Context-free Languages

Andreas Karwath & Malte Helmert

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 Concepts: Context-free Grammar

Grammar G_1 :

$A \rightarrow 0A1$

$A \rightarrow B$

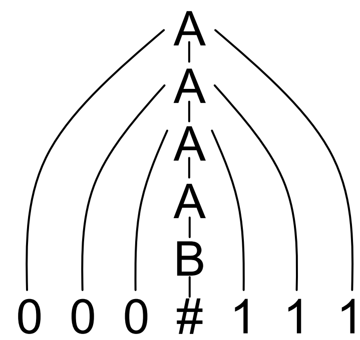
$B \rightarrow \#$

Terminals 0,1,# (correspond to the alphabet Σ)

Nonterminals / Variables A, B

Rules $Symbol \rightarrow String$

Startsymbol



A parse tree for 000#111 in grammar G_1

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

Example, derivation for 000#111:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

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

Language defined by grammar G_1

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**

a boy sees

the boy sees a flower

a girl with a flower likes the boy

<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**

Definition Context free grammar

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

1. V is a finite set called the variables
2. Σ is 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

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

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

Parsing

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

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

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

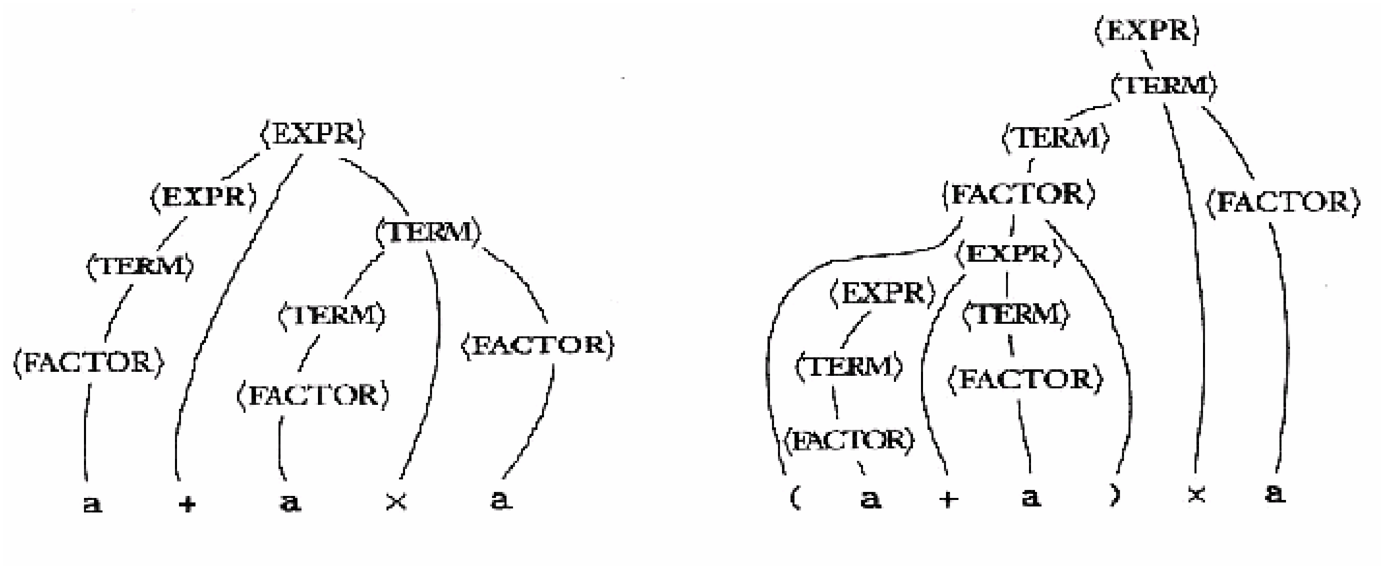
R is

$<Expr> \to <Expr> + <Term> \mid <Term>$

$<Term> \to <Term> \times <Factor> \mid <Factor>$

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

- ✶ 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

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

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

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

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 $R_i \rightarrow \varepsilon$

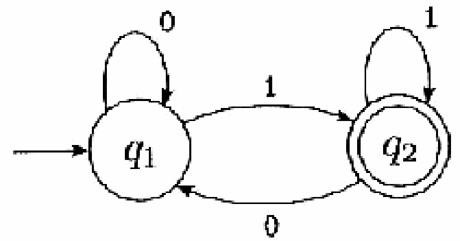


FIGURE 1.6
State diagram of the two-state finite automaton M_2

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

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

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

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

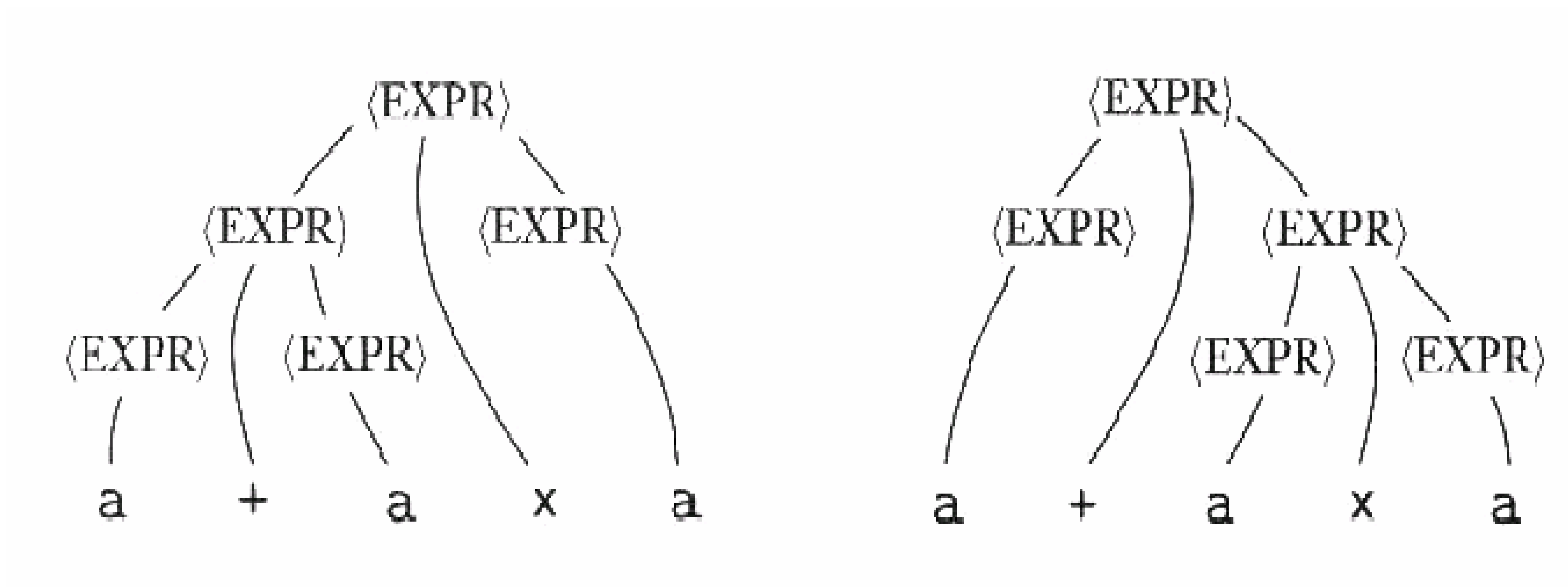
Ambiguity

- 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, ie. every grammar for the language is ambiguous

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

Chomsky-Normal-Form

Definition

A context-free grammar is in Chomsky-Normal-Form, if each rule is the following form:

- ★ **A → BC** or
- ★ **A → a** or
- ★ **S → ε**

where

- ★ A,B,C,S are the variables
- ★ a is a terminal
- ★ S is the start variable.
- ★ B,C are not the start variable,

Chomsky-Normal-Form

Theorem

Every context-free language is generated by a grammar in Chomsky-Normal-Form

Example

context-free grammar:

- G = ({A,B}, {0,1,#}, R, A}
- R = {A $\rightarrow$ 0A1, A $\rightarrow$ B, B $\rightarrow$ #}

A grammar of the same language in Chomsky-Normal-Form:

- G'= ({A,B,C,N,E}, {0,1,#}, R, S}
- R = {S $\rightarrow$ NC, N$\rightarrow$ 0,S $\rightarrow$ #,
  A $\rightarrow$ NC, C $\rightarrow$ AE, E $\rightarrow$1, A $\rightarrow$ #}

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 site of a rule**
   - Solution: introduce a new start variable and a new rule for the derivation
2. **Epsilon-Rules: A $\rightarrow \varepsilon$**
   - Solution: if A occurs in the right part of a rule, *introduce* new rules without A on the right part of the rule
3. **Unit-Rules: A $\rightarrow$ B**
   - Solution: directly replace B by its own production
4. **Long and/or mixed rules: A $\rightarrow$ aBcAbA**
   - Solution: new variables/new rules

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 refering 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 removed.

Proof by Construction (cont.)

4a. Convert remaining rules $A \rightarrow u_1 u_2 ... u_k$ 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)

Example 2.7

Let G_6 be the following CFG and convert it to Chomsky normal form by using the conversion precedure just given. The following series of grammars illustrates the steps in the conversion. Rules shown in **bold** have been just added. Rules shown in blue have just been removed.

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

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

Example 2.7

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

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

Example 2.7 (cont.)

3a. Remove unit rules $S \rightarrow S$, shown on the left, and $S_0 \rightarrow S$, shown on the right.

$$S_0 \rightarrow S$$
$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b$$

Example 2.7 (cont.)

3b. Remove unit rules $A \rightarrow B$ and $A \rightarrow S$.

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

Example 2.7 (cont.)

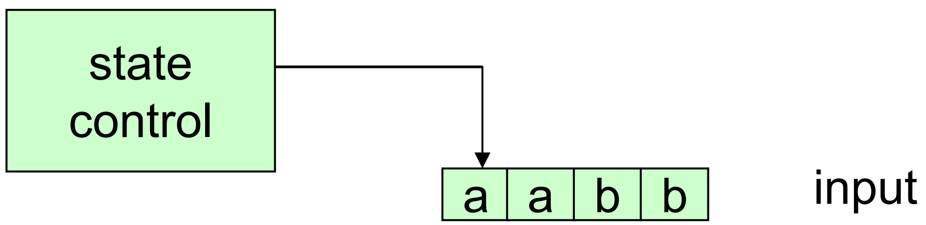
4.) Convert the remaining rules

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

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

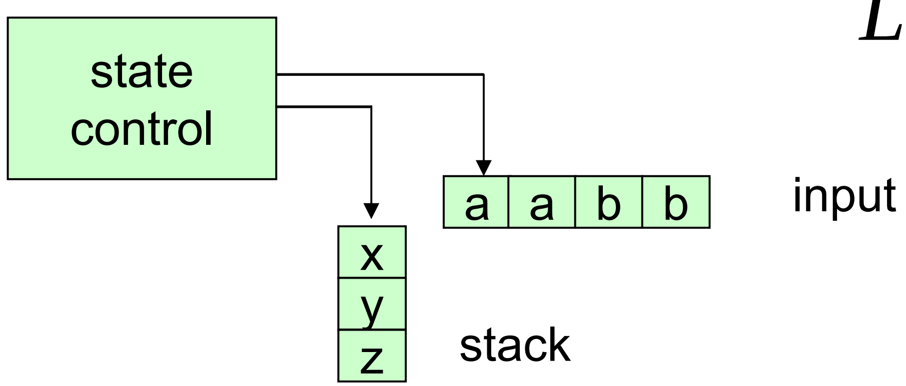
Pushdown automata

- Schema of a finite automaton



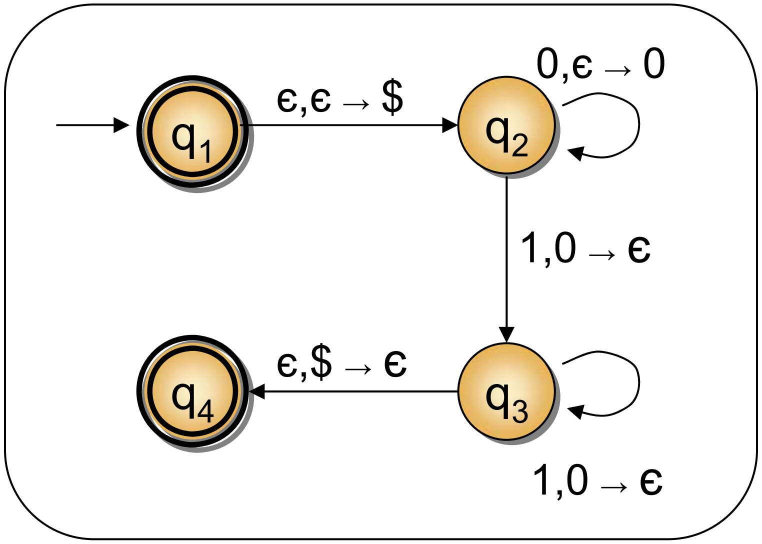
Pushdown automaton

- 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 $\{0^n1^n \mid n > 0\}$

Formal definition (Definition 2.8)

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. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathrm{P}(Q \times \Gamma_\varepsilon)$ 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 *nstackymbol*
input, stack, and *nstacksymbol* can be ε !

Example 2.9

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									

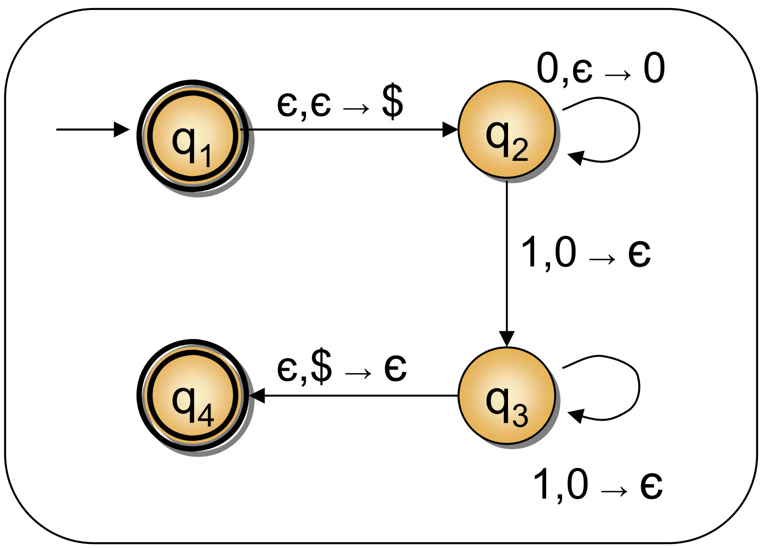
Computation with PDAs

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 !

$(0011, q_1, \varepsilon)$



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 $i = 0,...,n-1$

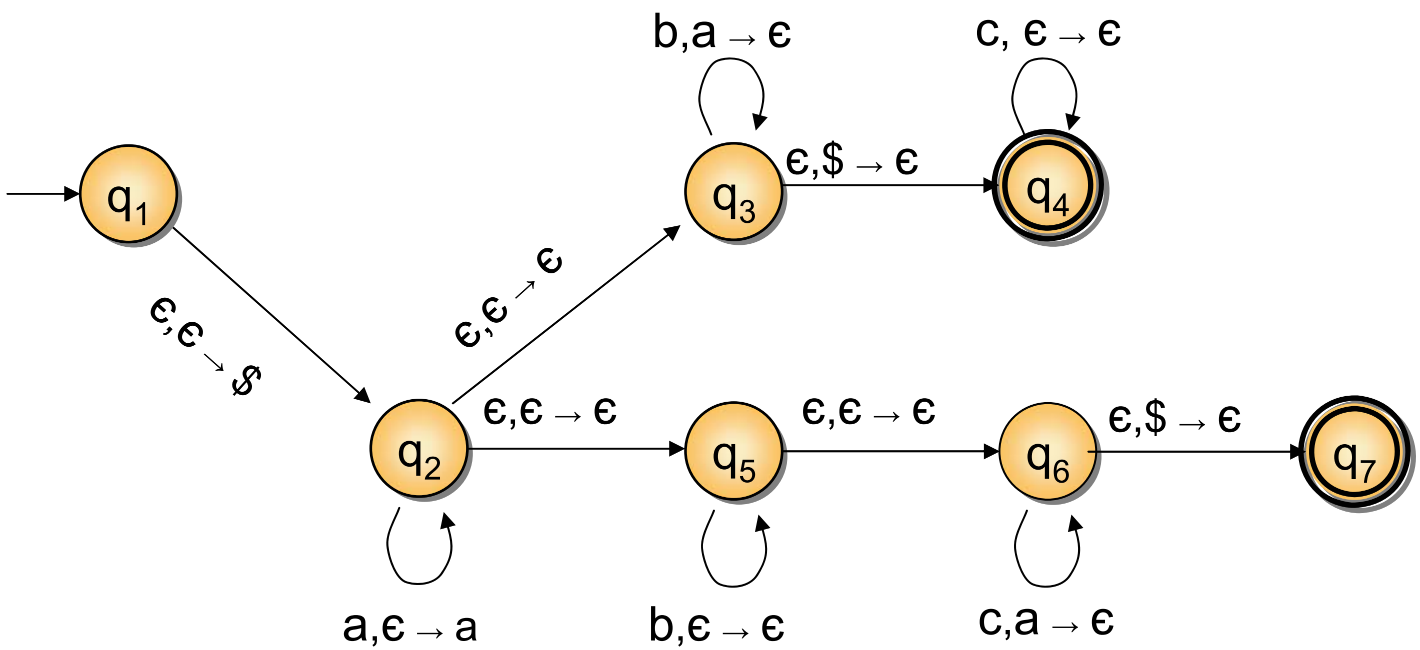
   $(r_{i+1},b) \in \delta(r_i,w_{i+1},a)$ where s_i=at and s_{i+1}=bt

   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 $\{a^i\, b^j\, c^k | i,j,k \geq 0 \text{ and } i = j \text{ or } i = k\}$

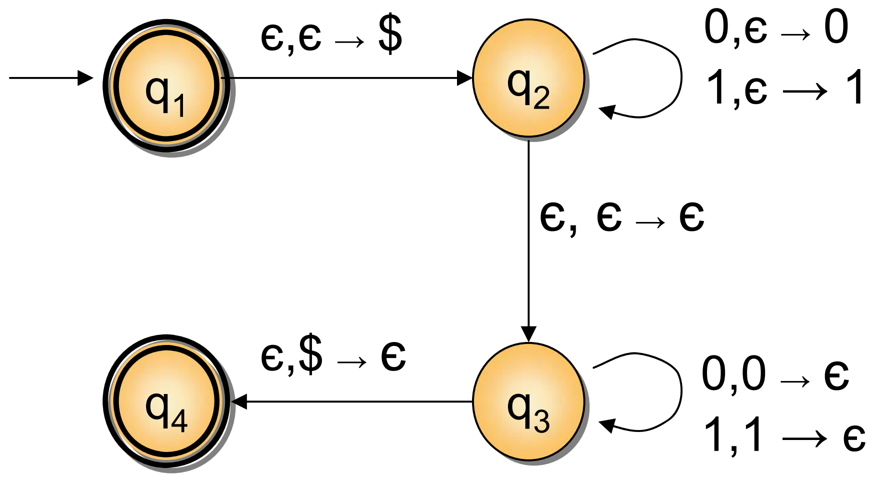


State diagram for PDA M2 that recognizes the language $\{a^i b^j c^k \mid i.j.k \geq 0 \text{ and } i = j \text{ or } i = k\}$

- *Non determinism* essential for this language

Another example

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



Theorem 2.12 and Lemma 2.13

Theorem 2.12

A language is context free if and only if some pushdown automaton recognizes it

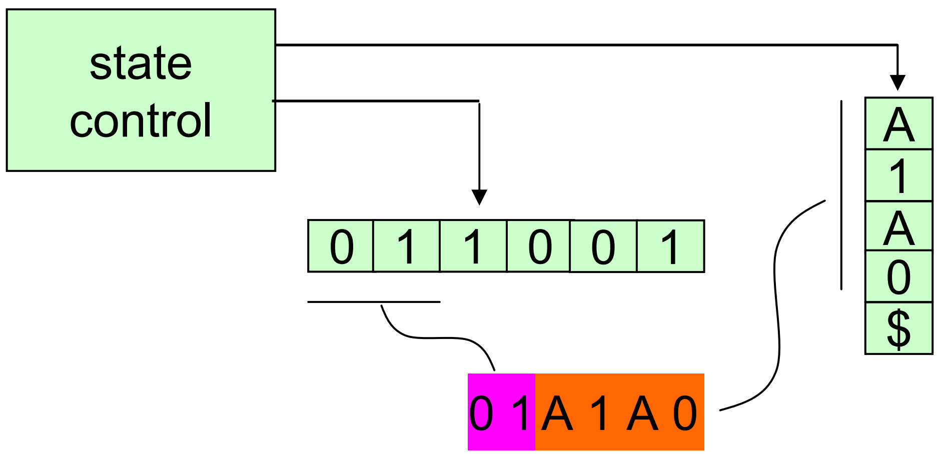
Lemma 2.13

If a language is context free then some pushdown automaton recognizes it

- 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 2.13 Proof idea



P presenting the intermediate string 01A1A0

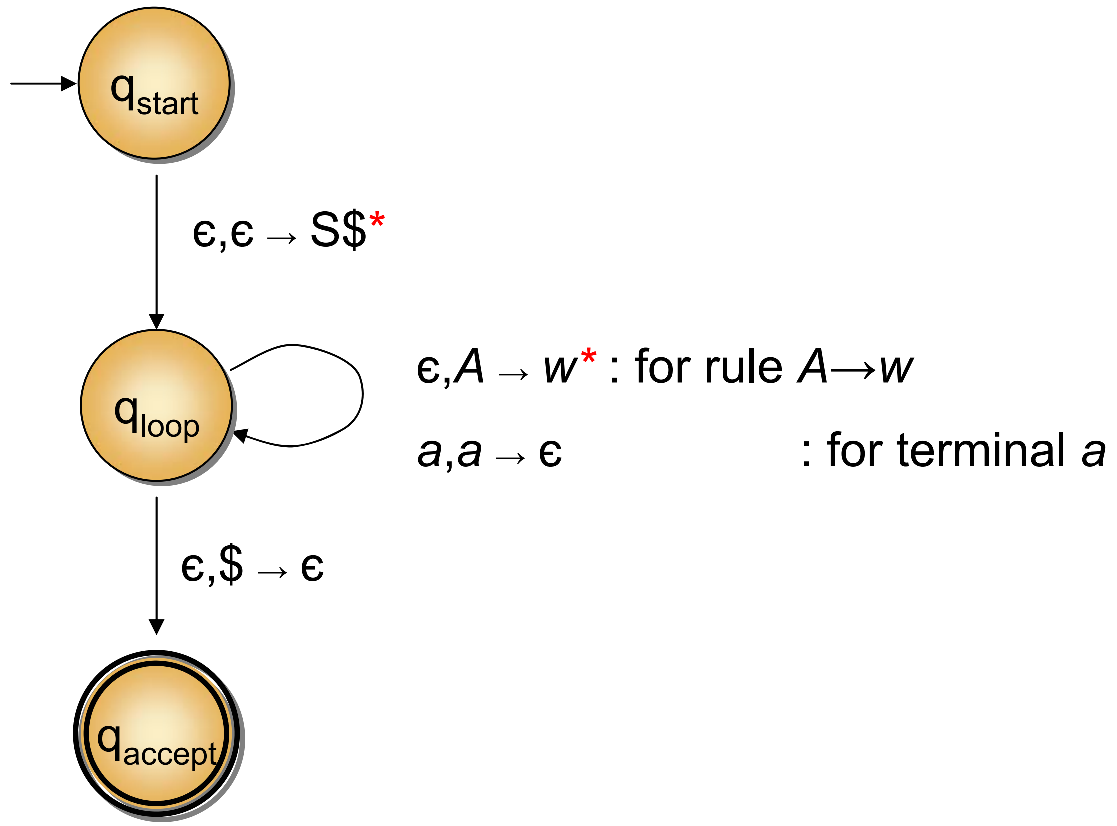
- Substitute variables by strings
- Replace top variable on stack by string

Lemma 2.13 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

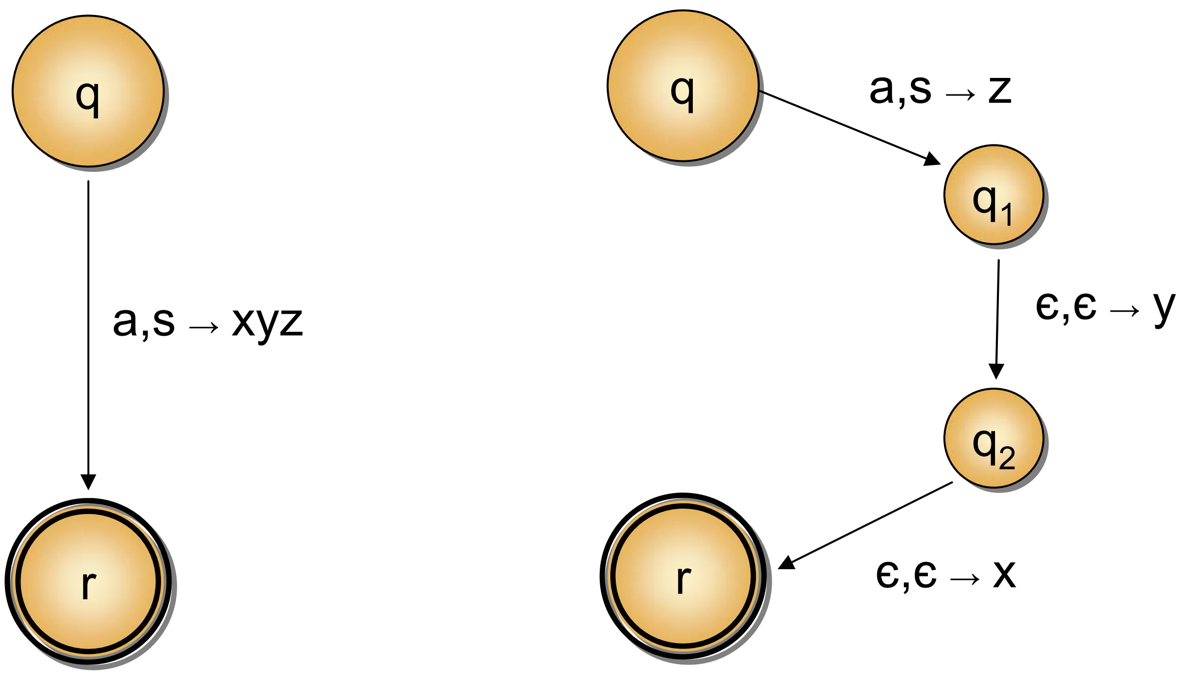
Resulting PDA



- State diagram of P

* shorthand notation

About „*Shorthand*“

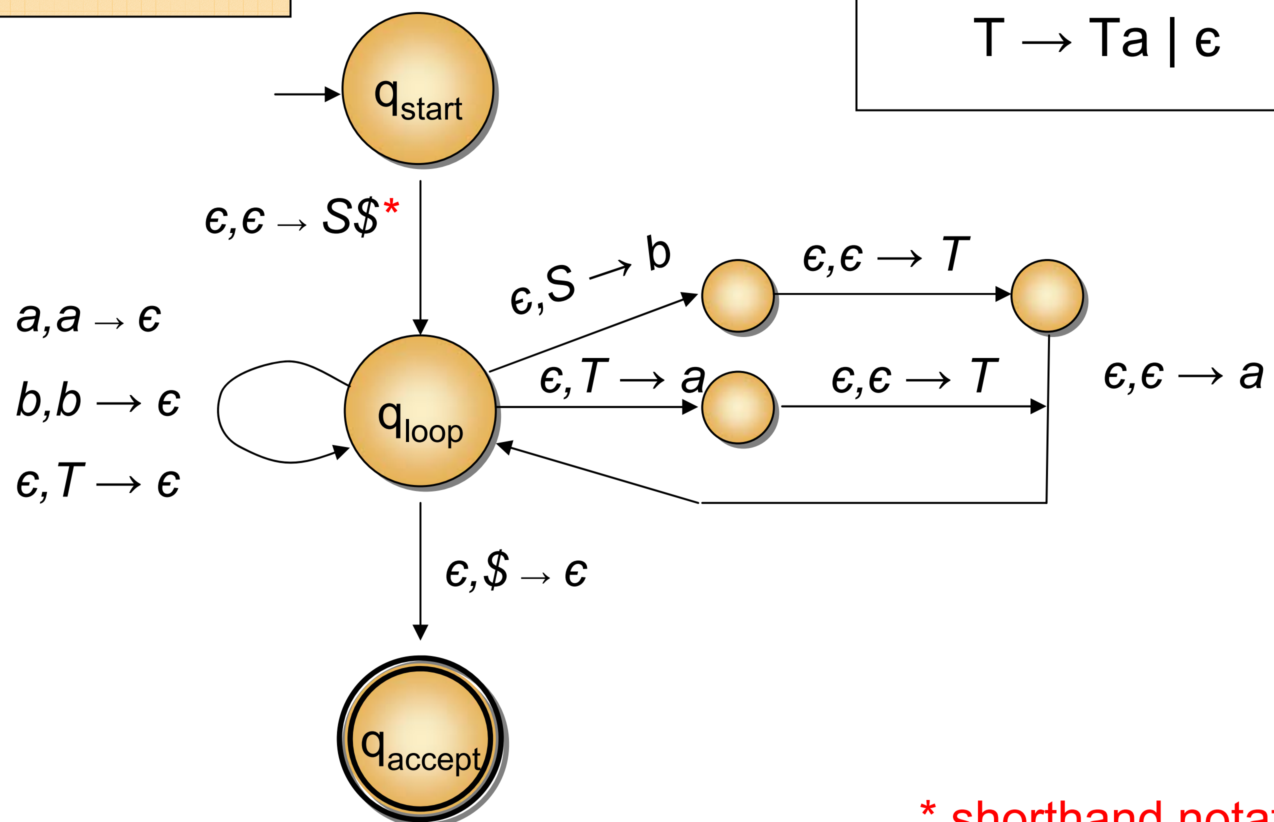


- Implementing shorthand: $(r,xyz) \in \delta(a,a,s)$

Resulting PDA

$\epsilon, A \rightarrow w$* : for rule $A \rightarrow w$
$a, a \rightarrow \epsilon$: for terminal a

CFG G:
S → aTb | b
T → Ta | ϵ



* shorthand notation

Lemma 2.15

Lemma 2.15:

If a pushdown automaton recognizes some language, then it is context-free.

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 $A_{pq} \to aA_{rs}b$ in G.
- For each $p, q, r \in Q$ put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$ put the rule $A_{pp} \to \varepsilon$ in G.

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

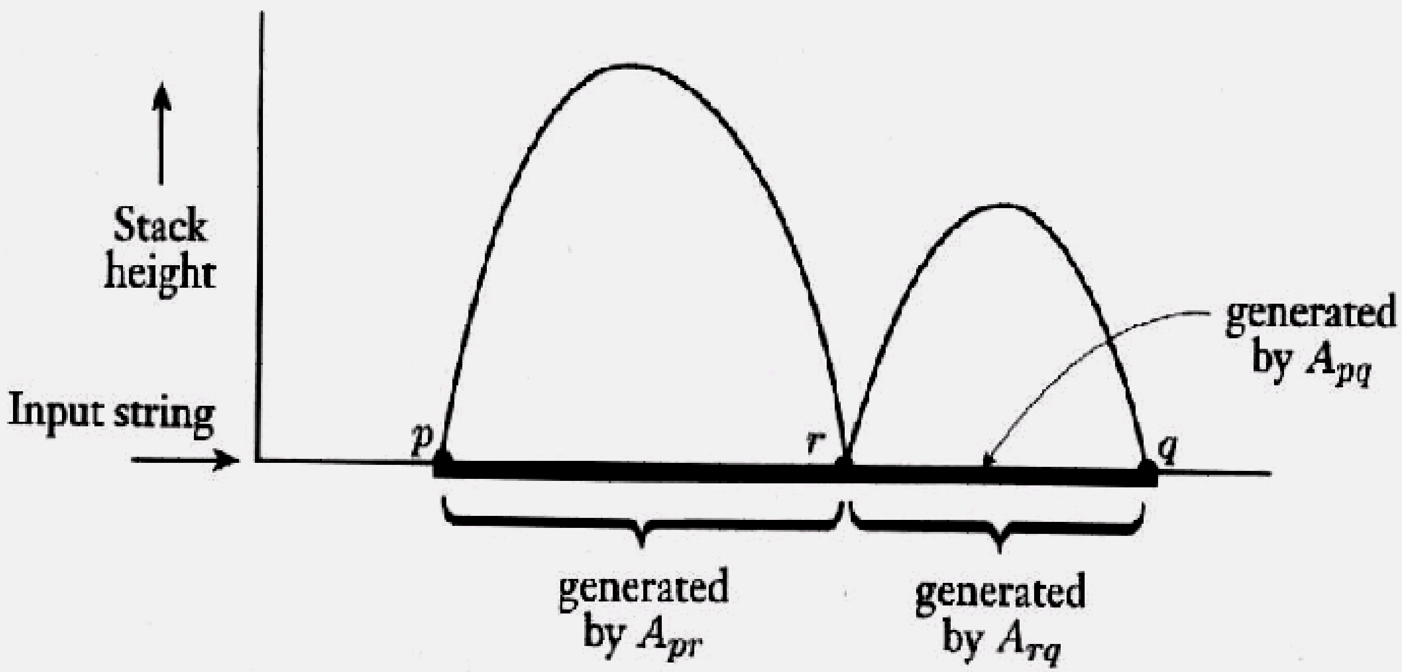


FIGURE 2.13

PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

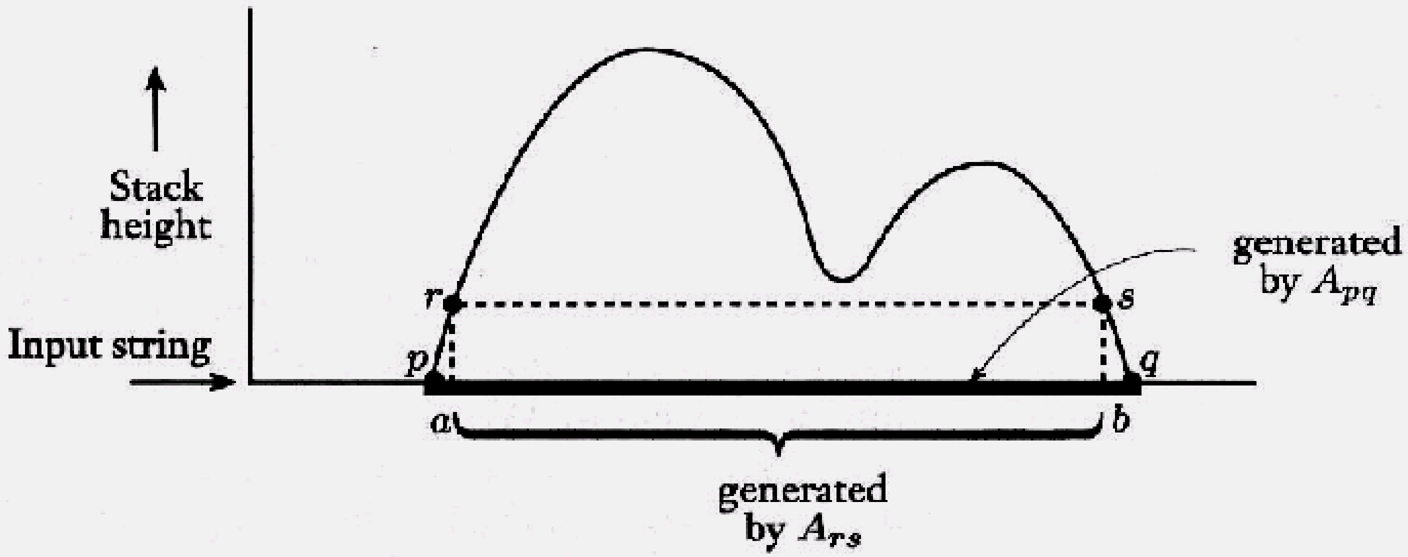


FIGURE 2.14
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

Claim 2.16

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

Proof

Basis : derivation has 1 step, i.e. $A_{pq} \Rightarrow x$ must use a rule with no variables in right hand side

only type $A_{pp} \rightarrow \varepsilon$

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

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

Case b. let $x = yz$ such that $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$

both derivations use at most k steps

Therefore x can bring P from p to q via r with empty stack

Claim 2.17

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

Proof

Basis : computation has 0 steps

Therefore, it starts and ends in same state, so we must prove that $A_{pp} \overset{*}{\Rightarrow} x$,

In 0 steps, x must be ε

This rule $A_{pp} \rightarrow \varepsilon$ is in G

Induction : Assume true for computations of length at most $k \geq 0$ and prove for $k+1$.

Suppose P has a computation where x brings p to q with emtpy stack in $k+1$ steps. Either stack is empty

a. only at the beginning and end, or b. also somewhere else.

Case a. symbol that is pushed first = symbol that is popped last = t

let a be the input read in first move, r be the state after first move

let b be the input read in last move, s be the state before last move

Then $\delta(p,a,\varepsilon) \ni (r,t)$ and $\delta(s,b,t) \ni (q,\varepsilon)$

So, $A_{pq} \rightarrow aA_{rs}b$ in G

Let $x = ayb$; then $A_{rs} \overset{*}{\Rightarrow} y$ in $k-1$ steps

So, $A_{pq} \overset{*}{\Rightarrow} x$

Case b. let r be the state where the stack becomes empty

then computations from p to r and from r to q take at most k steps

hence, $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$

Because $A_{pq} \rightarrow A_{pr}A_{rq}$ in G, $A_{pq} \overset{*}{\Rightarrow} x$

Every regular language is context-free

(because NFA is PDA without stack)

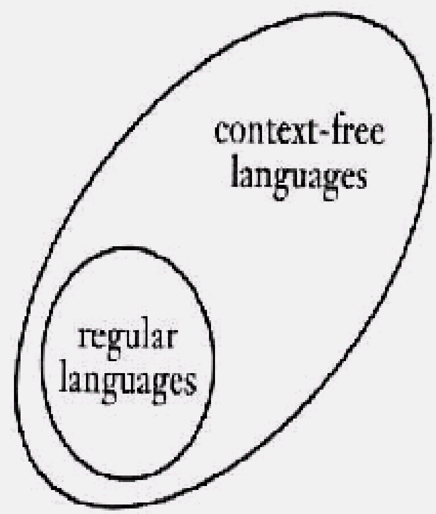


FIGURE 2.15
Relationship of the regular and context-free languages

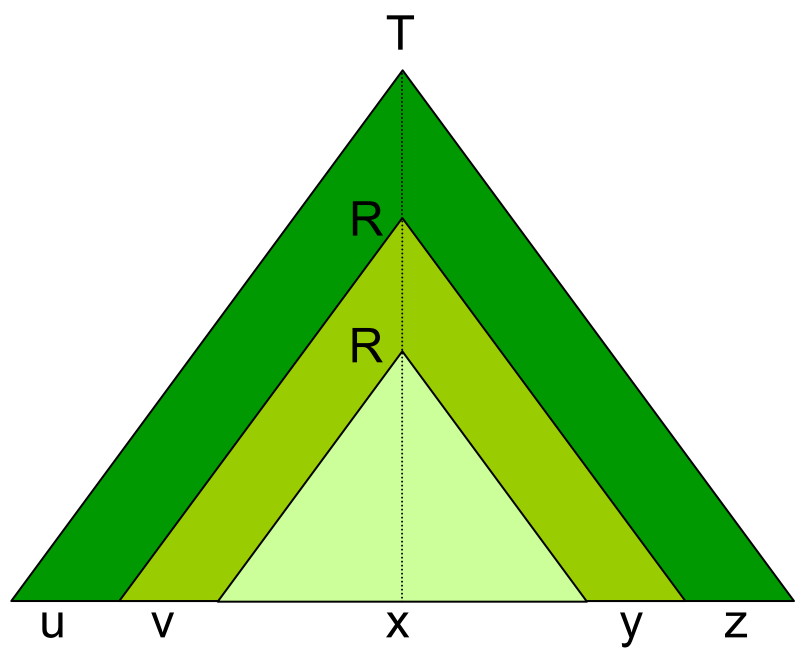
Pumping lemma

Theorem Pumping Lemma

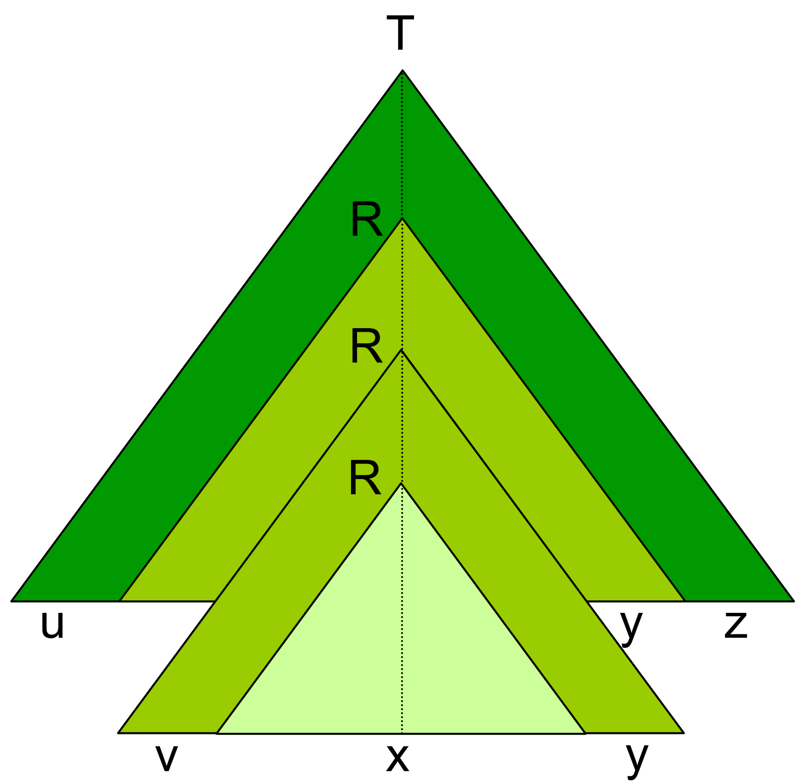
If A is a context free language, then there is a number p such that if s is any string in A of length at least p then s may be dived into $s = uvxyz$ such that

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

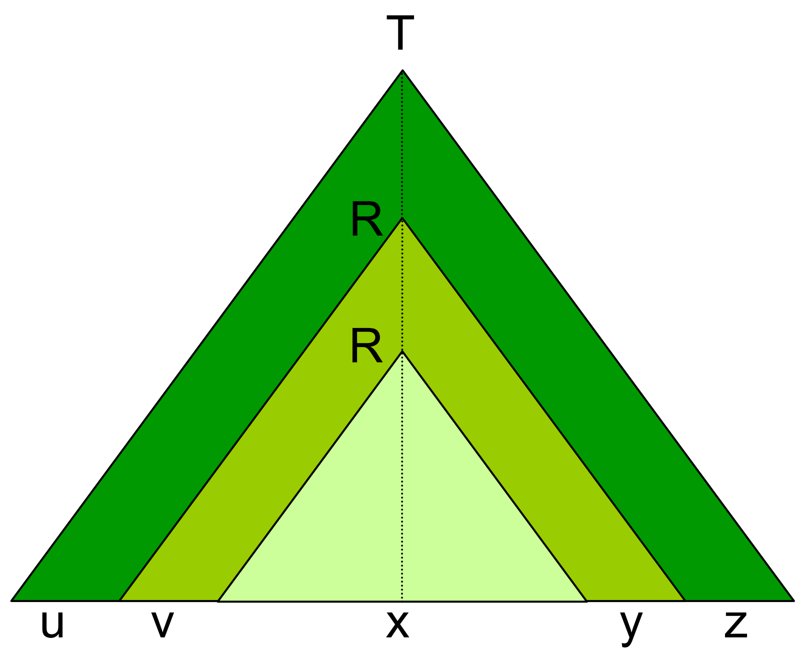
Proof Idea



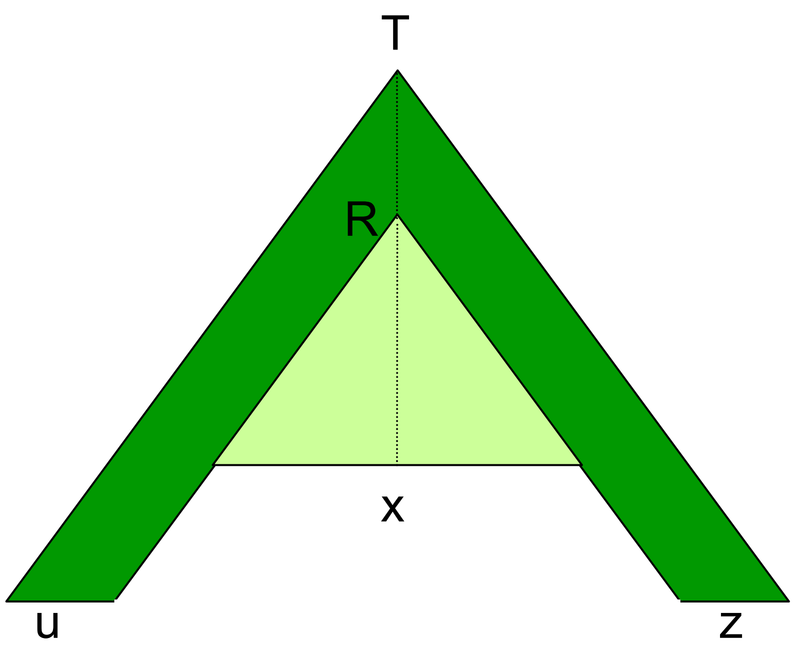
Proof Idea



Proof Idea



Proof Idea



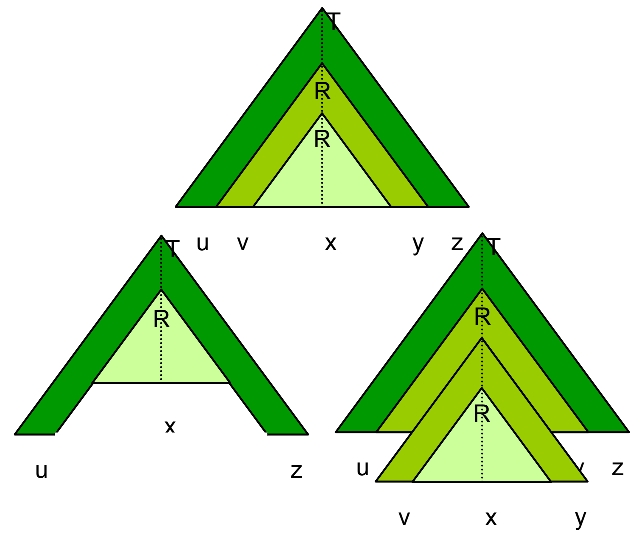
Proof elements

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

$b \geq 2$ because CFG (look at CNF)

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

hence, length of string in a parse tree of height $h \leq b^h$



$|V|$: number of vars in *Grammar*

choose $p = b^{|V|+2}$; so $p > b^{|V|+1}$ (because $b \geq 2$)

assume $|s| \geq p$

so, parse tree for s has height at least $|V| + 2$

take smallest parse tree for s

apply pigeonhole principle on longest path: R repeating var

Prove 1), 2) see figures

3) choose R in bottom $|V| + 1$ vars.

Subtree generating R has height at most $|V| + 2$

String vxy generated by R at most length $p = b^{|V|+2}$

$B = \{a^n b^n c^n \mid n \geq 0\}$ 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 less 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

Proof by Construction

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

4a. Convert remaining rules $A \to u_1 u_2 \ldots u_k$ where $k \geq 3$ into rules

$A \to u_1 A_1$

$A_1 \to u_2 A_2$

...

$A_{k-2} \to u_{k-1} u_k$

where the A_i are new variables

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

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