CS311—Computational Structures

More about PDAs & Context-Free Languages

Lecture 9

Andrew P. Black
Andrew Tolmach

Portland State University

Three important results

1. Any CFG can be “simulated” by a PDA
2. Any PDA can be “simulated” by a CFG
3. Pumping Lemma: not all languages are Context-free

but first:

some notation from Hopcroft et al.

PDA Acceptance, Revisted

- Consider a PDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$
- An **instantaneous description (ID)** of M has the form (q,w,t)

where $q \in Q$ is the current state,
$w \in \Sigma^*$ is the unread input,
$t \in \Gamma^*$ is the current stack

(with top of stack on the left)

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

Portland State UNIVERSITY

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

State

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q, aw, bt) \vdash (p, w, ct)$$

Input

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

Portland State University

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

Stack

iff

$$(p,c) \in \delta(q,a,b)$$

for some

$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

Portland State UNIVERSITY

PDA Acceptance, by $\vdash$

- We define a relation $\vdash$ on ID's; $\vdash$ captures what it means for the PDA to take a single step:

$$(q,aw,bt) \vdash (p,w,ct)$$

iff

$$(p,c) \in \delta(q,a,b)$$

for some
$p,q \in Q$; $a \in \Sigma_\varepsilon$; $w \in \Sigma^*$; $b,c \in \Gamma_\varepsilon$; $t \in \Gamma^*$

Portland State University

PDA Acceptance, by $\vdash$

- We write $\vdash^*$ to mean “zero or more steps using $\vdash$”
- Then we say M **accepts** w (by final state) iff $(q_0,w,\varepsilon) \vdash^* (q,\varepsilon,t)$ for some $q \in F$ and any $t \in \Gamma^*$
- As usual, the language accepted by M is just $\{w \mid w \text{ is accepted by } M\}$

Non-determinism is *fundamental*

- Unlike with finite automata, PDA non-determinism cannot be transformed away.
- Deterministic PDA's (DPDA's) recognize strictly fewer languages than non-deteristic ones
- DPDAs are useful in practice as the basis for language parser implementations

Portland State UNIVERSITY

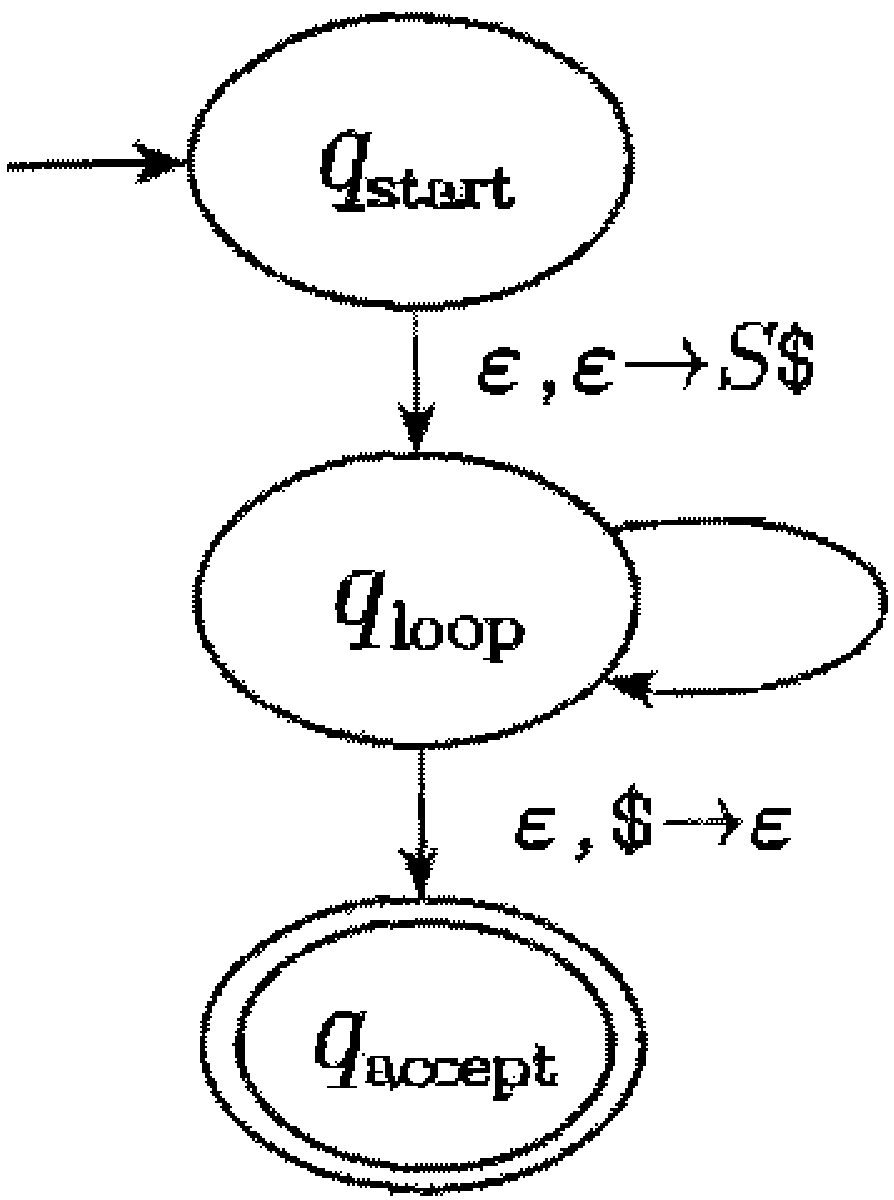
PDA’s and CFG’s are equivalent!

- If G is a CFG, we can build a (non-deterministic) PDA M with L(M) = L(G)
  - That is, we can build a **parser** for G
  - This is an easy construction
- If M is a PDA, we can construct a CFG G with L(G) = L(M)
  - This is harder

Parsing: PDA from CFG

- Parsing is the process of going from a sentence (string) in the language to a derivation tree.
- *Top-down* parsing starts at the "top", with the start-symbol of the grammar and derives a string
- Bottom-up parsing starts at the "bottom" with a string and figures out how to derive that string from the start-symbol.

PDAs for Top-down parsing



$\varepsilon, A \rightarrow w$ for rule $A \rightarrow w$

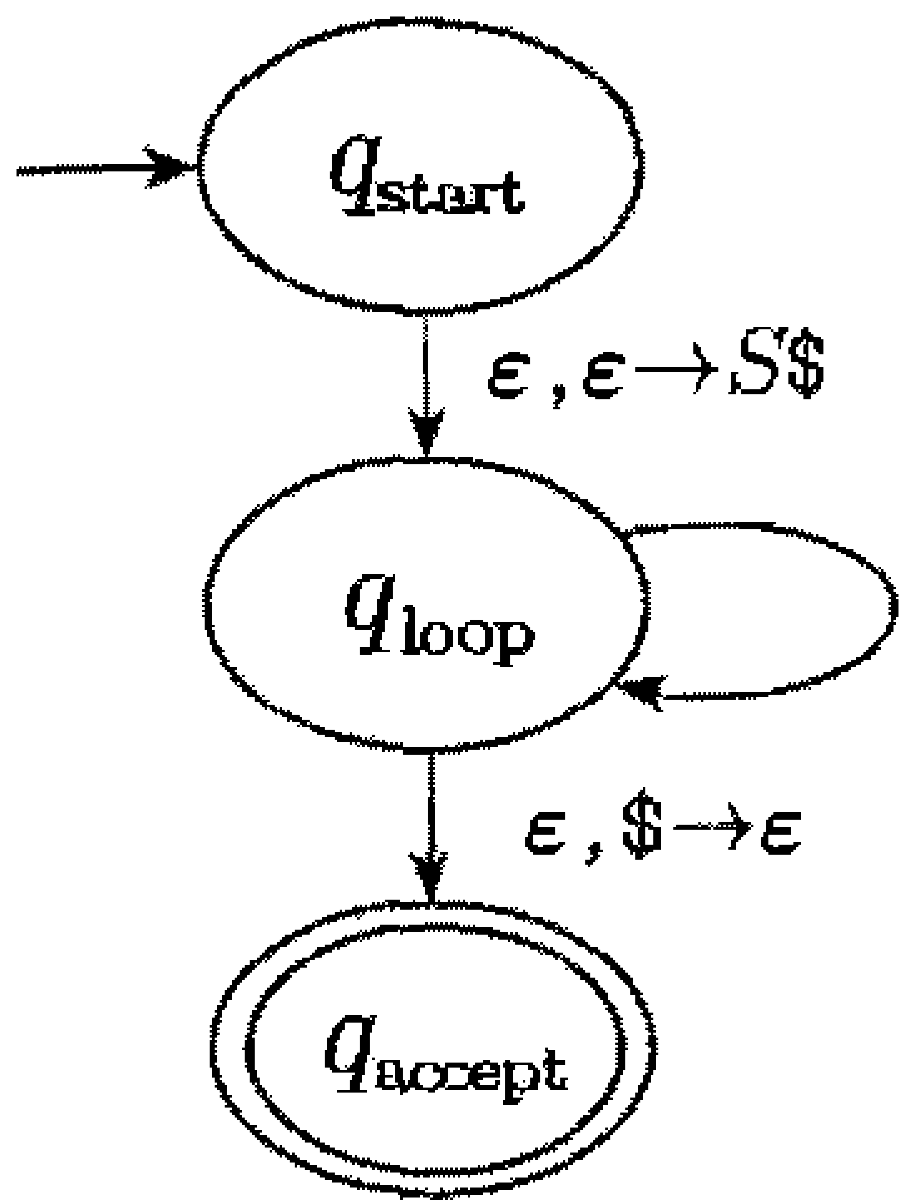
$a, a \rightarrow \varepsilon$ for terminal a

Note: we're allowing machine to push multiple symbols onto stack in one move

Portland State University

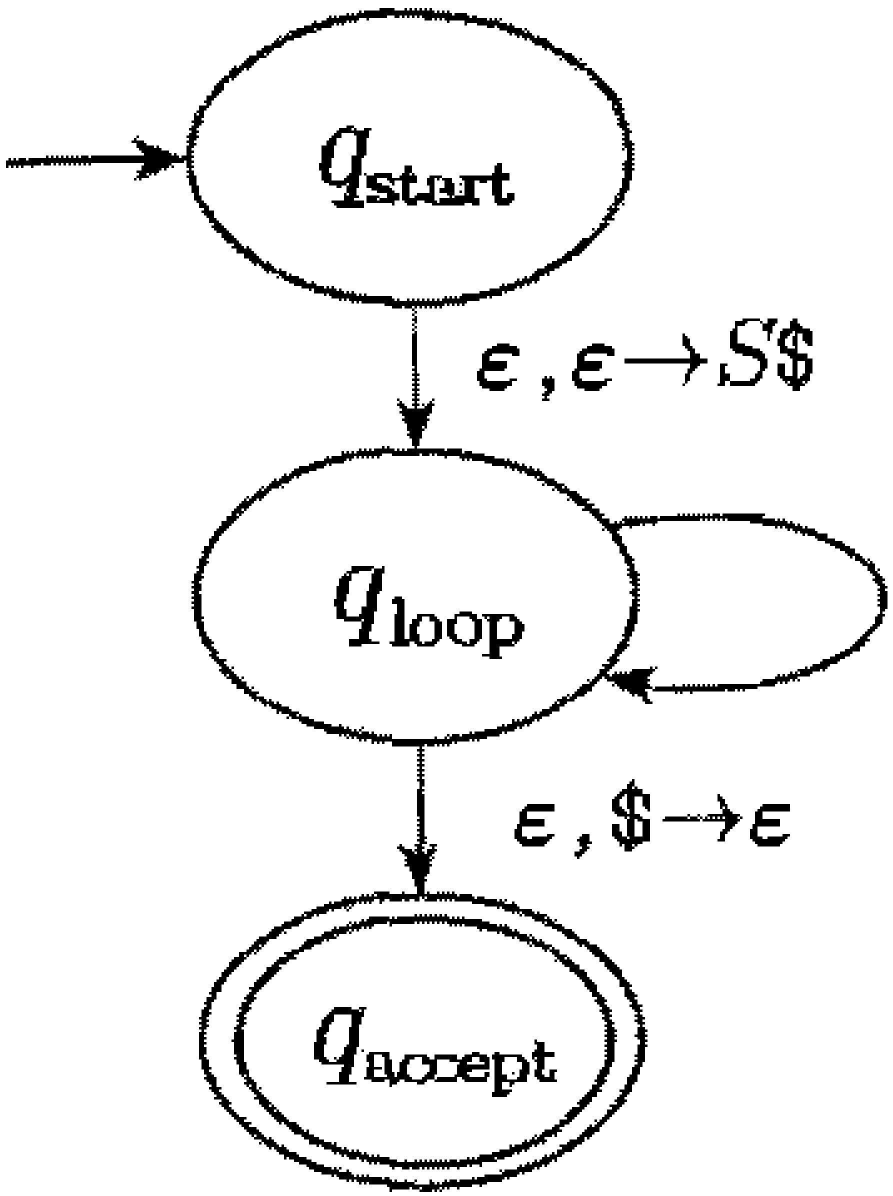
Top-down parsing PDA example

$S \rightarrow aSa$
$S \rightarrow B$
$B \rightarrow bB$
$B \rightarrow \varepsilon$



$\varepsilon, S \rightarrow aSa$
$\varepsilon, S \rightarrow B$
$\varepsilon, B \rightarrow bB$
$\varepsilon, B \rightarrow \varepsilon$
$a, a \rightarrow \varepsilon$
$b, b \rightarrow \varepsilon$

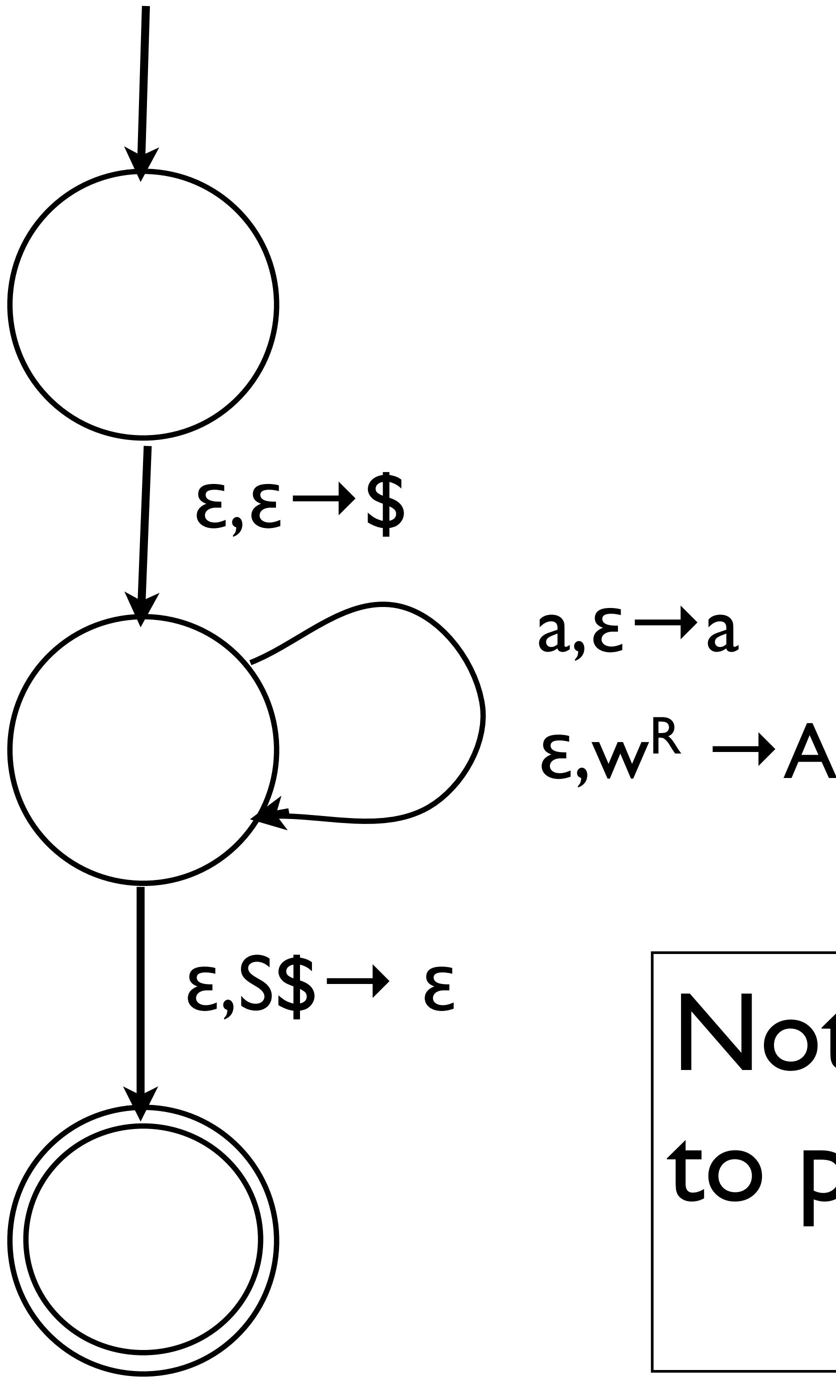
Top-down parsing PDA example



$\varepsilon, S \rightarrow aSa$
$\varepsilon, S \rightarrow B$
$\varepsilon, B \rightarrow bB$
$\varepsilon, B \rightarrow \varepsilon$
$a, a \rightarrow \varepsilon$
$b, b \rightarrow \varepsilon$

State	Input	Stack
start	aabaa	
loop	aabaa	S$
loop	aabaa	aSa$
loop	abaa	Sa$
loop	abaa	aSaa$
loop	baa	Saa$
loop	baa	Baa$
loop	baa	bBaa$
loop	aa	Baa$
loop	aa	aa$
loop	a	a$
loop		$
accept		

Alternative: Bottom-up parsing



for each a∈Σ

for each rule A → w

Note: we're allowing machine to pop multiple symbols from stack in one move

Building a CFG G from a PDA P
[method from Sipser; IALC is somewhat different]

Key idea: each string derived from A_{pq}, is capable of taking the PDA from state p with empty stack to state q with empty stack.

1. We seek to build a grammar that has the property in the box.
2. If an input string drives P from state p with empty stack to state q with empty stack, it will also move it from p to q with arbitrary stuff on the stack.

Building a CFG G from a PDA P

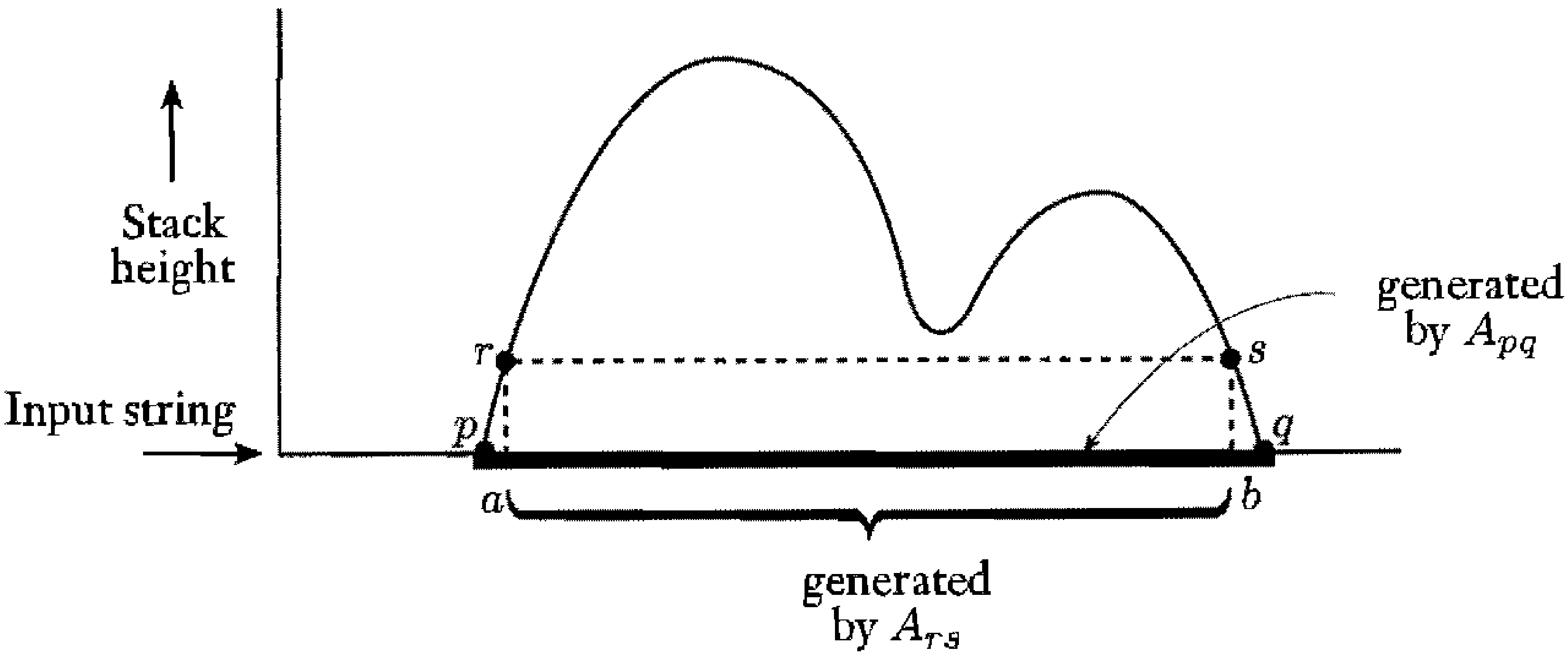
Invariant: each string derived from A_{pq}, is capable of taking the PDA from state p with empty stack to state q with empty stack.

- Start by simplifying the problem:
  - Modify P so that it has a start state σ, a single final state φ, so that it starts and finishes with an *empty stack,* and so that each transition *pushes* or *pops* a single symbol onto the stack.
- How to do this?
- Now we need to write a grammar with start symbol $A\sigma\varphi$, such that it satisfies the invariant

- How can P move from state p when its stack is empty?
  - First move must be to **push** some symbol onto the stack
  - Last move must be to pop a symbol off the stack.
  - Maybe the stack does not become empty in between … or maybe it does.
    - So, there are two cases

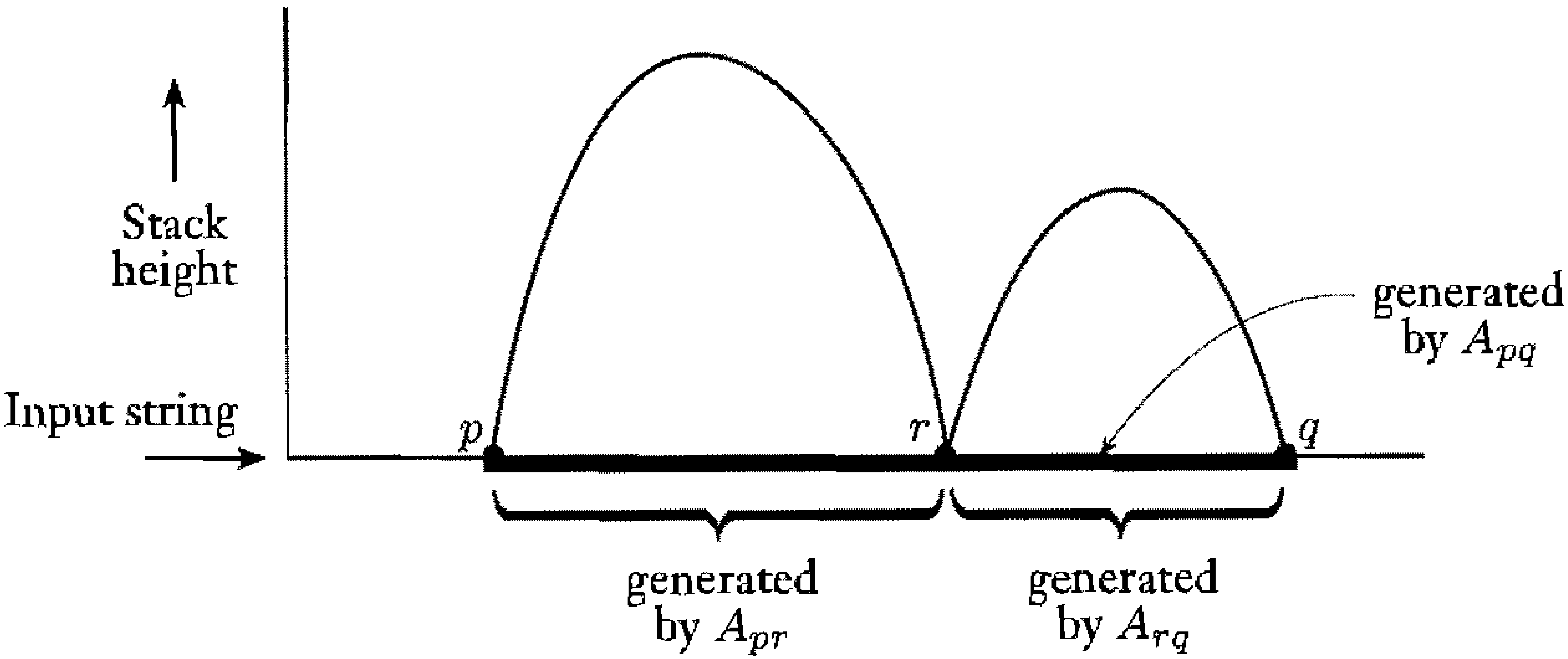
- Suppose that the stack does *not* become empty in between.
  - First, machine reads some *a*, pushes some *X*, and goes to some state, say r
  - Then it does something (maybe complicated), ending in some state s
  - Finally, it pops the same *X*, reads some *b* and goes to state q.
- This corresponds to the grammar production $A_{pq} \rightarrow aA_{rs}b$, where A_{rs} satisfies the invariant.
- Note that a and/or *b* might be ε

- In pictures:



Stack height
Input string
r
s
p
q
a
b
generated by A_{pq}
generated by A_{rs}

- Suppose that the stack becomes empty again in between



- then the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ does the job

Construction

- Let P = (Q, Σ, Γ, δ, σ, ε, $\{\varphi\}$). Construct G with variables $\{A_{pq} \mid p, q \in Q\}$, start symbol $A_{\sigma\varphi}$, terminals Σ, and rules R defined as follows:

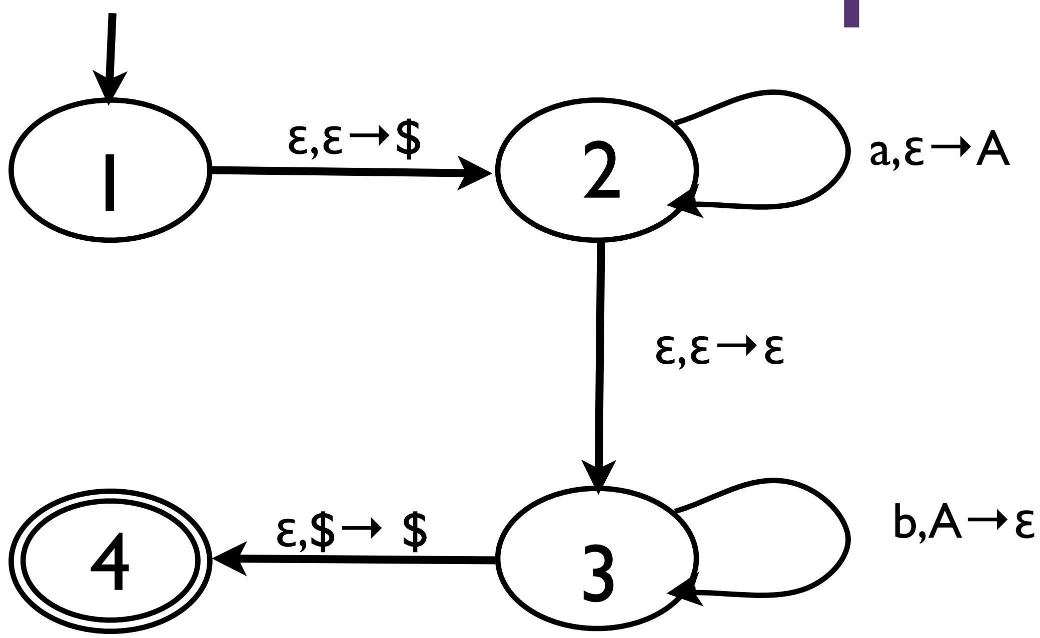
1. For each $p \in Q$, the rule $A_{pp} \rightarrow \varepsilon \in R$.
2. For each $p, q, r \in Q$, the rule $A_{pq} \rightarrow A_{pr}A_{rq} \in R$
3. For each $p, q, r, s \in Q$, $x \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, $\delta(p, a, \varepsilon) \ni (r, x)$ and $\delta(s, b, x) \ni (q, \varepsilon)$, the rule $A_{pq} \rightarrow aA_{rs}b \in R$

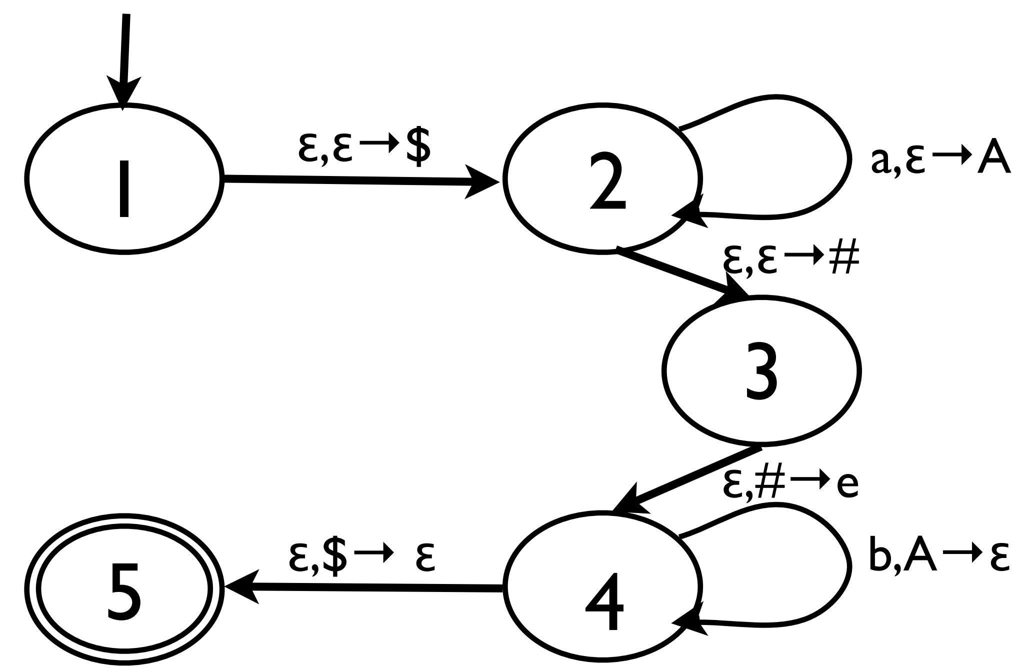
Proof Outline

- The proof that this construction works requires two things

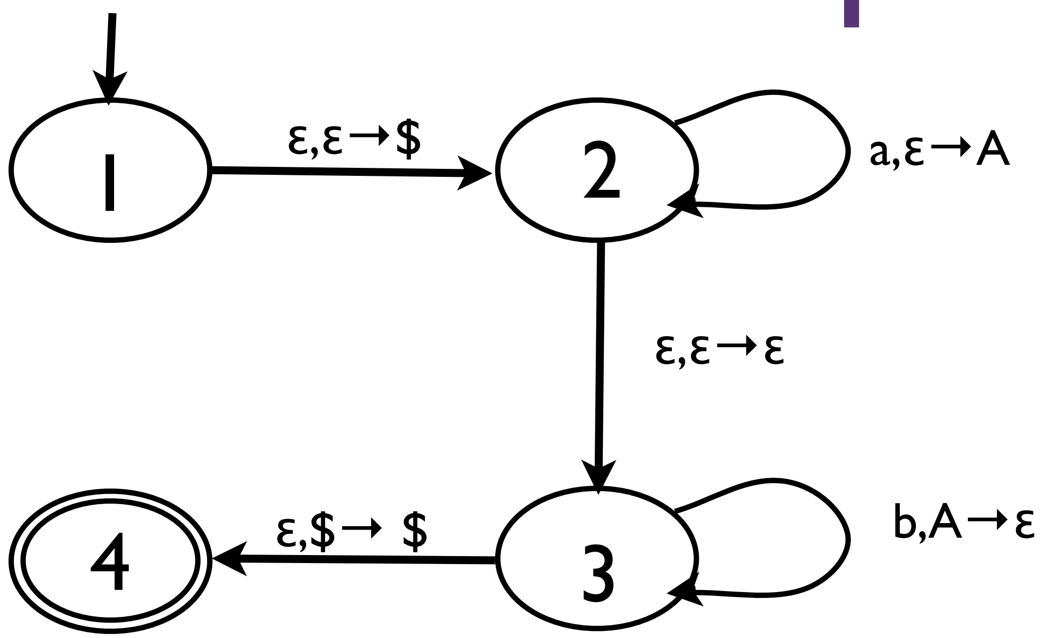
  1. Any string generated by A_{pq} will in fact bring P from state p with empty stack to state q with empty stack, and
  2. All strings capable of bringing P from state p with empty stack to state q with empty stack can in fact be generated by A_{pq}

Example: PDA for $a^n b^n$

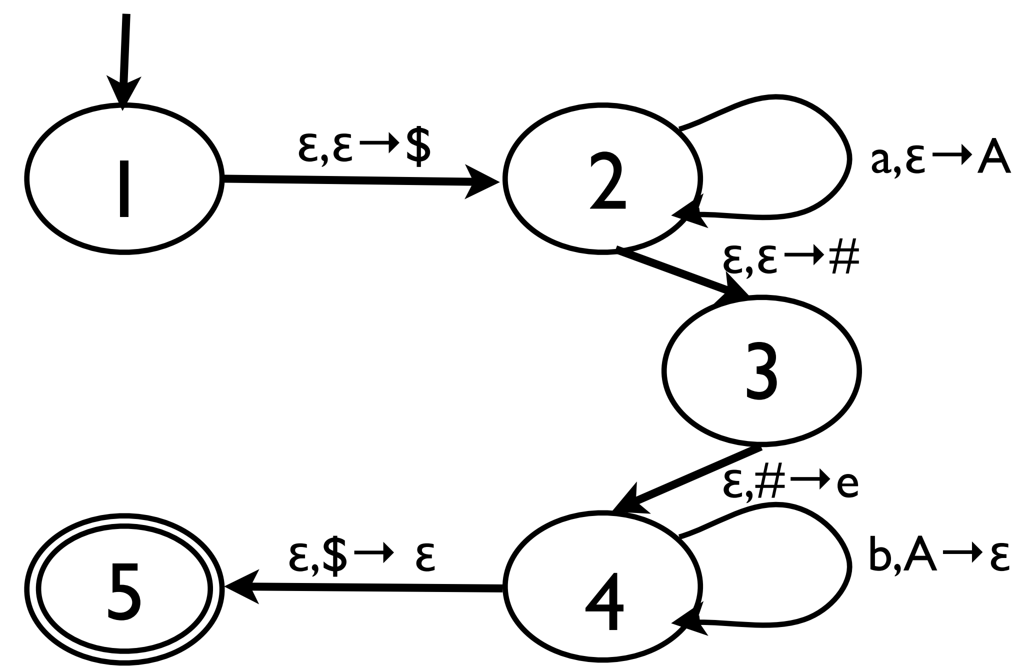




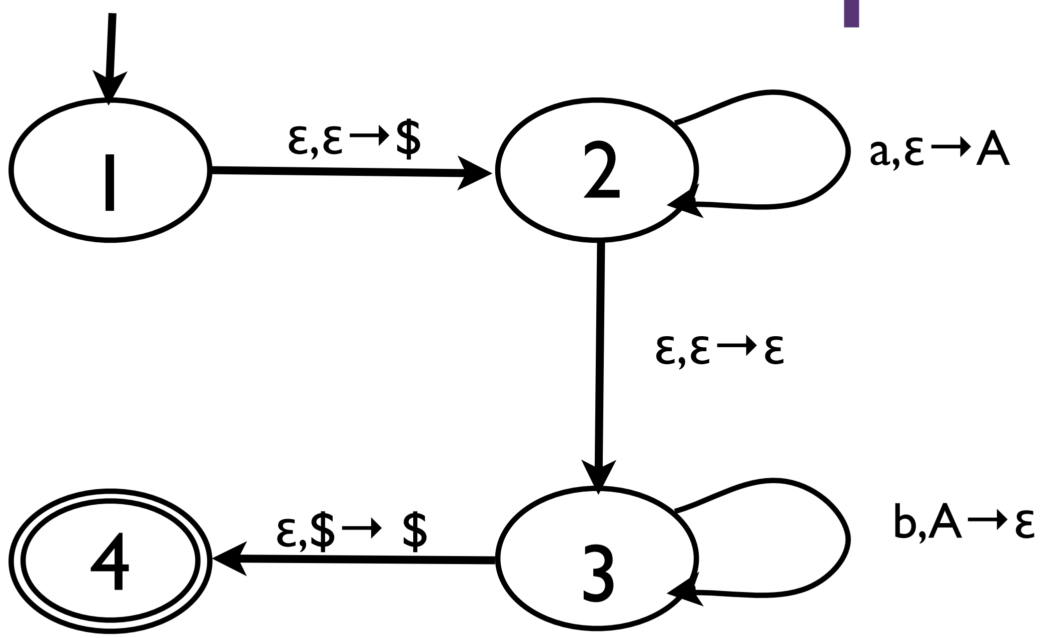
Example: PDA for $a^n b^n$



- Does not meet the restrictions

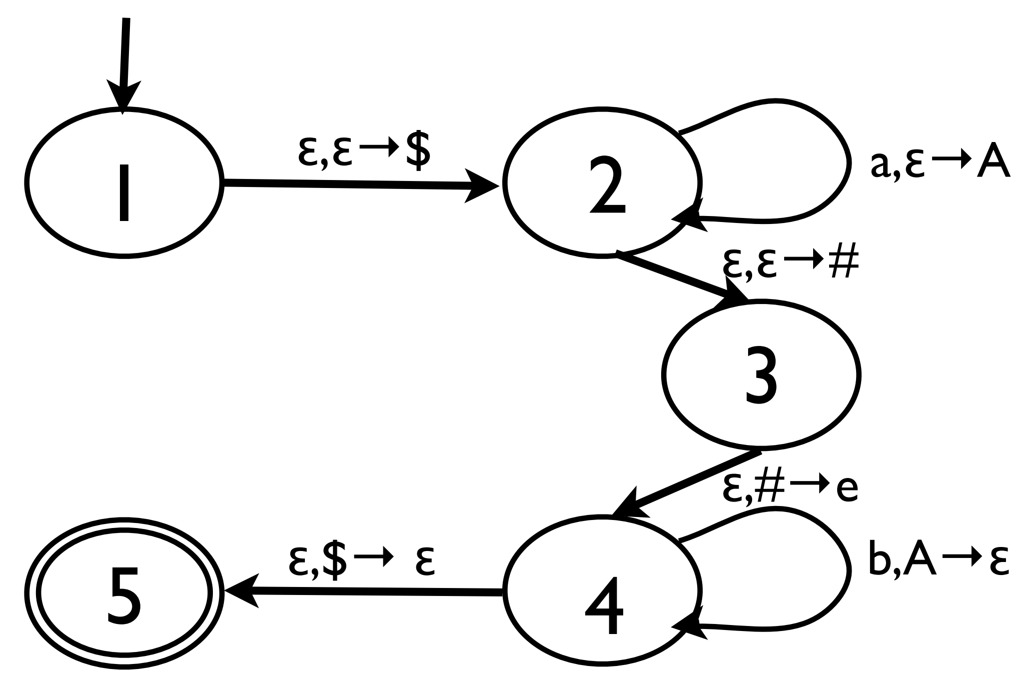


Example: PDA for $a^n b^n$

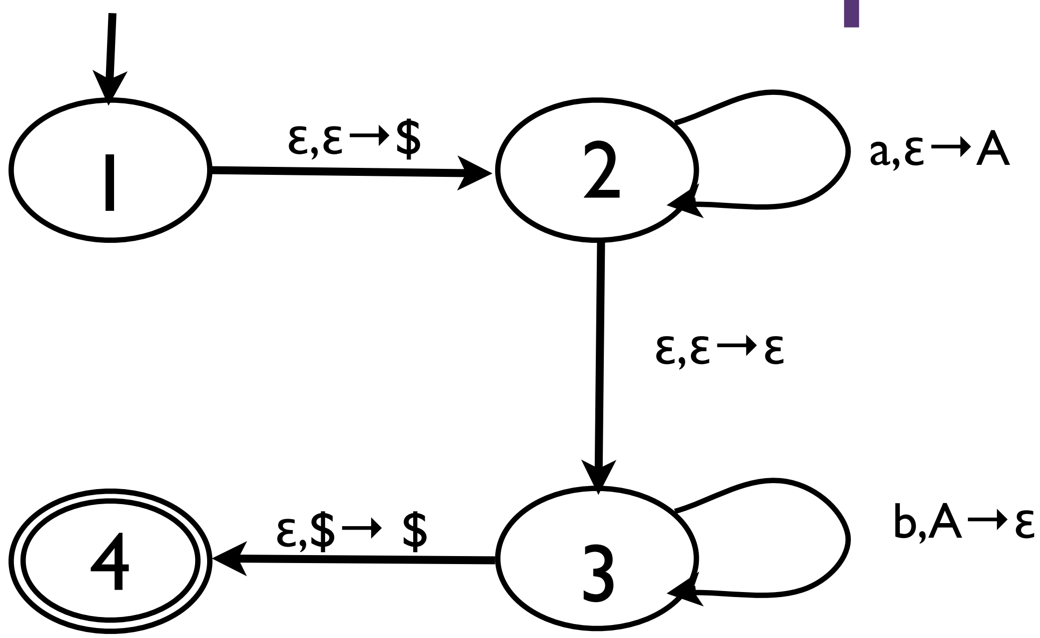


- Does not meet the restrictions

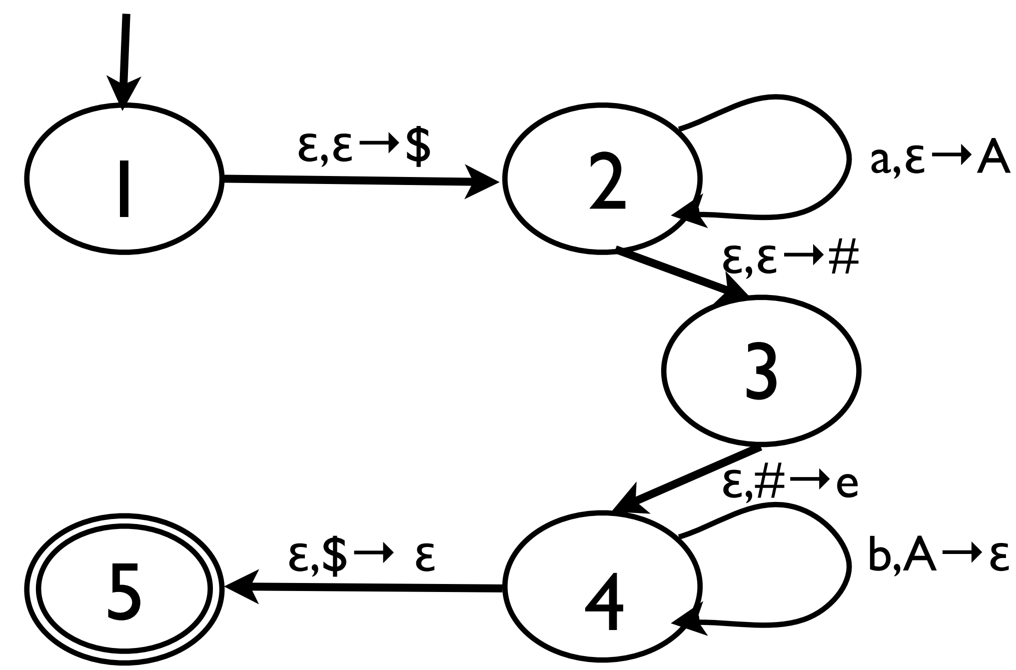
1. Stack must start and finish empty



Example: PDA for $a^n b^n$

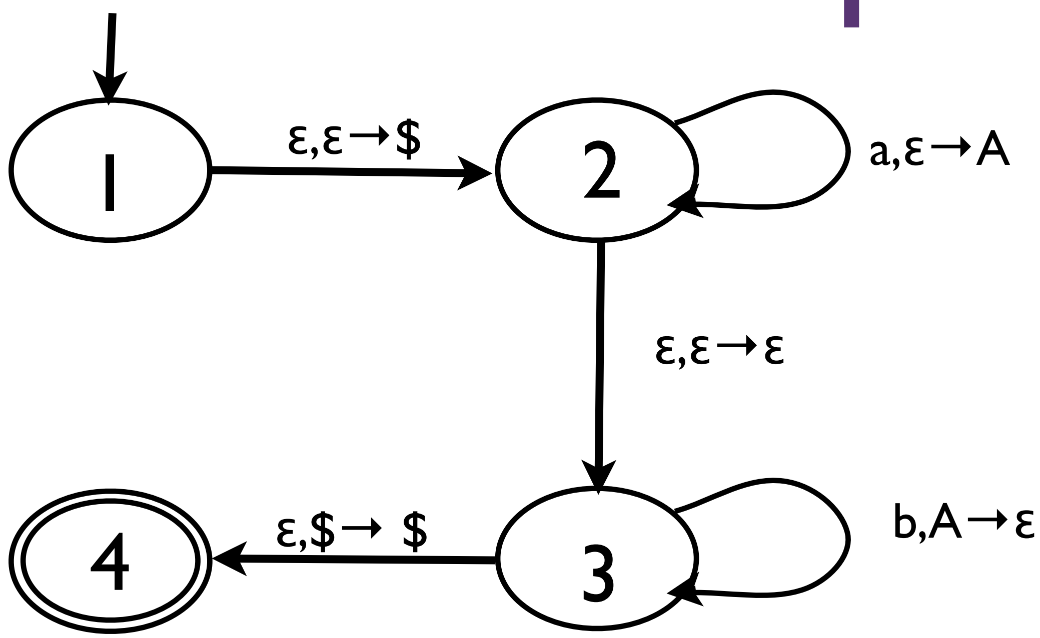


- Does not meet the restrictions

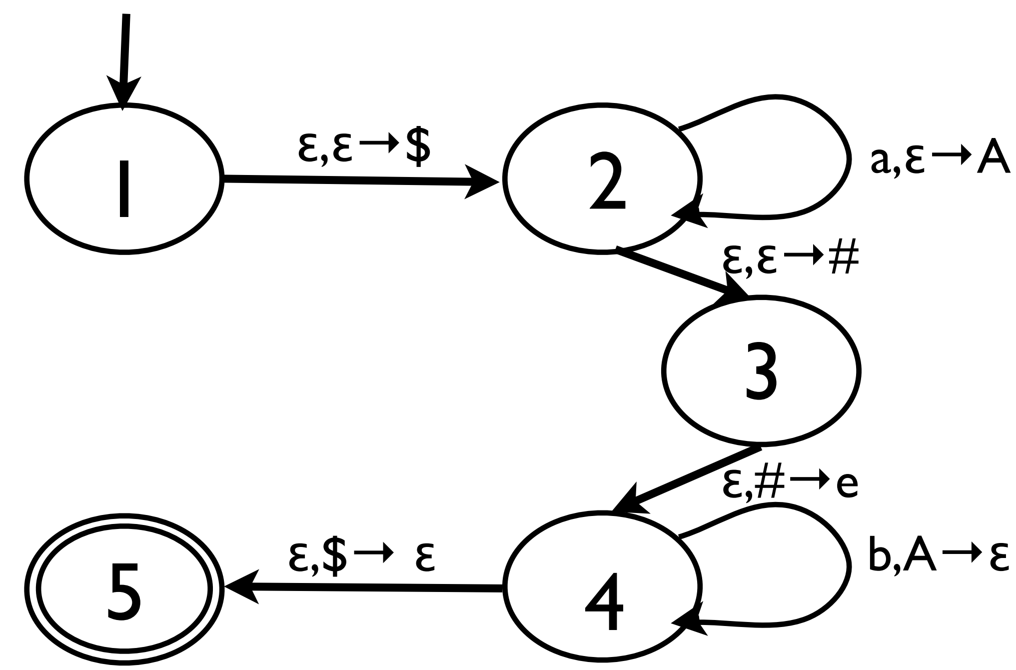


1. Stack must start and finish empty
2. Single final state

Example: PDA for $a^n b^n$

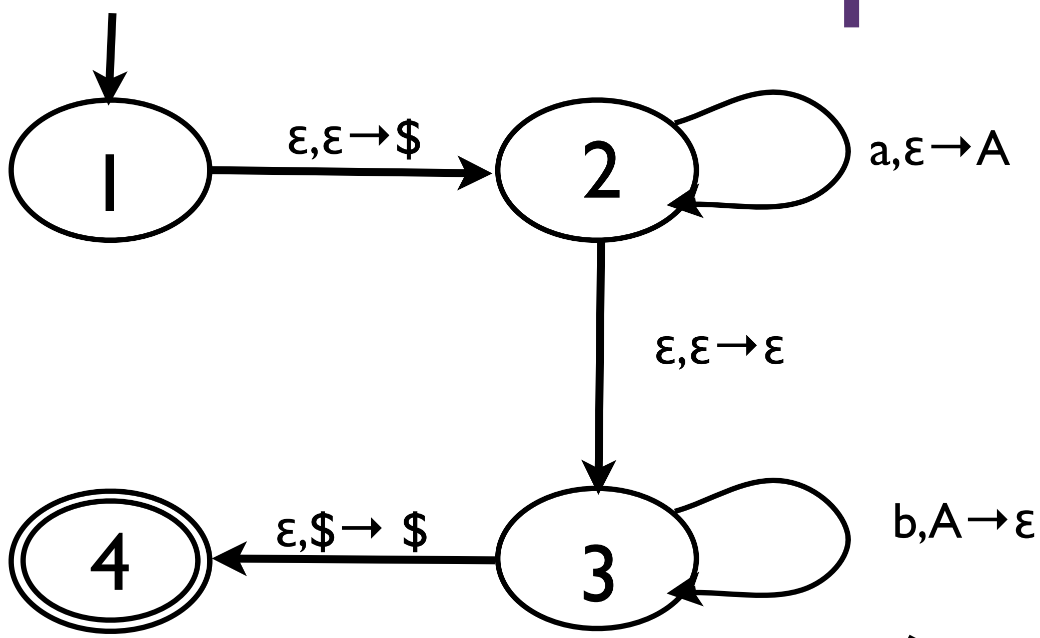


- Does not meet the restrictions

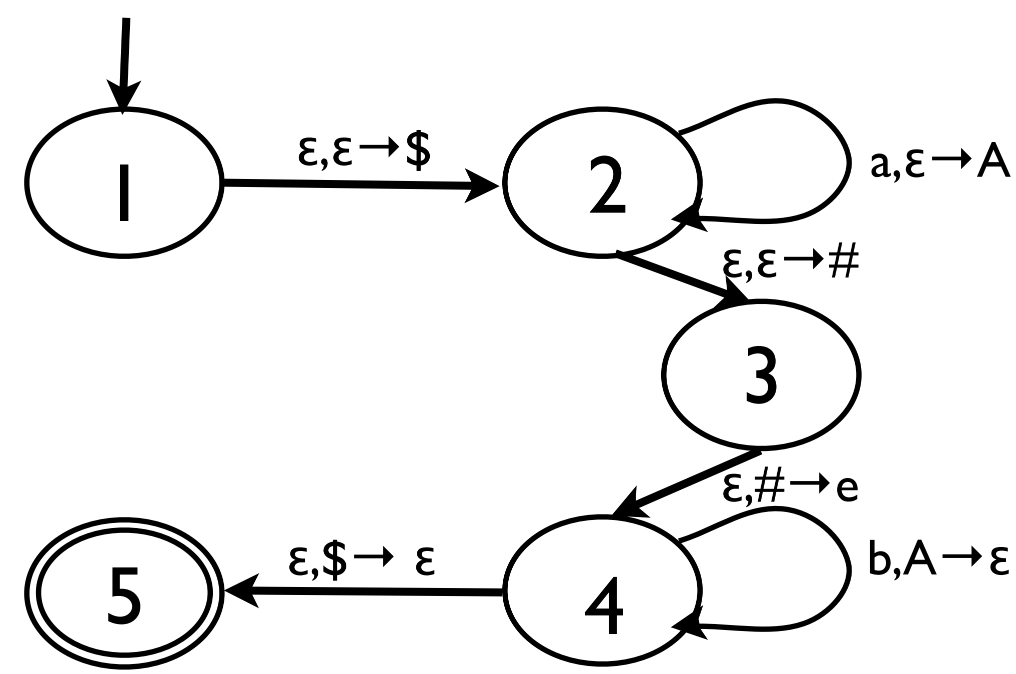


1. Stack must start and finish empty
2. Single final state
3. Every transition must be a push or a pop

Example: PDA for $a^n b^n$



- Does not meet the restrictions



1. Stack must start and finish empty
2. Single final state
3. Every transition must be a push or a pop

Grammar:

from rule 1:

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

$A_{33} \to \varepsilon \quad A_{44} \to \varepsilon \quad A_{55} \to \varepsilon$

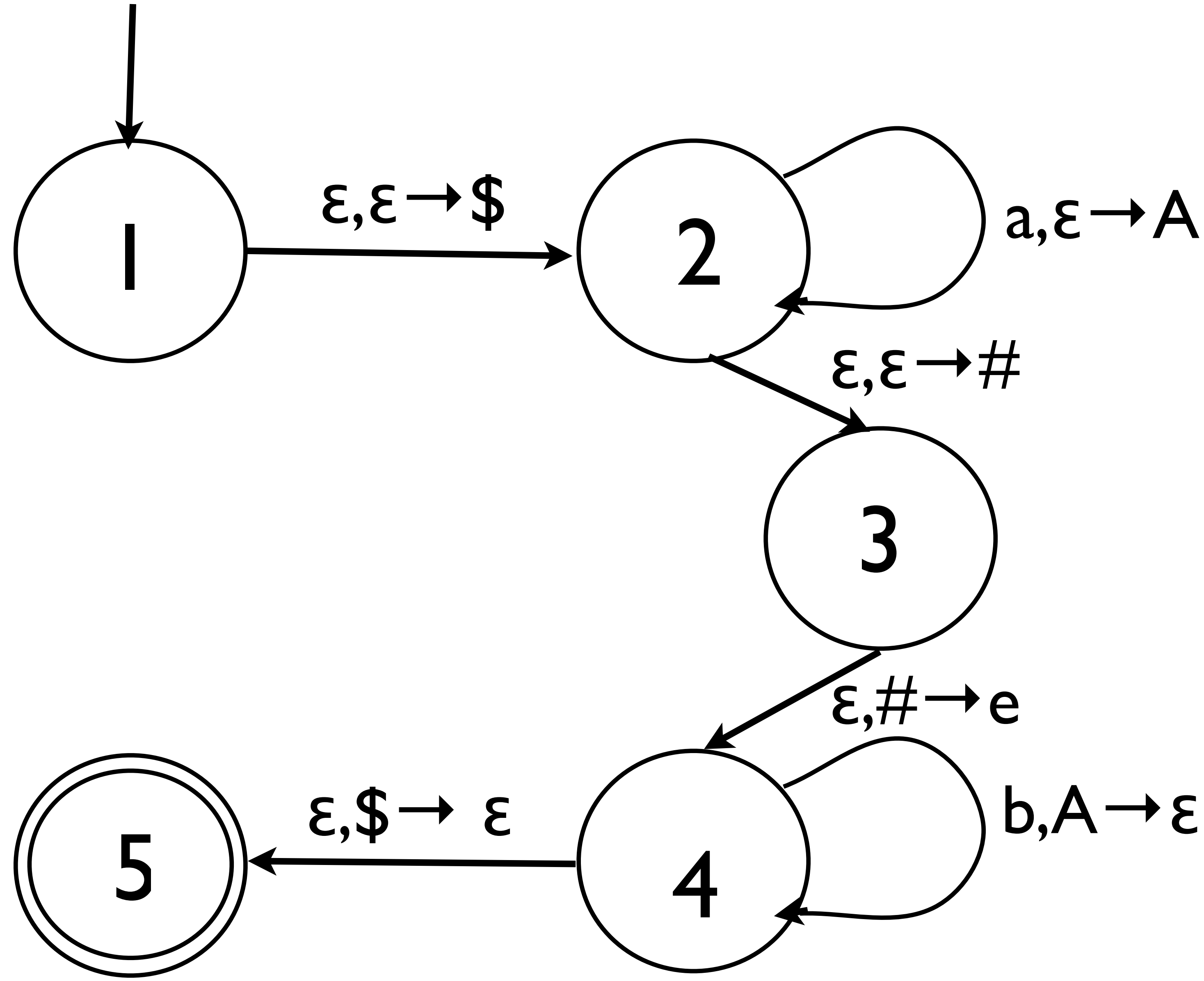
from rule 2:

$A_{15} \to A_{11} A_{15} \mid A_{12} A_{25} \mid A_{13} A_{35} \mid A_{14} A_{45} \mid A_{15} A_{55}$

$A_{24} \to A_{21} A_{14} \mid A_{22} A_{24} \mid A_{23} A_{34} \mid A_{24} A_{44} \mid A_{25} A_{54}$

$A_{33} \to A_{31} A_{13} \mid A_{32} A_{23} \mid A_{33} A_{33} \mid A_{34} A_{43} \mid A_{35} A_{53}$

etc.



from rule 3:

$A_{15} \to \varepsilon A_{24} \varepsilon$ (push/pop $)

$A_{24} \to a A_{24} b$ (push/pop A)

$\mid \varepsilon A_{33} \varepsilon$ (push/pop #)

the non-trivial part:

$A_{15} \to A_{24}$

$A_{24} \to a A_{24} b \mid A_{33}$

$A_{33} \to \varepsilon$

What do we know about CF Languages?

What do we know about CF Languages?

- Any CF language can be recognized by a PDA
- The language recognized by a PDA is CF
- (Some CF languages are deterministic, but not all)
- The union of two CF languages is CF
- The product of two CF languages is CF
- The Kleene closure (*) of a CF language is CF
- **Not all languages are CF**

Normal Forms

- When proving stuff using a grammar, the work is often simpler if the grammar is in a particular form
- Chomsky Normal Form is an example
  - There are others, e.g. Greibach Normal Form
- Key idea: the Normal Forms do *not* restrict the power of the grammar

Chomsky Normal Form (CNF)

- CNF is a restricted form of grammar in which all rules are in one of the following forms:
  - $A \rightarrow a$ ($a \in \Sigma$)
  - $A \rightarrow BC$ ($B,C \in V$ and are not the start symbol)
  - $S \rightarrow \varepsilon$ (allowed *only* if S is the start symbol)
- *Any* CFG can be rewritten to CNF

An application of CNF

- What is the shape of a CNF parse tree?
- Lemma: If G is a grammar in CNF, then for any string $w \in L(G)$ of length $n \geq 1$, any derivation of w requires exactly $2n$-1 steps. Proof: Homework!
- Theorem: For any grammar G and string w, we can determine in finite time whether or not $w \in L(G)$.
  - Proof: try all possible derivations of up to $2n$-1 steps!

Strategy: transforming to CNF

- Add new start symbol S_0 and rule $S_0 \to S$
  - only strictly necessary if S appears on a RHS
- Remove all rules of the form $A \to \varepsilon$
  - unless A is the start symbol
- Remove all **unit** rules of the form $A \to B$
- Arrange that RHS's of length ≥ 2 contain only variables
- Arrange that all RHS's have length ≤ 2
- We're done!

Remove ε-rules

- While there is a rule of the form $A \rightarrow \varepsilon$:
  - Remove the rule
  - Wherever an A appears in the RHS of a rule, add an instance of that rule with the A omitted
    - Ex: Given the rule $B \rightarrow uAv$, add the rule $B \rightarrow uv$
    - Ex. Given the rule $B \rightarrow uAvAw$, add the rules $B \rightarrow uvAw$, $B \rightarrow uAvw$, and $B \rightarrow uvw$
    - Ex. Given the rule $B \rightarrow A$, add the rule $B \rightarrow \varepsilon$ **unless** we have already removed that rule earlier

Remove unit-rules

- While there is a rule of the form $A \rightarrow B$:
  - Remove it
  - For every rule of the form $B \rightarrow u$, add a rule $A \rightarrow u$, unless this is a unit rule we previously removed

Require variables on RHS

- For each terminal $a \in \Sigma$ that appears on the right-hand side of some rule of the form $V \rightarrow w$ where $|w| \geq 2$:
  - Add a new variable A
  - Add a rule $A \rightarrow a$
  - Substitute A for all occurrences of *a* in rules of the above form

Portland State UNIVERSITY

Divide-up RHS

- For each rule of the form $A \rightarrow q_1q_2\ldots q_n$, where $n \geq 3$:
  - Remove the rule
  - Add variables $A_1,A_2,\ldots,A_{n-2}$
  - Add rules $A \rightarrow q_1A_1$, $A_1 \rightarrow q_2A_2$, ..., $A_{n-2} \rightarrow q_{n-1}q_n$

Portland State UNIVERSITY

Example: converting to CNF

- Initial grammar:

  $S \to aSb \mid T \qquad T \to cT \mid \varepsilon$

- After start variable introduction:

  $S_0 \to S \qquad S \to aSb \mid T \qquad T \to cT \mid \varepsilon$

- After ε-rule elimination:

  $S_0 \to S \mid \varepsilon \qquad S \to aSb \mid ab \mid T$

  $T \to cT \mid c$

- After unit-rule elimination:

  $S_0 \to aSb \mid ab \mid cT \mid c \mid \varepsilon$

  $S \to aSb \mid ab \mid cT \mid c \qquad T \to cT \mid c$

Example (continued)

- After variable introduction

  $S_0 \rightarrow ASB \mid AB \mid CT \mid c \mid \varepsilon$

  $S \rightarrow ASB \mid AB \mid CT \mid c$

  $T \rightarrow CT \mid c \qquad A \rightarrow a \qquad B \rightarrow b \quad C \rightarrow c$

- After RHS splitting

  $S_0 \rightarrow AD \mid AB \mid CT \mid c \mid \varepsilon$

  $S \rightarrow AD \mid AB \mid CT \mid c \qquad D \rightarrow SB$

  $T \rightarrow CT \mid c \qquad A \rightarrow a \qquad B \rightarrow b \quad C \rightarrow c$

The Pumping Lemma
for Context-free Languages

Portland State UNIVERSITY

The Pumping Lemma
for Context-free Languages

- If a CF language has arbitrarily long strings, any grammar for it must contain a recursive chain of productions

The Pumping Lemma
for Context-free Languages

- If a CF language has arbitrarily long strings, any grammar for it must contain a recursive chain of productions
- In the simplest case, it might contain a directly recursive production
  - e.g., $\quad S \rightarrow u R y$
    $R \rightarrow v R x \mid w$
    where either v or x must be non-empty

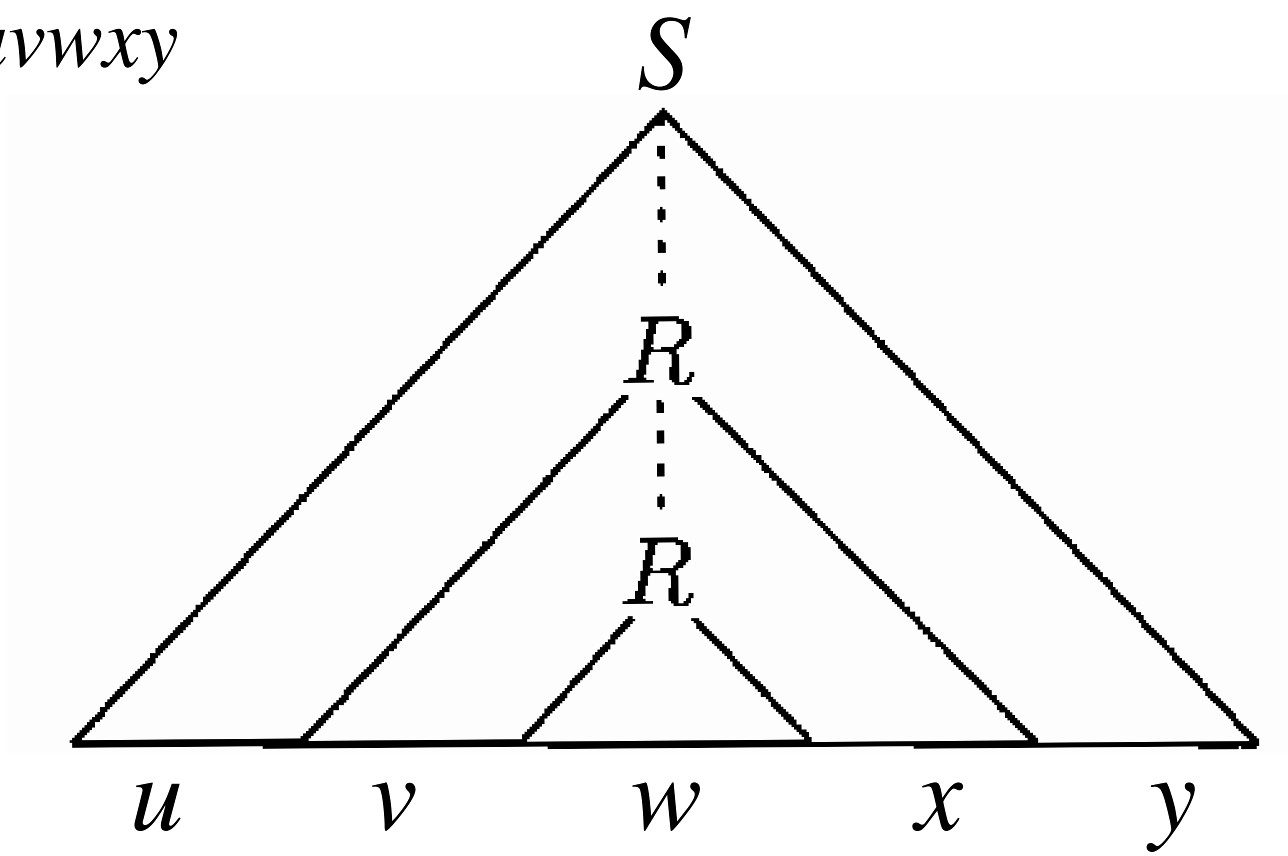
The Pumping Lemma
for Context-free Languages

- If a CF language has arbitrarily long strings, any grammar for it must contain a recursive chain of productions
- In the simplest case, it might contain a directly recursive production
  - e.g., $\text{S} \rightarrow u\text{R}y$
    $\text{R} \rightarrow v\text{R}x \mid w$

    where either v or x must be non-empty
- then we can derive:
  - $\text{S} \Rightarrow u\text{R}y \Rightarrow uwy$
  - $\text{S} \Rightarrow u\text{R}y \Rightarrow uv\text{R}xy \Rightarrow uvwxy$
  - $\text{S} \Rightarrow u\text{R}y \Rightarrow uv\text{R}xy \Rightarrow uvv\text{R}xxy \Rightarrow uvvwxxy$
  - $\text{S} \Rightarrow u\text{R}y \Rightarrow uv\text{R}xy \Rightarrow uvv\text{R}xxy \Rightarrow uvvv\text{R}xxxy \Rightarrow uvvvwxxxy$

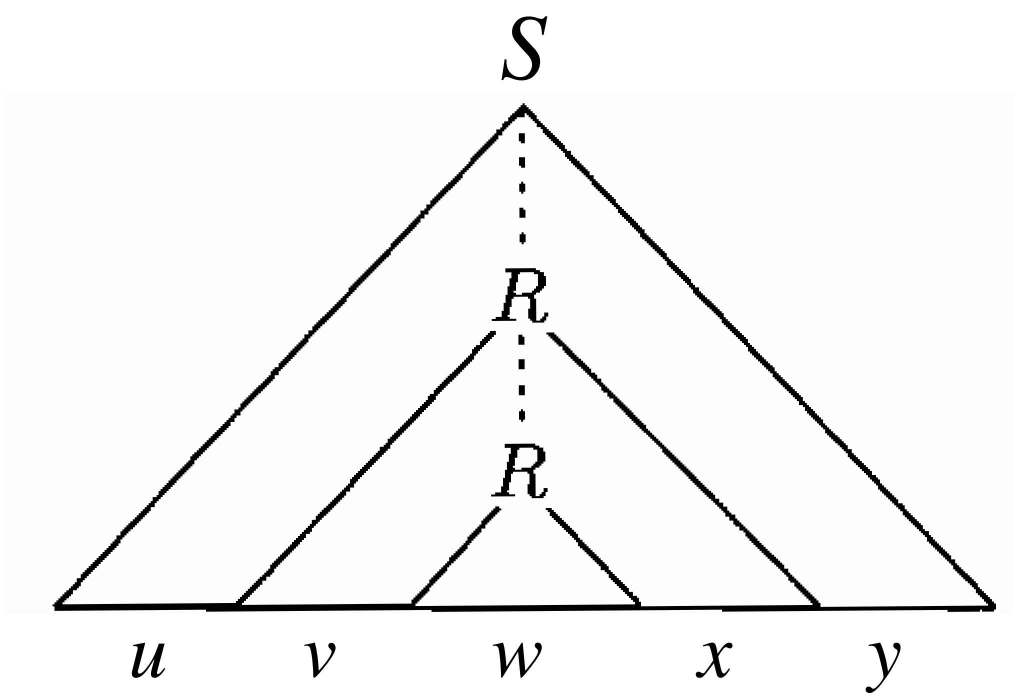
More generally:

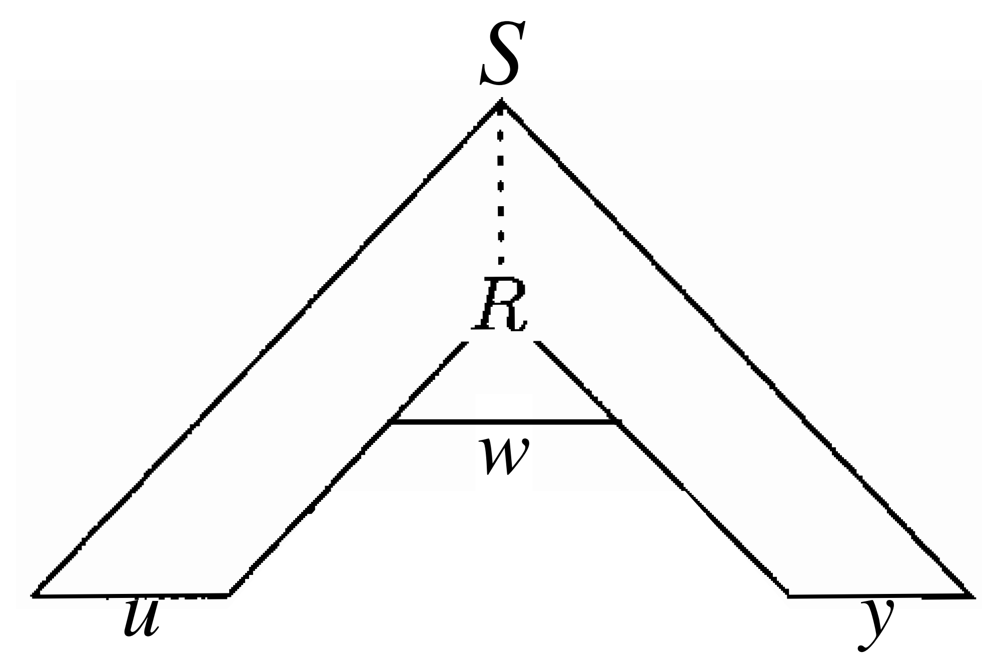
$S \Rightarrow^* uvwxy$

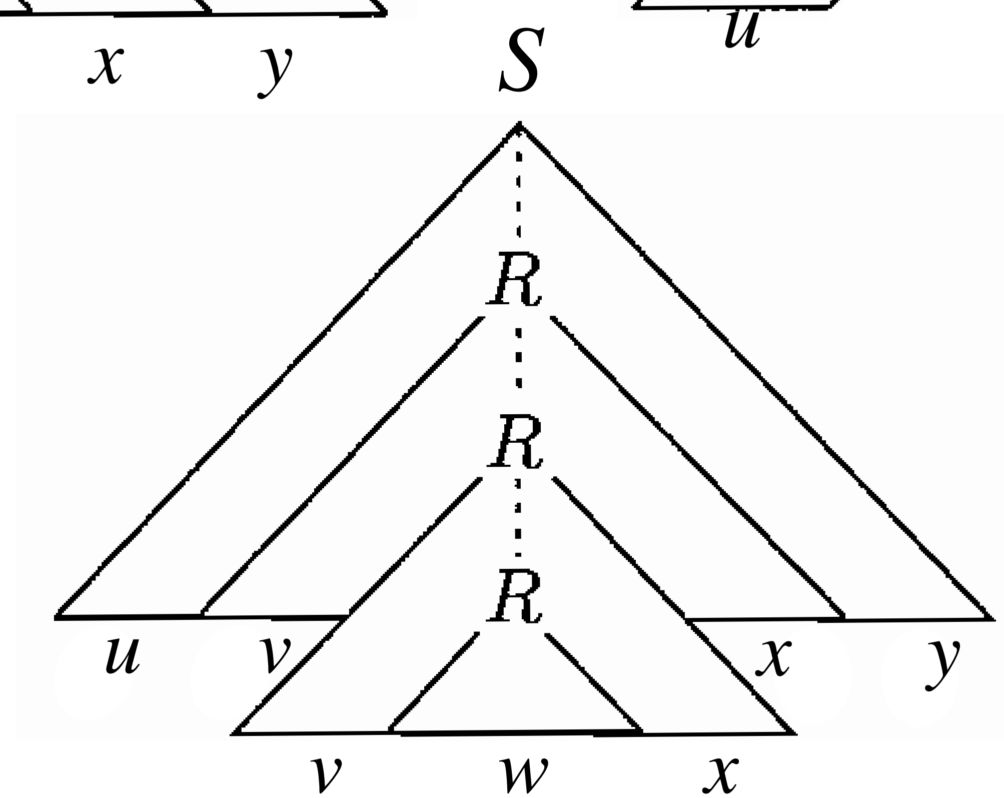


- If s is long enough, then some R must appear twice on the path from S to some terminal in s. Why?
- So we can write $s = uvwxy$ where...

More generally:







Theorem Statement

If L is a context-free language, then there is a number p (called the pumping length) such that for all strings $z \in L$, $|z| \geq p$, z can be divided into 5 pieces $z = uvwxy$ satisfying:

1. for each $i \geq 0$, $uv^iwx^iy \in L$
2. $|vx| > 0$
3. $|vwx| \leq p$

Proof

- Assume CFG for L is in Chomsky NF with variable set V.
- Take $p = 2^{|V|+1}$. Then if $|z| \geq p$, any parse tree for z has height at least $|V|+1$. **Why?**
- Choose a parse tree with fewest nodes.
- The longest path from root to a terminal must have at least $|V|+1$ variables. So some variable must appear at least twice among the bottom $|V|+1$ nodes. **Why?**
- Consider any such variable R and divide z into $uvwxy$ as in diagram. Can see that uv^iwx^iy is also in L for all $i \geq 0$.
- $|vwx| \leq p$ because path from R has height at most $|V|+1$.
- $|vx| > 0$; otherwise we could have a tree with fewer nodes

Portland State UNIVERSITY

Using the Pumping Lemma

- Prove that L = $\{a^n b^n c^n \mid n \geq 0\}$ is not CF
- Assume that L is CF and derive a contradiction:
  - pick $z = a^p b^p c^p$ where p is the pumping length
  - $|z| \geq p$, so we can write $z=uvwxy$ where $|vx| > 0$, $|vwx| \leq p$, and uv^iwx^iy is in L. In particular, take $i = 2$.
  - if v contains two letters, say a and b, then any string containing v^2 can't be in L. Same for x. **Why?**
  - so v and x must have the form a^j,or b^j,or c^j, or ε
    - but at most one of them can be ε. **Why?**
  - so at least one of the symbols a, b, or c does not appear in vx, but at least one does
  - so uv^2wx^2y can't have the same number of a's b's and c's

Pumping Lemma Example 2

Pumping Lemma Example 2

- Prove that $C = \{a^i b^j c^k \mid 0 \le i \le j \le k\}$ is not CF

Pumping Lemma Example 2

- Prove that C = $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ is not CF
  - Suppose C is CF.

Pumping Lemma Example 2

- Prove that $C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ is not CF
  - Suppose C is CF.
  - Let the pumping length be p, and again consider the string $z = a^p b^p c^p$ in C.

Pumping Lemma Example 2

- Prove that C = $\{a^i b^j c^k \mid 0 \le i \le j \le k\}$ is not CF
  - Suppose C is CF.
  - Let the pumping length be p, and again consider the string z = $a^p b^p c^p$ in C.
  - Then the pumping lemma says that we can divide $z = uvwxy$ where $|vx| > 0$, $|vwx| \le p$, and $uv^i wx^i y$ is in C for all i

Portland State UNIVERSITY

Pumping Lemma Example 2

- Prove that C = $\{a^i b^j c^k \mid 0 \le i \le j \le k\}$ is not CF
  - Suppose C is CF.
  - Let the pumping length be p, and again consider the string $z = a^p b^p c^p$ in C.
  - Then the pumping lemma says that we can divide $z = uvwxy$ where $|vx| > 0$, $|vwx| \le p$, and uv^iwx^iy is in C for all i
  - As in the previous example, at least one of the symbols a, b, or c does not appear in vx but at least one does.

Portland State UNIVERSITY

- Now there are three cases to consider:

- Now there are three cases to consider:

  1. The a's don't appear. Now consider uv^0wx^0y and compare with $uvwxy$. The string uv^0wx^0y still has p a's, but fewer b's and/or c's. Hence, $uv^0wx^0y \notin$ C

Portland State UNIVERSITY

- Now there are three cases to consider:
    1. The a's don't appear. Now consider uv^0wx^0y and compare with $uvwxy$. The string uv^0wx^0y still has p a's, but fewer b's and/or c's. Hence, $uv^0wx^0y \notin$ C
    2. The b's don't appear. So either a's or c's must appear in v or x. If a's appear, then uv^2wx^2y contains more a's than b's. If c's appear, then uv^0wx^0y contains fewer c's than b's. Either way, the pumped string $\notin$ C.

- Now there are three cases to consider:

  1. The a's don't appear. Now consider uv^0wx^0y and compare with $uvwxy$. The string uv^0wx^0y still has p a's, but fewer b's and/or c's. Hence, $uv^0wx^0y \notin$ C

  2. The b's don't appear. So either a's or c's must appear in v or x. If a's appear, then uv^2wx^2y contains more a's than b's. If c's appear, then uv^0wx^0y contains fewer c's than b's. Either way, the pumped string $\notin$ C.

  3. The c's don't appear. In this case, uv^2wx^2y contains more a's and/or bs than c's, and so the string $uv^2wx^2y \notin$ C

- Now there are three cases to consider:
  1. The a's don't appear. Now consider uv^0wx^0y and compare with $uvwxy$. The string uv^0wx^0y still has p a's, but fewer b's and/or c's. Hence, $uv^0wx^0y \notin$ C
  2. The b's don't appear. So either a's or c's must appear in v or x. If a's appear, then uv^2wx^2y contains more a's than b's. If c's appear, then uv^0wx^0y contains fewer c's than b's. Either way, the pumped string $\notin$ C.
  3. The c's don't appear. In this case, uv^2wx^2y contains more a's and/or bs than c's, and so the string $uv^2wx^2y \notin$ C
- Thus, z can't be pumped, and we have a contradiction. So C is not CF.

Pumping Lemma Example 3

- Prove that D = $\{ ww \mid w \in \{0, 1\}^* \}$ is not CF
  - Suppose D is CF with pumping length p.
  - Consider the string $z = 0^p1^p0^p1^p$ in D. Certainly $|z| \geq p$.
  - Then the pumping lemma says that we can divide $z = uvwxy$ where $|vx| > 0$, $|vwx| \leq p$, and uv^iwx^iy is in D for all i
  - Consider the following three mutually exclusive cases:
    - vwx falls in the first half of z. But then if we "pump up" to uv^2wx^2y, we'll move a 1 into the first position of the second half. The resulting string can't be in D.
    - vwx falls in the second half of z... a similar argument holds
    - vwx straddles the midpoint of z. But then if we "pump down" to uwy, we get a string of the form $0^p1^i0^j1^p$, where i and j cannot both be p. Resulting string can't be in D.