

Context-free Grammars $\hat{=}$ Languages

Def. A grammar $G = (V, T, S, P)$ is context-free (Type 2) if each rule is of the form:

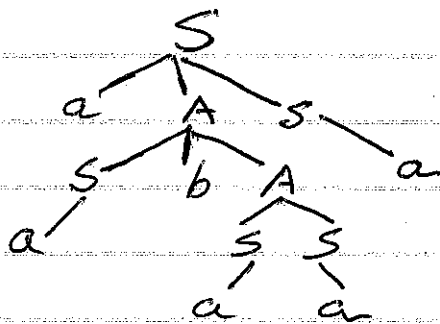
$$A \rightarrow \alpha \quad A \in V, \alpha \in \{V \cup T\}^*$$

Def. A language is a context-free language (cfl)
 $\Leftrightarrow \exists$ a cfg G st $L = L(G)$.

Example 1

$$\begin{aligned} S &\rightarrow aAS \mid a \\ A &\rightarrow sBA \mid SS \mid ba \end{aligned}$$

We associate a derivation tree with any (terminal) string w st. $S \Rightarrow^* w$



$$\begin{aligned} S &\Rightarrow aAS \Rightarrow aSbAS \\ &\Rightarrow aSbAa \Rightarrow aSbSSa \\ &\Rightarrow aabSSa \Rightarrow aabaaSa \\ &\Rightarrow aabaaa \end{aligned}$$

What is $L(G)$?

Def. If $S \Rightarrow^* \alpha$ for any $\alpha \in \{T \cup V\}^*$ then α is said to be a sentential form.

Example 2

$$\begin{aligned} S &\rightarrow AS_1 \mid S_1B \\ S_1 &\rightarrow aS_1b \mid \lambda \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

Roadmap

- (1) Derivation trees and context-free grammars (cfg)
- (2) cfg and normal forms
- (3) define a new class of machines: pushdown automata
- (4) relate pda's and cfg's.
- (5) Explore properties & decision algorithms for cfl's.

Example

$S \rightarrow aB$	$A \rightarrow bAA$	Claim: $L(G)$ is all words over $\{a, b\}$ that have an equal number of a's & b's. (excluding λ)
$S \rightarrow bA$	$B \rightarrow b$	
$A \rightarrow a$	$B \rightarrow bS$	
$A \rightarrow aS$	$B \rightarrow aBB$	

Proof (by induction on the size of $w \in L(G)$)

Induction hypothesis:

- ① $S \Rightarrow^* w$ iff w has an equal # of a's & b's.
- ② $A \Rightarrow^* w$ iff w has one more a's than b's.
- ③ $B \Rightarrow w$ iff w has one more b's than a's.

$|w| = 1$ Induction hypothesis can be checked (Basis)

Assume true for $|w| \leq k-1$. Show true for $|w| = k$.

- ① Suppose $S \Rightarrow^* w$, $|w| = k$. Then, the first derivation step must be $S \rightarrow aB$ or $S \rightarrow bA$.

Suppose it is $S \rightarrow aB$. Then $w = aw_1$ where $B \Rightarrow^* w_1$.

This last statement is using the induction principle for c.f.g.'s and is independent of the current proof.

By induction w_1 has one more b's than a's, since $|w_1| = k-1$. $\therefore w$ has an equal # a's and b's.

Similarly if $S \rightarrow bA$ in the first step.

① cont
(other
direction)

Assume $|w| = k$ and w has an equal number a's & b's.
Suppose $w = aw_1$. By induction $B \Rightarrow^* w_1$. Thus,
 $S \Rightarrow aB \Rightarrow^* aw_1 = w$. $\therefore S \Rightarrow^* w$. Similarly, if $w = bw_2$

② Suppose $A \Rightarrow^* w$, $|w| = k > 1$. Then the first derivation step must be $A \rightarrow aS$ or $A \rightarrow bAA$.

In the first case, $S \Rightarrow^* w_1$ with w_1 having equal a's & b's.

In the second case, on r.h.s. $A \Rightarrow^* w_1$ & $A \Rightarrow^* w_2$ with w_1 & w_2 having 1 more a's than b's. Thus the l.h.s. A has $A \Rightarrow^* bw_1w_2$ with 1 more a's than b's.

② cont
(opposite
direction)

Assume w has 1 more a's than b's, $|w| = k$. Let $w = aw_1$. By induction $S \Rightarrow^* w_1$. $\therefore A \Rightarrow aS \Rightarrow^* aw_1 = w$.
Let $w = bw_2$. Now, w_2 has two more a's than b's, and can be written as $w_2 = w_3w_4$ with w_3 having one more a's than b's and w_4 having 1 more a's than b's. (why is this so?) By induction

$A \Rightarrow^* w_3$ and $A \Rightarrow w_4$
 $\therefore A \Rightarrow bAA \Rightarrow^* bw_3w_4 = w$.

③ Proof of this is similar to proof on ②.

Leftmost and Rightmost Derivations

Definition

A derivation is said to be leftmost if at each step of the derivation, the variable replaced has no variables to its left in the sentential form from which the replacement is made.

(similarly for rightmost).

Example

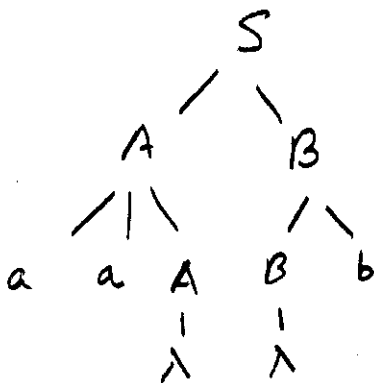
$$S \rightarrow AB \quad A \rightarrow aaA \mid \lambda \quad B \rightarrow Bb \mid \lambda$$

$$\text{Note: } L(G) = \{ a^{2n} b^m : n \geq 0, m \geq 0 \}$$

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab \quad (\text{leftmost})$$

$$S \Rightarrow AB \Rightarrow ABb \Rightarrow aaABb \Rightarrow aaAb \Rightarrow aab \quad (\text{neither})$$

Trees



trees are the same.

Lemma (Induction principle for c.f.g.s)

Let G be a c.f.g. and let $\alpha, \beta \in (VUT)^*$.
 If $\alpha \xRightarrow{r} \beta$ for $r \geq 0$, and if $\alpha = \alpha_1 \dots \alpha_n$ for $n \geq 1$,
 then $\exists t_i \geq 0, \beta_i \in (VUT)^*$ for $1 \leq i \leq n$ st.
 $\beta = \beta_1 \dots \beta_n$ and $\alpha_i \xRightarrow{t_i} \beta_i$ with $\sum_{i=1}^n t_i = r$.

Proof by induction on r , the length of the derivation.

Idea of Lemma: $\alpha_1 \dots \alpha_n \xRightarrow{\quad} \begin{matrix} B \\ \underline{b_1 b_2 b_3 \dots b_m} \\ \beta_1 \beta_2 \dots \beta_n \end{matrix}$

Can partition the derived string so each part is derived from a corresponding α_i .

Theorem

Given a c.f.g. G , if $S \xRightarrow{*} w$, then there is a leftmost derivation of w in G .

Proof by induction on # steps in the derivation. ($A \xRightarrow{*} w$)

True for step of length 1 ($A \rightarrow w$, is trivially leftmost).

Assume true for all derivations of length $\leq k$.

Let $A \xRightarrow{k+1} w$. \therefore

$A \Rightarrow X_1 \dots X_n \xRightarrow{k} w$ by induction principle of c.f.g.s.
 with $X_i \xRightarrow{t_i} \beta_i$ with $\sum t_i = k$ and $X_i \xRightarrow{*} \beta_i$.

But each X_i is either a terminal or a nonterminal,

\therefore by induction $X_i \Rightarrow \beta_i$ by a leftmost derivation

\therefore First use leftmost derivation for β_1 , then β_2, \dots then β_n .

This gives a leftmost derivation for w .

Derivation Trees

G :

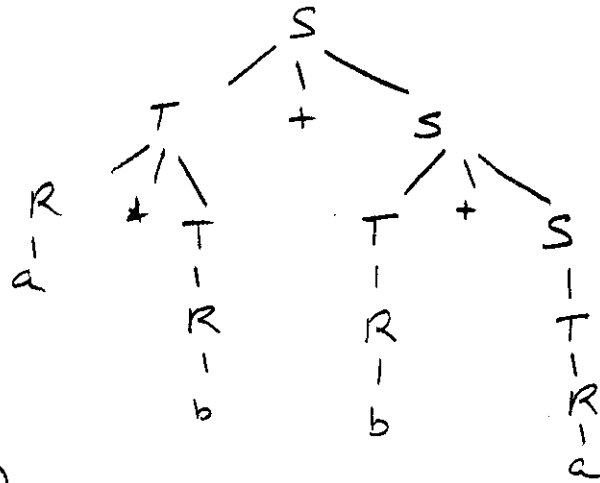
$$S \rightarrow T + S$$

$$S \rightarrow T$$

$$T \rightarrow R * T$$

$$T \rightarrow R$$

$$R \rightarrow a \mid b$$



$$a * b + b + a \in L(G)$$

For a grammar G ,

A derivation tree is an ordered tree in which root has label S , leaves are labelled from $T \cup \Sigma$, interior nodes are labelled from nonterminals V . and if an internal node is labelled A with children a_1, \dots, a_n , then \exists a production $A \rightarrow a_1, \dots, a_n$ in the grammar.

Furthermore a leaf labelled λ has no siblings.

Note: a partial derivation tree has a label from $V \cup T \cup \Sigma$.
ie. don't need to go to all terminals.

Note: reading leaves gives a sentence form. α .

Note: if $S \Rightarrow^* \alpha$, then α is the yield of the tree.

Note: if some other nonterminal A is used, it also has a subtree that is the derivation tree viewing A as root.

Thm

Let G be a c.f.g. Then for every $w \in L(G)$, \exists a derivation tree whose yield is w . Conversely, the yield of any derivation tree is in $L(G)$.
(Proof in book.)

Parsing and Ambiguity

The question of ambiguity is related to whether or not a string has two different derivation trees.

Def A string σ is ambiguous in grammar G if there exist at least two derivations of σ in G which have different derivation trees.

Def A c.f.g. G is ambiguous if $\exists w \in L(G)$ such that w is ambiguous.

Example

$$S \rightarrow S + S$$

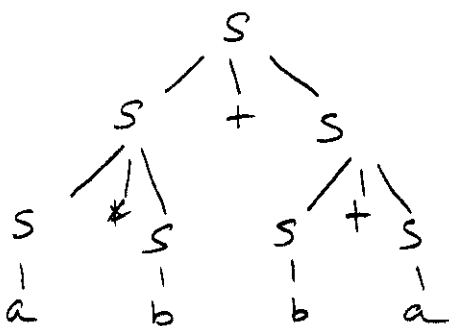
$$S \rightarrow S * S$$

$$S \rightarrow a | b$$

consider the string

$$a * b + b + a$$

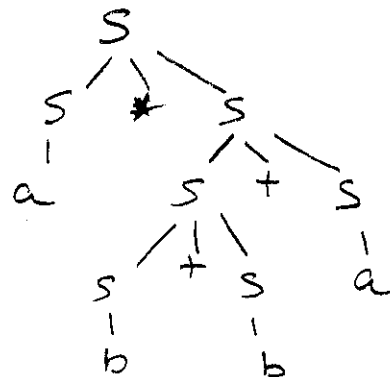
Tree 1



Leftmost derivation

$$\begin{aligned} S &\Rightarrow S + S \Rightarrow S * S + S \\ &\Rightarrow a * S + S \Rightarrow a * b + S \\ &\Rightarrow a * b + S + S \\ &\Rightarrow a * b + b + S \Rightarrow a * b + b + a \end{aligned}$$

Tree 2



Leftmost derivation Tree 2.

$$\begin{aligned} S &\Rightarrow S * S \Rightarrow a * S \\ &\Rightarrow a * S + S \Rightarrow a * S + S + S \\ &\Rightarrow a * b + S + S \Rightarrow a * b + b + S \\ &\Rightarrow a * b + b + a \end{aligned}$$

In this example G is ambiguous.
 But is $L(G)$ ambiguous? Note that a previous grammar generated the same language but was unambiguous.

Def A context-free language L is unambiguous if there exists a grammar G such that G is unambiguous and $L = L(G)$.

Def If every grammar that generates L is ambiguous, then L is said to be inherently ambiguous.

In general, it is very difficult to show if a grammar is ambiguous or not, and whether or not a language is inherently ambiguous.

Example $L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$ is inherently ambiguous.

Parsing A parsing of a string $w \in L(G)$ means to find a sequence of productions by which $S \Rightarrow^* w$ (or to determine that $w \notin L(G)$ if not known).

Example $S \rightarrow \overset{(1)}{SS} \mid \overset{(2)}{aSb} \mid \overset{(3)}{bSa} \mid \overset{(4)}{\lambda}$

Find parse for $w = aabb$. Could use (1) or (3) first.
 or (2) $S \Rightarrow aSb$ } at next step repeat this idea
 applying all valid rules.

$S \xrightarrow{(1)} SS \xrightarrow{(2)} SSS$	$\xrightarrow{(2)} aSbS$ $\xrightarrow{(2)} S'$	\checkmark \checkmark	
$\xrightarrow{(1)} aSb \xrightarrow{(2)} aSSb$	$\xrightarrow{(2)} a a S b b$	\checkmark \checkmark	(eliminate as sentential forms seen before)

Round 3

will find,

$\xrightarrow{(3)} aabb$

Def Above method is exhaustive search parsing.

Note: this is a form of topdown parsing; i.e., construction a derivation tree from the root down.

Theorem

Suppose a C.F.G. grammar has no rules of the form $A \rightarrow \lambda$ and $A \rightarrow B$.

Then the exhaustive search parsing method can be made into an algorithm to parse $w \in T^*$.

Proof

In each round, either the length of the sentential form increases, or the # of terminal symbols increases.

Each derivation choice increases one of these. \therefore The maximum length of a derivation is $2|w|$ rounds.

Therefore if w is parsed by then ok, else.

$w \notin L(G)$

Bottom Up Parsing An Example

Grammar

$$\begin{aligned}
 S &\rightarrow S + T \mid T \\
 T &\rightarrow T * R \mid R \\
 R &\rightarrow id
 \end{aligned}$$

Parse:

$$id_1 + id_2 * id_3$$

Idea:

Start with two stacks; defined logic decides to either shift or reduce based on item at top of left stack and item at top of right stack

		<u>Action</u>
\$	$id_1 + id_2 * id_3 \$$	shift
\$ id_1	$+ id_2 * id_3 \$$	reduce
\$ R	$+ id_2 * id_3 \$$	reduce
\$ T	$+ id_2 * id_3 \$$	reduce
\$ S	$+ id_2 * id_3 \$$	shift
\$ $S +$	$id_2 * id_3 \$$	shift
\$ $S + id_2$	$* id_3 \$$	reduce
\$ $S + R$	$* id_3 \$$	reduce
\$ $S + T$	$* id_3 \$$	shift !
\$ $S + T *$	$id_3 \$$	shift
\$ $S + T * id_3$	$\$$	reduce
\$ $S + T * R$	$\$$	reduce
\$ $S + T$	$\$$	reduce
\$ S	$\$$	(done)

Note

Tree is determined bottom up

Normal Forms for e.f.g.s.

Develop equivalent grammars that have restricted forms.
Two are particularly useful.

1. Chomsky Normal Form. CNF.

Productions of form

$$A \rightarrow BC$$

$$A, B, C \in N$$

$$A \rightarrow a$$

$$a \in T$$

2. Greibach Normal Form. GNF.

Productions of form

$$A \rightarrow ax$$

$$A \in N, a \in T, x \in N^*$$

Preliminaries

- { ① Algorithm for determining if $\lambda \in L(G)$.
- { ② Algorithm for eliminating λ -productions. ($A \rightarrow \lambda$)
- { ③ Algorithm for eliminating unit productions ($A \rightarrow B$)
- { ④ Algorithm for determining if $L(G)$ is empty.
- { ⑤ Algorithm for eliminating useless productions.

(a) A variable X is useful if

$$S \xRightarrow{*} \alpha X \beta \xRightarrow{*} w$$

$$w \in T^* \quad \alpha, \beta \in (N \cup T)^*$$

(b) Otherwise, the variable is useless.

(c) A production is useless if it contains useless variables.

Definition A variable A for which it is possible to derive λ , i.e. $A \xRightarrow{*} \lambda$ is nullable.

Lemma

Given a grammar G , the set of nullable variables in G , V_N , can be found using an algorithm.

Proof

1. For all productions $A \rightarrow \lambda$, put A into V_N .
2. Repeat the following until no new variables added to V_N .
For each production

$$B \rightarrow A_1 \dots A_n,$$

if $A_1, \dots, A_n \in V_N$, then put B into V_N .

Example

$$S \rightarrow ABaC$$

$$B \rightarrow b \mid \lambda$$

$$D \rightarrow d$$

$$A \rightarrow BC$$

$$C \rightarrow D \mid \lambda$$

Add B , add C , add A . That's all $\{A, B, C\} = V_N$.

Theorem

There exists an algorithm to determine if $\lambda \in L(G)$.

Proof

By Lemma determine the set of nullable variables
if $S \in V_N$ then $S \xRightarrow{*} \lambda$ and $\lambda \in L(G)$
else $\lambda \notin L(G)$.

Theorem

Let G be any context free grammar with $\lambda \notin L(G)$.

Then there exists an equivalent grammar \tilde{G} having no λ -productions.

Proof

Calculate nullable set V_N .

To construct \tilde{P} , for each production

$$A \rightarrow x_1 \dots x_m \quad m \geq 1, \quad x_i \in V \cup T$$

replace with $A \rightarrow x_1 \dots x_m$ and all combinations of nullable variable in $x_1 \dots x_m$ replaced by λ .

If all nullable do not include $A \rightarrow \lambda$.

Then prove that

$$A \xRightarrow{G}^* W \text{ and } W \neq \epsilon \iff A \xRightarrow{\tilde{G}}^* W$$

By induction ϵ using induction principle for c.f.g.'s.

Example (Previous)

A, B, C nullable. $\rightarrow S \rightarrow AaC$

$$S \rightarrow ABaC, \quad S \rightarrow BaC, \quad S \rightarrow ABa, \quad S \rightarrow aC, \quad S \rightarrow Ba,$$

$$S \rightarrow Aa, \quad S \rightarrow a$$

$$A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow BC$$

$$B \rightarrow b, \quad C \rightarrow D, \quad D \rightarrow d.$$

Theorem

Let G be a c.f.g. with $\lambda \in L(G)$. Then there exists a grammar \tilde{G} having no λ productions such that

$$L(G) = L(\tilde{G}) \cup \{\lambda\}.$$

Proof

Same as previous theorem, essentially.

Theorem

Let G be a c.f.g. with $\lambda \in L(G)$. Then there exists an equivalent grammar \tilde{G} with only the single λ production $S \rightarrow \lambda$.

Proof

Trivial using previous theorem.

Unit Productions

Trying to get rid of productions such as $A \rightarrow B$ have to be careful since $B \rightarrow A$ might also be a unit production.

Theorem

Let G be a c.f.g. without λ -productions. Then there exists an equivalent grammar G' without unit productions.

Proof

- (1) Eliminate productions of form $A \rightarrow A$.
 - (2) Add all non-unit productions of P .
 - (3) if $A \stackrel{*}{\Rightarrow} B$ $A \neq B$ (using a dependency graph this can be determined) add $A \rightarrow y_i$ whenever $B \rightarrow y_i \in P$ and is a nonunit production.
- This can be proven to be equivalent to original.

Example

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

Note $S \stackrel{*}{\Rightarrow} A$, $S \stackrel{*}{\Rightarrow} B$, $B \stackrel{*}{\Rightarrow} A$, $A \stackrel{*}{\Rightarrow} B$

\tilde{P} : $S \rightarrow Aa$ $B \rightarrow bb$ $A \rightarrow a \mid bc$

$S \rightarrow a \mid bc$ $B \rightarrow a \mid bc$

$S \rightarrow bb$ $A \rightarrow bb$

Eliminating useless productions

A variable is useful if $\exists w \in L(G)$ st.

$$S \xRightarrow{*} xAy \xRightarrow{*} w.$$

To be useful,

- (1) there must be a way of getting a terminal string from A .

$$S \rightarrow aAb$$

A is useless

$$A \rightarrow bAa$$

- (2) The variable must occur in a sentential form reachable from the start variable and the sentential form must be able to derive a terminal string. (Just consider reachability from start variable first).

Eg. $S \rightarrow AC$

$$C \rightarrow aAb$$

$$B \rightarrow Ab$$

B is not reachable from S .

Example

$$S \rightarrow aS / A / C$$

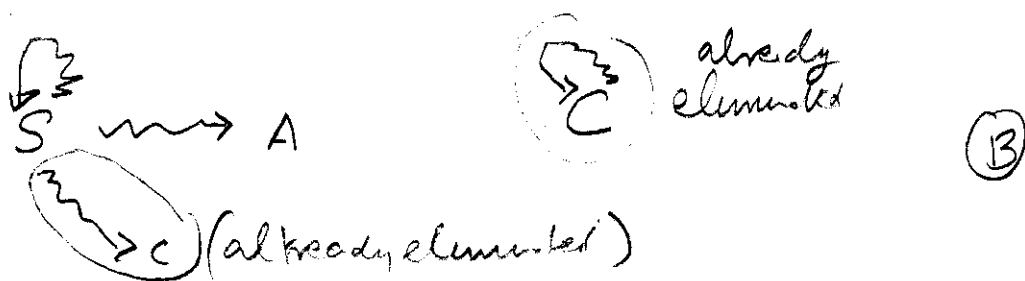
$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb$$

- (1) (a) A leads to terminal string, B , leads to terminal string
 (b) S leads to terminal string
 (c) no more additions to set. stop with $\{A, B, S\}$
 (d) C is useless.

- ② Create a dependency graph for the variables
 $C \rightsquigarrow D \iff$ for some strings x, y , $C \rightarrow x D y$
 $\{A, B, S\}$



\therefore B is not reachable. \therefore B should be eliminated and is useless.

Lemma 1.

Given a λ -free ^{c.f.g.} grammar G , there exists an equivalent grammar \hat{G} for which every variable A derives some terminal string $w \in L(G)$.

Procedure

1. Let $\hat{V} = \emptyset$
2. For every $A \in V$ with a production of the form $A \rightarrow x_1 \dots x_n$ with $x_i \in T \cup \hat{V}$ add A to \hat{V} .
3. Repeat until \hat{V} no longer changes.
4. Include all productions of P into \hat{P} if $p \in P$ is such that it contains only terminals or variables of \hat{V} .

Proof

It can be proven that G is equivalent to \hat{G} .

Lemma 2

Given a λ -free c.f.g. grammar G with each variable deriving some terminal string. Then, there exists an equivalent grammar \hat{G} such that each variable B is reachable from S . (unless c.f.g. is empty)

Procedure

- (1) Construct dependency graph of G .
- (2) Eliminate variables not reachable from S to get \hat{V} .
- (3) Eliminate all productions containing useless variables to get \hat{P} .

Proof. It can be shown that \hat{G} is equivalent to G .
 Furthermore if $S \xRightarrow{*} x_1 \dots x_n$ with $x_i \in T \cup \hat{V}$,
 then $S \xRightarrow{*} w \in T$. (since each nonterminal derives some terminal string.)

Theorem

Let G be a λ -free context-free grammar. There exists an equivalent grammar \hat{G} that contains no useless variables or productions.

Proof

Apply Lemma 1 and Lemma 2 in that order.

Theorem

There exists an algorithm to determine if $L(G)$ is empty.

Proof

Apply Lemma 1. S is useless $\Leftrightarrow L(G)$ is empty.

Chomsky Normal Form

A grammar is in Chomsky Normal Form if each production is of the form:

$$\left. \begin{array}{l} A \rightarrow BC \\ A \rightarrow a \end{array} \right\} \begin{array}{l} \text{where } A, B, C \in V \\ a \in T \end{array}$$

Example $S \rightarrow AB \quad B \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b \mid a$

Theorem Every context-free language L with $\lambda \in L$ is generated by a Chomsky Normal Form (CNF) grammar.

Proof Let G be a c.f.g. generating L . Without loss of generality assume G has no λ productions or unit productions. Now,

(1) Suppose G has a production of the form:

$A \rightarrow \alpha$ where α is 2 or more terminals & variables.

(a) Choose new non-terminals N_i for each terminal t_i ind.

(b) Replace each t_i by N_i in α , and add productions $N_i \rightarrow t_i$

(c) It should be clear that this new grammar is equivalent to the old grammar.

Example Consider the production $A \rightarrow b E F c$

Replace by: $A \rightarrow N_1 E F N_2$

$N_1 \rightarrow b$

$N_2 \rightarrow c$

(2) After step (1) each production is of the form

$$\left. \begin{array}{l} A \rightarrow K_1 K_2 K_3 \dots K_n \\ \text{or} \\ A \rightarrow a \end{array} \right\} \begin{array}{l} K_i \in V \\ a \in T. \end{array}$$

Replace the first production by:

$A \rightarrow K_1 Z_1$ where Z_i are new non-terminals.

$Z_1 \rightarrow K_2 Z_2$

$Z_2 \rightarrow K_3 Z_3$

\vdots

$Z_{n-3} \rightarrow K_{n-2} Z_{n-2}$

$Z_{n-2} \rightarrow K_{n-1} K_n$

It should be clear that this new grammar is equivalent to old.

Example $A \rightarrow BCDEF$ replaced by:

$$\left\{ \begin{array}{l} A \rightarrow BZ_1 \\ Z_1 \rightarrow CZ_2 \\ Z_2 \rightarrow DZ_3 \\ Z_3 \rightarrow EF \end{array} \right.$$

Example Consider the grammar $(\{S, A, B\}, \{a, b\}, P, S)$

with productions: $S \rightarrow bA \mid aB$

$A \rightarrow aS \mid bAA \mid a$

$B \rightarrow bS \mid aBB \mid b$

Find an equivalent C.N.F. grammar.

$S \rightarrow N_1 A \mid N_2 B$ $N_1 \rightarrow b$ $N_2 \rightarrow a$

$A \rightarrow N_3 S \mid N_4 AA \mid a$ $N_3 \rightarrow a$ $N_4 \rightarrow b$

$B \rightarrow N_5 S \mid N_6 BB \mid b$ $N_5 \rightarrow b$ $N_6 \rightarrow a$

Still need to fix up $A \rightarrow N_4 AA$ & $B \rightarrow N_6 BB$

$\therefore \left. \begin{array}{l} A \rightarrow N_4 Z_1 \\ Z_1 \rightarrow AA \\ B \rightarrow N_6 Z_2 \\ Z_2 \rightarrow BB \end{array} \right\} \text{replace these}$

Greibach Normal Form

A grammar is in Greibach Normal Form if each production is of the form: $A \rightarrow a \alpha$ where $a \in T$, $\alpha \in V^*$

Lemma 1. (Thm 6.1 in text; Substitution)

Let $G = (V, T, S, P)$ be a c.f.g.

Suppose $A \rightarrow \alpha_1 B \alpha_2$ is an A -production,

and $B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ are all the B -productions.

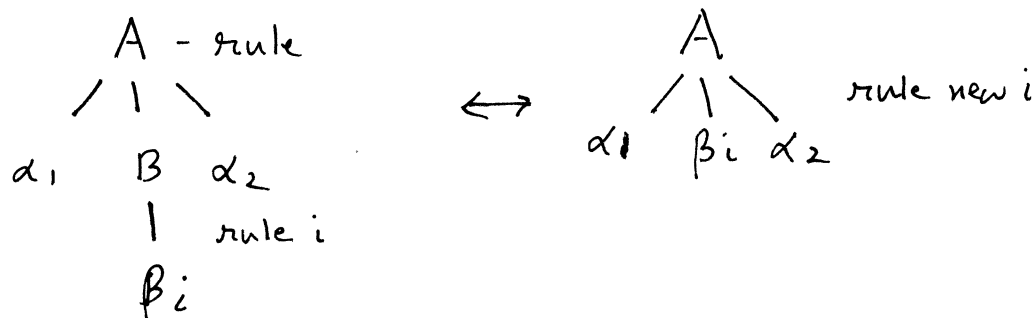
Define a new grammar $\hat{G} = (V, T, S, \hat{P})$ by:

(1) delete the production $A \rightarrow \alpha_1 B \alpha_2$

(2) add the productions $A \rightarrow \alpha_1 \beta_1 \alpha_2 \mid \dots \mid \alpha_1 \beta_n \alpha_2$

Claim: $L(G) = L(\hat{G})$.

"Proof"



These derive the same sentential forms

Example: $A \rightarrow ABa$

$B \rightarrow AA \mid b \mid Z_1 A_2$

\therefore we can delete

$A \rightarrow ABa$ by adding $A \rightarrow AAAa$

$A \rightarrow Aba$

$A \rightarrow AZ_1 A_2 a$

Lemma 2 (Thm 6.2 in text; removing left recursion)

Let $G = (V, T, S, P)$ be a c.f.g.

Let $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n$ be the set of A -productions that have A as the first symbol on the R.H.S. (left recursive in A).

Let $A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$ be all the other A -productions.

Let $\hat{G} = (V \cup \{Z\}, T, S, \hat{P})$ be the c.f.g. with:

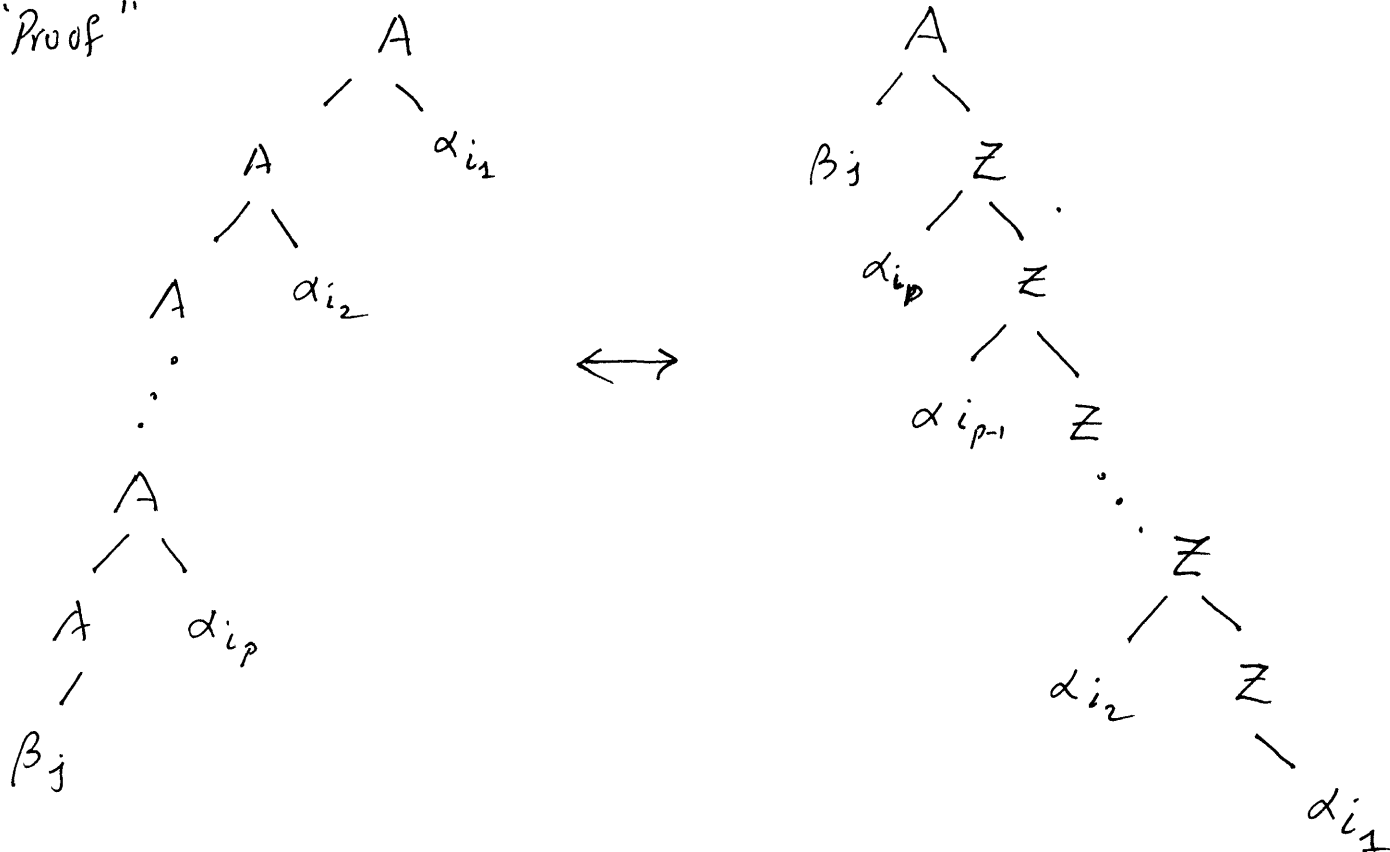
(1) "new productions"

(this already exists) \rightarrow $A \rightarrow \beta_i \quad 1 \leq i \leq m$ $Z \rightarrow d_i \quad 1 \leq i \leq n$
 $A \rightarrow \beta_i Z$ $Z \rightarrow d_i Z$

(2) delete all left recursive A -productions.

Claim: $L(G) = L(\hat{G})$.

"Proof"



Theorem: Every c.f.l. L with $\lambda \notin L$ is generated by a Greibach Normal Form grammar.

Proof

- ① Rewrite grammar into Chomsky Normal Form
- ② Relabel all variables as A_1, A_2, \dots, A_n .
- ③ Transform all productions into form:
 - (a) $A_i \rightarrow A_j x_j \quad i < j \text{ and } x_j \in V^*$
 - (b) or $A_i \rightarrow a x_j$
 - (c) or $Z_i \rightarrow A_j x_j$

Step (a) is either true for $i=1$ or there is a production of the form $A_1 \rightarrow A_1 x$ (say n of these)
Using Lemma 2, we can "remove" these left-recursions by:

replacing with	}	$A_1 \rightarrow \beta_i \quad 1 \leq i \leq m$	$Z \rightarrow \alpha_i \quad 1 \leq i \leq n$
		$A_1 \rightarrow \beta_i Z \quad 1 \leq i \leq m$	$Z \rightarrow \alpha_i Z \quad 1 \leq i \leq n$

After this step, there are no left recursive A_1 productions. Consider now an A_2 production. Either the r.h.s. starts with A_1 in which case we use Lemma 1 to substitute or "remove" this first A_1 .

($A_2 \rightarrow A_1 y \quad \& \quad A_1 \rightarrow \beta_i$ replace with
 $A_2 \rightarrow \beta_i y$, for all such r.h.s. starting with A_1).

Note that the new set must start with a terminal or $A_i, i \geq 2$.

\therefore all A_2 productions are recursive in A_2 or are of the form (b) or (c).

Now remove the recursive A_2 production.

Continue with $A_3 \dots A_n$, using substitution and left recursion removal.

When A_n is completed all productions are of the form (a), (b), or (c).

- (4) In the final step, note that an A_n -production must be of the form:

$$A_n \rightarrow a_n x_n \quad a_n \in T, x_n \in V^*$$

since no index is higher than n .

Thus, this production is in the correct G.N.F.

Consider an A_{n-1} production. It must be:

$$A_{n-1} \rightarrow a_{n-1} x_{n-1} \quad \text{this is in G.N.F.}$$

or $A_{n-1} \rightarrow A_n x_n$ Use lemma 1 again to substitute for A_n and thus A_{n-1} -productions can now be in correct G.N.F.

Continue with $A_{n-2} \dots A_1$ to get each of these in the correct G.N.F.

Finally use Lemma 1 again to fix all Z -productions, resulting in a Greibach Normal Form equivalent grammar.

Example $G = (\{A_1, A_2, A_3\}, \{a, b\}, A_1, P)$

$$P: \begin{aligned} A_1 &\rightarrow A_1 A_1 \quad | \quad A_2 A_3 \\ A_2 &\rightarrow A_3 A_2 \\ A_3 &\rightarrow A_1 A_2 \\ A_2 &\rightarrow a \\ A_3 &\rightarrow b \end{aligned}$$

Apply left recursion Lemma 2 to A_1 .

$$(\text{recursive } A_1 \text{ prod.}) \quad A_1 \rightarrow A_1 A_1 \quad A_1 \rightarrow A_2 A_3 \quad (\text{other } A_1 \text{ productions})$$

$$\text{Get:} \quad \left. \begin{aligned} A_1 &\rightarrow A_2 A_3 & Z_1 &\rightarrow A_1 \\ A_1 &\rightarrow A_2 A_3 Z_1 & Z_1 &\rightarrow A_2 Z_1 \end{aligned} \right\} \begin{aligned} A_2 &\rightarrow A_3 A_2 \quad | \quad a \\ A_3 &\rightarrow A_1 A_2 \quad | \quad b \end{aligned}$$

A_2 is already o.k.; need to fix up A_3 .

Apply Lemma 1 (substitution) to $A_3 \rightarrow \underline{A_1} A_2$ & delete this.

$$\left. \begin{aligned} A_3 &\rightarrow A_2 A_3 A_2 \\ A_3 &\rightarrow A_2 A_3 Z_1 A_2 \end{aligned} \right\} \begin{aligned} A_1 &\rightarrow A_2 A_3 & Z_1 &\rightarrow A_1 & A_2 &\rightarrow A_3 A_2 \quad | \quad a \\ A_1 &\rightarrow A_2 A_3 Z_1 & Z_1 &\rightarrow A_1 Z_1 & A_3 &\rightarrow \underline{A_1} A_2 \quad | \quad b \end{aligned}$$

(Note deleted $A_3 \rightarrow A_1 A_2$, left $A_3 \rightarrow b$)

Apply Lemma 1 to $\left\{ \begin{aligned} A_3 &\rightarrow A_2 A_3 A_2 \\ A_3 &\rightarrow A_2 A_3 Z_1 A_2 \end{aligned} \right.$ (then delete)

$$\text{Get:} \quad \left. \begin{aligned} A_3 &\rightarrow A_3 A_2 A_3 A_2 \\ A_3 &\rightarrow a A_3 A_2 \\ A_3 &\rightarrow A_3 A_2 A_3 Z_1 A_2 \\ A_3 &\rightarrow a A_3 Z_1 A_2 \\ A_3 &\rightarrow b \end{aligned} \right\} \begin{aligned} A_1 &\rightarrow A_2 A_3 \quad | \quad A_2 A_3 Z_1 \\ A_2 &\rightarrow A_3 A_2 \quad | \quad a \\ Z_1 &\rightarrow A_1 \quad | \quad A_1 Z_1 \end{aligned}$$

Apply Lemma 2 to:

$$A_3 \rightarrow A_3 A_2 A_3 A_2 \quad (\text{to be deleted})$$

$$A_3 \rightarrow A_3 A_2 A_3 Z_1 A_2$$

$$A_3 \rightarrow a A_3 A_2$$

$$A_3 \rightarrow a A_3 Z_1 A_2 \quad (\text{other } A_3 \text{ productions})$$

$$A_3 \rightarrow b$$

$$A_3 \rightarrow a A_3 A_2 \mid a A_3 Z_1 A_2 \mid b$$

$$A_3 \rightarrow a A_3 A_2 Z_2 \mid a A_3 Z_1 A_2 Z_2 \mid b Z_2$$

$$Z_2 \rightarrow A_2 A_3 A_2 \mid A_2 A_3 Z_1 A_2$$

$$Z_2 \rightarrow A_2 A_3 A_2 Z_2 \mid A_2 A_3 Z_1 A_2 Z_2$$

$$A_1 \rightarrow A_2 A_3 \mid A_2 A_3 Z_1$$

$$A_2 \rightarrow A_3 A_2 \mid a$$

$$Z_1 \rightarrow A_1 \mid A_1 Z_1$$

Now, productions are all in the form of step ③. ϵ A_3 is in G.N.F.

Using substitution (step ④) eliminate $A_2 \rightarrow A_3 A_2$ to get:

$$\{ A_2 \rightarrow a A_3 A_2 A_2 \mid a A_3 Z_1 A_2 A_2 \mid b A_2$$

$$\{ A_2 \rightarrow a A_3 A_2 Z_2 A_2 \mid a A_3 Z_1 A_2 Z_2 A_2 \mid b Z_2 A_2$$

$$A_2 \rightarrow a \quad (\text{from before}) \quad (\text{Note: } A_1, \epsilon Z_1 \text{ productions are the same and not shown here; similarly } A_3, Z_2.)$$

Next need to eliminate $A_1 \rightarrow A_2 A_3 \mid A_2 A_3 Z_1$ by substitution.

$$A_1 \rightarrow a A_3 A_2 A_2 A_3 \mid a A_3 Z_1 A_2 A_2 A_3 \mid b A_2 A_3$$

$$A_1 \rightarrow a A_3 A_2 Z_2 A_2 A_3 \mid a A_3 Z_1 A_2 Z_2 A_2 A_3 \mid b Z_2 A_2 A_3$$

$$A_1 \rightarrow a A_3$$

$$A_1 \rightarrow a A_3 A_2 A_2 A_3 Z_1 \mid a A_3 Z_1 A_2 A_2 A_3 Z_1 \mid b A_2 A_3 Z_1$$

$$A_1 \rightarrow a A_3 A_2 Z_2 A_2 A_3 Z_1 \mid a A_3 Z_1 A_2 Z_2 A_2 A_3 Z_1 \mid b Z_2 A_2 A_3 Z_1$$

$$A_1 \rightarrow a A_3 Z_1$$

Now all of A_1, A_2, A_3 productions are in G.N.F.

Need to fix up the $Z_1, \epsilon Z_2$ productions by substitution.

$$Z_1 \rightarrow A_1 \mid A_1 Z_1$$

$$Z_2 \rightarrow A_2 A_3 A_2 \mid A_2 A_3 Z_1 A_2$$

$$Z_2 \rightarrow A_2 A_3 A_2 Z_2 \mid A_2 A_3 Z_1 A_2 Z_2$$

→ substitute for leading

$A_1, \epsilon A_2$

(not shown.)