Context-free Languages

Bernhard Nebel and Christian Becker-Asano

Overview

- Context free grammars
- Pushdown Automata
- Equivalence of PDAs and CFGs
- Non-context free grammars
  - Pumping lemma

Context free languages

- Extend regular languages
- First studied for natural languages
- Often used in computer languages
  - Compilers
  - Parsers
- Pushdown automata

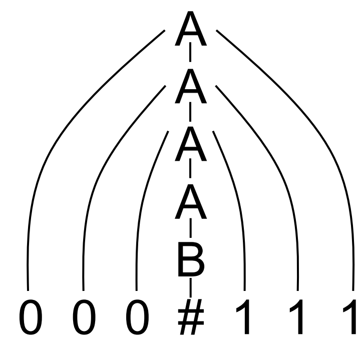
Key concept: context-free grammar

Example grammar G_1:

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$



Parse tree for 000#111 in G_1

- Terminals: 0, 1, # (correspond to alphabet Σ)
- Nonterminals / variables: A, B
- Rules: $Symbol \rightarrow String$
- Startsymbol

The sequence of substitutions to obtain a string is called a **derivation**.

E.g. derivation of 000#111: A → 0A1 → 00A11 → 000A111 → 000#111

- **Language defined by G_1:** $L(G_1) = \{0^n\#1^n | \, n \geq 0\}$

Natural language example:

⟨SENTENCE⟩	→	⟨NOUN-PHRASE⟩⟨VERB-PHRASE⟩
⟨NOUN-PHRASE⟩	→	⟨CMPLX-NOUN⟩|⟨CMPLX-NOUN⟩⟨PREP-PHRASE⟩
⟨VERB-PHRASE⟩	→	⟨CMPLX-VERB⟩|⟨CMPLX-VERB⟩⟨PREP-PHRASE⟩
⟨PREP-PHRASE⟩	→	⟨PREP⟩⟨CMPLX-NOUN⟩
⟨CMPLX-NOUN⟩	→	⟨ARTICLE⟩⟨NOUN⟩
⟨CMPLX-VERB⟩	→	⟨VERB⟩|⟨VERB⟩⟨NOUN-PHRASE⟩
⟨ARTICLE⟩	→	a | the
⟨NOUN⟩	→	boy | girl | flower
⟨VERB⟩	→	touches | likes | sees
⟨PREP⟩	→	with

Example sentences:

1. a boy sees
2. the boy sees the flower
3. a girl with a flower likes the boy

Context-free grammar

DEFINITION 2.2:

A context-free grammar is a 4-tuple (V, Σ, R, S) with:

1. V a finite set called the **variables**
2. Σ a finite set, disjoint from V, called the **terminals**
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals
4. $S \in V$ is the **start symbol**

Example: $G_3 = (\{S\}, \{a, b\}, R, S)$

$S \rightarrow aSb \mid SS \mid \varepsilon$

Parsing

$G_3 = (V, \Sigma, R, <Expr>)$

$V = \{<Expr>, <Term>, <Factor>\}$

$\Sigma = \{a, +, \times, (,)\}$

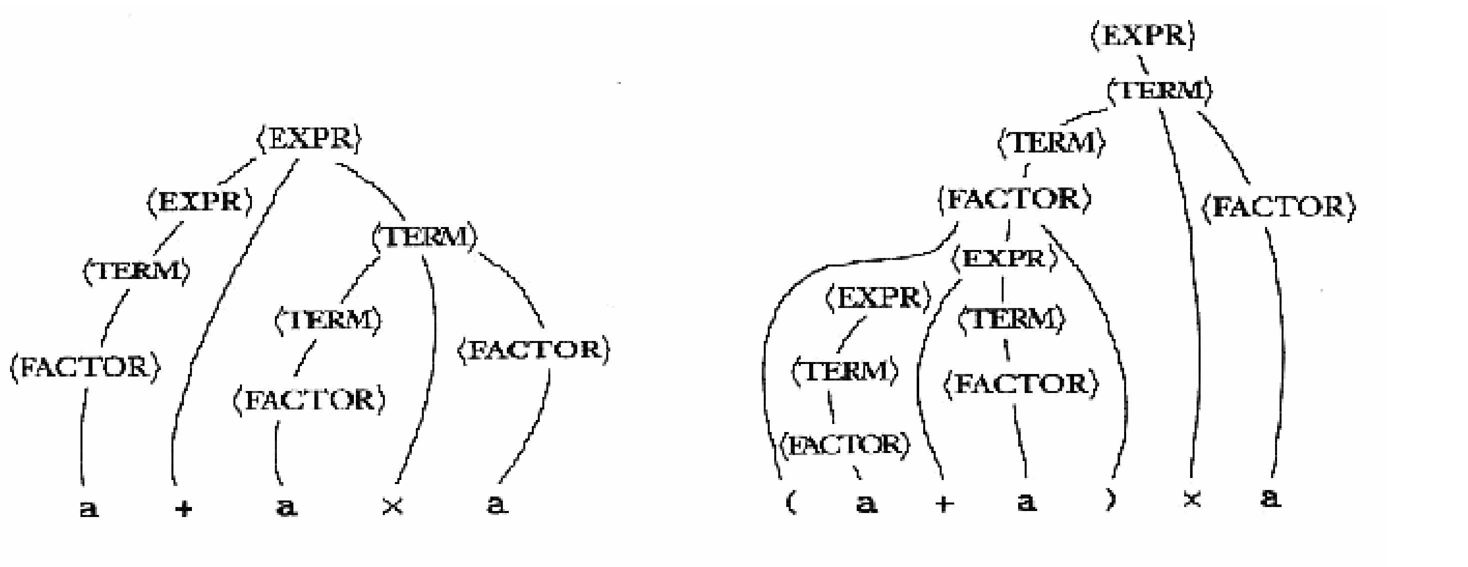
R is

$<Expr> \rightarrow <Expr> + <Term> | <Term>$

$<Term> \rightarrow <Term> \times <Factor> | <Factor>$

$<Factor> \rightarrow (<Expr>) | a$

- ✶ Construct meaning (parse tree)



- ✶ Parse trees for the strings **a + a x a** and **(a + a) x a**

Constructing CFGs

- As the union of simpler CFGs

$$S_1 \to 0S_1 1 \mid \varepsilon \qquad L(G_1) = \{0^n 1^n \mid n \geq 0\}$$

$$S_2 \to 1S_2 0 \mid \varepsilon \qquad L(G_2) = \{1^n 0^n \mid n \geq 0\}$$

$$S \to S_1 \mid S_2 \qquad L(G) = L(G_1) \cup L(G_2)$$

Constructing CFGs

- When given a DFA (i.e. constructing a CFG for reg. languages)

For each state q_i

Make a variable R_i

For each transition $\delta(q_i, a) = q_j$

Add the rule $R_i \rightarrow aR_j$

For each accept state q_i

Add the rule $R_i \rightarrow \varepsilon$

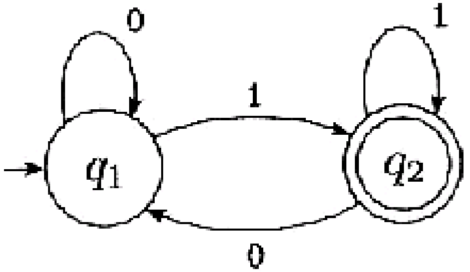


FIGURE 1.6
State diagram of the two-state finite automaton M_2

Constructing CFGs

- Languages consisting of “linked” strings

$$L(G_1) = \{0^n 1^n \mid n \geq 0\}$$

Use rules of the form

$$R \rightarrow uRv$$

$$S_1 \rightarrow 0S_1 1 \mid \varepsilon$$

Constructing CFGs

- Strings that may contain structures that appear recursively as part of other (or the same) structures

$$< Expr > \rightarrow < Expr > + < Term > | < Term >$$

$$< Term > \rightarrow < Term > \times < Factor > | < Factor >$$

$$< Factor > \rightarrow (< Expr >) \mid a$$

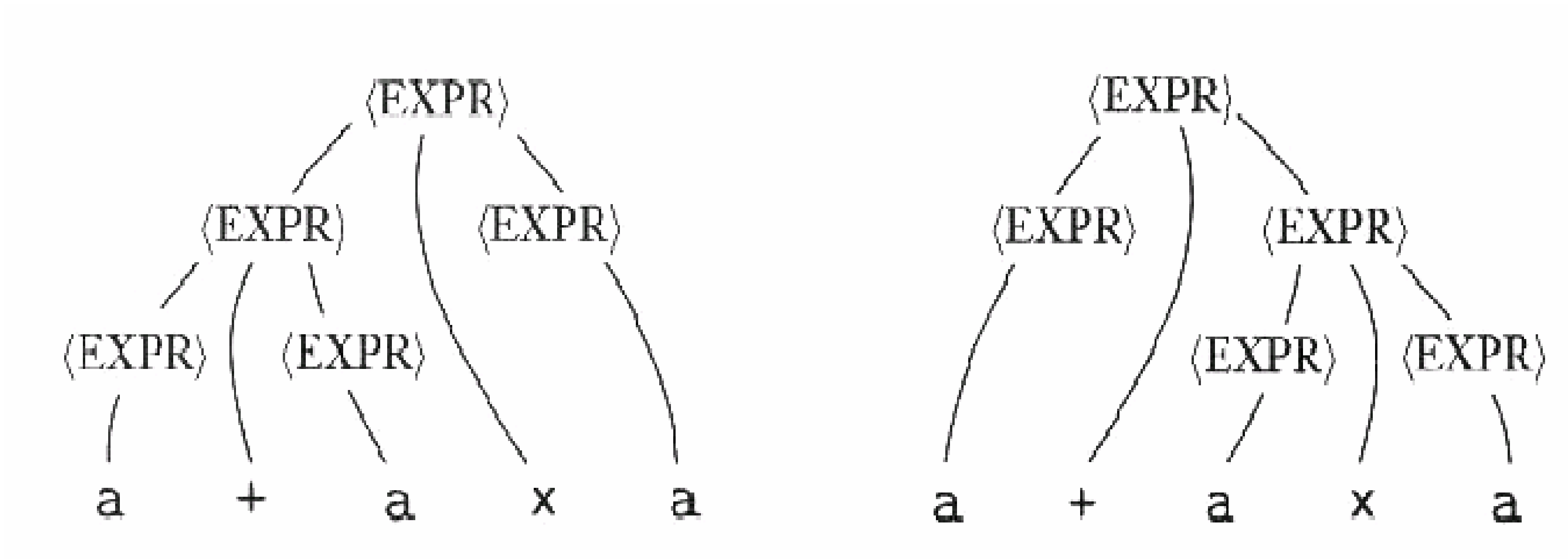
Ambiguity

- If a CFG generates the same string in several ways, then the grammar is *ambiguous*
- E.g. grammar G_5:

$$< Expr > \rightarrow < Expr > + < Expr > | < Expr > \times < Expr > | (< Expr >) | a$$

- The grammar does not capture usual precedence relations
- One of the main problems in natural language processing
- “the boy touches the girl with the flower”

$$< Expr > \rightarrow < Expr > + < Expr > | < Expr > \times < Expr > | (< Expr >) | a$$



The two parse trees for the string **a + a x a** in grammar G_5

Defining ambiguity

- *Leftmost* derivation :
  - At every step in the derivation the leftmost variable is replaced
- A string is derived *ambiguously* in a CFG if it has two or more different *leftmost* derivations
- A grammar is *ambiguous* if it generates *some* string ambiguously
- Some context free languages are *inherently* ambiguous, i.e. every grammar for the language is ambiguous

$$\{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$$

Chomsky Normal Form (CNF)

DEFINITION 2.8:

A context-free grammar is in **Chomsky normal form** if every rule is of the form

$$A \rightarrow BC$$
$$A \rightarrow a$$

where a is any terminal and $A, B,$ and C are any variables—except that B and C may not be the start variable. In addition we permit the rule $S \rightarrow \varepsilon$, where S is the start variable.

Theorem 2.9:

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Chomsky normal form: proof idea

- Rewrite all rules, which are not conform with the Chomsky normal form
- If necessary, introduce new variables

Four problems:

1. Start variable is on the right side of a rule
   → *Introduce a new start variable and a new rule for the derivation*
2. Epsilon-rules, like $A \rightarrow \varepsilon$
   → *If A occurs on the right part of a rule, introduce new rules without A on the right part of the rule*
3. Unit-rules, like $A \rightarrow B$
   → *directly replace B by its own production*
4. Long and/or mixed rules, like $A \rightarrow aBcAbA$
   → *new variables/new rules*

CNF: proof by construction

1. Add a new start symbol S_0 and the rule $S_0 \rightarrow S$, where S is the old start symbol.
2. Remove all rules $A \rightarrow \varepsilon$:
   For each occurrence of A in a rule $R \rightarrow uAv$ add $R \rightarrow uv$ (if u and v are ε, then add $R \rightarrow \varepsilon$). Repeat this step until all such rules (except a rule referring to the start variable) are removed.
3. Remove all unit rules $A \rightarrow B$: Whenever $B \rightarrow u$ appears, then add $A \rightarrow u$. Repeat this step until all unit rules are removed.

4a. Convert remaining rules $A \rightarrow u_1 u_2 \dots u_k$, where $k \geq 3$, into rules
$A \rightarrow u_1 A_1$,
$A_1 \rightarrow u_2 A_2$, …,
$A_{k-2} \rightarrow u_{k-1} u_k$, where the A_i are new variables.

4b. If $k = 2$, then replace any terminal u_i in the rules with a new variable U_i and the new rule $U_i \rightarrow u_i$.

Do not allow for cycles (i.e. first remove, then add rule)!

CNF: example 2.10

Let G_6 be the following CFG and convert it into CNF by using the conversion procedure just given. The following series of grammars illustrates the steps in the conversion. Rules set in **bold** have just been **added**. Rules or symbols struck ~~through~~ have just been ~~removed~~.

1. The original CFG G_6 is shown below on the left. The result of applying the first step to make a new start symbol appears on the right.

$$S \rightarrow ASA \mid aB$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b \mid \varepsilon$$

$$\boldsymbol{S_0 \rightarrow S}$$
$$S \rightarrow ASA \mid aB$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b \mid \varepsilon$$

CNF: example 2.10 (ctd.)

2. Remove ε rule $B \to \varepsilon$, shown on the left, and then also $A \to \varepsilon$, shown on the right.

$$S_0 \to S \qquad\qquad S_0 \to S$$
$$S \to ASA \mid aB \mid \boldsymbol{a} \qquad\qquad S \to ASA \mid aB \mid a \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S}$$
$$A \to B \mid S \mid \boldsymbol{\varepsilon} \qquad\qquad A \to B \mid S \mid \cancel{\varepsilon}$$
$$B \to b \mid \cancel{\varepsilon} \qquad\qquad B \to b$$

3. (a) Remove unit rules $S \to S$, shown left, and $S_0 \to S$, shown right.

$$S_0 \to S \qquad\qquad S_0 \to \cancel{S} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS}$$
$$S \to ASA \mid aB \mid a \mid SA \mid AS \mid \cancel{S} \qquad\qquad S \to ASA \mid aB \mid a \mid SA \mid AS$$
$$A \to B \mid S \qquad\qquad A \to B \mid S$$
$$B \to b \qquad\qquad B \to b$$

CNF: example 2.10 (ctd.)

3. (b) Remove unit rules $\mathrm{A} \rightarrow B$ and $A \rightarrow S$.

$$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$A \rightarrow \cancel{B} \mid S \mid \boldsymbol{b}$$
$$B \rightarrow b$$

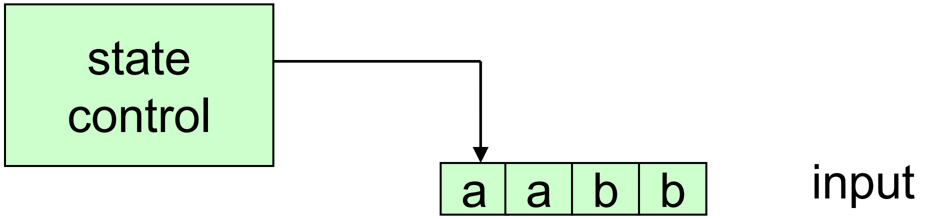
$$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$A \rightarrow \cancel{S} \mid b \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS}$$
$$B \rightarrow b$$

4. Convert the remaining rules.

$$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$$
$$S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$$
$$A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$$
$$A_1 \rightarrow SA$$
$$U \rightarrow a$$
$$B \rightarrow b$$

Pushdown automata: introduction

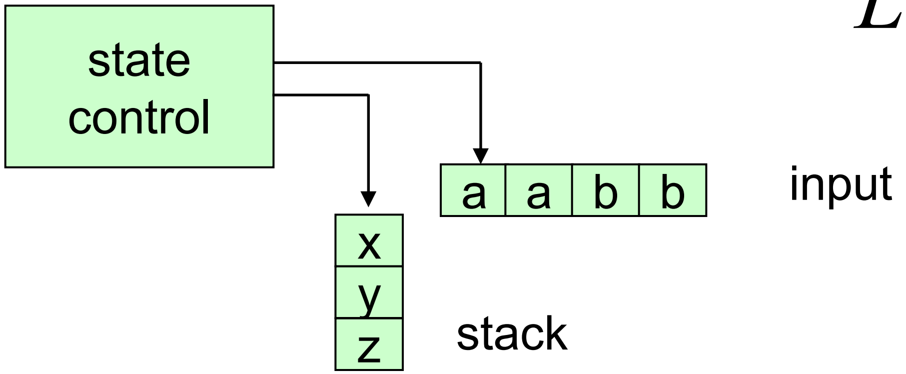
- Schema of a finite automaton



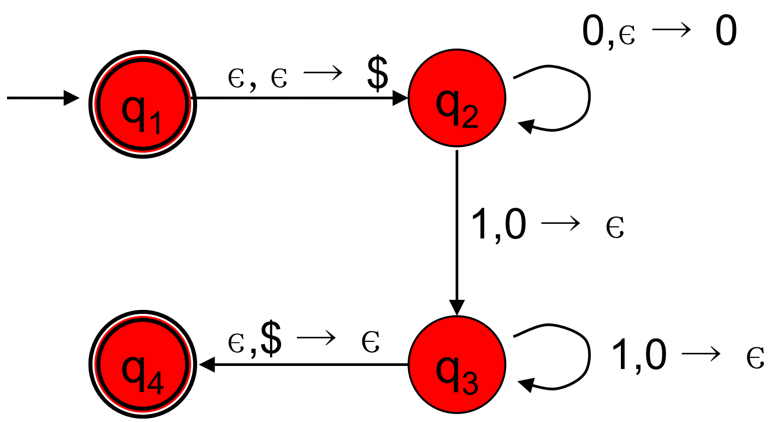
Pushdown automaton

- Includes a stack
  - Push something on top of stack
  - Pop something from top of stack
  - Last in first out principle
  - As in cafeteria – tray
  - Schematic of a pushdown automaton:

$$L(G_1) = \{0^n 1^n \mid n \geq 0\}$$



An example PDA



State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

Formal definition (Definition 2.13)

A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

1. Q is a finite set of states
2. Σ is a finite set, the input alphabet
3. Γ is a finite set, the stack alphabet
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ is the transition function
5. $q_o \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states

Transition function

maps (*state, inputsymbol, stacksymbol*) onto set of (*nstate, nstacksymbol*)

Meaning:

stacksymbol is replaced by *nstacksymbol*
input, stack, and *nstacksymbol* can be ε !

Example 2.14 (PDA M_1)

The following is the formal description of a PDA that recognizes the language $\{0^n 1^n \mid n \geq 0\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0, 1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$.

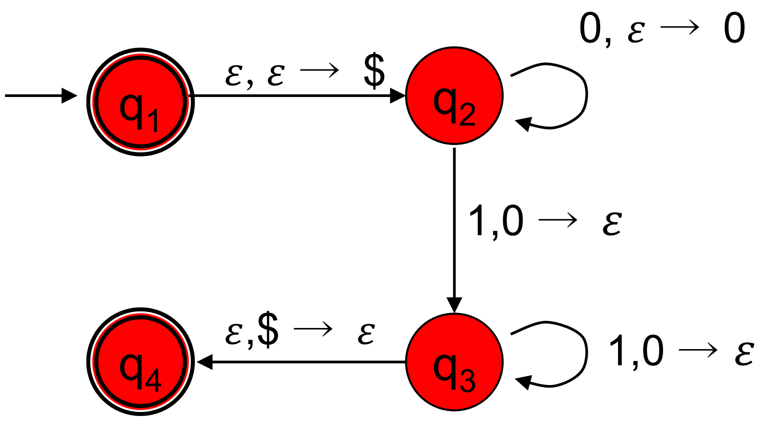
Input	0			1			ϵ		
Stack	0	$	ϵ	0	$	ϵ	0	$	ϵ
q_1									$\{(q_2,\$)\}$
q_2			$\{(q_2,0)\}$	$\{(q_3,\epsilon)\}$					
q_3				$\{(q_3,\epsilon)\}$				$\{(q_4,\epsilon)\}$	
q_4									

Computation with PDA M_1

To compute, one can keep track of

1. rest of the input string (to read)
2. state of PDA
3. string on stack

Use a tree structure as for NFAs !



$$(0011, q_1, \varepsilon)$$
$$\downarrow$$
$$(0011, q_2, \$)$$
$$\downarrow$$
$$(011, q_2, 0\$)$$
$$\downarrow$$
$$(11, q_2, 00\$)$$
$$\downarrow$$
$$(1, q_3, 0\$)$$
$$\downarrow$$
$$(\varepsilon, q_3, \$)$$
$$\downarrow$$
$$(q_4, \varepsilon)\text{ accept}$$

Formal Definition of Computation

Let M be a pushdown automaton $(Q,\Sigma,\Gamma,\delta,q_0,F)$

Let $w = w_1....w_n$ be a string over Σ

M accepts w if $w \in \Sigma^*$ and $w = w_1....w_n$ where $w_i \in \Sigma_\varepsilon$ and a sequence of states $r_0,...,r_n$ exists in Q and strings $s_0,...,s_n$ exists in Γ^* such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$
2. for all $i = 0,...,n-1$

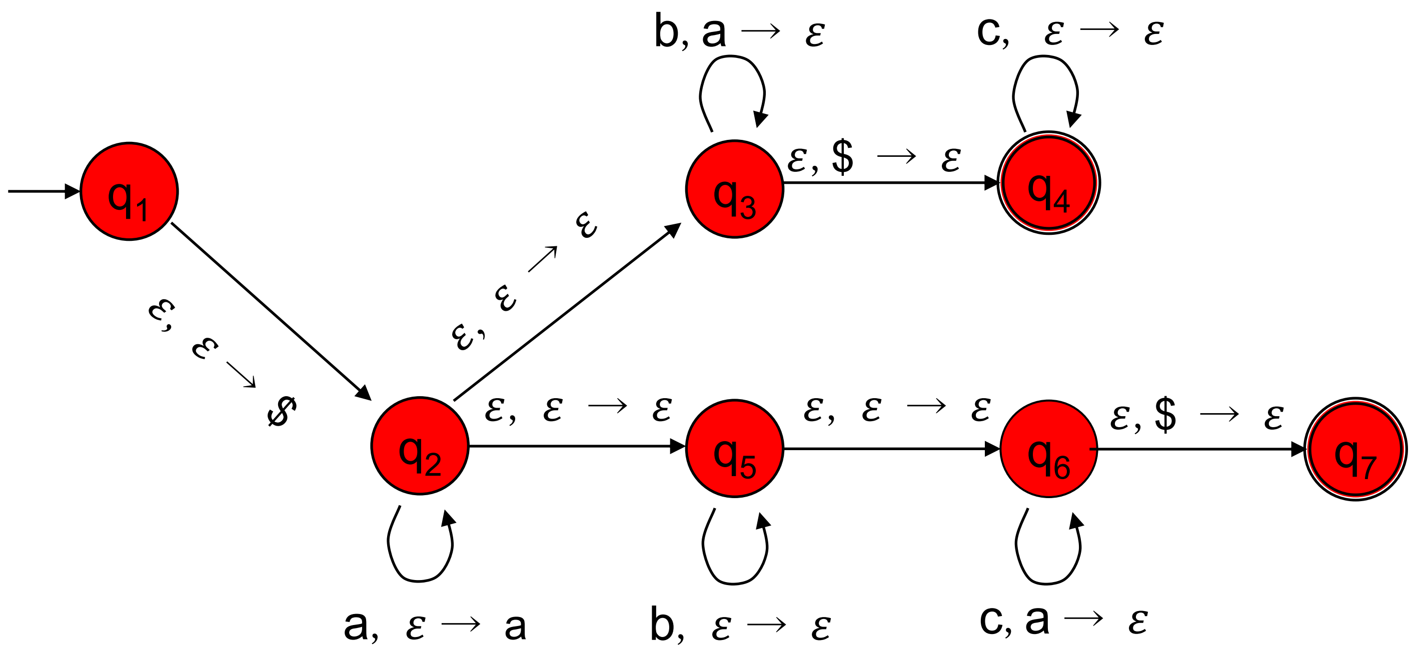
   $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where s_i=at and s_{i+1}=bt

   for some $a,b \in \Gamma_\varepsilon$ and some $t \in \Gamma^*$
3. $r_n \in F$

No explicit test for empty stack and end of input

Another example

PDA M_2 recognizing $\{a^i\, b^j\, c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

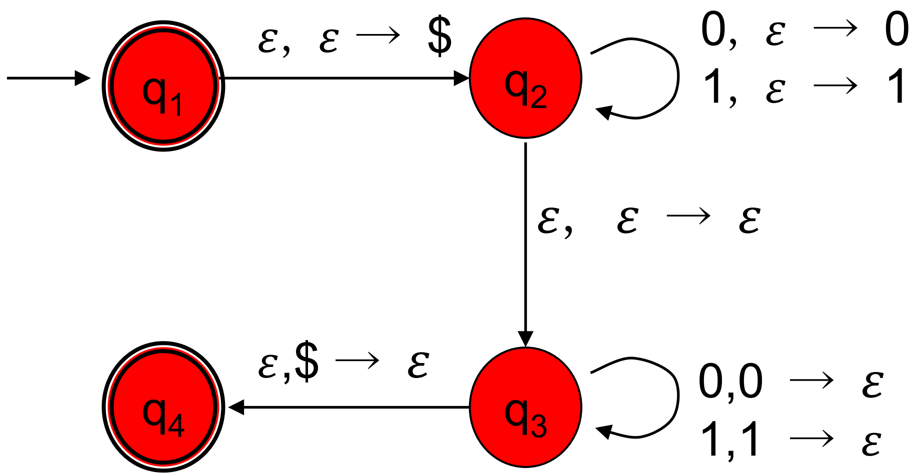


State diagram for PDA M_2 that recognizes the language $\{a^i b^j c^k \mid i.j.k \geq 0 \text{ and } i = j \text{ or } i = k\}$

- *Non determinism* essential for this language!

Another example

PDA M_3 recognizing $\{ww^R | w\ 2\ \{0,1\}^*\}$



Theorem 2.20 and Lemma 2.21

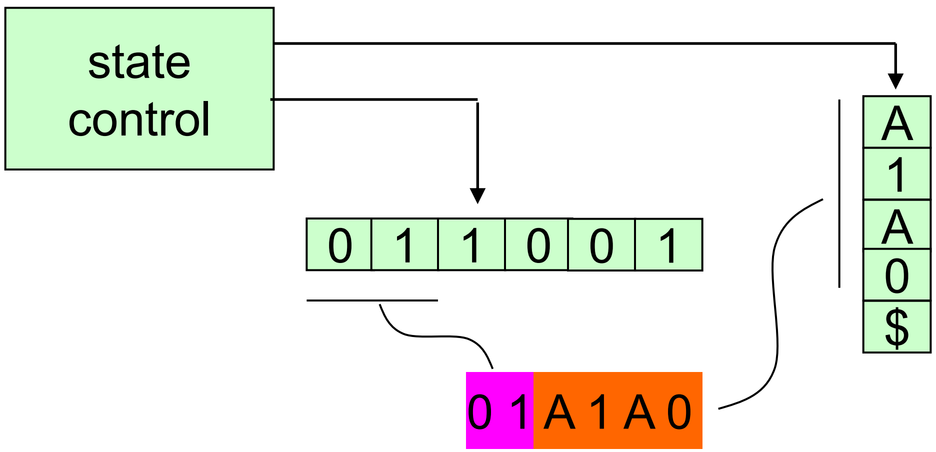
Theorem 2.20:
A language is context free if and only if some pushdown automaton recognizes it.

Lemma 2.21: If a language is context free, then some pushdown automaton recognizes it. (Forward direction of proof)

- A CFL accepts a string if there exists a derivation of the string
- Involves intermediate strings
- Represent intermediate strings on PDA

<SENTENCE> ⇒ <NOUN-PHRASE><VERB-PHRASE>
⇒ <CMPLX-NOUN><VERB-PHRASE>
⇒ <ARTICLE><NOUN><VERB-PHRASE>
⇒ a <NOUN><VERB-PHRASE>
⇒ a boy <VERB-PHRASE>
⇒ a boy <CMPLX-VERB>
⇒ a boy <VERB>
⇒ a boy sees

Lemma 2.21 Proof idea



P presenting the intermediate string 01A1A0

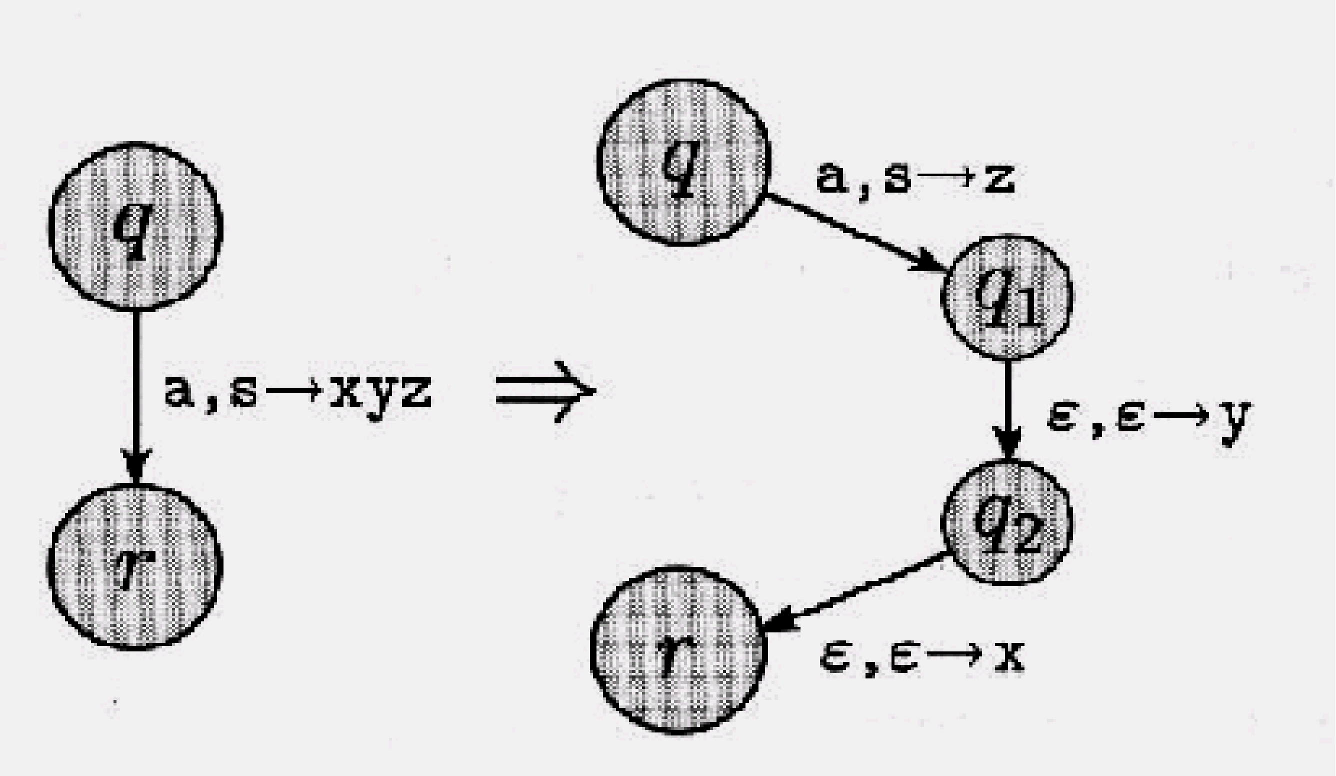
- Substitute variables by strings
- Replace top variable on stack by string

Lemma 2.21 Proof by construction

Construction

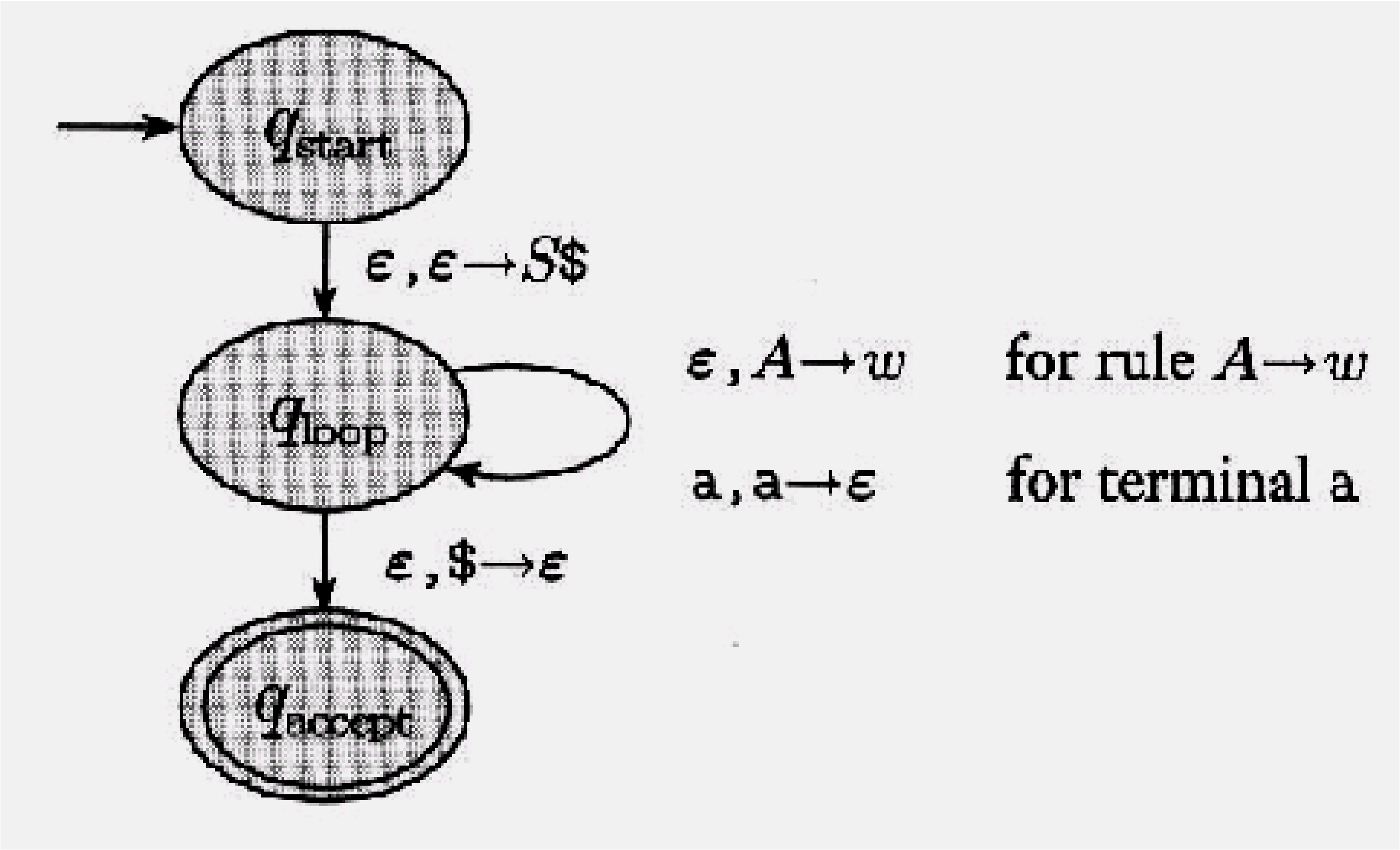
1. Place the marker $ and the start symbol on the stack
2. Repeat forever
   a. if top(stack)=variable *A*
      then non-deterministically select one of the rules for *A*
      and substitute *A* by right hand side of rule
   b. if top(stack)=terminal symbol *a*
      then read next input symbol be *i*
         if *a* <> *i* then fail
   c. if top(stack)=$ and all input read
      then enter accept state

Lemma 2.21: Proof (ctd.)



- A construction to substitute a variable by a *string*

Lemma 2.21: Proof, resulting PDA

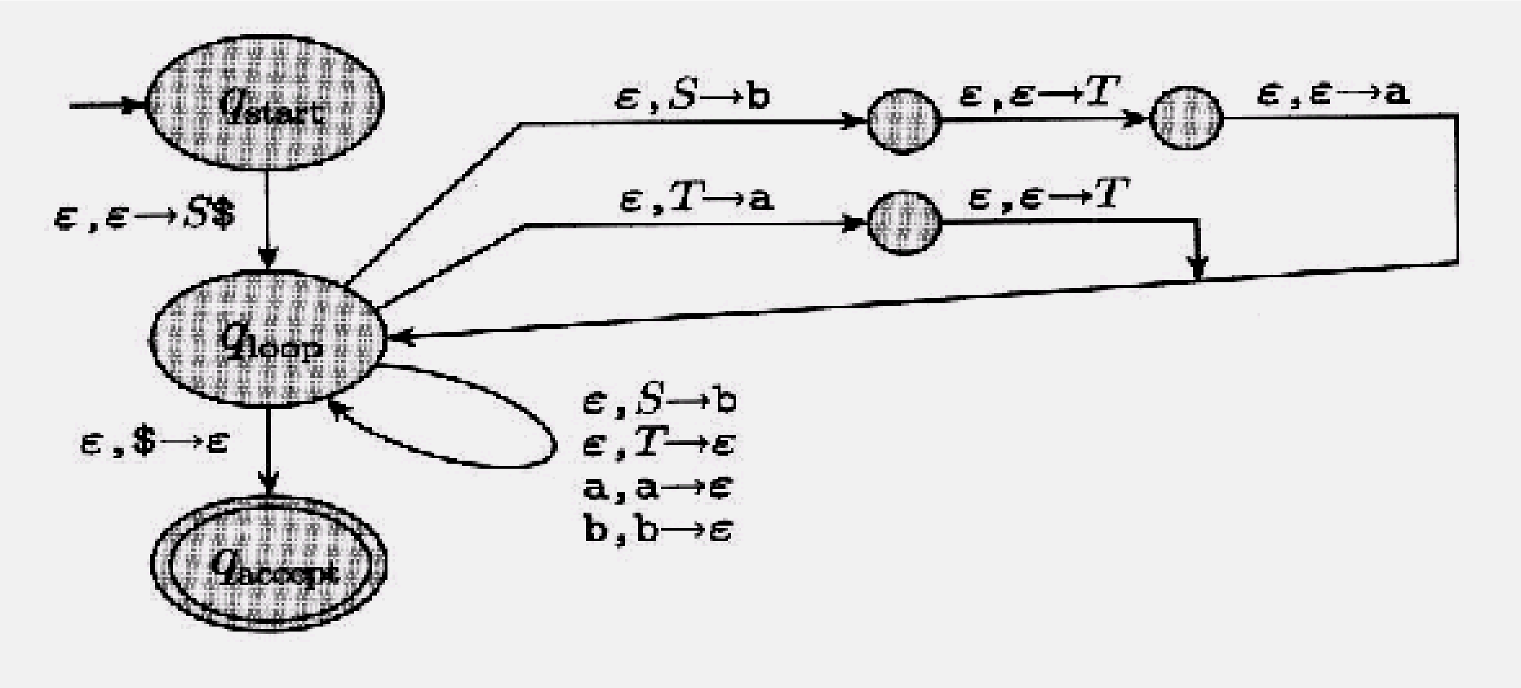


Example 2.25

We use the procedure to construct a PDA P1 from the following CFG G.

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \varepsilon$$

The transition function is shown in the following diagram:



Lemma 2.27

Lemma 2.27:

If a pushdown automaton recognizes some language, then it is context-free. (Backward direction)

Construction

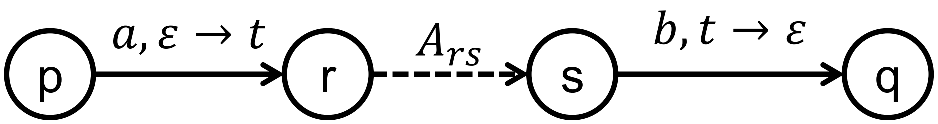
Assume PDA satisfies the following conditions

1. It has a single accept state, q_{accept}
2. It empties the stack before accepting
3. Each transition either pushes symbol onto the stack or removes a symbol from the stack

Proof

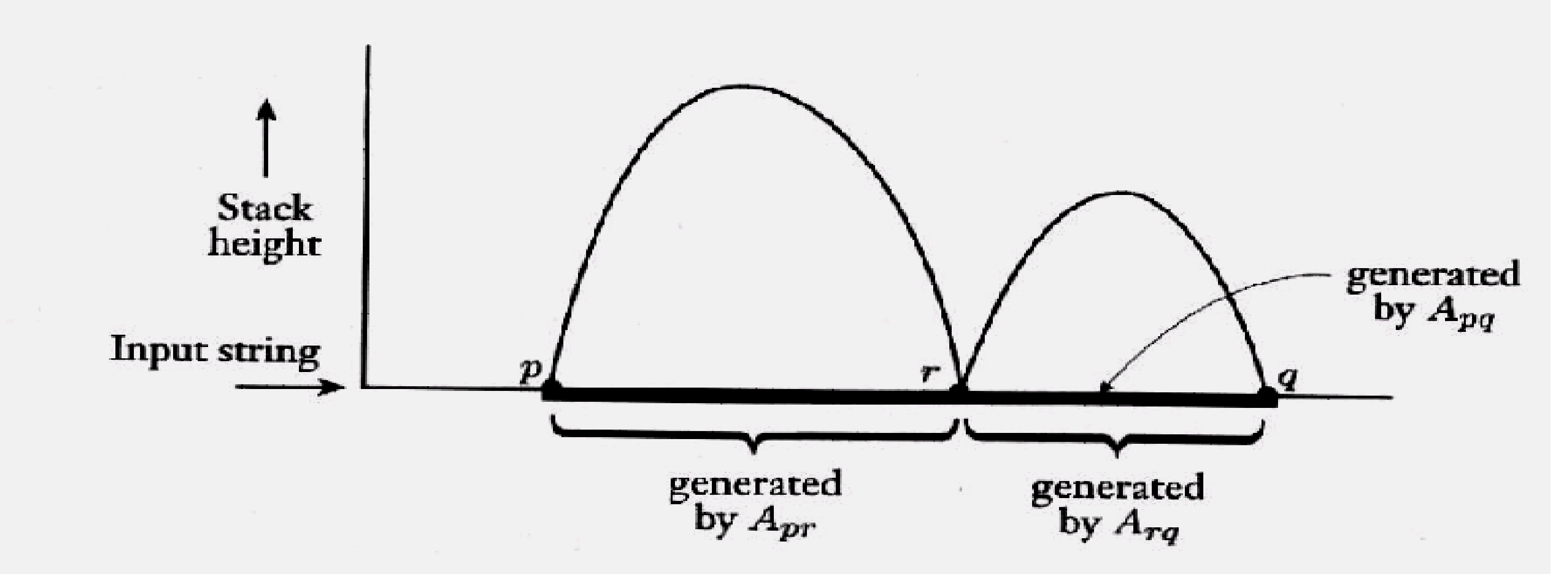
Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ and construct G. The variables of G are $\{A_{pq} \mid p,q \in Q\}$. The start variable is $A_{q_0, q_{accept}}$.

Now we describe G's rules.

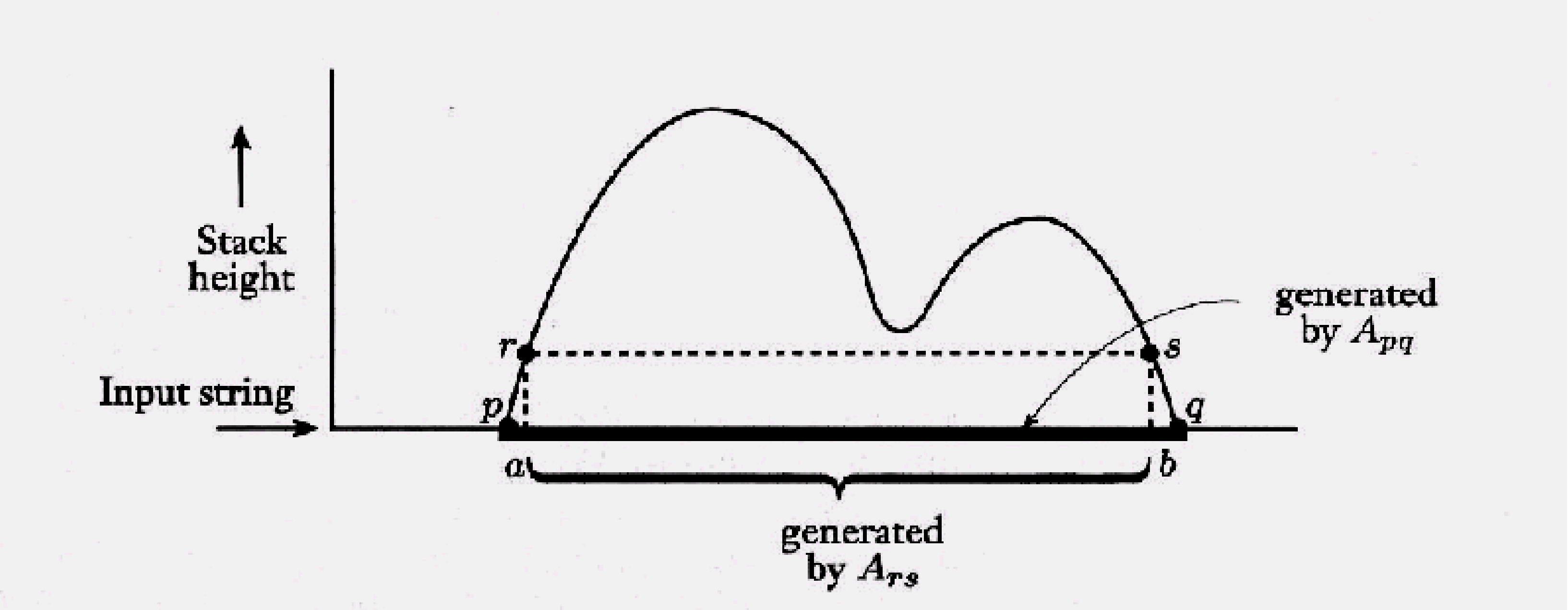


- For each $p,q,r,s \in Q; t \in \Gamma$, and $a,b \in \Sigma_\varepsilon$, if $\delta(p,a,\varepsilon)$ contains (r,t) and $\delta(s,b,t)$ contains (q,ε) put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
- For each $p,q,r \in Q$ put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$ put the rule $A_{pp} \rightarrow \varepsilon$ in G.

You may gain some intuition for this construction from the following figures.



Corresponding to: $A_{pq} \rightarrow A_{pr}A_{rq}$



Corresponding to: $A_{pq} \rightarrow aA_{rs}b$

Claim 2.30

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack

Proof

Basis: derivation has one step, i.e. $A_{pq} \Rightarrow x$ must use a rule with no variables in right hand side → only type $A_{pp} \to \varepsilon$.

Induction: Assume true for derivations of length at most $k \geq 1$ and prove for $k+1$.

Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. Then first step is either

a) $A_{pq} \Rightarrow aA_{rs}b$, or

b) $A_{pq} \Rightarrow A_{pr}A_{rq}$.

Case a): $x = ayb$ and $A_{rs} \overset{*}{\Rightarrow} y$ in k steps with empty stack

Now, because $A_{pq} \Rightarrow aA_{rs}b$ in G, we have $\delta(p, a, \varepsilon) \ni (r, t)$ and $\delta(s, b, t) \ni (q, \varepsilon)$

Therefore, x can bring P from p to q with empty stack.

Claim 2.30

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack

Proof

Basis: derivation has one step, i.e. $A_{pq} \Rightarrow x$ must use a rule with no variables in right hand side → only type $A_{pp} \to \varepsilon$.

Induction: Assume true for derivations of length at most $k \geq 1$ and prove for $k + 1$.

Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k + 1$ steps. Then first step is either

a) $A_{pq} \Rightarrow aA_{rs}b$, or

b) $A_{pq} \Rightarrow A_{pr}A_{rq}$.

Case b): $x = yz$ such that $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} y$ and both derivations use at most k steps.

Therefore, x can bring P from p to q with empty stack.

(**Claim 2.31** "If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x", likewise. See page 123 in Sipser.)

Every regular language is context-free

(.. because NFA is PDA without a stack!)

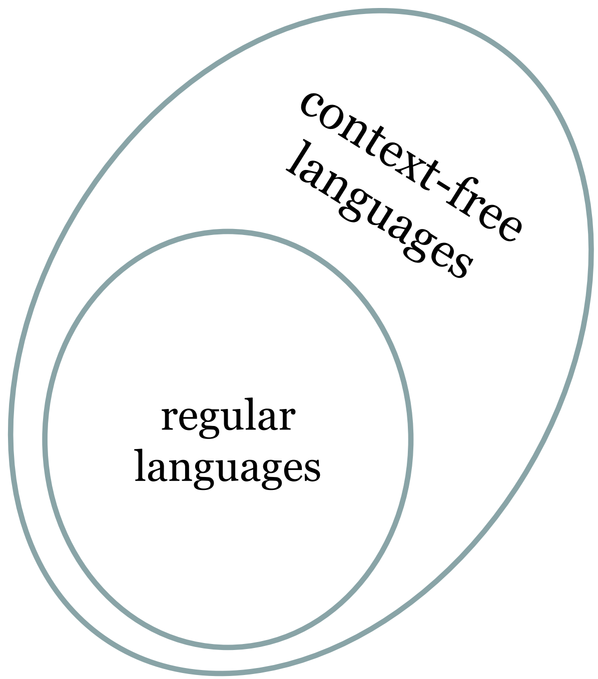


Figure 2.33: Relationship of the regular and context-free languages

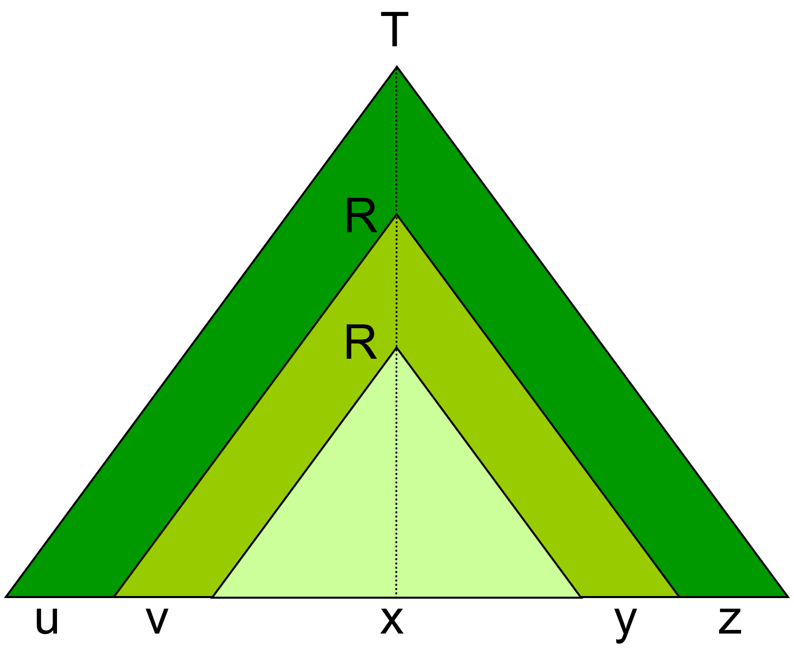
Pumping lemma

Theorem Pumping Lemma

If A is a context free language, then there is a number p such that if s is any string in A of length at least p then s may be dived into $s = uvxyz$ such that

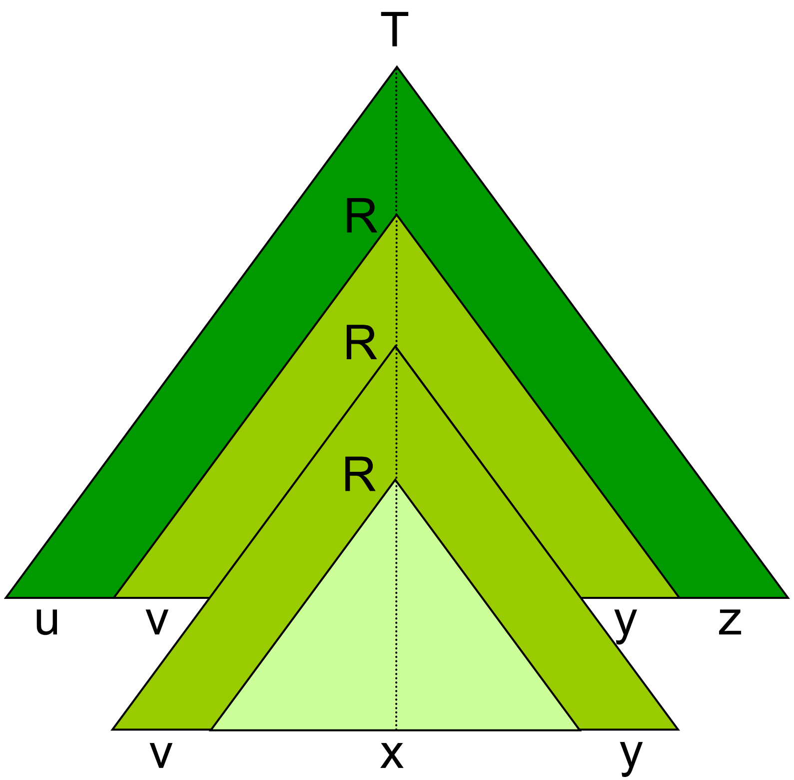
1. For each $i \geq 0$; $uv^i xy^i z \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$

Proof Idea

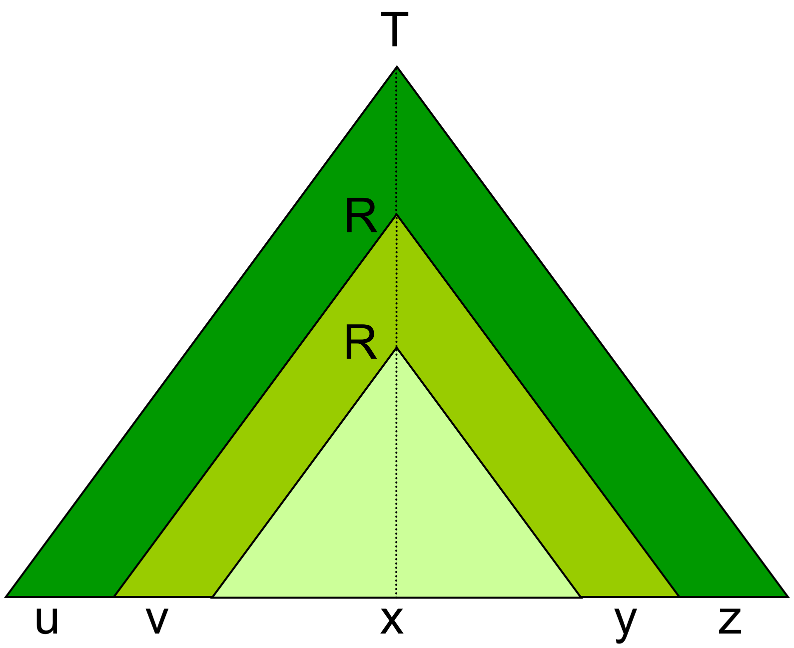


Proof Idea

uv^2xy^2z

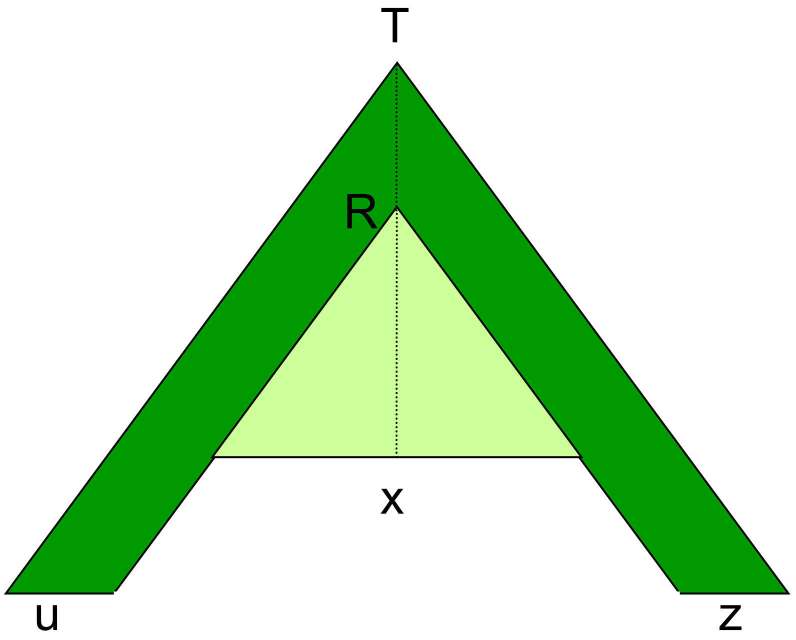


Proof Idea

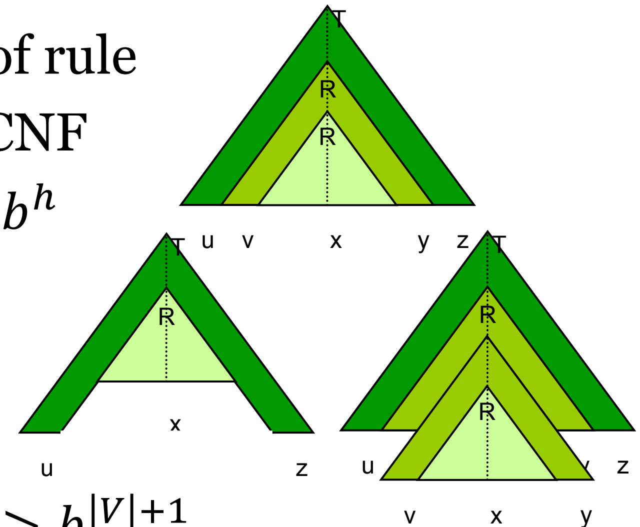


Proof Idea

$$uv^0xy^0z = uxz$$



Proof of pumping lemma (outline)



b: max number of symbols on right hand side of rule

$b \geq 2$ because any CFG can be converted into CNF

number of leaves in a parse tree of height h: $\leq b^h$

hence, for string s of such parse tree: $|s| \leq b^h$

$|V|$: number of variables in CFG G

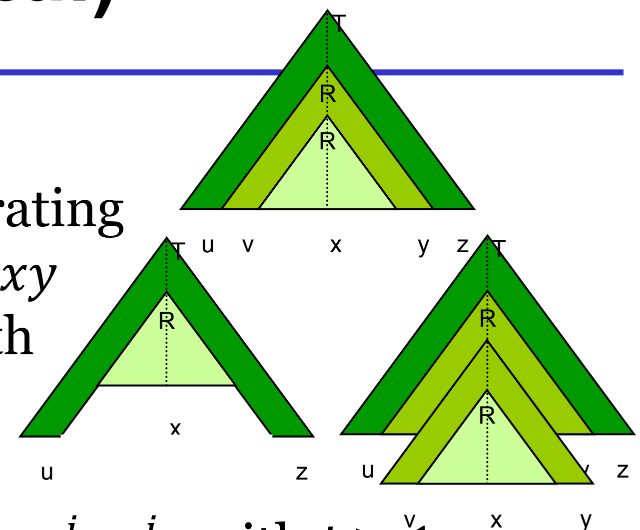
choose pumping length $p = b^{|V|+2}$ such that $p > b^{|V|+1}$

for any $|s| \geq p$: possible parse trees for s have height at least $|V| + 2$

let τ be the parse tree for s with smallest number of nodes:

- → must be at least $|V| + 1$ high
- → must contain a path P from root to a leaf of length at least $|V| + 1$
- → P has at least $|V| + 2$ nodes: one terminal and the rest variables
- → P has at least $|V| + 1$ variables → some variable must be doubled!

Proof of pumping lemma (ctd.)



Divide s into $uvxyz$ as in picture to the right.

Each occurance of R has subtree under it, generating a part of string s. Upper occurrence generates vxy with larger subtree, lower occurrence just x, with smaller subtree. Both are generated by R, thus, we can substitute one for the other.

→ pumping down gives uxz; pumping up gives uv^ixy^iz with $i \geq 1$

→ **condition 1** is satisfied: for each $i \geq 0$, $uv^ixy^iz \in A$

condition 2: $|vy| > 0$

→ must be sure that both v and y are not ε.

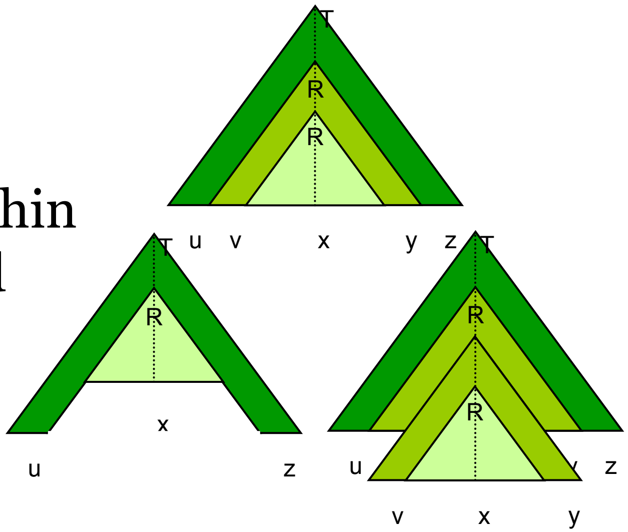
→ Assuming they were ε, substituting smaller for bigger subtree would lead to parse tree with <u>fewer</u> nodes than τ that would still generate s.

→ contradiction: τ chosen to be parse tree with <u>fewest</u> number of nodes

Proof of pumping lemma (ctd.)

condition 3: $|vxy| \leq p$

- upper occurrence of R generates vxy
- R chosen such that both occurrences fall within the bottom $|V| + 1$ variables on the path and chose longest path in parse tree
- subtree where R generates vxy is at most $|V| + 2$ high.
- Any such tree of height $|V| + 2$ can only generate strings of length at most $b^{|V|+2} = p$



■

$B = \{a^n b^n c^n \mid n \geq 0\}$ is not context free

1. For each $i \geq 0$; $uv^i xy^i z \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$

choose $s = a^p b^p c^p$

clearly in B

because 2) either v or y not empty

Consider two cases :

A. both v and y contain only one type of alphabet symbol

Then $uv^2xy^2z \notin B$ (does not contain equal no. of a, b, c)

B. either v or y contain more than one type of symbol

Then $uv^2xy^2z \notin B$ (does not have right order of a, b, c)

$C = \{a^i b^j c^k \mid 0 \le i \le j \le k\}$ is not context free

1. For each $i \ge 0$; $uv^i xy^i z \in A$
2. $|vy| > 0$
3. $|vxy| \le p$

choose $s = a^p b^p c^p$; clearly in C

because 2) either v or y not empty; Consider two cases :

A. both v and y contain only one type of alphabet symbol

Three subcases :

A1. a does not appear in v and y

Then $uv^0 xy^0 z \notin B$ (contains fewer b, c)

A2. b does not appear in v and y

If a appears then $uv^2 xy^2 z \notin B$ (contains more a than b)

If c appears then $uv^0 xy^0 z \notin B$ (contains more c than b)

A3. c does not appear in v and y

Then $uv^2 xy^2 z \notin B$

B. either v or y contain more than one symbol

Then $uv^2 xy^2 z \notin B$ (does not have right order of a, b, c)

Overview

- Context free grammars
- Pushdown Automata
- Equivalence of PDAs and CFGs
- Non-context free grammars
  - Pumping lemma