (2)\\
&\Rightarrow_R \text{a if b then if b then a else a} && (1)
\end{aligned}
$$



Notation: [state, stack, input]
bot→top

Correctness of shift-reduce construction

CLAIM: For all $\gamma \in (V \cup \Sigma)^*$ and all $w \in \Sigma^*$,

$$\gamma \Rightarrow_R^k w \text{ if and only if } [q, \epsilon, w] \vdash^{k+|w|} [q, \gamma, \epsilon].$$

COROLLARY: $L(M) = L(G)$

PDA Configurations:
[state, stack, input]
bot→top

PDA Moves:
[q, γα, ay] ⊢ [q’, γβ, y]
(like slide 5, except stack reversed)

Proof:

$$S \Rightarrow_R^k w \text{ if and only if } [q, \epsilon, w] \vdash^{k+|w|} [q, S, \epsilon]$$

CLAIM: $\forall \gamma \in (V \cup \Sigma)^*, \forall w \in \Sigma^*, \gamma \Rightarrow_R^k w$ only if $[q, \epsilon, w] \vdash^{k+|w|} [q, \gamma, \epsilon]$.
Basis ($k = 0$):

$$\gamma \Rightarrow_R^0 w \text{ so } \gamma = w, \text{ so } [q, \epsilon, w] \vdash^{|w|} [q, \gamma, \epsilon] \quad \text{(via } |w| \text{ shifts)}$$

Induction: Assume the claim for some $k \geq 0$. Suppose

$$\gamma \Rightarrow_R^{k+1} w$$

Let its first step be $A \to \beta$. $\exists \alpha \in (V \cup \Sigma)^*$, $\exists x, y \in \Sigma^*$ s.t.

$$\gamma = \alpha A y \Rightarrow_R \alpha \beta y \Rightarrow_R^k xy = w \text{ so } \alpha A \Rightarrow_R \boxed{\alpha \beta \Rightarrow_R^k x}$$

By the induction hypothesis, and the definition of "reduce moves":

$$\boxed{[q, \epsilon, x] \vdash^{k+|x|} [q, \alpha\beta, \epsilon]} \text{ and } [q, \alpha\beta, \epsilon] \vdash [q, \alpha A, \epsilon]$$

So

$$[q, \epsilon, xy] \vdash^{k+|x|} [q, \alpha\beta, y] \vdash^1 [q, \alpha A, y] \vdash^{|y|} [q, \alpha A y, \epsilon]$$

Thus

$$[q, \epsilon, w] \vdash^{k+1+|w|} [q, \gamma, \epsilon]$$

$\textsc{Claim}$: $\forall \gamma \in (V \cup \Sigma)^*, \forall w \in \Sigma^*, \gamma \Rightarrow_R^k w$ if $[q, \epsilon, w] \vdash^{k+|w|} [q, \gamma, \epsilon]$.

Proof of this direction is similar, and is left as an exercise.
Hint: Again induction on k; consider the last "reduce" step in the PDA's computation.

Notes

Both top-down & bottom up PDA's above are *non*deterministic. With a carefully designed grammar, and by being able to "peek" ahead at the next input symbol, it may be possible to tell *deterministically* which action to take. The CFG's for which this is possible are called LL(1) (top-down case) or LR(1) (shift-reduce case) grammars, and are important for programming language design.
Every language accepted by a deterministic PDA has an LR (1) grammar, but not all grammars for a given language are LR(1), and for some CFL's *no* grammar is LR(1).

Some PDA Facts

$\forall y$, $\boxed{[p, \alpha, x] \vdash^* [q, \beta, \varepsilon]}$ if and only if $\boxed{[p, \alpha, xy] \vdash^* [q, \beta, y]}$

Why? PDA can't test "end of input" or "peek ahead," so presence/absence of y is invisible. (A bit like the "context-free" property in a CFG.)

$\forall \gamma$, $\boxed{[p, \alpha, x] \vdash^* [q, \beta, \varepsilon]}$ implies $\boxed{[p, \gamma\alpha, x] \vdash^* [q, \gamma\beta, \varepsilon]}$

Why? γ is "buried" on bottom of stack, so computation allowed in its absence is still valid in its presence. Note the *converse* is, in general, *false!* Computation on right might pop part of γ, then push it back, whereas one at left would block at the attempted pop. Important special case: $\alpha = \beta = \varepsilon$:

$\boxed{p \rightarrow^{x}\rightarrow q \text{ allowed on empty stack} \Rightarrow \text{allowed on any stack}}$

Q: What L solves this equation?

$$L \subseteq \{a,b\}^*$$

$$L = \{\varepsilon\} \cup \{a\} \bullet L \bullet \{b\}$$

Answer:

$$L = \{\, a^n b^n \mid n \geq 0 \,\}$$

Compare to:

$$S \rightarrow \varepsilon \mid a\, S\, b$$

Q: What L solves this equation?

$L, X \subseteq \Sigma^*$ (X fixed, e.g. "palindromes" or "odd len")

$L = \{\varepsilon\} \cup X \cup L \bullet L$

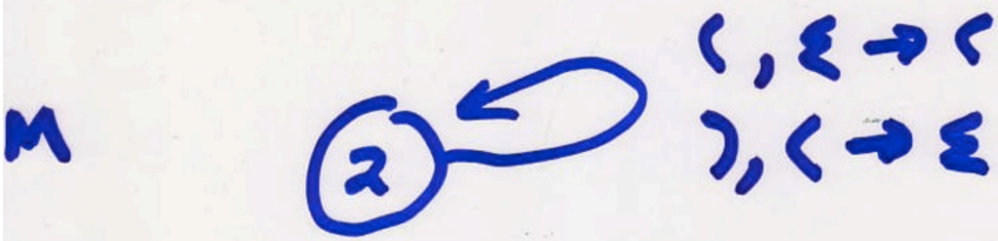
Alt phrasing: the smallest set containing ε and all of X and is closed under concatenation?

Answer:

$L = X^*$

Compare to:

$S \to \varepsilon \mid S_{grammar\_for\_X} \mid S\ S$



$$L_{22} = \{ x \mid [2, \varepsilon, x] \vdash^* [2, \varepsilon, \varepsilon] \}$$

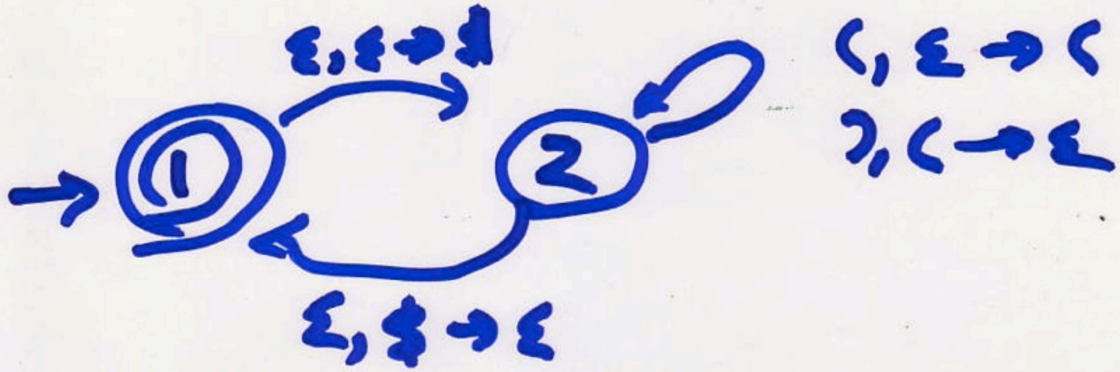
In English? L_{22} = the set of input strings x that allow M to go from state 2 to state 2, starting & ending with empty stack.

An equation?

$$L_{22} = \{\varepsilon\} \cup L_{22} \cdot L_{22} \cup (\cdot L_{22} \cdot)$$

$$S \to \varepsilon \mid SS \mid (S)$$

$$L(M) \stackrel{?}{=} \phi$$



L_{22} – as before

$\{\varepsilon\} \cup L_{22} \cdot L_{22} \cup (L_{22})$

$L_{11} = \{\varepsilon\} \cup L_{11} \cdot L_{11} \cup \varepsilon L_{22} \varepsilon$

$A_{22} \to \varepsilon \mid A_{22} A_{22} \mid (A_{22})$

$A_{11} \to \varepsilon \mid A_{11} A_{11} \mid A_{22}$

$L_{12} = \emptyset = L_{21}$

PDA to CFG, general construction

I. WLOG, assume PDA:

a) has only one final state

b) accepts only when stack is empty, and

c) all transitions either push or pop, never both/neither

II. $\forall\, p, q \in Q$

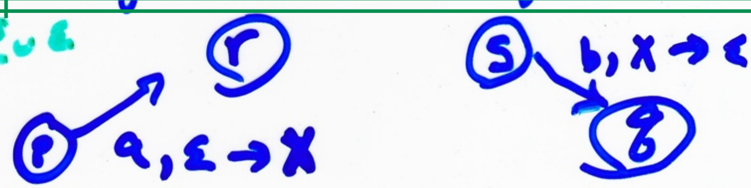
nonterminal A_{pq} ← Goal: A_{pq} gives inputs allowing M to go from state p to state q, *starting & ending with empty stack.*

$A_{pp} \to \varepsilon \quad \forall p \in Q$ ← read nothing/no transition

$A_{pq} \to A_{pr} A_{rq} \quad \forall p, q, r \in Q$ ← p to q via any intermediate r

$\forall\, a, b \in \Sigma \cup \varepsilon$

$X \in \Gamma$



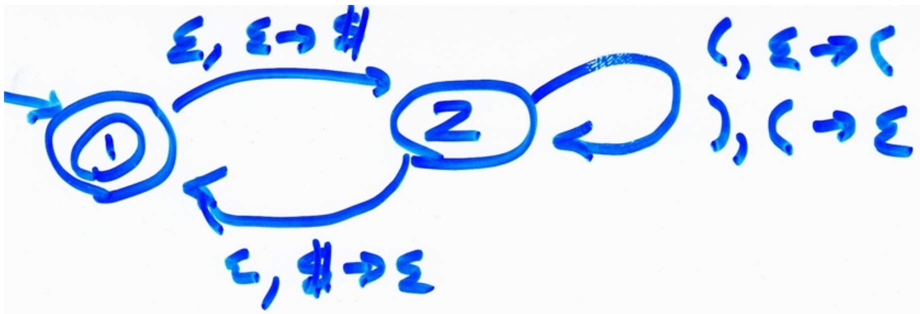
$A_{pq} \to a\, A_{rs}\, b$

← 1. read a + push X

2. go from r to s *on empty stack*, so X re-exposed

3. read b + pop X

Grammar start symbol = $A_{\text{start-state, final-state}}$



$$\boxed{A_{pp} \to \varepsilon \quad \forall p \in Q}$$

$A_{11} \to \varepsilon$

$A_{22} \to \varepsilon$

$\forall a, b \in \Sigma \cup \varepsilon$

$X \in \Gamma$

$p \xrightarrow{a,\, \varepsilon \to X} r \qquad s \xrightarrow{b,\, X \to \varepsilon} q$

$$A_{pq} \to a\, A_{rs}\, b$$

p	a	X	r	s	b	q
1	ε	$	2	2	ε	1
2	(	(	2	2	)	2

$A_{11} \to \varepsilon A_{22} \varepsilon$

$A_{22} \to (A_{22})$

$$\boxed{A_{pq} \to A_{pr} A_{rq} \quad \forall p, q, r \in Q}$$

$A_{11} \to A_{11} A_{11} \mid A_{12} A_{21}$

$A_{12} \to A_{11} A_{12} \mid A_{12} A_{22}$

$A_{21} \to A_{21} A_{11} \mid A_{22} A_{21}$

$A_{22} \to A_{21} A_{12} \mid A_{22} A_{22}$

NB: G can be simplified. E.g., remove A_{12}, A_{21} & rules using them, since, e.g., $\nexists\, x \in \Sigma^*$ s.t. $A_{21} \Rightarrow^* x$.

This is just fine in the construction, since there is also no x s.t. $[2, \varepsilon, x] \vdash^* [1, \varepsilon, \varepsilon]$.

Easier to construct useless rules locally than to sort out such ramifications globally.

Claim $\forall x \in \Sigma^* \ \forall p,q \in Q \quad A_{pq} \stackrel{*}{\Rightarrow} x$

iff $[p, \varepsilon, x] \vdash^* [q, \varepsilon, \varepsilon]$ ← I.e., A_{pq} gives set of inputs that allow M to go from state p to state q, starting & ending with empty stack.

Cor $L(G) = L(M)$

Since $L(G) = \{x \mid A_{init,final} \stackrel{*}{\Rightarrow} x\}$ ↑ defn.

$= \{x \mid [init, \varepsilon, x] \vdash^* [final, \varepsilon, \varepsilon]\}$ ↑ by claim

$= L(M)$ ↑ defn. (and fact that M's stack is empty when it enters F)

Claim $\forall x \in \Sigma^* \ \forall p,q \in Q \quad A_{pq} \Rightarrow^* x$ iff $[p, \varepsilon, x] \vdash^* [q, \varepsilon, \varepsilon]$

I.e., A_{pq} gives set of inputs that allow M to go from state p to state q, starting & ending with empty stack.

Claim ($\Longleftarrow$) induct on deriv length

basis

$A_{pq} \Rightarrow^0 x$: impossible; nothing to prove

$A_{pq} \Rightarrow^1 x$: must be $x = \varepsilon$, $p = q$

$[p, \varepsilon, \varepsilon] \vdash^* [q, \varepsilon, \varepsilon]$

Ind $\Rightarrow^{k+1}$ either (i) $A_{pq} \to A_{pr} A_{rq} \Rightarrow^k x$

(ii) $A_{pq} \to a A_{rs} b \Rightarrow^k x$

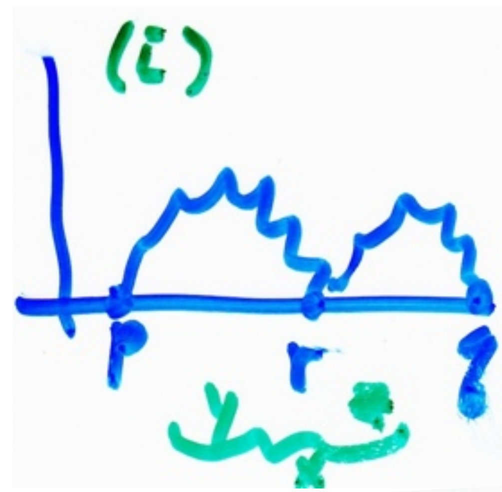
Case (i): exercise

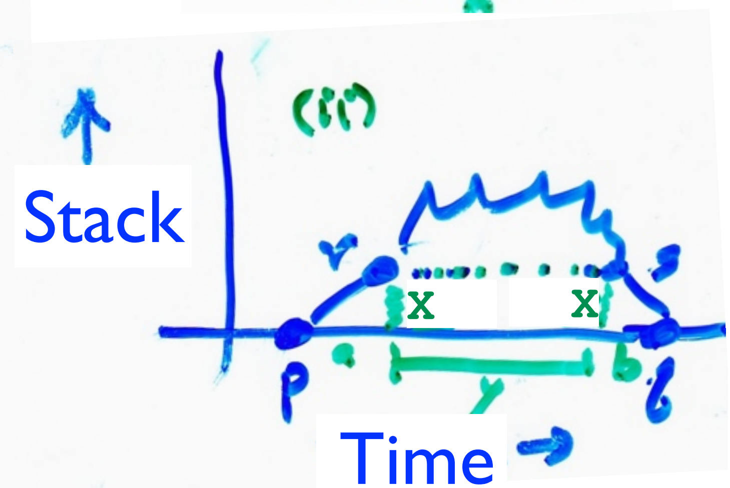
case (ii):

$x = ayb$ & $A_{rs} \Rightarrow^k y$

by ind $[r, \varepsilon, y] \vdash^* [s, \varepsilon, \varepsilon]$

since ⊛ing $[p, \varepsilon, ayb] \vdash [r, X, yb] \vdash^* [s, X, b] \vdash [q, \varepsilon, \varepsilon]$





Claim $\forall x \in \Sigma^*$ $\forall p,q \in Q$ $A_{pq} \stackrel{*}{\Rightarrow} x$ iff $[p, \epsilon, x] \vdash^* [q, \epsilon, \epsilon]$

I.e., A_{pq} gives set of inputs that allow M to go from state p to state q, starting & ending with empty stack.

$\Leftarrow$ direction of claim is similar, by induction on # of steps in $\vdash^*$

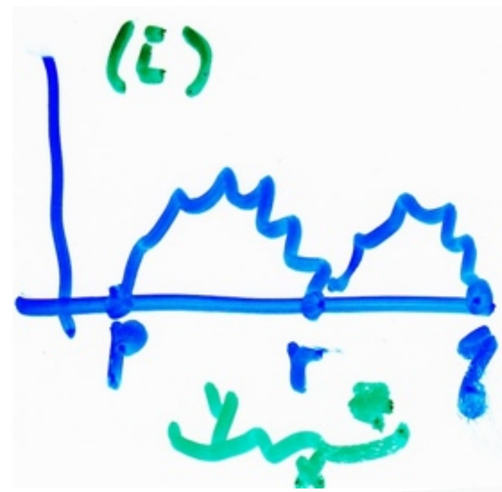
basis: 0 steps, use ϵ rule in G

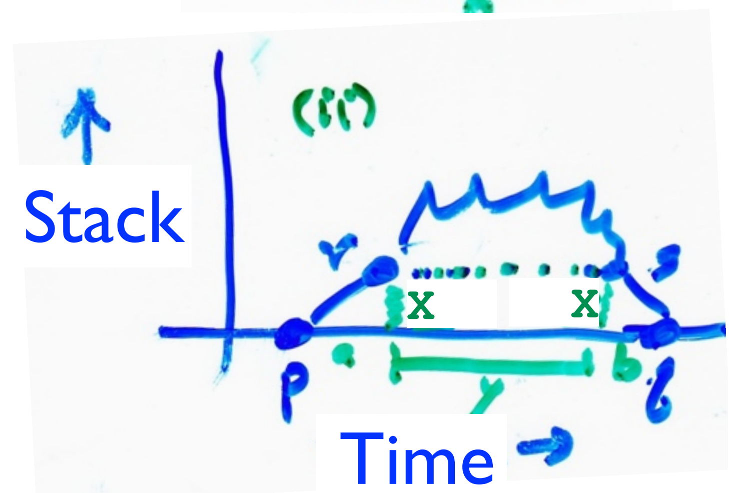
ind: $k+1 > 0$ steps, then

Stack either is (case i) or is not (case ii) empty at some intermediate step.

In case i, I.H. & construction give $A_{pq} \to A_{pr} A_{rq}$ etc.

In case ii, $A_{pq} \to a A_{rs} b$ etc.





This construction & proof are just like the text's version, so more details there.

Summary: PDA $\equiv$ CFG

Pushdown stack conveniently allows simulation of recursion in CFG

E.g., $\{a^n b^n\}$ or $\{ww^R\}$ or balanced parens, etc.: push some, match later

Nondeterminism sometimes essential

- e.g., "guess middle"; there is *no* "subset constr" for NPDA

$G \subseteq M$: guess deriv., using stack carefully ($\Rightarrow_L$ or $\Rightarrow_R$)

- basis for parsers in most compilers, e.g.

$M \subseteq G$: $A_{pq} = \{x \mid$ go from p to q *on empty stack*$\}$