# Context-free Grammars & Languages

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

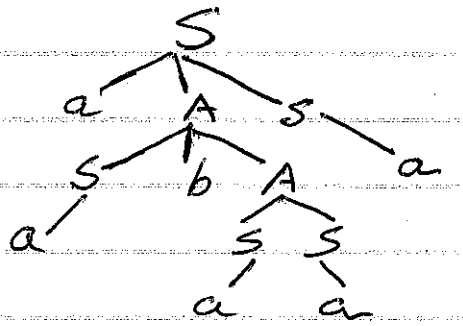
$$A \to \alpha \qquad A \in V,\ \alpha \in \{V \cup T\}^*$$

**Def.** A language is a <u>context-free language</u> (cfl) $\iff \exists$ a cfg $G$ st $L = L(G)$.

**Example 1**

$$S \to aAS \mid a$$
$$A \to SbA \mid SS \mid ba$$

We associate a <u>derivation tree</u> with any (terminal) string $w$ s.t. $S \Rightarrow^* w$




$S \Rightarrow aAS \Rightarrow aSbAS$
$\Rightarrow aSbAa \Rightarrow aSbSSa$
$\Rightarrow aabSSa \Rightarrow aababSa$
$\Rightarrow aababaaa$

What is $L(G)$ ?

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

**Example 2**

$$S \to AS_1 \mid S_1B$$
$$S_1 \to aS_1b \mid \lambda$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

## 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

$S \to aB$ $\quad$ $A \to bAA$ $\quad$ Claim: $L(G)$ is all words over $\{a,b\}$ that have an equal number of a's & b's. (excluding $\lambda$)

$S \to bA$ $\quad$ $B \to b$

$A \to a$ $\quad$ $B \to bS$

$A \to aS$ $\quad$ $B \to aBB$

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

Induction hypothesis:

① $S \overset{*}{\Rightarrow} w$ iff $w$ has an equal # of a's & b's.

② $A \overset{*}{\Rightarrow} w$ 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 $S \overset{*}{\Rightarrow} w$, $|w| = k$. Then, the first derivation step must be $S \to aB$ or $S \to bA$.

Suppose it is $S \to aB$. Then $w = aw_1$ where $B \overset{*}{\Rightarrow} w_1$.

This last statement is using the induction principal for cfg'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 $S \to bA$ 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 $A \to aS$ or $A \to bAA$.
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. $A \Rightarrow^* w_1$ & 2nd $A \Rightarrow^* w_2$ 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

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

③ Proof of this is similar to proof on ②.

## Leftmost and Rightmost Derivations

### Definition

A derivation is said to be <u>leftmost</u> 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

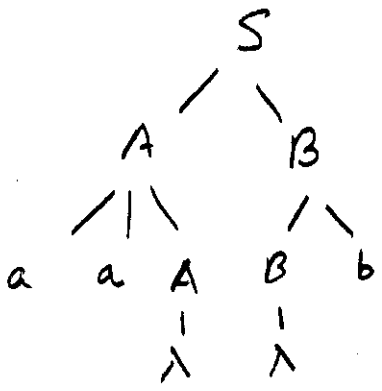
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

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

$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$
(leftmost)

$S \Rightarrow AB \Rightarrow ABb \Rightarrow aaABb \Rightarrow aaAb \Rightarrow aab$
(neither)

Trees




trees are the same.

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

Let $G$ be a c.f.g. and let $\alpha, \beta \in (V \cup T)^*$
If $\alpha \overset{r}{\Rightarrow} \beta$ for $r \geq 0$, and if $\alpha = \alpha_1 \ldots \alpha_n$ for $n \geq 1$,
then $\exists\ t_i \geq 0$, $\beta_i \in (V \cup T)^*$ for $1 \leq i \leq n$ s.t.

$$\beta = \beta_1 \cdots \beta_n \quad \text{and} \quad \alpha_i \overset{t_i}{\Rightarrow} \beta_i \quad \text{with} \quad \sum_{i=1}^{n} t_i = r.$$

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

Idea of Lemma: $\alpha_1 \ldots \alpha_n \Rightarrow \underbrace{b_1 b_2}_{\beta_1}\ \underbrace{b_3 \cdots}_{\beta_2}\ \underbrace{\cdots b_m}_{\beta_n}$

Can partition the derived string so each part is derived from a corresponding $\alpha_i$.

**Theorem**

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

Proof by induction on # steps in the derivation. ($A \overset{*}{\Rightarrow} w$)
True for step of length 1 ($A \to w$ is trivially leftmost).
Assume true for all derivations of length $\leq k$.
Let $A \overset{k+1}{\Rightarrow} w$. $\therefore$

$$A \Rightarrow X_1 \cdots X_n \overset{k}{\Rightarrow} w$$

($X_i \in T$ or $V$.)

by induction principle of c.f.g.s.
with $X_i \overset{t_i}{\Rightarrow} \beta_i$ with $\Sigma t_i = k$ and $X_i \overset{*}{\Rightarrow} \beta_i$
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 $\beta_1$, then $\beta_2$, ... then $\beta_n$.
This gives a leftmost derivation for $w$.

## Derivation Trees

**G:**

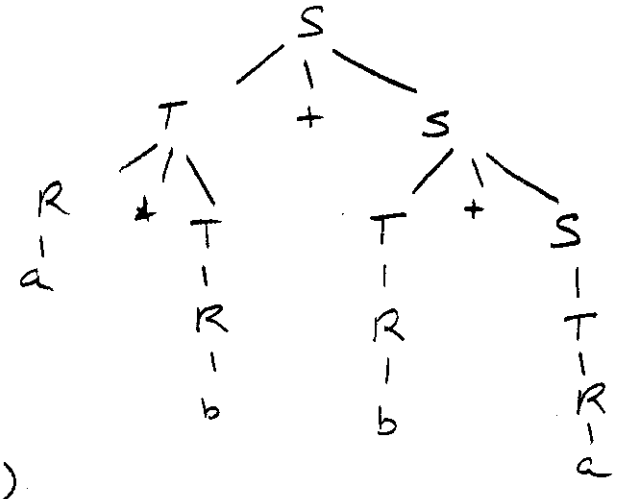
$S \to T + S$

$S \to T$

$T \to R * T$

$T \to R$

$R \to a \mid b$




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

For a grammar $G$,

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

$A \to a_1 \ldots a_n$ in the grammar.

For theorem a leaf labelled $\lambda$ has no siblings.

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

Note: reading leaves gives a sentential form. $\alpha$

Note: if $S \Rightarrow^* \alpha$, then $\alpha$ is the <u>yield</u> 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.

**<u>Thm</u>**

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 $\sigma$ is ambiguous in grammar $G$ if there exist at least two derivations of $\sigma$ 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.

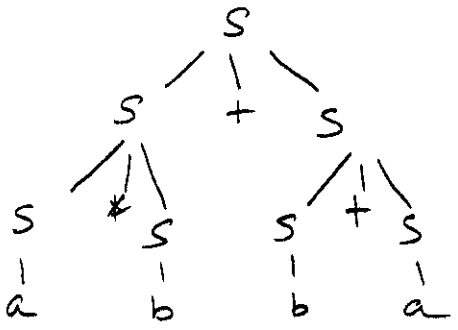
**Example**

$S \to S+S$
$S \to S*S$
$S \to a \mid b$

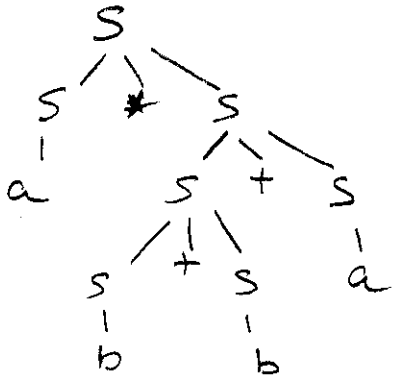
consider the string

$a * b + b + a$

Tree 1




Tree 2




Leftmost derivation

$S \Rightarrow S+S \Rightarrow S*S+S$
$\Rightarrow a*S+S \Rightarrow a*b+S$
$\Rightarrow a*b+S+S$
$\Rightarrow a*b+b+S \Rightarrow a+b+b+a$

Leftmost derivation tree 2.

$S \Rightarrow S*S \Rightarrow a*S$
$\Rightarrow a*S+S \Rightarrow a*S+S+S$
$\Rightarrow a*b+S+S \Rightarrow a*b+b+S$
$\Rightarrow a*b+b+a$

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 <u>unambiguous</u> 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 <u>inherently ambiguous</u>.

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* $S \to \overset{①}{SS} \mid \overset{②}{aSb} \mid \overset{③}{bSa} \mid \overset{④}{\lambda}$

Find parse for $w = aabb$. Could use ① or ② first.

① $S \Rightarrow SS$
or ② $S \Rightarrow aSb$

} at next step repeat this idea applying all valid rules.

$S \overset{①}{\Rightarrow} SS \overset{②}{\Rightarrow} SSS$ ✓

$\overset{②}{\Rightarrow} aSbS$ ✓

$\overset{②}{\Rightarrow} S'$ (eliminate as sentential form seen before)

$\overset{①}{\Rightarrow} aSb \overset{②}{\Rightarrow} aSSb$ ✓

$\overset{②}{\Rightarrow} aaSbb$ ✓

Round 3

$\overset{③}{\Rightarrow} aabb$

will find.

Def Above method is exhaustive search parsing

Note: this is a form of top-down parsing; ie., construction a derivation tree from the root down.

Theorem

Suppose a C.f.g. grammar has no rules of the form $A \to \lambda$ and $A \to B$.
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. ∴ The maximum length of a derivation is $2|w|$ rounds. Therefore if $w$ is parsed by then O/K. else. $w \notin L(G)$

# Bottom Up Parsing
## An Example

Grammar

$S \rightarrow S + T \mid T$
$T \rightarrow T * R \mid R$
$R \rightarrow id$

Parse: $id_1 + id_2 * id_3$

Idea: Start with two stacks; defined logic decides to either <u>shift</u> or <u>reduce</u> based on item at top of left stack and item at top of right stack

| | | Action |
|---|---|---|
| $ | $id_1 + id_2 * id_3$ $ | 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 c.f.g.s.

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

1. Chomsky Normal Form. CNF

Productions of form

$A \to BC$ $\quad A, B, C \in N$

$A \to a$ $\quad a \in T$

2. Greibach Normal Form. GNF.

Productions of form

$A \to ax$ $\quad A \in N,\ a \in T,\ x \in N^*$

## Preliminaries

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

(a) A variable $X$ is <u>useful</u> if

$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ $\quad w \in T^*$ $\quad \alpha, \beta \in (N \cup T)^*$.

(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 $\lambda$, ie $A \overset{*}{\Rightarrow} \lambda$ 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 $A \to \lambda$, put $A$ into $V_N$.
2. Repeat the following until no new variables added to $V_N$.
   For each production
   $$B \to A_1 \dots A_n,$$
   if $A_1, \dots, A_n$ in $V_N$, then put $B$ into $V_N$.

**Example**

$S \to ABaC$ $\quad$ $B \to b \mid \lambda$ $\quad$ $D \to d$

$A \to BC$ $\quad$ $C \to D \mid \lambda$

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 $S \overset{*}{\Rightarrow} \lambda$ 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 $\hat{G}$ having no $\lambda$-productions.

**Proof**

Calculate nullable set $V_N$.

To construct $\hat{P}$, for each production

$$A \to x_1 \cdots x_m \qquad m \geq 1, \qquad x_i \in V \cup T$$

replace with $A \to x_1 \cdots x_m$ and all combinations of nullable variable in $x_1 \cdots x_m$ replaced by $\lambda$. If all nullables do not include $A \to \lambda$.

Then prove that

$$A \overset{*}{\underset{G}{\Rightarrow}} w \text{ and } w \neq \epsilon \iff A \overset{*}{\underset{\hat{G}}{\Rightarrow}} w$$

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

**Example** (Previous) $A, B, C$ nullable $\rightarrow S \to AaC$

$S \to ABaC$, $S \to BaC$, $S \to ABa$, $S \to aC$, $S \to Ba$, $S \to Aa$, $S \to a$

$A \to B$, $A \to C$, $A \to BC$

$B \to b$, $C \to D$, $D \to d$.

**Theorem**

Let $G$ be a cfg with $\lambda \in L(G)$. Then there exists a grammar $\hat{G}$ having no $\lambda$ productions such that

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

**Proof**

Same as previous theorem, essentially.

Theorem

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

Proof

Trivial using previous theorem.

## Unit Productions

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

Theorem

Let $G$ be a c.f.g. without $\lambda$-productions Then there exists an equivalent Grammar $G'$ without unit productions.

Proof

(1) Eliminate productions of form $A \to A$.

(2) Add all non-unit productions of $P$.

(3) if $A \overset{*}{\Rightarrow} B$ $A \neq B$ (using a dependency graph this can be determined) add $A \to y_i$ whenever $B \to y_i \in P$ and is a non-unit production

This can be proven to be equivalent to original.

Example

$S \to Aa \mid B \quad B \to A \mid bb \quad A \to a \mid bc \mid B$

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

$\hat{P}$: $S \to Aa \quad B \to bb \quad A \to a \mid bc$

$S \to a \mid bc \quad B \to a \mid bc$

$S \to bb \quad A \to bb$

## Eliminating useless productions

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

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

To be useful,

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

$$S \to aAb \qquad A \text{ is useless}$$
$$A \to bAa$$

(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.
$$S \to AC$$
$$C \to aAb \qquad B \text{ is not reachable from } S.$$
$$B \to Ab$$

### Example

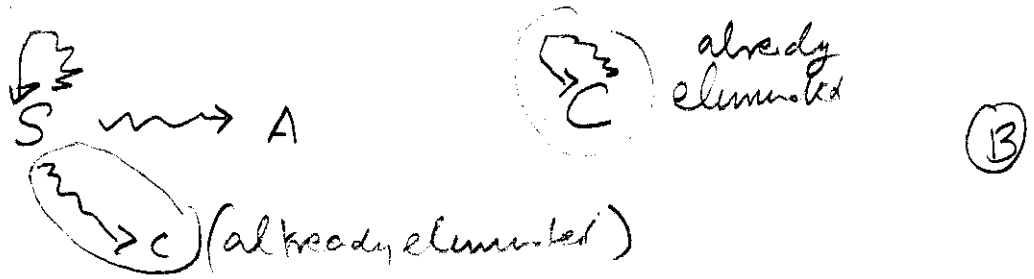
$$S \to aS \mid A \mid C$$
$$A \to a$$
$$B \to aa$$
$$C \to aCb$$

(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 $x, y$, $C \to x D y$

$\{A, B, S\}$




∴ B is not reachable. ∴ 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 $\hat{V} = \Phi$
2. For every $A \in V$ with a production of the form $A \to x_1 \cdots x_n$ 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 $\lambda$-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 if $S \Rightarrow^* x_1 \cdots x_n$ with $x_i \in T \cup \hat{V}$, then $S \Rightarrow^* w \in T$. (since each nonterminal derives some terminal string.)

Theorem

Let $G$ be a $\lambda$-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:

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

where $A, B, C \in V$, $a \in T$

*Example* $S \to AB \quad B \to AB \quad A \to a \quad B \to b \mid a$

*Theorem* Every context-free language $L$ with $\lambda \notin L$ 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 $\lambda$ productions or unit productions. Now,

(1) Suppose $G$ has a production of the form:
$A \to \alpha$ where $\alpha$ is 2 or more terminals & variables.
(a) Choose new non-terminals $N_i$ for each terminal $t_i$ in $\alpha$.
(b) Replace each $t_i$ by $N_i$ in $\alpha$, and add productions
$N_i \to t_i$
(c) It should be clear that this new grammar is equivalent to the old grammar.

*Example* Consider the production $A \to bEFc$

Replace by: $A \to N_1 E F N_2$
$N_1 \to b$
$N_2 \to c$

(2) After step (1) each production is of the form $A \to K_1 K_2 K_3 \cdots K_N$ or $A \to a$ } $K_i \in V$, $a \in T$.

Replace the first production by:

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

$Z_1 \to K_2 Z_2$

$Z_2 \to K_3 Z_3$

$\vdots$

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

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

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

Example $A \to BCDEF$ replaced by: $\begin{cases} A \to B Z_1 \\ Z_1 \to C Z_2 \\ Z_2 \to D Z_3 \\ Z_3 \to EF \end{cases}$

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

$S \to bA \mid aB$

$A \to aS \mid bAA \mid a$

$B \to bS \mid aBB \mid b$

Find an equivalent C.N.F. grammar.

$S \to N_1 A \mid N_2 B$ $\quad N_1 \to b$ $\quad N_2 \to a$

$A \to N_3 S \mid N_4 AA \mid a$ $\quad N_3 \to a$ $\quad N_4 \to b$

$B \to N_5 S \mid N_6 BB \mid b$ $\quad N_5 \to b$ $\quad N_6 \to a$

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

$\therefore$ $A \to N_4 Z_1$ $\quad B \to N_6 Z_2$ } replace these

$Z_1 \to AA$ $\quad Z_2 \to BB$

## Greibach Normal Form

A grammar is in Greibach Normal Form if each production is of the form: $A \to a\alpha$ 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 $A \to \alpha_1 B \alpha_2$ is an $A$-production,

and $B \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$ are all the $B$-productions.

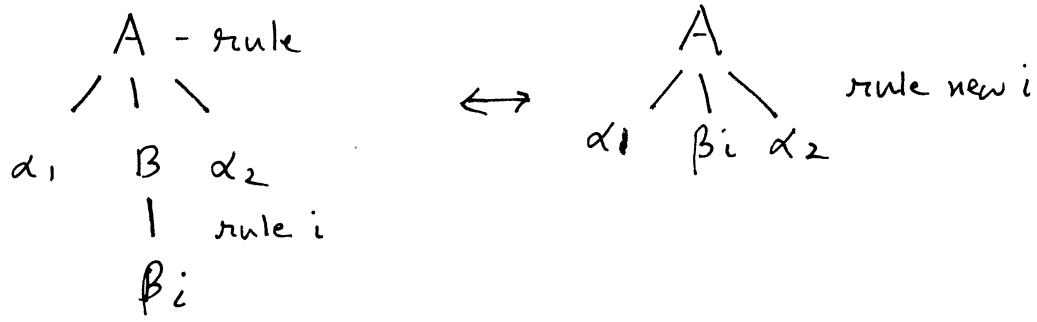
Define a new grammar $\hat{G} = (V, T, S, \hat{P})$ by:

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

(2) add the productions $A \to \alpha_1 \beta_1 \alpha_2 \mid \cdots \mid \alpha_1 \beta_n \alpha_2$

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

"Proof"




These derive the same sentential forms

Example: $A \to ABa$

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

∴ we can delete $A \to ABa$ by adding $A \to AAAa$

$A \to Aba$

$A \to AZ_1A_2a$

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

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

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

Let $A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$ 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"

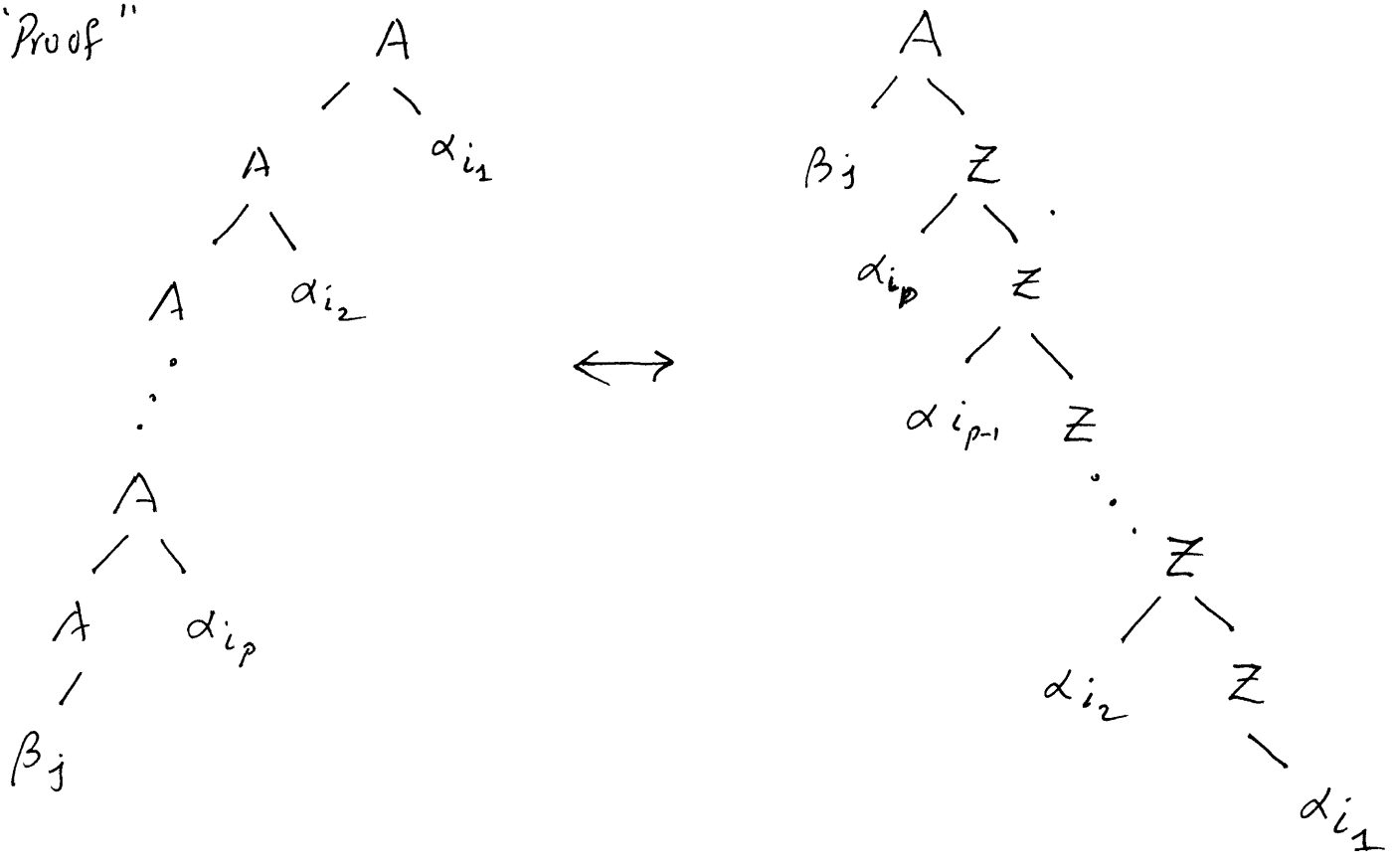
(this already exists) $A \to \beta_i \quad 1 \le i \le m$ $\qquad Z \to \alpha_i \quad 1 \le i \le n$

$A \to \beta_i Z$ $\qquad Z \to \alpha_i Z$

(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 $A_1, A_2, \ldots, A_n$.

③ Transform all productions into form:

(a) $A_i \to A_j x_j \quad i < j$ and $x_j \in V^*$

(b) or $A_i \to a x_j$

(c) or $Z_i \to A_j x_j$

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

replacing with

$A_1 \to \beta_i \quad 1 \le i \le m \qquad Z \to \alpha_i \quad 1 \le i \le n$

$A_1 \to \beta_i Z \quad 1 \le i \le m \qquad Z \to \alpha_i Z \quad 1 \le i \le n$

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 $A_1$.
( $A_2 \to A_1 y$ & $A_1 \to \beta_i$ replace with
$A_2 \to \beta_i y$, for all such r.h.s. starting with $A_1$).
Note that the new set must start with a terminal or $A_i, i \ge 2$.
∴ all $A_2$ productions are recursive in $A_2$ or are of the form (b) or (a).
Now remove the recursive $A_2$ production.

Continue with $A_3 \ldots A_n$, 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:

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

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:

$A_{n-1} \rightarrow a_{n-1} x_{n-1}$ 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 $A_{n-2} \cdots A_1$ 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: $A_1 \to A_1 A_1 \mid A_2 A_3$

$A_2 \to A_3 A_2$

$A_3 \to A_1 A_2$

$A_2 \to a$

$A_3 \to b$

Apply left recursion Lemma 2 to $A_1$.

(recursive $A_1$ prod.) $A_1 \to A_1 A_1$ $\qquad$ $A_1 \to A_2 A_3$ (other $A_1$ productions)

Get:

$A_1 \to A_2 A_3$ $\qquad$ $Z_1 \to A_1$

$A_1 \to A_2 A_3 Z_1$ $\qquad$ $Z_1 \to A_1 Z_1$

$A_2 \to A_3 A_2 \mid a$

$A_3 \to A_1 A_2 \mid b$

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

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

$A_3 \to A_2 A_3 A_2$

$A_3 \to A_2 A_3 Z_1 A_2$

$A_1 \to A_2 A_3$ $\quad$ $Z_1 \to A_1$ $\quad$ $A_2 \to A_3 A_2 \mid a$

$A_1 \to A_2 A_3 Z_1$ $\quad$ $Z_1 \to A_1 Z_1$ $\quad$ $A_3 \to \cancel{A_1 A_2} \mid b$

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

Apply Lemma 1 to

$A_3 \to A_2 A_3 A_2$ (then delete)

$A_3 \to A_2 A_3 Z_1 A_2$

Get:

$A_3 \to A_3 A_2 A_3 A_2$

$A_3 \to a A_3 A_2$

$A_3 \to A_3 A_2 A_3 Z_1 A_2$

$A_3 \to a A_3 Z_1 A_2$

$A_3 \to b$

$A_1 \to A_2 A_3 \mid A_2 A_3 Z_1$

$A_2 \to A_3 A_2 \mid a$

$Z_1 \to A_1 \mid A_1 Z_1$

Apply Lemma 2 to:

$A_3 \to A_3 A_2 A_3 A_2$ (to be deleted)
$A_3 \to A_3 A_2 A_3 Z_1 A_2$

$A_3 \to a A_3 A_2$
$A_3 \to a A_3 Z_1 A_2$
$A_3 \to b$
(other $A_3$ productions)

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

$A_1 \to A_2 A_3 \mid A_2 A_3 Z_1$
$A_2 \to A_3 A_2 \mid a$
$Z_1 \to A_1 \mid A_1 Z_1$

Now, productions are all in the form of step ③. & $A_3$ is in G.N.F.
Using substitution (step ④) eliminate $A_2 \to A_3 A_2$ to get:

$A_2 \to a A_3 A_2 A_2 \mid a A_3 Z_1 A_2 A_2 \mid b A_2$
$A_2 \to a A_3 A_2 Z_2 A_2 \mid a A_3 Z_1 A_2 Z_2 A_2 \mid b Z_2 A_2$
$A_2 \to a$ (from before.) (Note: $A_1$ & $Z_1$ productions are the same and not shown here; similarly $A_3, Z_2$.)

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

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

Now all of $A_1, A_2, A_3$ productions are in G.N.F.
Need to fix up the $Z_1$ & $Z_2$ productions by substitution.

$Z_1 \to A_1 \mid A_1 Z_1$
$Z_2 \to A_2 A_3 A_2 \mid A_2 A_3 Z_1 A_2$
$Z_2 \to A_2 A_3 A_2 Z_2 \mid A_2 A_3 Z_1 A_2 Z_2$

→ substitute for leading $A_1$ & $A_2$ (not shown.)