Theory of Languages and Automata

Chapter 2- Context-free Languages

Sharif University of Technology

Table of contents

Context-Free Grammars

- A collection of substitution rules (production)
  - Each rule: a line in the grammar
  - Each line: a variable, an arrow, a string
  - The string: variables and terminals

Example

- G1: $A \longrightarrow 0A1$
  $A \longrightarrow B$
  $B \longrightarrow \#$

- Three rules
- Variables: A,B
- Start Variable: A
- Terminals: 0,1,#

Another Example

- G2 (describes a fragment of the English language):

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\rightarrow \texttt{with}
\end{aligned}
$$

Derivation

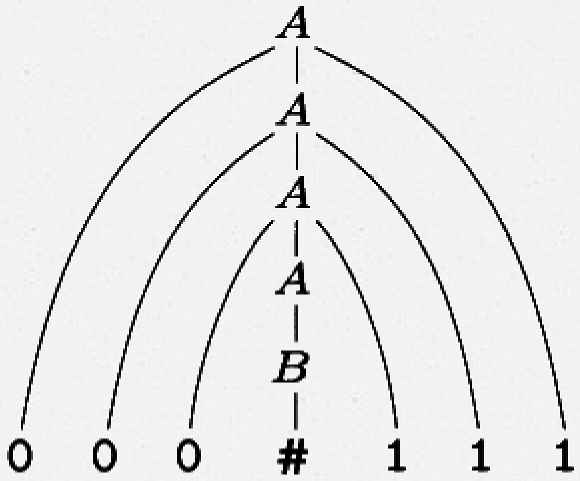
- Generating the strings of the language:
  1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
  2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of the rule.
  3. Repeat step 2 until no variables remain.

Example

- G1:

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

- Parse Tree:



Another Example

- Derivation of the sentence "a boy sees" in G2

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a}\ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}
$$

Context-Free Language

- All Strings generated in this way

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

Context-Free Grammar (Formal Definition)

- 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, and
  4. $S \in V$ is the start variable.

Context-Free Language (Formal Definition)

- u derives v:

  $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

  $$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

- The language of the grammar:

  $$\{ w \in \Sigma^* / S \overset{*}{\Rightarrow} w \}$$

Example

- G3 = ({S}, {a,b}, *R*, *S*)

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

➜ Strings such as: abab, aaabbb, aababb

! If a → (

and b →),

then: L(G3) → language of all strings of properly nested parentheses

Another Example

- G4 = (V, $\sum$, R, $\langle$EXPR$\rangle$)

  V = {$\langle$EXPR$\rangle$, $\langle$TERM$\rangle$, $\langle$FACTOR$\rangle$}

  $\sum$ = {a, +, ×, (,)}

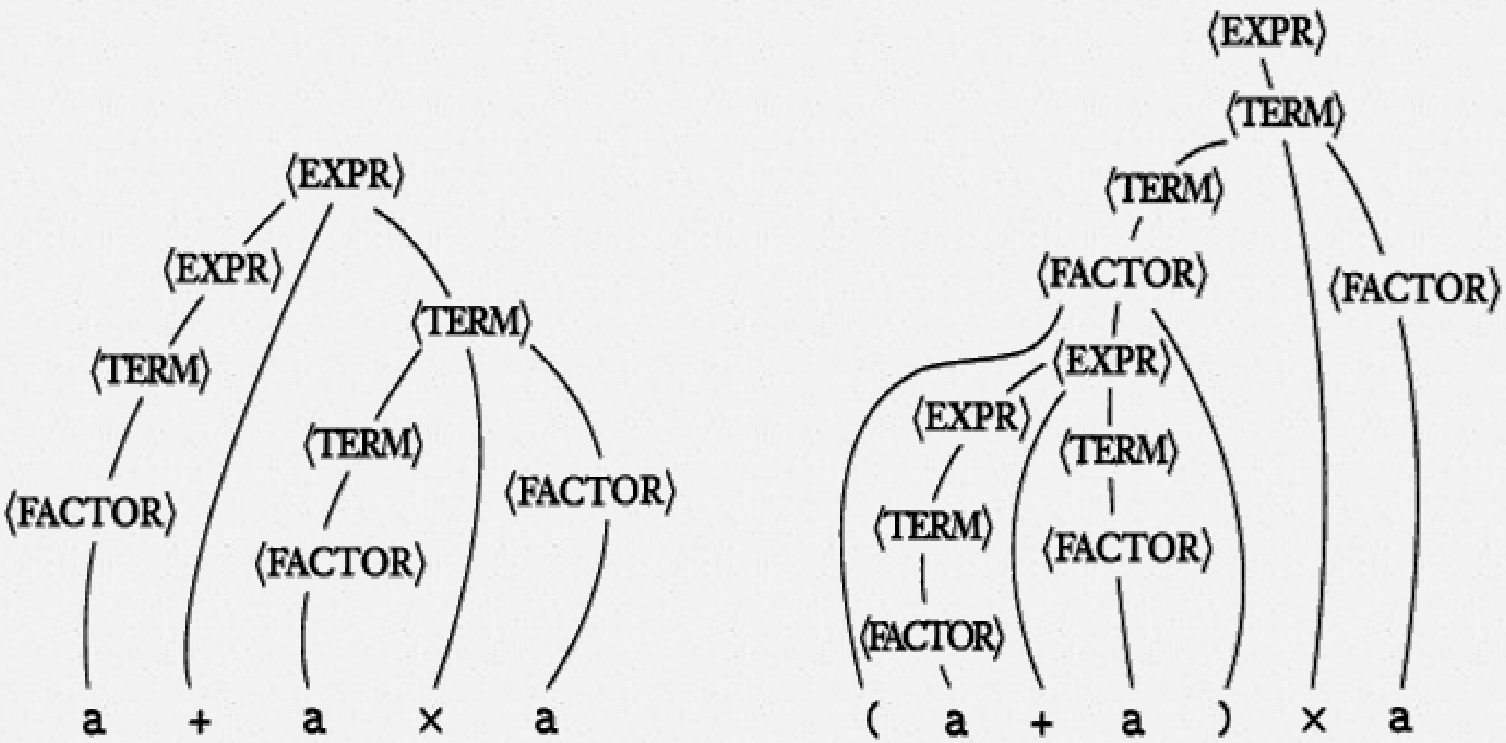
  R: $\langle$EXPR$\rangle \rightarrow \langle$EXPR$\rangle$ + $\langle$TERM$\rangle$ | $\langle$TERM$\rangle$

  $\langle$TERM$\rangle \rightarrow \langle$TERM$\rangle$ × $\langle$FACTOR$\rangle$ | $\langle$FACTOR$\rangle$

  $\langle$FACTOR$\rangle \rightarrow$ ($\langle$EXPR$\rangle$) | a

Another Example (cont.)

- Parse trees for the strings a+a×a and (a+a)×a



CFG Design

- Break the CFL into simpler pieces, if possible
- Construct grammar for each piece
- Combine the rules and the new rule $S \rightarrow S_1|S_2|\ldots|S_k$, where the S_i are the start variables for the individual grammars.

Example

- $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$
  - ✓ $\{0^n1^n \mid n \geq 0\} : S_1 \rightarrow 0S_11 \mid \varepsilon$
  - ✓ $\{1^n0^n \mid n \geq 0\} : S_2 \rightarrow 1S_20 \mid \varepsilon$
  - ✓ Add the rule: $S \rightarrow S_1 \mid S_2$

➜ G:

$$S \rightarrow S_1 \mid S_2$$
$$S_1 \rightarrow 0S_11 \mid \varepsilon$$
$$S_2 \rightarrow 1S_20 \mid \varepsilon$$

CFG Design (cont.)

- Any DFA can be converted to an equivalent CFG
  - ✓ Make a variable R_i for each state q_i
  - ✓ Add the rule $R_i \rightarrow aR_j$, if $\delta(q_i, a) = q_j$
  - ✓ Add the rule $R_i \rightarrow \varepsilon$, if q_i is an accept state

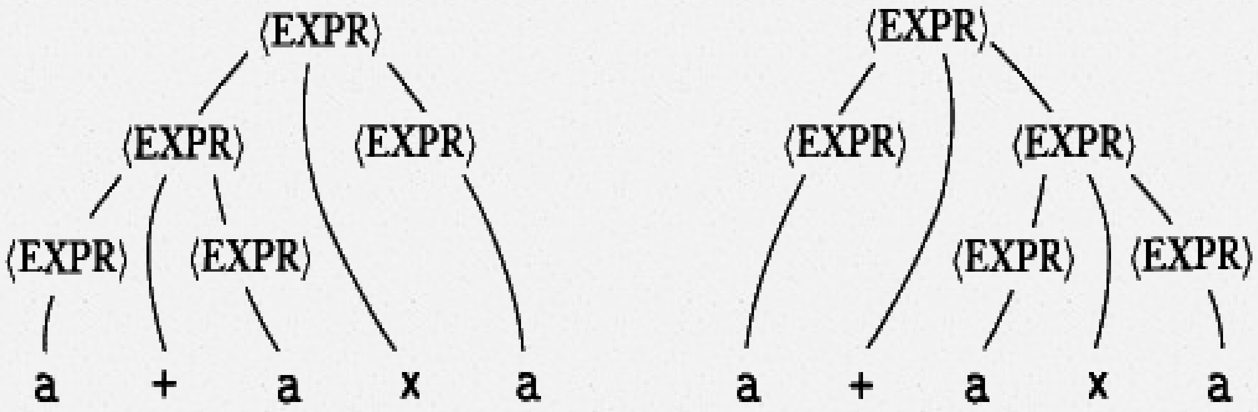
Ambiguity

- When a grammar can generate the same string in different ways

- Undesirable for applications such as programming languages

Example

- $\langle \mathrm{EXPR} \rangle \to \langle \mathrm{EXPR} \rangle + \langle \mathrm{EXPR} \rangle \mid \langle \mathrm{EXPR} \rangle \times \langle \mathrm{EXPR} \rangle \mid (\langle \mathrm{EXPR} \rangle) \mid a$
- This grammar generates the string a+a×a ambiguously



Ambiguity: Formal Definition

- Definition: A string w is ambiguous if it can be derived by at least two derivation (parse) trees.

Leftmost (Rightmost) Derivation

- Leftmost (Rightmost) Derivation: at every step, the leftmost (rightmost) remaining variable is replaced.

Ambiguity (Equivalent Definition)

- A string w is derived ***ambiguously*** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ***ambiguous*** if it generates some string ambiguously. Otherwise it is ***unambiguous***.

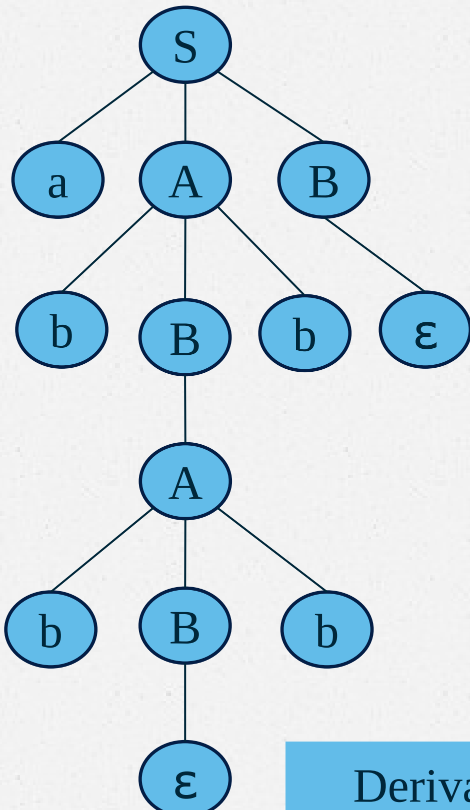
Example:

- $G: S \rightarrow aAB$

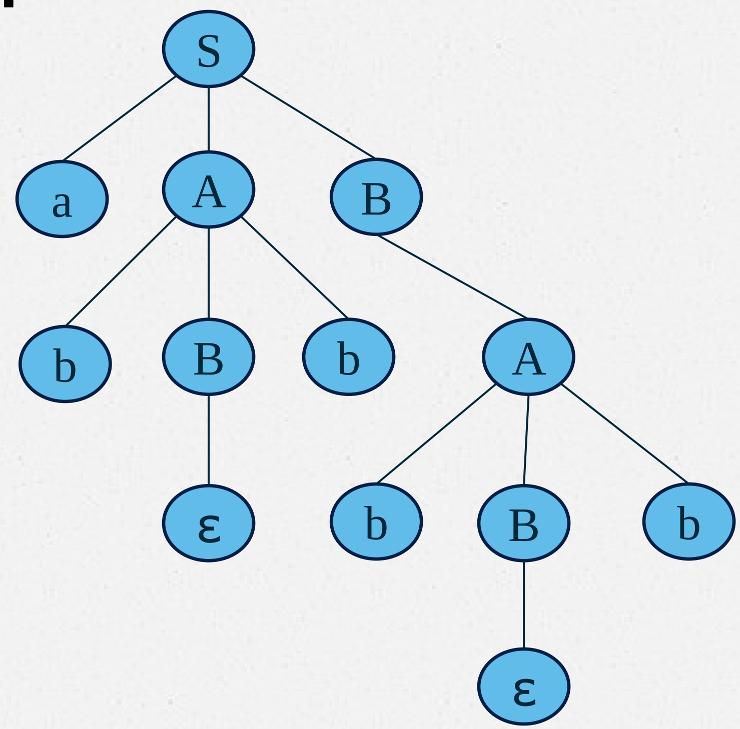
  $A \rightarrow bBb$

  $B \rightarrow A \mid \varepsilon$

Derivation Trees for the Previous Example



Derivation Tree 1



Derivation Tree 2

Rightmost and Leftmost Derivations for Tree 1

- Rightmost derivation

  $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$

- Leftmost derivation

  $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$

Rightmost and Leftmost Derivations for Tree 2

- Rightmost derivation

  $S \Rightarrow aAB \Rightarrow aAA \Rightarrow aAbBb \Rightarrow aAbb \Rightarrow abBbbb \Rightarrow abbbb$

- Leftmost derivation

  $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abbA \Rightarrow abbbBb \Rightarrow abbbb$

Inherently Ambiguous Languages

- A context-free language is ***(inherently) ambiguous*** if all context-free grammars generating it will be ambiguous.

Examples

- The following context-free languages are inherently ambiguous:
  - $L1 = \{a^i b^j c^k \mid i=j \text{ or } j=k \text{ where } i,j,k \geq 0\}$
  - $L2 = \{a^i b^j c^k d^m \mid i=j, k=m \text{ or } i=m, j=k \text{ where } i,j,k,m \geq 1\}$

Simplification of CFG

- Let $G = (V, \Sigma, R, S)$ be a CFG. Suppose R contains a rule of the form

$$A \to x_1Bx_2$$

- Assume that A and B are different variables and that

$$B \to y_1 \mid y_2 \mid \ldots \mid y_n$$

is the set of all rules in R which has B as the left side.

Let $G' = (V, \Sigma, R', S)$ be the grammar in which R' is constructed by deleting $A \to x_1Bx_2$ from R and adding to it

$$A \to x_1y_1 x_2 \mid x_1y_2 x_2 \mid \ldots \mid x_1y_n x_2.$$

- Then

$$L(G') = L(G).$$

Example

- Consider grammar

$$G: A \rightarrow a \mid aaA \mid abBc,$$
$$B \rightarrow abbA \mid b$$

- Using the substitution property, we get

$$G': A \rightarrow a \mid aaA \mid ababbAc \mid abbc,$$
$$B \rightarrow abbA \mid b$$

- The new grammar G' is equivalent to G.

Useful Variable

- Let G = (V, Σ, R, S) be a CFG. A variable $A \in V$ is said to be **useful** if and only if there is at least one $w \in L(G)$ such that

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w$$

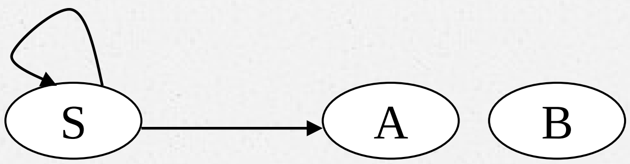
with $x, y \in (V \cup \Sigma)^*$. A variable that is not useful is called **useless**. A rule is **useless** if it involves any useless variable.

Example

G: $S \rightarrow aS \mid A \mid C$
$A \rightarrow a$
$B \rightarrow aa$
$C \rightarrow aCb$

$S \rightarrow aS \mid A$
$A \rightarrow a$
$B \rightarrow aa$

$S \rightarrow aS \mid A$
$A \rightarrow a$



Dependency Graph

Theorem

- Let $G = (V, \Sigma, R, S)$ be a CFG. Then there exists an equivalent grammar $G' = (V', \Sigma', R', S)$ that does not contain any useless variable or rules.

ε rules

- Any rule of a context-free grammar of the form

$$A \rightarrow \varepsilon$$

is called an **ε rule**. Any variable A for which the derivation

$$A \stackrel{*}{\Rightarrow} \varepsilon$$

is called **nullable.**

Example

G: $S \rightarrow ABaC$

$A \rightarrow BC$

$B \rightarrow b \mid \varepsilon$

$C \rightarrow D \mid \varepsilon$

$D \rightarrow d$

A, B, C are nullable variables above. Then we have:

$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$

$A \rightarrow B \mid C \mid BC$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

Theorem

- Let G be any context-free grammar with ε not in L(G). Then there exists an equivalent grammar G' having no ε rules.

Unit rules

- Any rule of a context-free grammar of the form

$$A \rightarrow B$$

where $A, B \in V$ is called a **unit rule**.

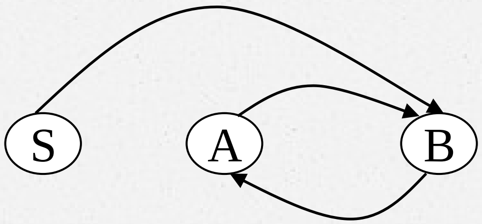
Example

G: $S \rightarrow Aa \mid B$
$B \rightarrow A \mid bb$
$A \rightarrow a \mid bc \mid B$

$S \overset{*}{\Rightarrow} A,\ S \overset{*}{\Rightarrow} B,\ B \overset{*}{\Rightarrow} A,\ A \overset{*}{\Rightarrow} B$

$S \rightarrow Aa$
$B \rightarrow bb$
$A \rightarrow a \mid bc$

$S \rightarrow a \mid bc \mid bb \mid Aa$
$B \rightarrow a \mid bb \mid bc$
$A \rightarrow a \mid bb \mid bc$



Dependency graph for unit rules

Theorem

- Let G be any context-free grammar without any ε rule. Then there exists a context-free grammar G’ that does not have any unit rules and that is equivalent to G.

Theorem

- Let L be a context-free language that does not contain ε. Then there exists a context-free grammar that generates L and does not have any useless rules, ε rules, or unit rules.
- **Solution:**
  1. Remove ε rules
  2. Remove unit rules
  3. Remove useless rules.

Chomsky Normal Form (Formal Definition)

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

$$A \to BC$$
$$A \to a$$

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

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

Example

- Step 1: (make a new start variable)

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

$$S_0 \to S$$
$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

- Step 2: Remove ε rules B→ε , shown on the left and A→ε , shown on the right

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

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

Example (cont.)

- Step 3a: Remove unit rules S→S, shown on the left and S_0→S, shown on the right

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A &\to B \mid S \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to S \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to B \mid S \\ B &\to b \end{aligned}$$

- Step 3b: Remove unit rules A→B and A→S

$$\begin{aligned} S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to B \mid S \mid \mathbf{b} \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to S \mid b \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ B &\to b \end{aligned}$$

Example (cont.)

- Step 4: convert the remaining rules into the proper form by adding variables and rules

$$
\begin{aligned}
S_0 &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
A &\rightarrow \mathtt{b} \mid AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow \mathtt{a} \\
B &\rightarrow \mathtt{b}
\end{aligned}
$$

CYK Algorithm

(J. Cooke, D.H. Younger, and T. Kasami)

- Let $G = (V, \Sigma, R, S)$ be a context-free grammar (in Chomsky normal form) and w be a string of length n

$$w = a_1\ a_2\ \dots a_n.$$

The membership problem is the problem of finding if $w \in L(G)$.

CYK Algorithm (cont.)

- Define

$$w_{ij} = a_i \ldots a_j$$

and subsets of V

$$V_{ij} = \{A \in V \mid A \overset{*}{\Rightarrow} w_{ij}\}$$

We can show

$$V_{ij} = \bigcup_{k \in \{i, i+1, \ldots, j-1\}} \{A \mid A \to BC, B \in V_{ik}, C \in V_{k+1,j}\}$$

CYK Algorithm (cont.)

- Then, the algorithm to compute the problem is as follows with the time complexity $O(n^3)$:

1. Compute $V_{11}, V_{22}, \ldots, V_{nn}$
2. Compute $V_{12}, V_{23}, \ldots, V_{n-1,n}$
3. Compute $V_{13}, V_{24}, \ldots, V_{n-2,n}$

and so on.

Example

- Determine whether the string w = aabbb is in the language generated by the grammar G below

$$S \rightarrow AB$$
$$A \rightarrow BB|a$$
$$B \rightarrow AB|b$$

- We have:
  - $V_{11} = V_{22} = \{A\}$, $V_{33} = V_{44} = V_{55} = \{B\}$,
  - $V_{12} = \emptyset$, $V_{23} = \{S, B\}$, $V_{34} = V_{45} = \{A\}$,
  - $V_{13} = = V_{35} = \{S, B\}$, $V_{24} = \{A\}$,
  - $V_{14} = \{A\}$, $V_{25} = \{S, B\}$,
  - $V_{15} = \{S, B\}$,
- so that $w \in L(G)$.

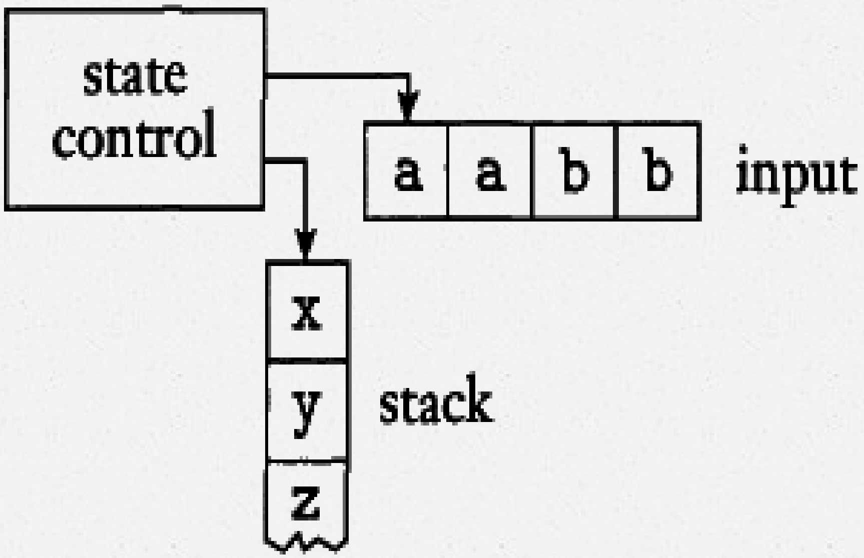
Pushdown Automata

- Like nondeterministic finite automata, with an extra component: stack
- The stack provides additional memory
- Pushdown automata are equivalent in power to context-free

Pushdown Automata (cont.)

- Push: Writing a symbol on the stack
- Pop: Removing a symbol
- Recognizes $\{0^n1^n \mid n \geq 0\}$
- As each 0 is read from the input, push it onto the stack.
- Pop a 0 off the stack for each 1 read.
- Reading the input is finished exactly when the stack becomes empty $\rightarrow$ Accept

Schematic of a pushdown automaton



Pushdown Automata (Formal Definition)

- A ***pushdown automaton*** is a 6-tuple $(Q, \sum, \Gamma, \delta, q_0, F)$, where $Q, \sum$, and Γ are finite sets, and
  1. Q is the set of states,
  2. $\sum$ is the input alphabet,
  3. Γ is the stack alphabet,
  4. $\delta : Q\times \sum_\varepsilon\times \Gamma_\varepsilon \rightarrow P(Q\times\Gamma_\varepsilon)$ is the transition function,
  5. $q_0 \in Q$ is the start state, and
  6. $F \subseteq Q$ is the set of accept states.

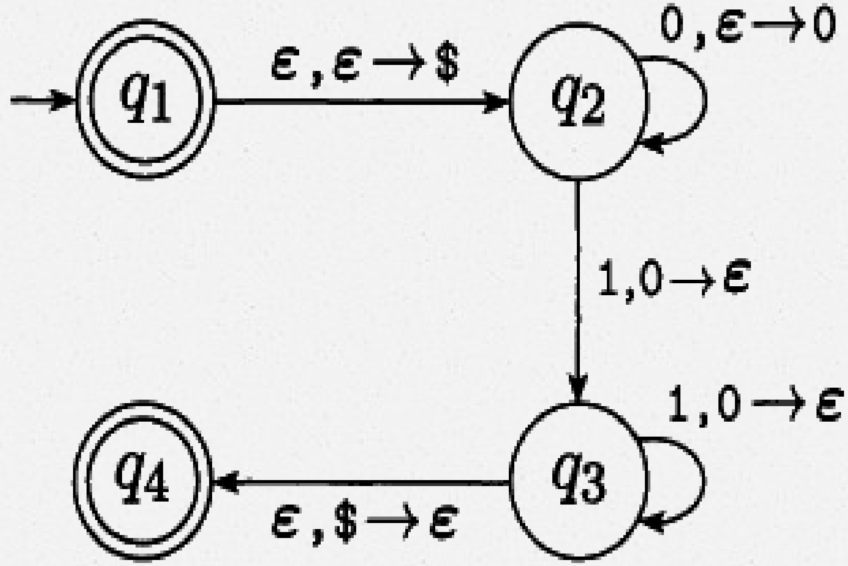
Example 1

- $L=\{0^n1^n | n\geq 0\}$
- $Q=\{q_1,q_2,q_3,q_4\}$
- $\Sigma=\{0,1\}$
- $\Gamma=\{0,\$\}$
- $F=\{q_1,q_4\}$
- δ:

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

Example 1 (cont.)

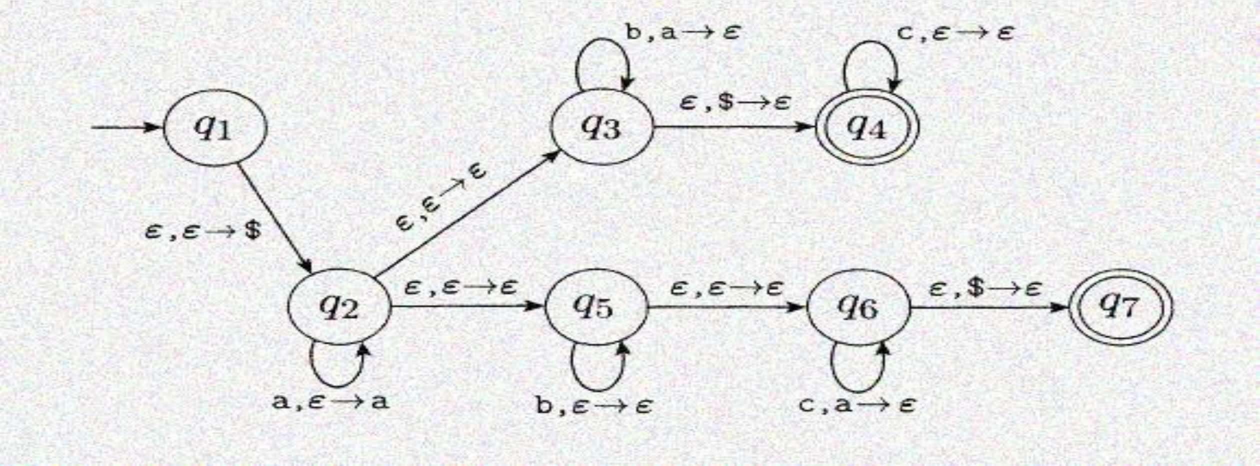
- State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$



Example 2

Give a pushdown automaton for the language:

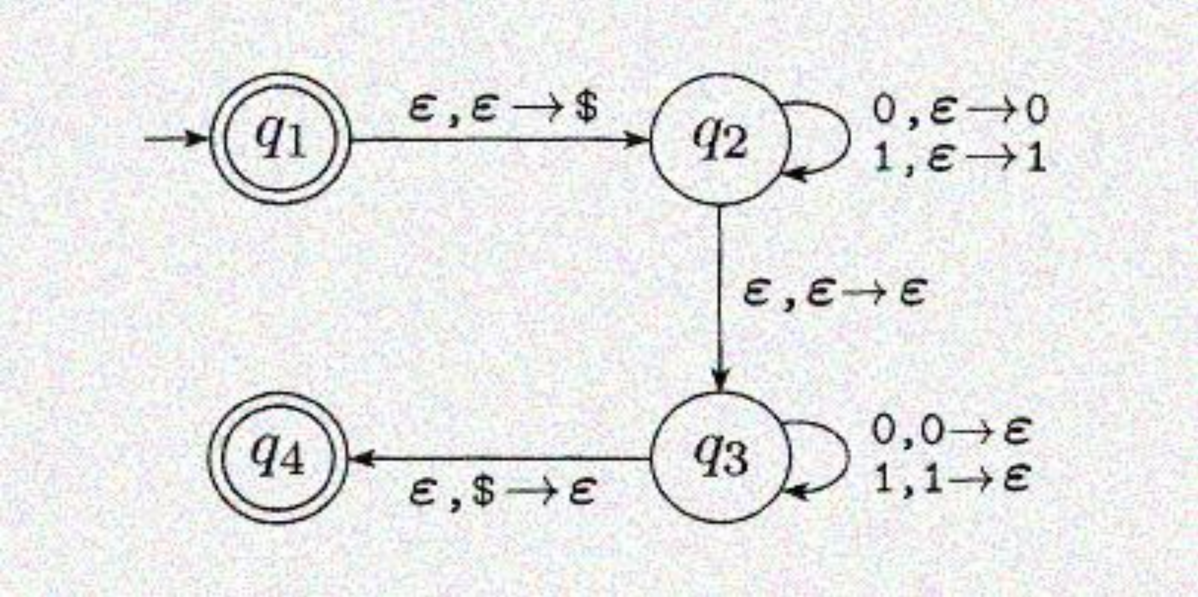
$\{a^i b^j c^k \mid i,j,k \geq 0 \text{ and } i = j \text{ or } i = k\}$



Example 3

- Give a pushdown automaton for the language:

$$\{ ww^R \mid w \in \{0, 1\}^* \}$$

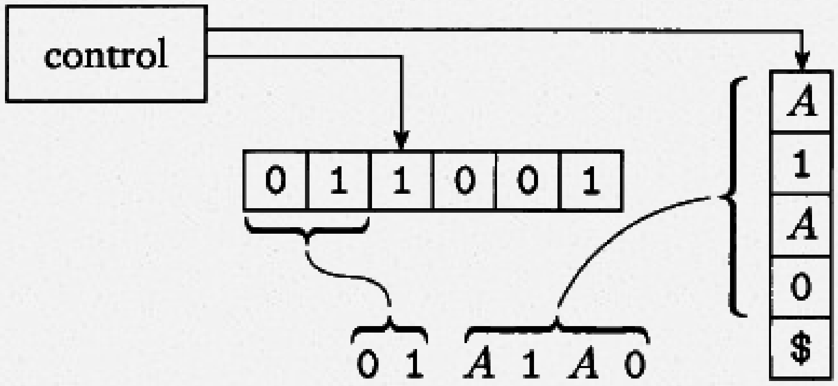


Equivalence with context-free grammars

- A language is context free if and only if some pushdown automaton recognizes it.
- Proof Idea:
  - Only if part : Let A be a CFL. Show how to convert the A's CFG, G into a PDA P that accepts all strings that G generates.
  - If part : Let P be a PDA. Show how to convert P into a CFG G that generates all strings that P accepts.

Proof of "Only If" Part

- Any terminal symbols appearing before the first variable are matched with input.



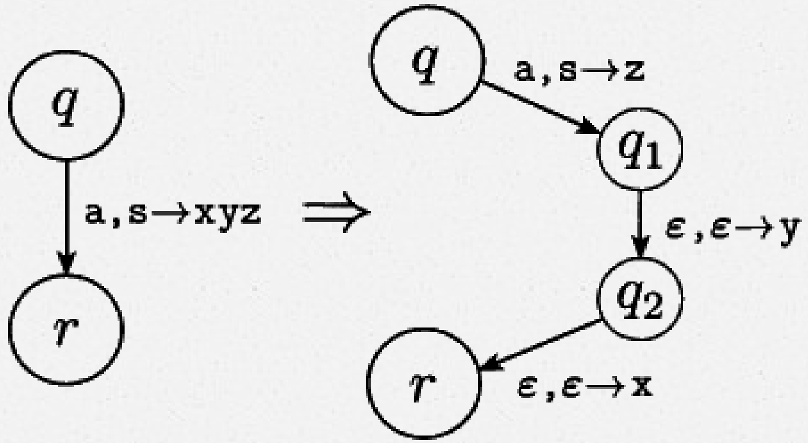
P representing the intermediate string $01A1A0$

Proof

1. Place the marker symbol $ and the start variable on the stack.
2. Repeat the following steps forever.
   - **a.** If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
   - **b.** If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
   - **c.** If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

Proof (Cont.)

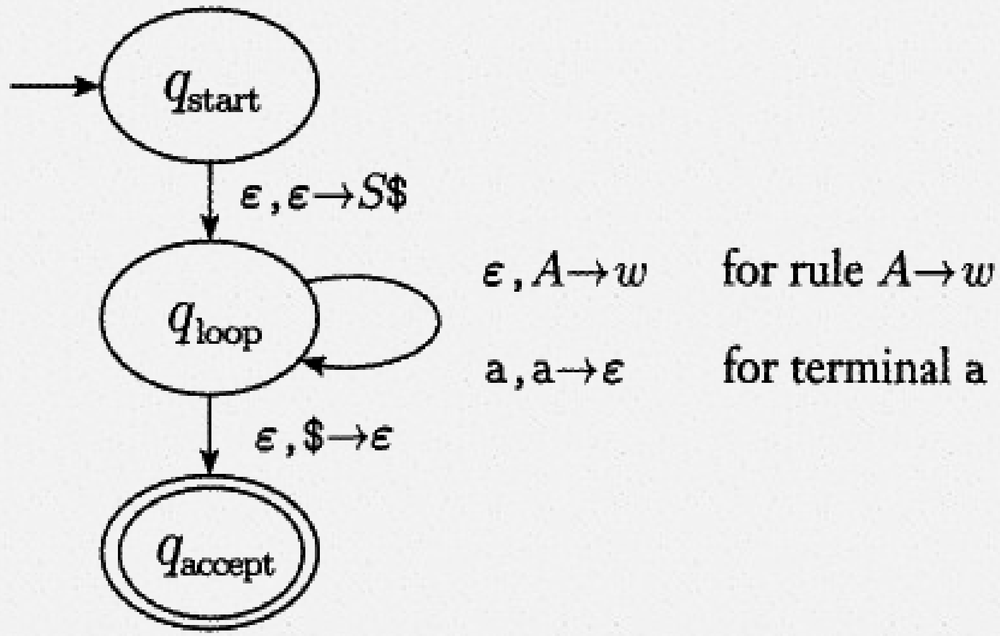
- $(r,u) \in \delta(q,a,s)$ means that when q is the state of the automaton, a is the next input symbol, and s is on the top of the stack, push the string u onto the stack and go on to the state r.



Proof (Cont.)

- $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$
  where E is the set of states needed for implementing the shorthand described before.
- Step 1: $\delta(q_{start},\varepsilon,\varepsilon)=\{(q_{loop},S\$)\}$
- Step 2:
  1. (a) $\delta(q_{loop},\varepsilon,A)=\{(q_{loop},w)|\text{where } A\rightarrow w \text{ is a rule}\}$
  2. (b) $\delta(q_{loop},a,a)=\{(q_{loop},\varepsilon)\}$
  3. (c) $\delta(q_{loop},\varepsilon,\$)=\{(q_{accept},\varepsilon)\}$

Proof (Cont.)



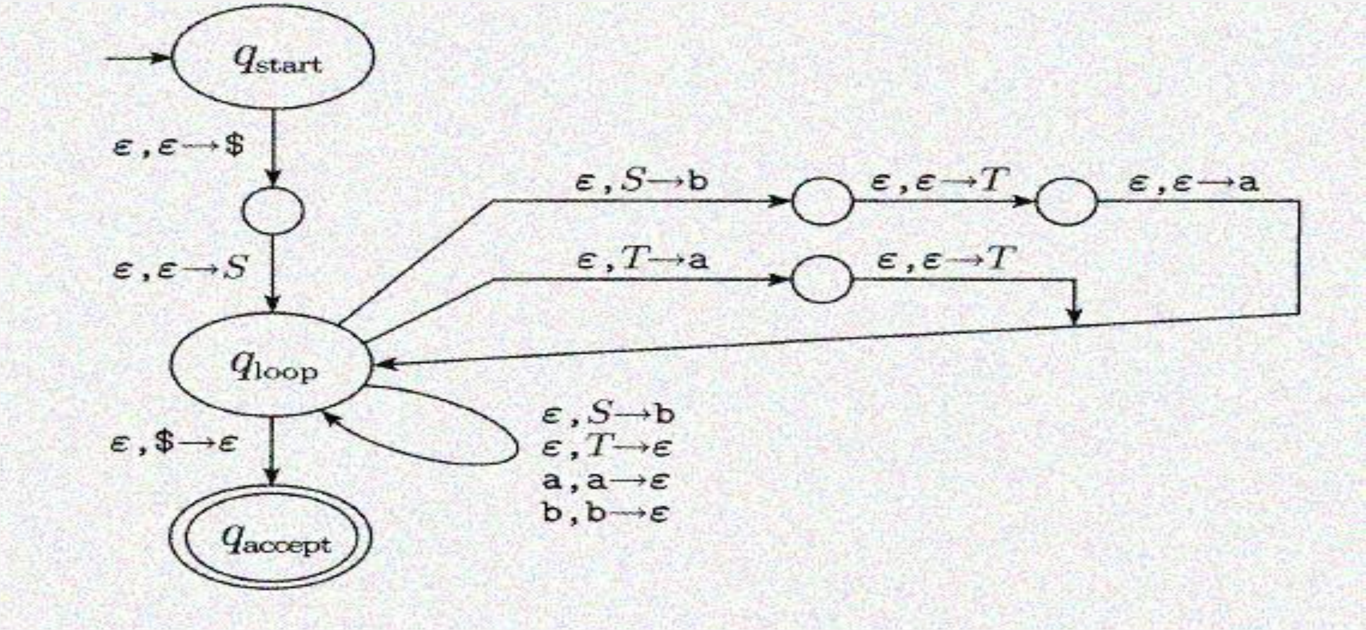
State diagram of P

Example

Construct a PDA for the CFG grammar:

S → aTb | b

T → Ta | ε

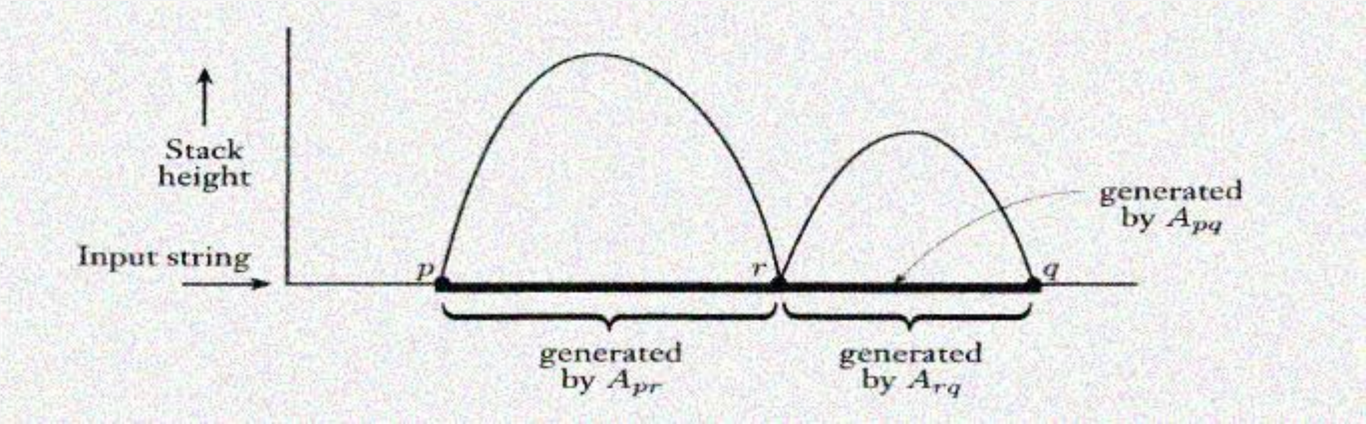


Proof of "IF" Part

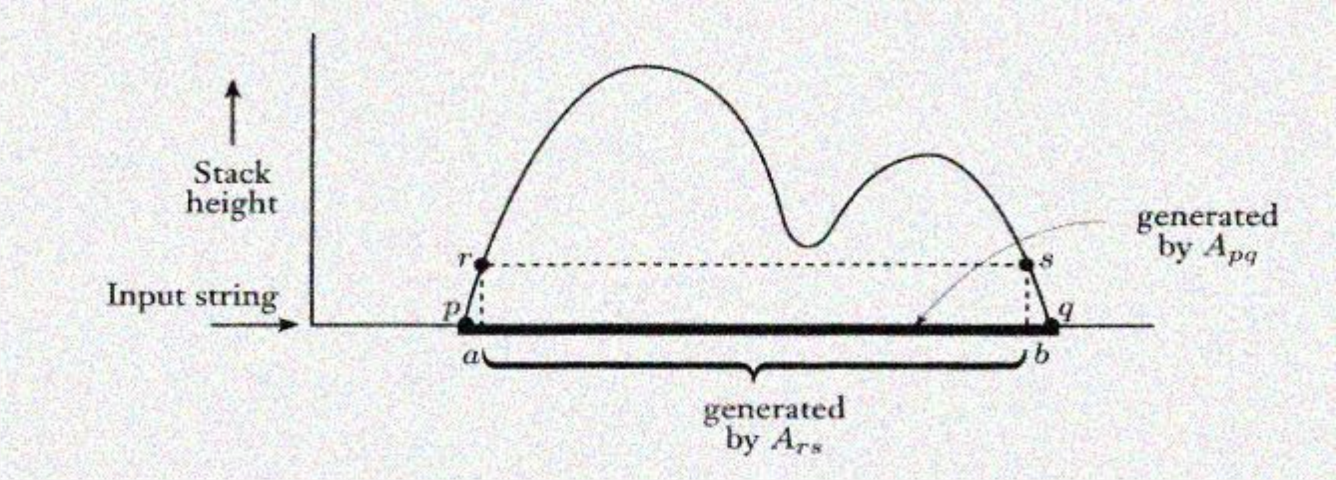
- First, we simplify our task by modifying P slightly to give it the following three features.
  1. It has a single accept state, q_{accept}.
  2. It empties its stack before accepting.
  3. Each transition either pushes a symbol onto a stack (a *push* move) or pops one off the stack (a *pop* move), but does not do both at the same time.
- Next, for each pair of states p and q in P the grammar will have a variable A_{pq}. This variable generates all strings that can take P from p with an empty stack to q with an empty stack. Observe that such strings can also take P from p to q, regardless of the stack contents at p, leaving the stack at q in the same condition as it was at p.

Proof

- Suppose $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$. We can construct CFG 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 rule $A_{pq} \rightarrow aA_{rs}b$ in G.
  - For each $p,q,r \in Q$, put 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.



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



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

Claims

- Claim 1: If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

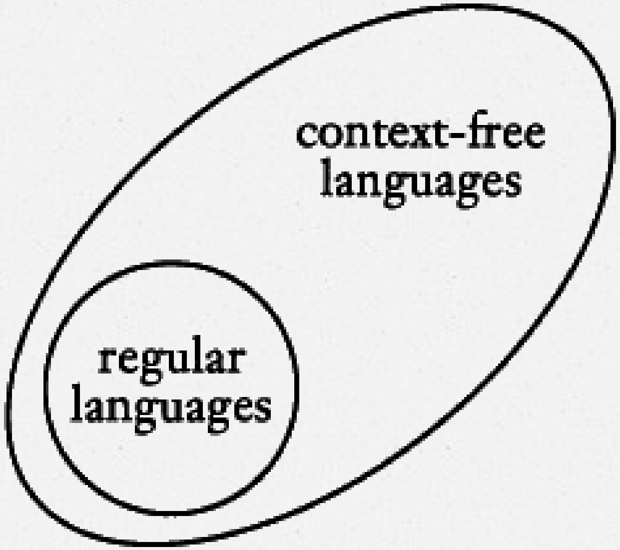
  The proof is done by induction on the number of steps in the derivation of x from A_{pq}.

- Claim 2: If x can bring P from p with empty stack to q with empty stack, A_{pq} generates x.

  The proof is done by induction on the number of steps in the computation of P that goes from p to q with empty stacks on input x.

Relationship of the regular and context-free languages

Every **regular** language is context-free.



Deterministic PDAs

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is said to be deterministic if it has the following properties:

- $\forall q \in Q, \forall A \in \Gamma_\varepsilon, \forall a \in \Sigma, \delta(q, \varepsilon, A) \neq \Phi \rightarrow \delta(q, a, A) = \Phi$
- $\forall q \in Q, \forall A \in \Gamma, \forall a \in \Sigma_\varepsilon, \delta(q, a, \varepsilon) \neq \Phi \rightarrow \delta(q, a, A) = \Phi$
- $\forall q \in Q, \forall A \in \Gamma_\varepsilon, \forall a \in \Sigma, \delta(q, \varepsilon, \varepsilon) \neq \Phi \rightarrow \delta(q, a, A) = \Phi$
- $\forall q \in Q, \forall A \in \Gamma_\varepsilon, \forall a \in \Sigma_\varepsilon, |\delta(q, a, A)| \leq 1$

Deterministic PDAs

DEFINITION 2.39

A ***deterministic pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The transition function δ must satisfy the following condition. For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$, exactly one of the values

$$\delta(q, a, x), \delta(q, a, \varepsilon), \delta(q, \varepsilon, x), \text{ and } \delta(q, \varepsilon, \varepsilon)$$

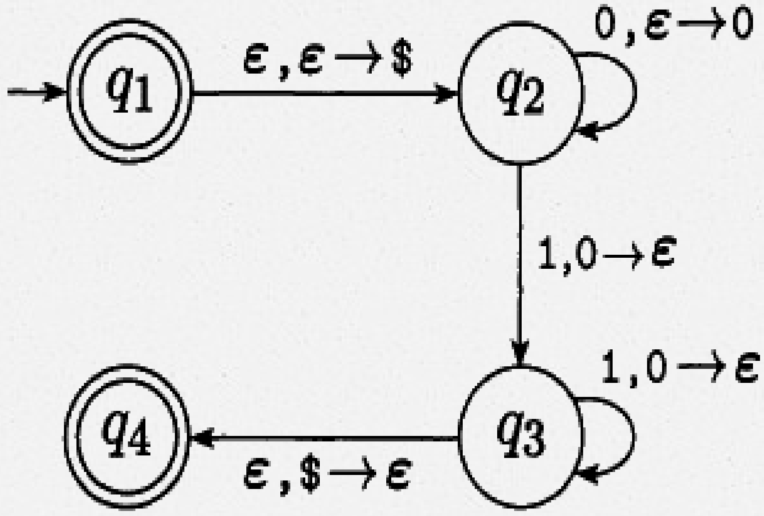
is not $\emptyset$.

Deterministic Context-free Languages

- **Definition:** A context-free language is said to be <u>deterministic</u> if it can be accepted by some deterministic pushdown automaton.

Example 1

- $L = \{a^n b^n : n \geq 0\}$ is a deterministic language because it can be accepted by the following deterministic automaton:



Example 2

- It can be shown that the language

$$L_1 = \{a^n b^n : n \geq 0\} \cup \{a^n b^{2n} : n \geq 0\}$$

is **non**deterministic.

NON-CONTEXT-FREE LANGUAGES

- Pumping Lemma for context-free languages:
  - If A is a context-free language, then there is **a** number *p* (the pumping length) where, if s is **any** string in A of length at least *p*, then s may be divided into five pieces s=uvxyz satisfying the conditions:
    - For each $i \geq 0$, $uv^ixy^iz \in A$
    - $|vy|>0$
    - $|vxy| \leq p$

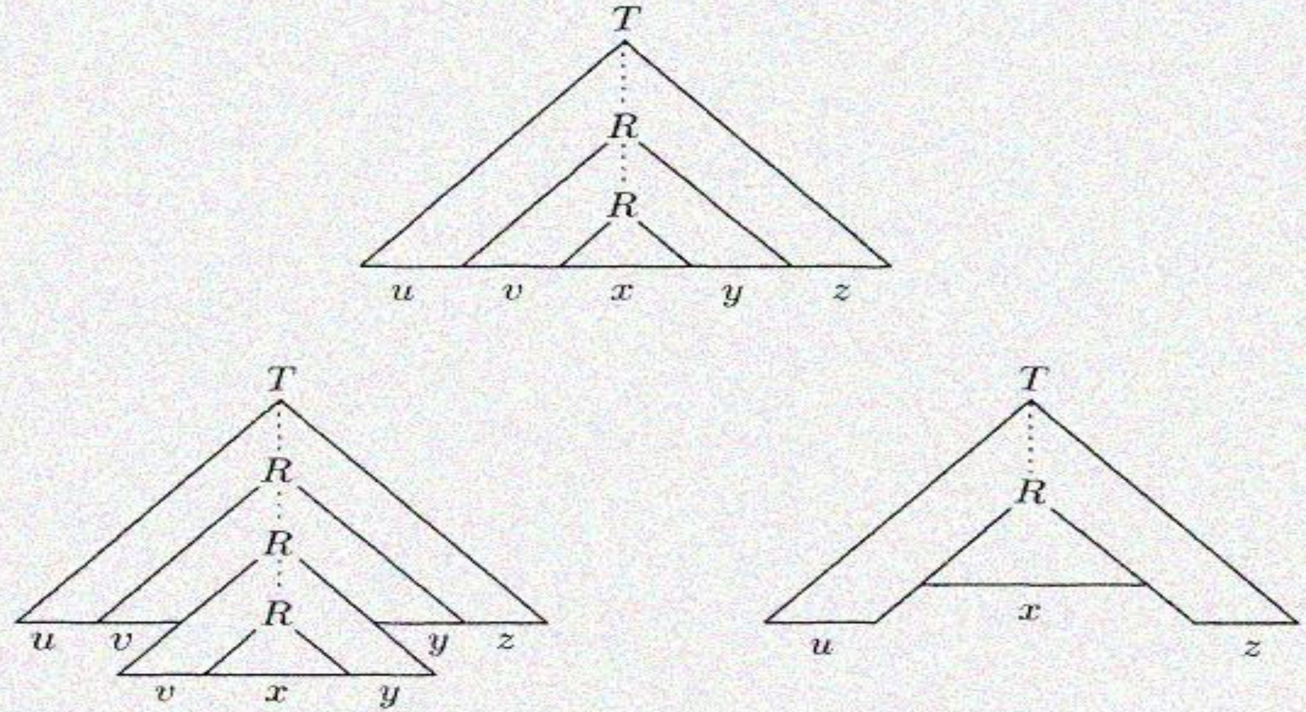
Proof

- Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule (assume **at least 2**).

- Then, if the height of the parse tree is at most h, the length of the string generated is at most b^h. Conversely, if a generated string is at least b^h+1 long, each of its parse tree must be at least h+1 high.

- Let |V| be the number of variables in G. Then, we set p to be $b^{|V|+1}$. Now if s is a string in A and its length is p or more, any of its parse trees must be at least |V| +1 high, because $b^{|V|+1} \geq b^{|v|}+1$. We choose the smallest of such parse trees.

Proof (Cont.)



Surgery on parse trees

Example 1

- $B=\{a^n b^n c^n | n \geq 0\}$

Assume that B is a CFL and p is the pumping length.

- $s=a^p b^p c^p$: a member of B and of length at least p

We show that it can not be pumped

Example(Cont.)

Consider: s=uvxyz

First Condition:

- v and y contain one type pf symbol SO

  uv^2xy^2z cannot contain equal number of a's, b's and c's.
- Either v or y contain more than one type of symbol SO uv^2xy^2z cannot contain a's, b's and c's in the correct order.

Example 2

- Let D = {ww | w {0,1}*}
- Assume that D is a CFL and p is the pumping length.
- s= 0^p10^p1: a member of D and of length at least p;

It will not work!

$$\overbrace{\underbrace{000\cdots000}_{u}\;\underbrace{0}_{v}\;\underbrace{1}_{x}}^{0^p1}\;\overbrace{\underbrace{0}_{y}\;\underbrace{000\cdots0001}_{z}}^{0^p1}$$

Example 2

- Let us choose $s = 0^p1^p0^p1^p$

It will work!

We show that it cannot be pumped

Note that

$$S=uvxyz = 0^p1^p0^p1^p$$

End of chapter 2 ☺