Ch. 2 – Phrase-Structured Grammars

(17 pages)

§1. Generating languages with phrase-structured grammars

Consider the English sentence, "Jane ate an apple." We can replace the noun "Jane" by any noun phrase, the verb "ate" by any appropriate phrase, the "an" by any other article, and the "apple" by any other noun phrase – and we will still get a grammatically correct sentence. For example we can easily get, "John quickly swallowed the two grapes." Roughly speaking a natural grammar such as English grammar consists of the following:

a) A set of phrase denoting variables – ⟨sent.⟩, ⟨noun⟩, ⟨verb⟩...
b) A set of terminal symbols – Jane, John, ate, an, ...
c) A starting variable – ⟨sent.⟩
d) A set of rewriting rules – ⟨verb⟩ → ⟨adverb⟩⟨verb⟩
⟨noun phrase⟩ → ⟨noun phrase⟩⟨conj.⟩⟨noun⟩

Def. A phrase-structured grammar (PSG) is a 4-tuple $G = \langle V, T, \mathcal{S}, \mathcal{P} \rangle$ where
$V = V(G)$ is a finite set of variables
$T = T(G)$ is a finite set of terminal symbols
$\mathcal{S} = \mathcal{S}(G)$ is a finite set of starting strings, and
$\mathcal{P} = \mathcal{P}(G)$ is a finite set of productions. (see def. below)

Def. A production is an ordered pair $\langle \alpha A \beta, \alpha \varphi \beta \rangle$ where

A is a variable in V and α, β & φ are strings from $(V \cup T)^*$.

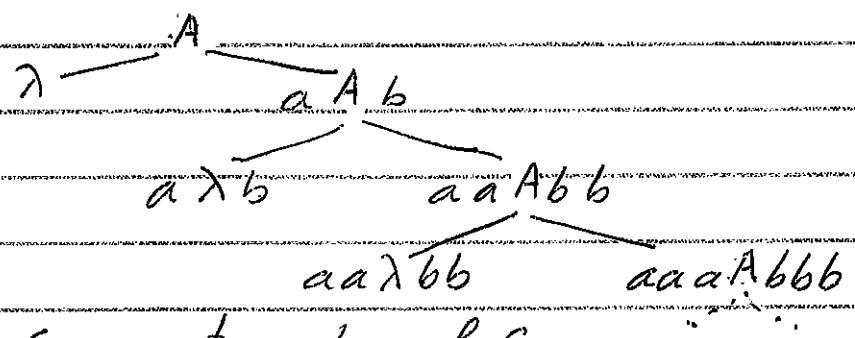
We usually write the production $\langle \alpha A \beta, \alpha \varphi \beta \rangle$ as $\alpha A \beta \to \alpha \varphi \beta$ and interpret it as meaning that A can be replace by φ when it is in the context α followed by β. If α and β are both λ, then $\alpha A \beta = A$ and the production $A \to \varphi$ is said to be <u>context-free</u>. $\mathcal{S}$ usually consists of a single variable – just like a natural grammar like English gramma

Ex1. Let $G_1 = \langle V, T, \mathcal{S}, \mathcal{P} \rangle$ where

$V = \{A\}$, $T = \{a, b\}$, $\mathcal{S} = \{A\}$ and

$\mathcal{P} = \{A \to aAb,\ A \to \lambda\}$.

Then G is a PSG. The set of all strings of terminal symbols that can be "generated" by G will be denoted by $L(G_1)$.



Generating tree of G_1.

$L(G_1) = \{\lambda, ab, aabb, a^3b^3, \ldots\}$

$= \{a^n b^n : n \geq 0\}$.

We will now specify what "generate" means and use it to precisely define $L(G_1)$. We often call a phrase structured grammar just a <u>grammar</u>

Let G be a PSG and $\alpha A \beta \to \alpha \varphi \beta$ be a production of G. Also let $w_1 = \psi_1 \alpha A \beta \psi_2 \in (V \cup T)^*$ & $w_2 = \psi_1 \alpha \varphi \beta \psi_2 \in (V \cup T)^*$. We say that w_2 is <u>immediately derivable</u> from w_1 and write $w_1 \underset{G}{\Rightarrow} w_2$ to mean that w_1 can be transformed into w_2 in G. If $w_1, w_2, \ldots, w_n$ is a sequence of strings in $(V \cup T)^*$ such that

$$w_1 \underset{G}{\Rightarrow} w_2,\ w_2 \underset{G}{\Rightarrow} w_3,\ \ldots,\ w_{n-1} \underset{G}{\Rightarrow} w_n$$

then say that w_n is <u>derivable</u> from w_1 and write $w_1 \overset{*}{\underset{G}{\Rightarrow}} w_n$ to mean that w_1 can be transformed into w_n in G in a finite number of steps. We also say that $w_1 \overset{*}{\underset{G}{\Rightarrow}} w_1$ for any PSG, G in zero steps.

The sequence $\langle w_1, w_2, \ldots, w_n \rangle$ is called a <u>derivation</u> of w_n from w_1 in G. We often leave out the "G" from under the "$\Rightarrow$" when we are discussing only a single PSG. Also we often write a derivation $\langle w_1, w_2, \ldots w_n \rangle$ as $w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$.

<u>Def.</u> The language defined by a grammar G (we often say "grammar" instead of PSG) is defined by

$$L(G) = \{ w \in T^* : w \text{ is derivable from at least one string in } \mathcal{S}(G) \}$$

In other words $L(G)$ = set of all terminal strings that are derivable from $\mathcal{S}(G)$. We also say that G <u>generates</u> the language $L(G)$.

Notational conventions: In order to be able specify grammars more simply, we shall use the following conventions. We will denote

a) Variables by upper case Roman letters: $A, B, C, \ldots$
b) Terminal symbols by lower case Roman letters: $0, 1, a, b, c, \ldots$
c) strings from $(V \cup T)^*$ by $\varphi, \psi, \chi, \omega \ldots$
and strings from T^* by $\alpha, \beta, \gamma, \ldots$.

We will also put an arrow "$\rightarrow$" infront of a string to indicate that it is a starting string in $\mathcal{S}(G)$.

Ex. 2 Let G_2 be the grammar with starting strings & productions as follows:

$$\rightarrow A, \ \rightarrow \lambda, \ \rightarrow ab, \ A \rightarrow aB, \ B \rightarrow bB, \ B \rightarrow aC, \ C \rightarrow b.$$

Then $V(G_2) = \{A, B, C\}$, $T(G_2) = \{a, b\}$, $\mathcal{S}(G_2) = \{\lambda, ab, A\}$ and $\mathcal{P}(G) = \{A \rightarrow aB, B \rightarrow bB, B \rightarrow aC, C \rightarrow b\}$.

$$\begin{array}{ll}
A & \\
| & \leftarrow A \rightarrow aB \\
aB & \\
| & \\
abB & \\
\vdots & \\
ab^nB & \\
| & \leftarrow B \rightarrow aC \\
ab^naC & \\
| & \leftarrow C \rightarrow b \\
ab^nab &
\end{array}
\qquad
\left.\begin{array}{l} aB \\ | \\ abB \\ \vdots \\ ab^nB \end{array}\right\} B \rightarrow bB \ (n \text{ times})$$

So $L(G_2) = \{\lambda, ab\} \cup \{ab^nab : n \geq 0\}$

We can write the answer in this case by using the regular expression,

$$\lambda + ab + ab^*ab$$

but most languages generated by PSGs cannot be described by regular expressions.

§ 2. Classification of Phrase-Structured Grammars

The phrase-structure grammars can be classified according to the type of productions that are allowed.

	TYPE	PRODUCTIONS ALLOWED
PSG	Phrase-structured	$\psi A \chi \to \psi \varphi \chi$ with $A \in V$, and $\varphi, \psi, \chi \in (V \cup T)^*$.
CSG	Context-sensitive	$\psi A \chi \to \psi \varphi \chi$ with $\varphi \neq \lambda$; and $A \in V$ & $\varphi, \psi, \chi \in (V \cup T)^*$.
CFG	Context-free	$A \to \varphi$; $A \in V$ & $\varphi \in (V \cup T)^*$
BLG	Bi-linear (or Linear)	$A \to \alpha B \beta$ or $A \to \alpha$; with $A, B \in V$ & $\alpha, \beta \in T^*$
RLG	Right-Linear	$A \to \alpha B$ or $A \to \alpha$; $A, B \in V$ & $\alpha \in T^*$
LLG	Left-Linear	$A \to B\alpha$ or $A \to \alpha$; $A, B \in V$ & $\alpha \in T^*$.

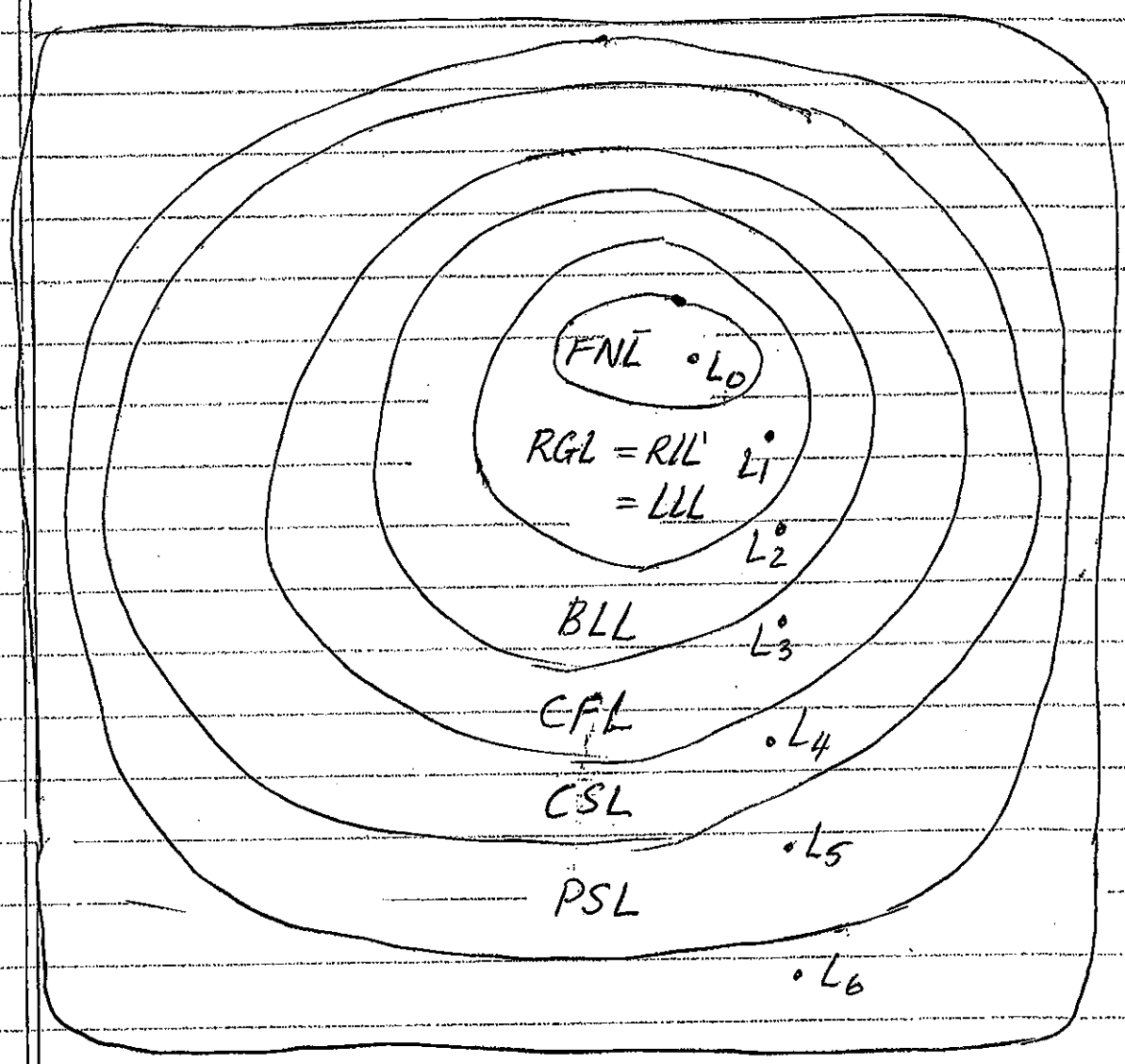
Examples

1.	$\to A$, $A \to 0A$, $A \to 10B$, $B \to \lambda$	RLG
2.	$\to B$, $B \to Ba$, $B \to Cbb$, $C \to a$	LLG
3.	$\to C$, $C \to 0S11$, $C \to 0$	BLG
4.	$\to D$, $D \to aD$, $D \to Db$, $D \to \lambda$	BLG
5.	$\to A$, $A \to AB$, $B \to bB$, $B \to aC$, $C \to \lambda$	CFG
6.	$\to B$, $B \to bC$, $bC \to bb$, $C \to D$, $D \to a$	CSG
7.	$\to C$, $C \to bCD$, $bC \to bb$, $C \to D$, $D \to \lambda$	PSG

Def A language L_1 is said to be phrase-structured if we can find a phrase-structured grammar (PSG) G_1 such that $L(G_1) = L_1$.

We define <u>context-sensitive</u>, <u>context-free</u>, <u>bi-linear</u>, <u>right-linear</u> and <u>left-linear</u> languages in a similar ways by using CSG, CFG, BLG, RLG and LLG instead of PSG. This leads to a strict hierarchy of classes of languages.

Let $V = \{a, b, c\}$. Then $\mathcal{L}(V)$ is as shown below:



FNL = class of finite languages
RGL = class of regular languages
BLL = class of bi-linear languages

The languages L_i $(i = 0, 1, 2, \ldots, 6)$ show that the classes are all non-empty and that the hierarchy is strict. Unfortunately we do not the machinery to describe L_5 and L_6 as yet. Below we will specify $L_0, \ldots, L_4$ and find grammars of the appropriate type for them.

$L_0 = \{a, ab\}$

G_0: $\to A$, $A \to a$, $A \to ab$.

$L_1 = \{a^n : n \geq 1\}$

G_1: $\to B$, $B \to aB$, $B \to a$

$L_2 = \{a^n b^n : n \geq 1\}$

G_2: $\to C$, $C \to aCb$, $C \to ab$.

$L_3 = \{a^n b^n a^k b^k : n \geq 1, k \geq 1\}$

G_3: $\to A$, $A \to BC$, $B \to aBb$, $C \to aCb$, $B \to ab$, $C \to ab$.

$L_4 = \{a^n b^n c^n : n \geq 1\}$. The CSG for L_4 is a little bit complicated.

G_4: $\to A$, $A \to abC$, $A \to aABC$, $bB \to bb$, $bC \to bc$, $cC \to cc$, $\underbrace{CB \to XB,\ XB \to XY,\ XY \to BY,\ BY \to BC}_{\text{equiv. to } CB \to BC \text{ but this is not a legal production.}}$.

Two derivations in G_4.

$\to A \Rightarrow abC \Rightarrow abc$

$\to A \Rightarrow aABC \Rightarrow aabCBC \Rightarrow aabXBC \Rightarrow aabXYC \Rightarrow aabBYC$
$\Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow aabbcc.$

We will now find grammars of various types which generate certain specific languages.

Ex. 1(a) Find an RLG which generates the language
$L_1 = \mathcal{L}(\underline{a}^* \underline{c} (\underline{ba})^* \underline{ab}) = \{a^k c (ba)^l ab : k, l \geq 0\}$.

G_a : $\to A$, $A \to aA$, $A \to cB$, $B \to baB$, $B \to ab$

Ex 1(b) Find an LLG which generates the language
$L_1 = \mathcal{L}(\underline{a}^* \underline{c} (\underline{ba})^* \underline{ab})$

G_b : $\to B$, $B \to Cab$, $C \to Cba$, $C \to Dc$,
$D \to Da$, $D \to \lambda$

Ex. 2 Find a CFG which generates the language
$L_2 = \{a^n b^k : k \geq 2n,\ n \geq 0\}$.

G_2 : $\to C$, $C \to aCbb$, $C \to D$,
$D \to Db$, $D \to \lambda$

$$\begin{array}{l} C \\ \vdots \\ aCbb \\ \vdots \\ a^n C b^{2n} \end{array} \Big\} \text{ n times}$$

Ex. 3 Find a CFG which generates the language
$L_3 = \{a^n b^k : n \geq 0,\ 0 \leq k \leq 2n\}$.

G_3 : $\to D$, $D \to aDBB$, $D \to \lambda$,
$B \to b$, $B \to \lambda$.

$$\begin{array}{l} D \\ \vdots \\ aDBB \\ \vdots \\ a^n D B^{2n} \\ \vdots \\ a^n . \lambda . b^k, \quad k \leq 2n \end{array}$$

$$\begin{array}{l} a^n D b^{2n} \\ a^n D b b^{2n} \\ \vdots \\ a^n D b^l b^{2n} \\ \vdots \\ a^n . \lambda . b^{2n+l} \end{array} \Big\} \ l \text{ times}$$

Note: G_2' : $\to C$, $C \to aCbb$, $C \to bC$, $C \to \lambda$ does not generate L_2
but G_2'' : $\to C$, $C \to aCbb$, $C \to Cb$, $C \to \lambda$ generates L_2.

§ 3. Parsing & Ambiguity in CFGs

PSGs are the most general grammars that we consider. CSGs are almost as complicated as PSGs but they are needed to understand the grammars of natural languages. For the rest of the chapter we shall only consider CFGs because they are a lot simpler and are exactly the kind of grammars that we need in Computer Science for designing Programming languages. (RLGs are very simple CFGs.)

Qu: Suppose G is a CFG and φ is a terminal string. How can we tell if $\varphi \in L(G)$? And if $\varphi \in L(G)$, how can we find if there is essentially more than one derivation of φ in G?

Ans: If G is of a CFG of special format, then there are parsing algorithms that can tell us whether or not $\varphi \in L(G)$. The parsing algorithms is basically a breadth-first search algorithm and after a certain point, if we do not find a derivation of φ, then it means there is none. But the CFG has to be of this special format, otherwise the algorithm won't work. (Parsing is the process of understanding how a terminal string was generated.) So we will now indicate what these special formats are. They are called normal forms of CFGs.

Def Two grammars G_1 & G_2 are equivalent if $L(G_1) = L(G_2)$.

Def. Let G be a CFG. An <u>unreachable production</u> of G is one which involves a variable that cannot be reached from the starting strings of G. (To be more specific $B \to \varphi$ is unreachable if none of the strings that are derivable in G includes the variable B.)

Ex. 1 G_1: $\to A$, $A \to aC$, $B \to bB$, $C \to Cb$, $C \to \lambda$, $B \to a$
In G_1, $B \to bB$ and $B \to a$ are unreachable prod.
An equivalent CFG G_1' can be obtained by deleting these two unreachable productions
G_1': $\to A$, $A \to aC$, $C \to Cb$, $C \to \lambda$

Def. A <u>non-terminating</u> production is one which involves a variable which does not eventually terminate. (Again to be more specific, $C \to \varphi$ is non-terminating if there is no terminal string that can be derived by starting with $C \to \varphi$.)

Ex. 2 G_2: $\to B$, $B \to aC$, $D \to bD$, $B \to Eb$, $E \to aE$, $E \to \lambda$.
Here $B \to aC$ and $D \to bD$ are non-terminating productions. ($D \to bD$ is also unreachable.)
An equiv. CFG G_2' is as shown below:
G_2': $\to B$, $B \to Eb$, $E \to aE$, $E \to \lambda$.

Def. A <u>useless production</u> in a CFG is one that is unreachable or one that is non-terminating.

Def. A <u>unit production</u> is one of the form $B \to C$ where B and C are variables.

Ex.3 G_3: $\to A$, $A \to aB$, $B \to C$, $C \to Cb$, $C \to \lambda$
Here $B \to C$ is a unit production. We can eliminate $B \to C$ to get an equivalent CFG
G_3': $\to A$, $A \to aB$, $B \to Cb$, $B \to \lambda$, $C \to Cb$, $C \to \lambda$

Def. A λ-production is one of the form $B \to \lambda$ where B is a variable.

Ex.4 G_4: $\to A$, $A \to bA$, $A \to aB$, $B \to Bab$, $B \to \lambda$
Here $B \to \lambda$ is a λ-production. We can eliminate $B \to \lambda$ to get an equivalent CFG
G_4': $\to A$, $A \to bA$, $A \to aB$, $A \to a$, $B \to Bab$, $B \to ab$

Def. A <u>terminal production</u> in a CFG is a production of the form $B \to \alpha$ where $B \in V$ and $\alpha \in T^*$.
A CFG G is <u>increasing</u> if whenever $A \to \varphi \in G$, then $|\varphi| \geq 1$. It is <u>strictly increasing</u> if it is increasing and $A \to \varphi \in G$ implies $|\varphi| \geq 2$.

The parsing algorithms will work for any strictly increasing CFG. We can also prove the following.

<u>Normal Form Theorem for CFG</u>: Any CFG G is equivalent to one in which $\mathcal{S}(G) \subseteq \{A, \lambda\}$ and all productions are of the form $B \to CD$ or $B \to a$ with $B, C, D \in V$ and $a \in T$.

(12)

Consider the grammar G below.

G: $\to A$, $A \to BC$, $B \to aBb$, $B \to \lambda$, $C \to aCb$, $C \to \lambda$.

Let $\varphi = abab$. Then

(a) $\to A \Rightarrow BC \Rightarrow aBbC \Rightarrow abC \Rightarrow abaCb \Rightarrow abab$ &

(b) $\to A \Rightarrow BC \Rightarrow BaCb \Rightarrow Bab \Rightarrow aBbab \Rightarrow abab$

are both derivations of φ in G.

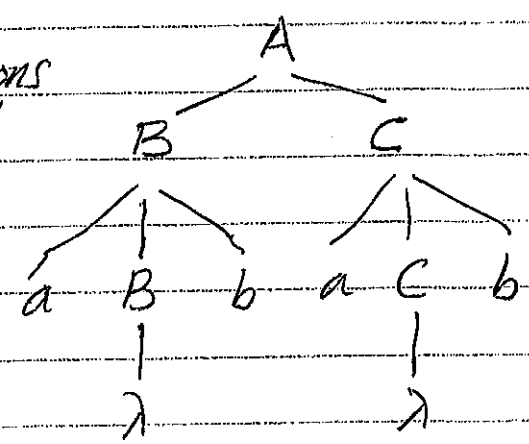
Although (a) & (b) look different, they are <u>essentially</u> the same derivation. (a) is called a <u>left-most</u> derivation because the left-most variable is replaced at each step. (b) is called a <u>right-most</u> derivation for a similar reason. We can even have other derivations of φ such as:

(c) $\to A \Rightarrow BC \Rightarrow BaCb \Rightarrow aBbaCb \Rightarrow aBbab \Rightarrow abab$

which is neither leftmost nor rightmost.

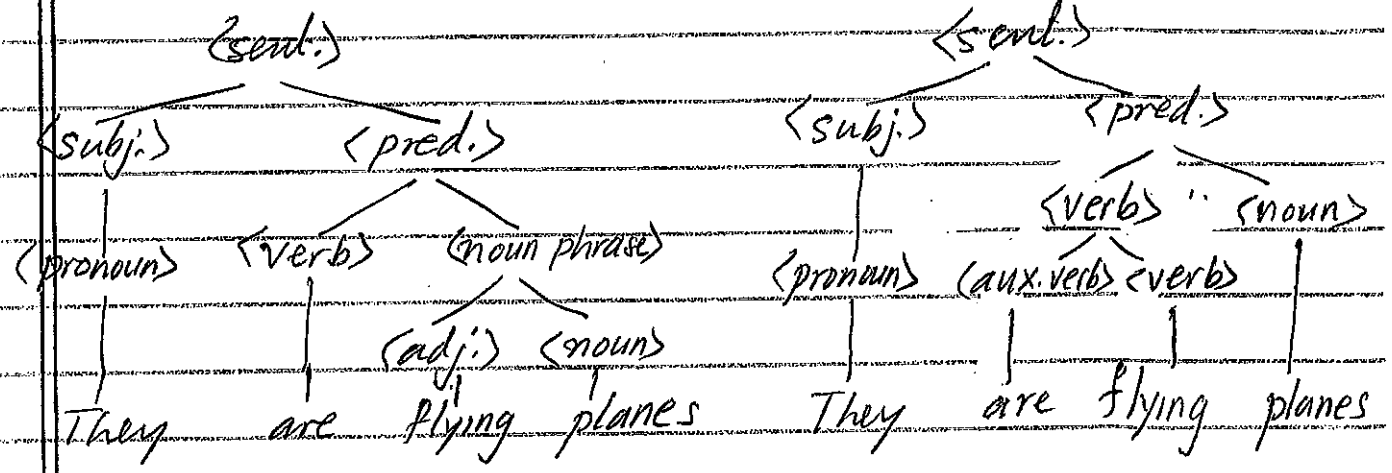
We can see that all three derivations of φ are <u>essentially</u> the same by drawing the <u>derivation tree</u> of each of the derivations (a), (b) & (c).

For each of the derivations (a), (b) & (c) we will end up with the same tree on the right.



The string derived is $a\lambda ba\lambda b = abab$.

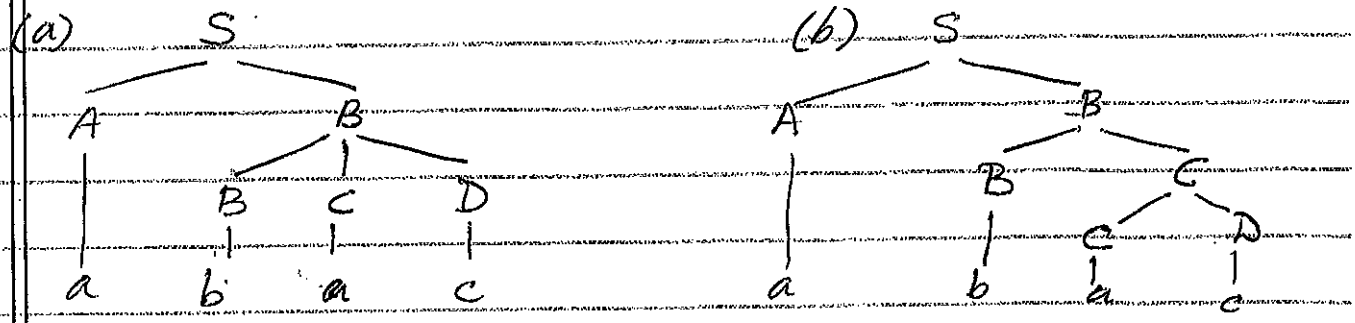
Consider the sentence, "They are flying planes". This sentence can be parsed in two different ways and each way has a different meaning.



So we say that the sentence is ambiguous.

Def. A CFG G is <u>ambiguous</u> if it generates a string φ which has at least two non-identical (different) derivation trees. (Equivalently, we can say that G is ambiguous if it generates a string φ which has at least 2 leftmost derivations.) A CFG is <u>un-ambiguous</u> if it is <u>not</u> ambiguous.

Ex. 1 G : $\to S$, $S \to AB$, $A \to a$, $B \to BC$, $B \to BCD$
$B \to b$, $C \to CD$, $C \to a$, $D \to c$.



So G is an ambiguous CFG.

We can also see this by writing down the equivalent left-most derivations.

(a) $\rightarrow S \Rightarrow AB \Rightarrow aB \Rightarrow aBCD \Rightarrow abCD \Rightarrow abaD \Rightarrow abac$

(b) $\rightarrow S \Rightarrow AB \Rightarrow aB \Rightarrow aBC \Rightarrow abC \Rightarrow abCD \Rightarrow abaD \Rightarrow abac$

Now it would be nice if we can find an unambiguous CFG which is equivalent to any ambiguous CFG, but unfortunately, this is not so.

Ex.2 Let $L_2 = \{a^n b^n c^k : n, k \geq 0\} \cup \{a^n b^k c^k : n, k \geq 0\}$. Then the CFG G_2 below will generate L_2.

G_2: $\rightarrow A$, $A \rightarrow B$, $B \rightarrow Bc$, $B \rightarrow C$, $C \rightarrow aCb$, $C \rightarrow \lambda$
$A \rightarrow D$, $D \rightarrow aD$, $D \rightarrow E$, $E \rightarrow bEc$, $E \rightarrow \lambda$

Now G_2 is ambiguous because abc has two different (non-identical) derivation trees.

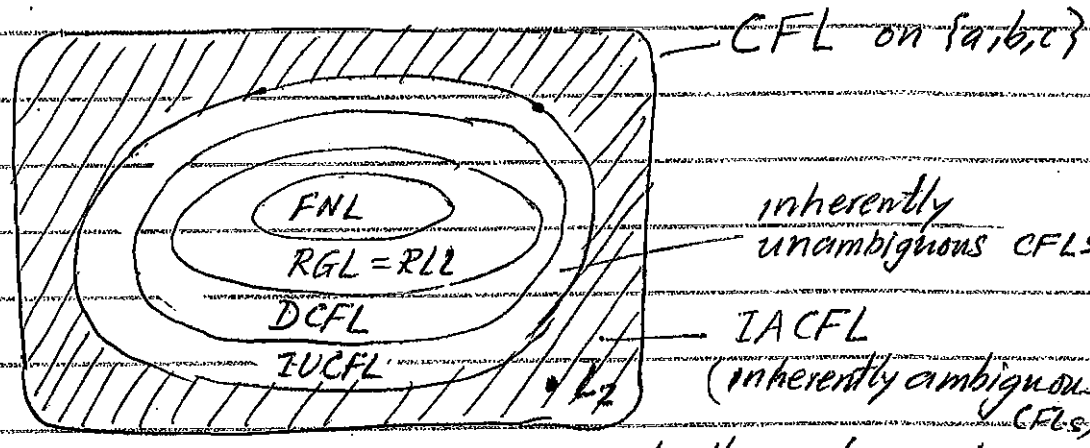
(a) $\rightarrow A \Rightarrow B \Rightarrow Bc \Rightarrow Cc \Rightarrow aCbc \Rightarrow abc$

(b) $\rightarrow A \Rightarrow D \Rightarrow aD \Rightarrow aE \Rightarrow abEc \Rightarrow abc$.

But it can be shown that there is no un-ambiguous CFG which can generated L_2. In otherwords every CFG which generates L_2 is ambiguous.

Def. A context-free language (CFL) L is <u>inherently ambiguous</u> if there is no un-ambiguous CFG which can generate L.

It can be shown that any ambiguous RLG G equivalent to an un-ambiguous RLG G' and we will see this in Ch.4.



We cannot say what DCFL are at this stage because it depends on theory of machines known as DPDA.

§4. Other Properties of CFG's and RLGs

It is easy to show that any CFG is equivalent to one in which the set of starting strings consist of just one variable. From now on we shall assume this.

Qu1 Given CFGs G_1 and G_2, how can we find CFGs which will generate

(a) $L(G_1) \cup L(G_2)$? (b) $L(G_1) . L(G_2)$? (c) $L(G_1)^*$?

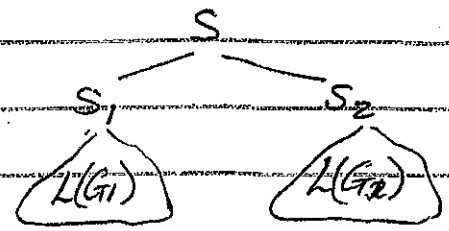
H.W.

Add (d) $L(G_1)^R$ as H.W.

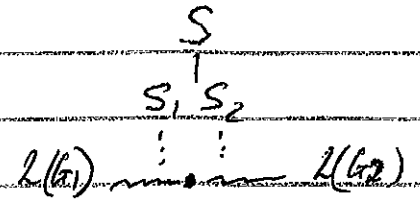
Let $G_1 = \langle V_1, T_1, \{S_1\}, P_1 \rangle$ & $G_2 = \langle V_2, T_2, \{S_2\}, P_2 \rangle$.

First we must rename the variables in V_2 to get V_2' to ensure that $V_1 \cap V_2' = \phi$. Also let P_2' be P_2 with the renamed variables and S be a new variable which is not already in $V_1 \cup V_2'$.

(a) Put $G_a = \langle V_1 \cup V_2' \cup \{S\},\ T_1 \cup T_2,\ \{S\},\ \mathcal{P} \rangle$ where
$\mathcal{P} = \{S \to S_1,\ S \to S_2\} \cup \mathcal{P}_1 \cup \mathcal{P}_2'$.
Then $L(G_a) = L(G_1) \cup L(G_2)$.



(b) Put $G_b = \langle V_1 \cup V_2' \cup \{S\},\ T_1 \cup T_2,\ \{S\},\ \mathcal{P} \rangle$
where $\mathcal{P} = \{S \to S_1 S_2\} \cup \mathcal{P}_1 \cup \mathcal{P}_2'$.
Then $L(G_b) = L(G_1) . L(G_2)$



(c) Put $G_c = \langle V_1 \cup \{S\},\ T_1,\ \{S\},\ \mathcal{P} \rangle$
where $\mathcal{P} = \{S \to \lambda,\ S \to SS_1\} \cup \mathcal{P}_1$. Then $L(G_c) = L(G_1)^*$

Ex.1 G_1: $S_1 \to aA$, $A \to bA$, $A \to aaB$, $B \to bB$, $B \to \lambda$
G_2: $S_2 \to bA$, $A \to aA$, $A \to bC$, $C \to baC$, $C \to \lambda$
Then $V_1 = \{S_1, A, B\}$ & $V_2 = \{S_2, A, C\}$. We rename the A in V_2 as A' to get $V_2' = \{S_2, A', C\}$.

(a) G_a: $S \to S_1$, $S_1 \to aA$, $A \to bA$, $A \to aaB$, $B \to bB$, $B \to \lambda$,
$S \to S_2$, $S_2 \to bA'$, $A' \to aA'$, $A' \to bC$, $C \to baC$, $C \to \lambda$.
Then $L(G_a) = L(G_1) \cup L(G_2)$

(b) G_b: $S \to S_1 S_2$, $S_1 \to aA$, $A \to bA$, $A \to aaB$, $B \to bB$, $B \to \lambda$,
$S_2 \to bA'$, $A' \to aA'$, $A' \to bC$, $C \to baC$, $C \to \lambda$.
Then $L(G_b) = L(G_1) . L(G_2)$.

(c) G_c: $S \to \lambda$, $S \to SS_1$, $S_1 \to aA$, $A \to bA$, $A \to aaB$, $B \to bB$, $B \to \lambda$.
Then $L(G_c) = L(G)^*$.

Qu2 Given RLGs G_1 & G_2, how can we find RLGs which will generate
(a) $L(G_1) \cup L(G_2)$? (b) $L(G_1) . L(G_2)$? (c) $L(G_1)^*$?

This is the same question as the previous one except that CFGs are replaced by RLGs. The solution to part (a) is the same because $S \to S_1$ & $S \to S_2$ are allowed in a RLG — but the solutions to parts (b) & (c) have to be changed because $S \to S_1 S_2$ & $S \to SS_1$ are not allowed in a RLG.

2(b) Put $G_b = \langle V_1 \cup V_2' \cup \{S\}, T_1 \cup T_2, \{S\}, \mathcal{P} \rangle$ where $\mathcal{P} = \{S \to S_1\} \cup \mathcal{P}_1'' \cup \mathcal{P}_2'$. Here $\mathcal{P}_1''$ consists of the same productions as $\mathcal{P}_1$ except that all the terminal productions will have an S_2 added to their right-hand side. (If $A \to a$ or $A \to \lambda$ is in $\mathcal{P}_1$, then $A \to aS_2$ & $A \to \lambda S_2 \in \mathcal{P}_1''$.)

(c) Put $G_c = \langle V_1 \cup \{S\}, T_1, \{S\}, \mathcal{P} \rangle$ where $\mathcal{P} = \{S \to \lambda, S \to S_1\} \cup \mathcal{P}_1'''$. Here $\mathcal{P}_1'''$ consists of the same productions as $\mathcal{P}_1$ except all terminal productions will have S added to their right-hand side.

Ex. 2 G_1: $S_1 \to aA$, $A \to cA$, $A \to aaB$, $B \to ba$, $B \to \lambda$
G_2: $S_2 \to bA'$, $A \to abA$, $A \to bC$, $C \to b$, $C \to \lambda$

RLG for $L(G_1) \cup L(G_2)$ (a) G_a: $S \to S_1$, $S_1 \to aA$, $A \to cA$, $A \to aaB$, $B \to ba$, $B \to \lambda$,
$S \to S_2$, $S_2 \to bA'$, $A' \to abA'$, $A' \to bC$, $C \to b$, $C \to \lambda$.

RLG for $L(G_1) \cdot L(G_2)$ (b) G_b: $S \to S_1$, $S_1 \to aA$, $A \to cA$, $A \to aaB$, $B \to baS_2$, $B \to S_2$,
$S_2 \to bA'$, $A' \to abA'$, $A' \to bC$, $C \to b$, $C \to \lambda$.

RLG for $L(G_1)^*$. (c) G_c: $S \to \lambda$, $S_1 \to aA$, $A \to cA$, $A \to aaB$, $B \to baS$, $B \to S$,
$S \to S_1$. Then $L(G_c) = L(G_1)^*$.

Notice how we had to change the variable A in G_2 into A' to avoid mixing it up with the A from G_1.

END OF Ch. 2