

### Context Free Grammars

Context Free Grammars (CFGs) can be recognized by PushDown Stack Machines known as PushDown Automata (PDAs). Regular Grammars are a subset of CFGs.

Slide Lecture 8 –215

A Context Free Grammar (CFG) is defined as a 4-tuple

$G = (V, \Sigma, S, R)$  where:

$V$  and  $\Sigma$  are finite sets with  $V \cap \Sigma = \emptyset$ .

1.  $V$  is the set of *nonterminal symbols* representing *variables* of the *stack alphabet*.
2.  $\Sigma$  is the set of *terminals* representing the *input alphabets*.
3.  $S \in V$  is the *start* or *sentence* symbol
4.  $R$  is a finite set of *productions* (aka the rewrite rules or *grammar rules*) of the form  $A \rightarrow \alpha$  where  $A \in V$  and  $\alpha \in \{V \cup \Sigma\}$

NOTE: CFG have a non-contracting form.

Slide Lecture 8 –216

A Context Free Grammar for

$$L = \{0^n 1^n | n \geq 0\}$$

$$V = \{S, A\}$$

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

$$R = S \rightarrow \lambda$$

$$S \rightarrow A$$

$$A \rightarrow 0A1$$

$$A \rightarrow 01$$

Slide Lecture 8 -217

or it could be written

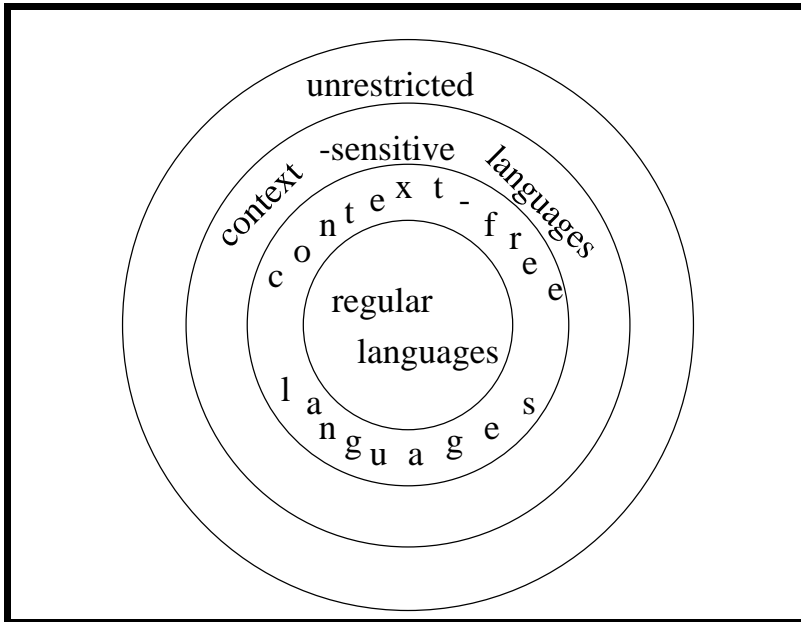
$$R = S \rightarrow \lambda$$

$$S \rightarrow 0S1$$

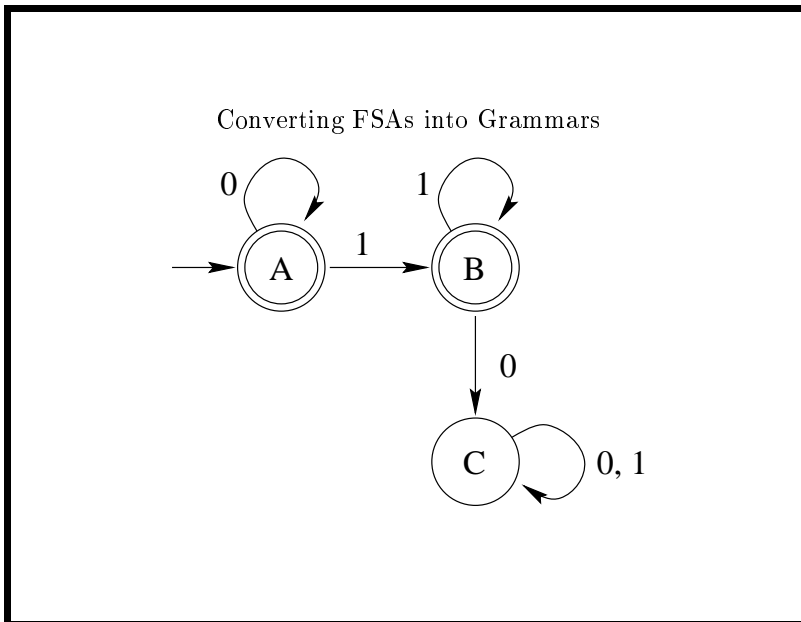
Clearly, Context Free Grammars can generate things not  
recognized by a Finite State Machine.

But Regular Grammars are also Context Free Grammars.

Slide Lecture 8 -218



Slide Lecture 8 -219



Slide Lecture 8 -220

Consider the Set Equations:

$$A = 0A + 1B + \lambda$$

$$B = 1B + 0C + \lambda$$

$$C = 0C + 1C$$

Which implies

$$(A = 0A) + (A = 1B) + (A = \lambda)$$

**Slide Lecture 8 -221**

We can rewrite the set equations as productions.

Let  $S$  be the “start” symbol

$S \rightarrow A$ , the initial state

$$S \rightarrow A$$

$$A \rightarrow 0A$$

$$A \rightarrow 1B$$

$$A \rightarrow \lambda$$

$$B \rightarrow 1B$$

$$B \rightarrow 0C$$

$$B \rightarrow \lambda$$

$$C \rightarrow 0C$$

$$C \rightarrow 1C$$

Note that the productions for “ $C$ ” involve infinite recursion.

Why? ... What do we do?

**Slide Lecture 8 -222**

“C” is a trap state.

$$S \longrightarrow A$$

$$A \longrightarrow \lambda \mid 0A \mid 1B$$

$$B \longrightarrow 1B \mid \lambda$$

Since the machine was deterministic, so is the grammar.

Does this language include the string 000111?

**Slide Lecture 8 –223**

Yes. This also means that the corresponding grammar generates this string:

$$S \longrightarrow A$$

$$A \longrightarrow 0A$$

$$0A \longrightarrow 00A$$

$$00A \longrightarrow 000A$$

$$000A \longrightarrow 0001B$$

$$0001B \longrightarrow 00011B$$

$$00011B \longrightarrow 000111$$

**Slide Lecture 8 –224**

Note that all productions are **right linear**,

$$A \rightarrow 0A$$

$$A \rightarrow 1B$$

FSA's have regular grammars of the following form

$$S \rightarrow \lambda$$

$$S \rightarrow A$$

$$A \rightarrow aB$$

$$A \rightarrow a$$

where  $S$  is the start or "sentence symbol"

and  $A$  and  $B$  are "nonterminals" (that correspond to States)

and  $a$  is a "terminal" ( $a \in \Sigma$ ) +  $\lambda$

Slide Lecture 8 -225

What about  $A \rightarrow 101B$  ?

$$A \rightarrow 1Z$$

$$Z \rightarrow 0Y$$

$$Y \rightarrow 1B$$

Slide Lecture 8 -226

For each right linear grammar there is a corresponding left linear grammar (which generates right to left).

$$S \rightarrow 1B \quad S \rightarrow B1$$
$$S \rightarrow 1 \quad S \rightarrow 1$$
$$A \rightarrow 1B \quad A \rightarrow B1$$
$$A \rightarrow 1 \quad A \rightarrow 1$$
$$B \rightarrow 0A \quad B \rightarrow A0$$
$$1(01)^* \quad (10)^*1$$

Note that these left linear and right linear grammars are not the same, but recognized equivalent languages.

**Slide Lecture 8 -227**

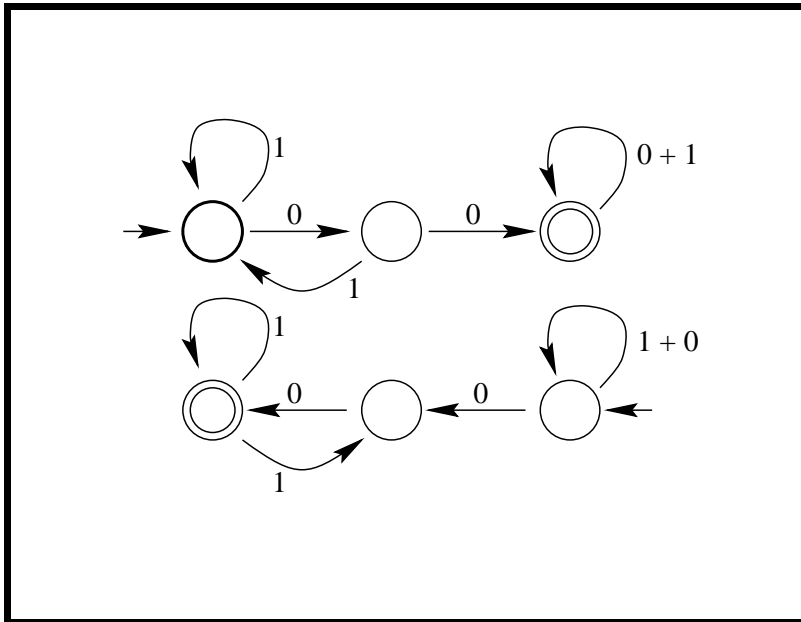
A left linear language is like running the finite state machine backwards while scanning right to left.

Initial States are Final States

Final States are Initial States

Then just change the direction on the transition arcs.

**Slide Lecture 8 -228**



Slide Lecture 8 -229

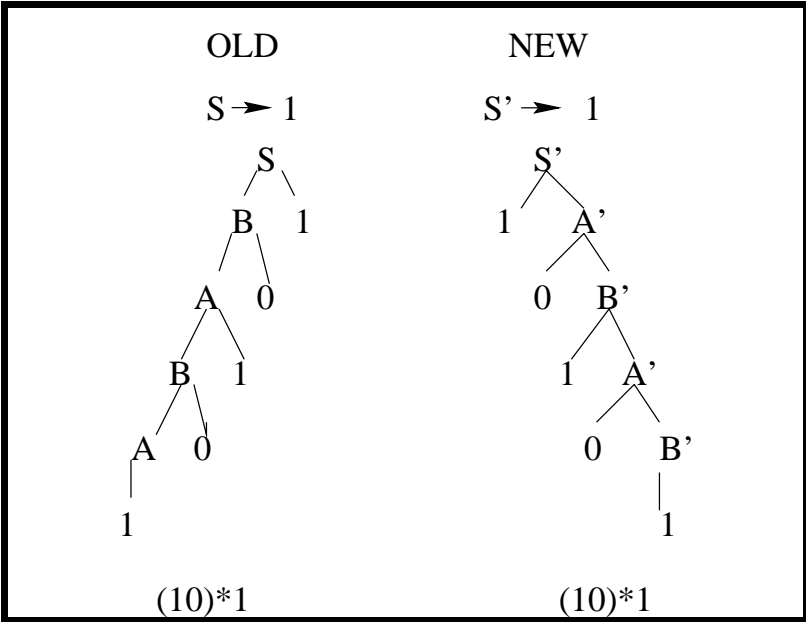
To transform from left linear to right linear:

1.  $A \rightarrow \alpha$   
becomes  $A \rightarrow \lambda \alpha$
2.  $S$  becomes  $\lambda$   
 $\lambda$  becomes  $S$
3.  $A \rightarrow B\alpha$  becomes  $B' \rightarrow \alpha A'$
4. Drop  $\lambda'$

Slide Lecture 8 -230

$S \rightarrow B1$	$S \rightarrow B1$
$S \rightarrow 1$	$S \rightarrow \lambda 1$
$A \rightarrow B1$	$A \rightarrow B1$
$A \rightarrow 1$	$A \rightarrow \lambda 1$
$B \rightarrow A0$	$B \rightarrow A0$
$\lambda \rightarrow B1$	$B' \rightarrow 1\lambda'$
$\lambda \rightarrow S1$	$S' \rightarrow 1\lambda'$
$A \rightarrow B1$	$B' \rightarrow 1A'$
$A \rightarrow S1$	$S' \rightarrow 1A'$
$B \rightarrow A0$	$A' \rightarrow 0B'$

Slide Lecture 8 -231



Slide Lecture 8 -232

Example:

$$\Sigma = \{a, b\}$$

$$S \rightarrow Sb$$

$$S \rightarrow a$$

defines  $S$  so as to generate  $ab^*$

Slide Lecture 8 -233

$$S \rightarrow a = a$$

$$S \rightarrow Sb \quad \text{yields } Sb$$

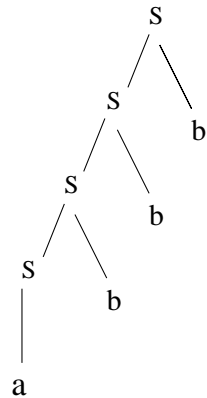
$$S \rightarrow Sb \quad \text{yields } Sbb$$

$$S \rightarrow Sb \quad \text{yields } Sbbb$$

$$S \rightarrow a \quad \text{yields } abbb$$

Slide Lecture 8 -234

Parse tree for abbb



Slide Lecture 8 -235

We define two relations

$$A \xrightarrow{G} B$$

$$A \xrightarrow{G^*} B$$

where \* indicates 0 or more applications of generator G.

If  $A \rightarrow B$  is a production,  
and  $\alpha$  and  $\omega$  are any strings in  $V \cup \Sigma$   
and  $A \in V$  and  $B \in V \cup \Sigma$   
then

$$\alpha A \omega \xrightarrow{G} \alpha B \omega$$

that is,  $\alpha A \omega$  directly (or immediately) derives  $\alpha B \omega$ .

Slide Lecture 8 -236

If  $\alpha_i \in (V \cup \Sigma)^*$  and  $\alpha_1 \xrightarrow{G} \alpha_2 \xrightarrow{G} \alpha_3 \xrightarrow{G} \dots \xrightarrow{G} \alpha_n$  then  
 $\alpha_1 \xrightarrow{G^*} \alpha_n$

$\xrightarrow{G^*}$  is the reflexive and transitive closure of  $\xrightarrow{G}$

The language generated by G is defined as

$$\{\omega \in \Sigma^* \mid S \xrightarrow{G^*} \omega\}$$

We may write  $\Rightarrow$  for  $\xrightarrow{G}$  when no confusion arises.

Any sequence  $\omega_0 \Rightarrow \omega_1 \Rightarrow \dots \Rightarrow \omega_n$   
 is a derivation of  $\omega_n$  from  $\omega_0$  in G.

A language is context free IFF there exists a context free  
 grammar that generates L.

Slide Lecture 8 -237

### Grammars and Natural Languages

P ... noun phrase

A ... adjective

N ... noun

V ... verb

S ... sentence

S  $\rightarrow$  PVN      V  $\rightarrow$  eat

P  $\rightarrow$  AN      A  $\rightarrow$  big

P  $\rightarrow$  AAN      A  $\rightarrow$  gray

P  $\rightarrow$  N      N  $\rightarrow$  peanuts

N  $\rightarrow$  elephants

Slide Lecture 8 -238

S  $\rightarrow$  PVN  $\rightarrow$  PV peanuts  $\rightarrow$  P eat peanuts  $\rightarrow$   
 AN eat peanuts  $\rightarrow$  A elephants eat peanuts  
 $\rightarrow$  gray elephants eat peanuts

S  $\rightarrow$  PVN  $\rightarrow$  PV elephants  $\rightarrow$  P eat elephants  $\rightarrow$   
 AAN eat peanuts  $\rightarrow$  A A peanuts eat elephants  
 $\rightarrow$  gray A peanuts eat elephants  
 $\rightarrow$  gray big peanuts eat elephants

In artificial intelligence natural languages are discussed; we are interested in formal languages that are well defined and useful for programming.

Slide Lecture 8 -239

Example:

$G = (V, \Sigma, R, S)$

$V = (T, S, F)$

$\Sigma = \{X, 1, 2, +, *, (, )\}$

generate  $(x1 * x2 + x1) * (x1 + x2)$  using the following

R= {S  $\rightarrow$  S + T R1  
 S  $\rightarrow$  T R2  
 T  $\rightarrow$  T \* F R3  
 T  $\rightarrow$  F R4  
 F  $\rightarrow$  (S) R5  
 F  $\rightarrow$  X1 R6  
 F  $\rightarrow$  X2} R7

Slide Lecture 8 -240

S	→	T	R2
T	→	T * F	R3
	→	T * (S)	R5
	→	T * (S + T)	R1
	→	T * (S + F)	R4
	→	T * (S + X2)	R7
	→	T * (T + X2)	R2

Slide Lecture 8 -241

→	T * (F + X2)	R4
→	T * (X1 + X2)	R6
→	F * (X1 + X2)	R4
→	(S) * (X1 + X2)	R5
→	(S + T) * (X1 + X2)	R1
→	(S + F) * (X1 + X2)	R4

Slide Lecture 8 -242

$\longrightarrow (S + X1) * (X1 + X2) \quad R6$   
 $\longrightarrow (T + X1) * (X1 + X2) \quad R2$   
 $\longrightarrow (T * F + X1) * (X1 + X2) \quad R3$   
 $\longrightarrow (T * X2 + X1) * (X1 + X2) \quad R7$   
 $\longrightarrow (F * X2 + X1) * (X1 + X2) \quad R4$   
 $\longrightarrow (X1 * X2 + X1) * (X1 + X2) \quad R6$

Similar to logic for propositional calculus. Notice forward and backward chaining.

Slide Lecture 8 -243

**Theorem:** If  $L_1$  and  $L_2$  are Context Free, then the languages  $L_1 \cup L_2$ ,  $L_1 L_2$  and  $L_1^*$  are also Context Free.

Proof by construction.

Let  $G_1$  for  $L_1$  and  $G_2$  for  $L_2$  be defined as

$$G_1 = (V_1, \Sigma_1, S_1, P_1)$$

$$G_2 = (V_2, \Sigma_2, S_2, P_2)$$

We assume  $V_1$  and  $V_2$  are named (or renamed) such that

$$V_1 \cup V_2 = \emptyset$$

Slide Lecture 8 -244

1. We construct the context free grammar:

$$G_u = (V_u, \Sigma_u, S_u, P_u)$$

generating  $(L_1 \cup L_2)$  as follows.

$$V_u = V_1 \cup V_2 \cup S_u$$

$$\Sigma_u = \Sigma_1 \cup \Sigma_2$$

$$P_u = P_1 \cup P_2 \cup \{S_u \rightarrow S_1 \mid S_2\}$$

Forall  $x \in L_u$ ,

the derivation  $S_u \Rightarrow x$  must begin

$$S_u \Rightarrow S_1 \xrightarrow{G^*} x \text{ or } S_u \Rightarrow S_2 \xrightarrow{G^*} x$$

and thus it follows  $x \in L_1$  or  $x \in L_2$ .

**Slide Lecture 8 -245**

2. We construct the context free grammar:

$$G_c = (V_c, \Sigma_c, S_c, P_c)$$

generating  $(L_1 L_2)$  as follows.

$$V_c = V_1 \cup V_2 \cup S_c$$

$$\Sigma_c = \Sigma_1 \cup \Sigma_2$$

$$P_c = P_1 \cup P_2 \cup \{S_c \rightarrow S_1 S_2\}$$

Forall  $x \in L_c$ ,

there Exists  $y$  and  $z$  such that  $x = yz$ .

$$S_c \xrightarrow{G^*} yz \text{ must begin}$$

$$S_c \Rightarrow S_1 S_2 \xrightarrow{G^*} yz$$

where  $y \in L_1$  and  $z \in L_2$ .

**Slide Lecture 8 -246**

3. We construct the context free grammar:

$$G_k = (V_k, \Sigma_k, S_k, P_k)$$

generating  $(L_1)^*$  as follows.

$$V_k = V_1 \cup S_k$$

$$\Sigma_k = \Sigma_1$$

$$P_k = P_1 \cup \{S_k \longrightarrow S_1 S_k | \lambda\}$$

Forall  $x \in L_k$ , there Exists a derivation

$$x = \lambda \text{ or } x = (S_1)^i.$$