CONTEXT FREE GRAMMARS/LANGUAGES and PUSH DOWN AUTOMATA

We'll now study the class of "context free languages" (CFLs). This class

- includes the class of regular languages.
- is characterized by "context free grammars" (CFGs) in syntactic manner
- is characterized (equivalently) by "push down automata" (PDAs) in computational manner.

We start with grammars (CFGs).

Motivating example from natural language (English)

Sentence $\longrightarrow$ Subject Verb Object

Sentence $\longrightarrow$ Sentence and Sentence

Subject $\longrightarrow$ Jim | Lisa

Verb $\longrightarrow$ ate | threw | killed

Object $\longrightarrow$ apple | ball | rat

This grammar generates sentences such as

- Jim ate apple
- Jim ate apple and Lisa threw ball

- Lisa threw apple and Jim killed rat and Jim ate ball

Fact Programming Languages (such as C) are described by a grammar that generates all syntactically correct programs.

Exercise Look up the C-grammar.

Example of a CFG

$$G: \quad \begin{aligned} S &\to A \\ A &\to 0A1 \\ A &\to \# \end{aligned}$$

- 3 "rules".
- S, A variables
- S : start variable
- 0, 1, # terminals

Idea Start with the string S.

At any step, replace a variable Var by string α if $\text{var} \to \alpha$ is a rule.

Stop if the string only has terminals and no variable.

Eg. $S \to A \to 0A1 \to 00A11 \to 00\#11$

We say that grammar G derives generates string 00#11

Clearly this grammar generates precisely the set of strings (referred to as the language $L(G)$ generated by the grammar)
described

$$L(G) = \{ 0^n \# 1^n \mid n \geq 0 \}.$$

Note that $L(G)$ is not regular.

<u>Parse tree</u>

The parse tree shows graphical (and structural) representation of the derivation. (of string <u>000#111</u> here).

<u>Formal Definition of CFG</u>

<u>Def</u> A CFG G is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables
- Σ " of terminals, disjoint from V.
- $S \in V$ is the start variable.

- $\mathcal{R}$ is a finite set of rules of type

Variable $\rightarrow$ String of variables and terminals

i.e. $A \rightarrow (\mathcal{V} \cup \Sigma)^*$, $A \in \mathcal{V}$.

Usually the start variable appears as the left side of the topmost/first rule.

<u>Def</u> Suppose $u, v, w \in (\mathcal{V} \cup \Sigma)^*$ and $A \to w$ is a rule. Then we say that

$uAv \Rightarrow uwv$ "uAv yields uwv".

I.e. one substitutes A by the right side of the rule w.

<u>Def</u> $u \overset{*}{\Rightarrow} v$ if there is a (finite) sequence of strings $u_1, \ldots, u_k \in (\mathcal{V} \cup \Sigma)^*$ such that

$$u_1 = u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v.$$

<u>Def</u> Language generated by grammar G is

$$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}.$$

<u>Note</u>. $L(G)$ is (only) over the alphabet Σ of terminals.

<u>Example</u>

G:

$$S \to (S)$$
$$S \to SS \qquad \Sigma = \{ (,) \}.$$
$$S \to \varepsilon$$

E.g. $\underline{S} \to (\underline{S}) \to (\underline{S}S) \to ((S)\underline{S})$

$\to ((\underline{S})(S)) \to (()(\underline{S})) \to (()())$

$L(G)$ is the language of all "matched parentheses".

The substituted variable is underlined in each step above.

<u>Example</u>

G_{Add}:

$$E \to E+E \qquad \Sigma = \{ +, 1, 2, 3 \}$$
$$E \to 1 \mid 2 \mid 3$$

$L(G_{Add})$ is all expressions of type

$1 \qquad 1+2 \qquad 1+2+3 \qquad 2+1+1+3+2$ etc.

Example

$G_{Add\text{-}Mult}$:

$$E \to E + E \qquad \Sigma = \{+, \times, 1, 2, 3\}$$
$$E \to E \times E$$
$$E \to 1 \mid 2 \mid 3$$

$L(G_{Add\cdot Mult})$ has arithmetic expressions such as $1 + 2 \times 3$ $\quad 2+3 \times 1+2$ $\quad$ etc.

$$\underline{E} \to \underline{E} \times E \to \underline{E} + E \times E \to 1 + E \times \underline{E}$$
$$\to 1 + \underline{E} \times 3 \to 1 + 2 \times 3$$

Example

G_{Arith} :

$$E \to (E + E)$$
$$E \to E \times E \qquad \Sigma = \{+, \times, (,), 1, 2, 3\}$$
$$E \to 1 \mid 2 \mid 3$$

$L(G_{Arith})$ has arithmetic expressions with parentheses, e.g. $(1+2) \times 3$

$$\underline{E} \to \underline{E} \times E \to (\underline{E} + E) \times E \to (1 + \underline{E}) \times E$$
$$\to (1+2) \times \underline{E} \to (1+2) \times 3$$

Example

$G_{\text{Arith-Int}}$:

$$E \to (E+E)$$
$$E \to E \times E$$
$$E \to I$$
$$I \to 0 \mid 1T \mid 2T \mid \cdots \mid 9T$$
$$T \to \varepsilon \mid T0 \mid T1 \mid \cdots \mid T9$$

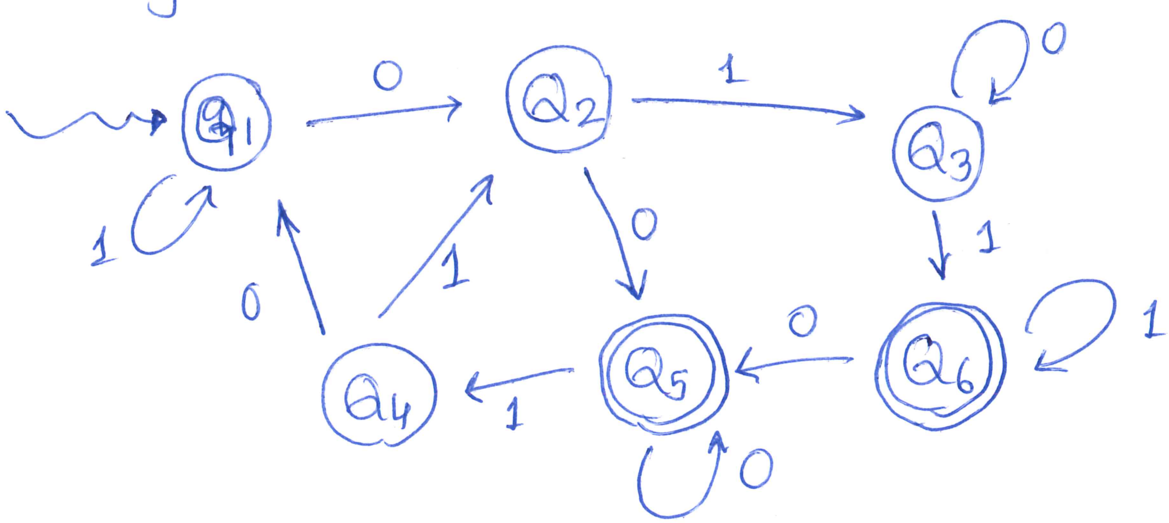
This generates expressions with decimal integers

e.g. $(214 + 109) \times (51080 + 76541)$

It is easy to show that every regular language is context free.

Theorem Every regular language is context free.

Proof (by example). Suppose a language L such as is recognized by a DFA



The following grammar G "simulates" the DFA.

$$G: \quad \left.\begin{array}{l} Q_1 \to 0Q_2 \mid 1Q_1 \\ Q_2 \to 0Q_5 \mid 1Q_3 \\ Q_3 \to 0Q_3 \mid 1Q_6 \\ Q_4 \to 0Q_1 \mid 1Q_2 \\ Q_5 \to 0Q_5 \mid 1Q_4 \\ Q_6 \to 0Q_5 \mid 1Q_6 \end{array}\right\} \text{simulate transitions of the DFA}$$

$$\left.\boxed{\begin{array}{l} Q_5 \to \varepsilon \\ Q_6 \to \varepsilon \end{array}}\right\} \text{Add these only for accept states of the DFA.}$$

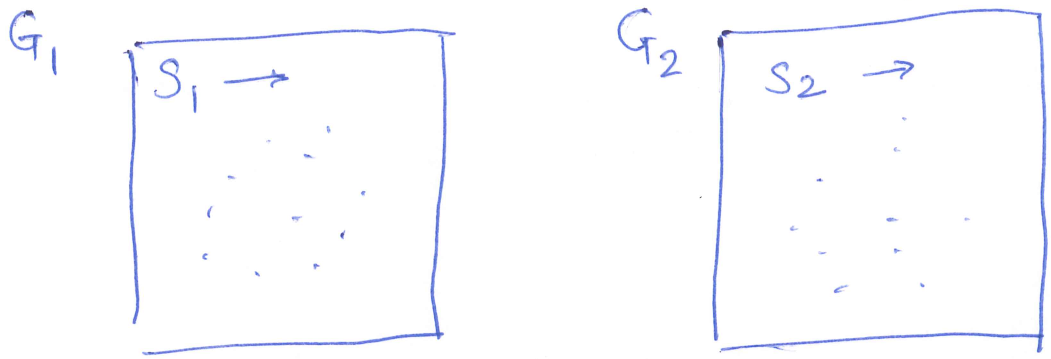
Here $Q_1, Q_2, \ldots, Q_6$ are variables of the grammar and Q_1 is the start variable (state).

<u>Exercise</u> Convince yourself that $L = L(G)$!

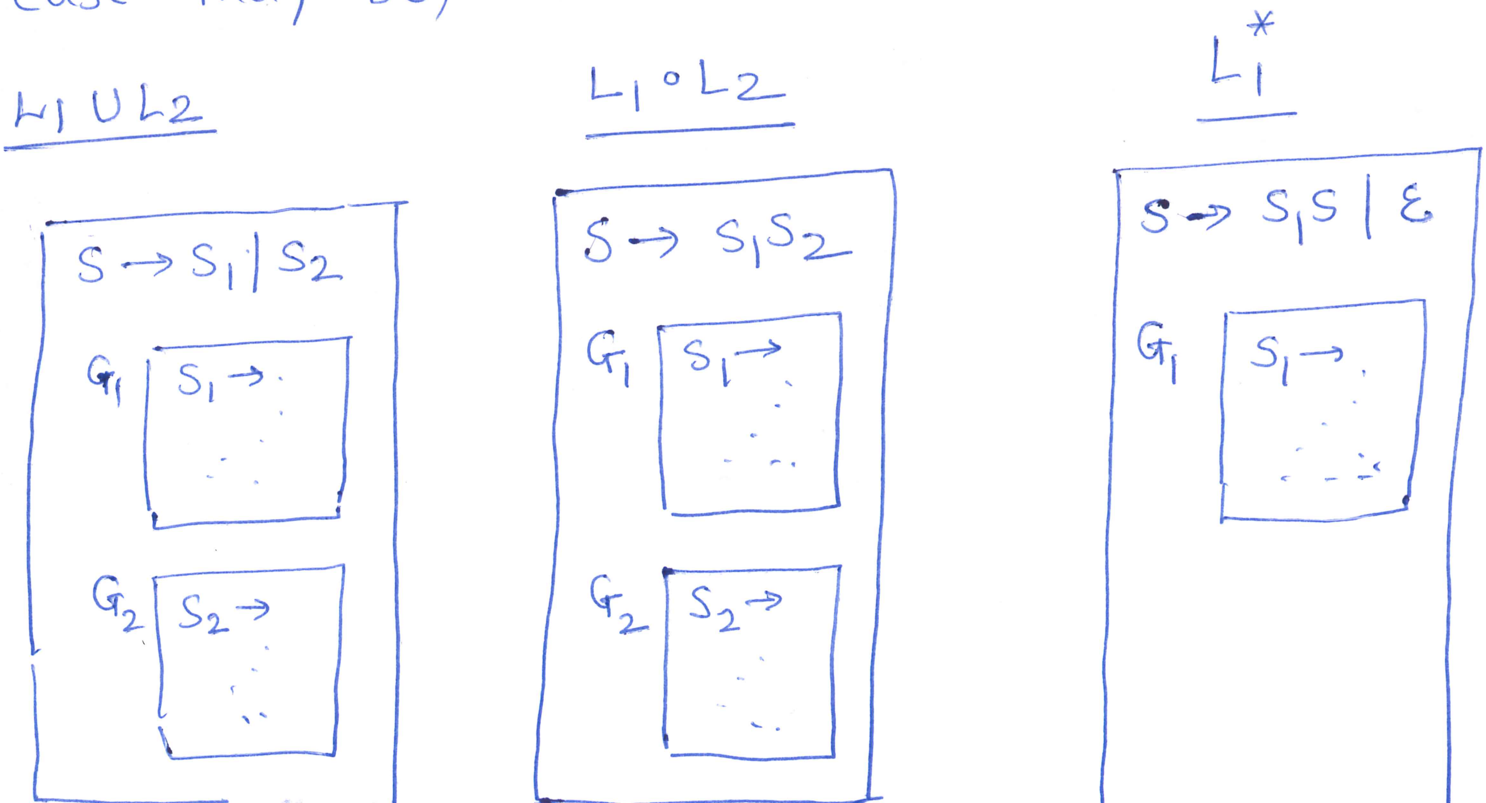
It is easily seen that the class of CFLs is closed under $\cup, \circ, *$.

<u>Theorem</u> If L_1, L_2 are context free languages, then so are $L_1 \cup L_2$, $L_1 \circ L_2$, L_1^*.

<u>Proof</u> Let the grammars for L_1, L_2 be G_1, G_2 respectively.



where S_1, S_2 are their start variables respectively. We may assume that their sets of variables are disjoint (but the set of terminals Σ is the same and L_1, L_2 are languages over Σ). The grammar G, with a new start variable S, for languages $L_1 \cup L_2$, $L_1 \circ L_2$, L_1^* (as the case may be) is:

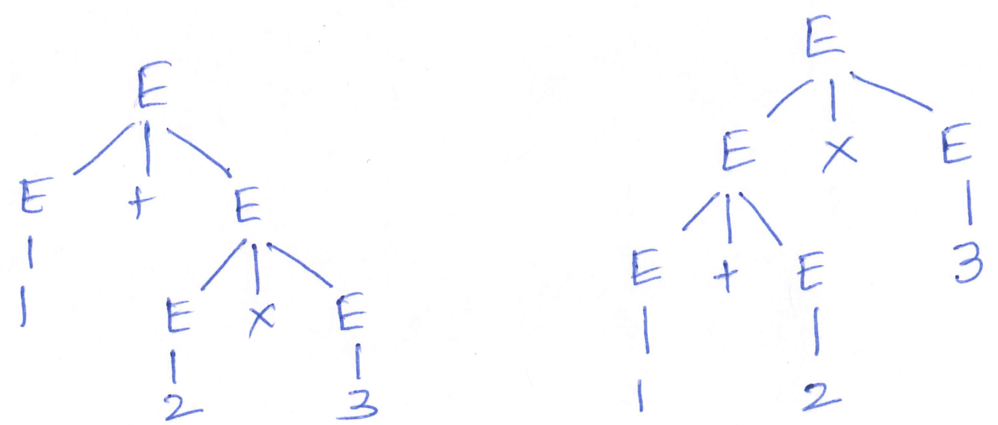


Ambiguity of Grammars

Consider the grammar:

$G_{Add\cdot Mult}$:
$$E \to E+E$$
$$E \to E \times E$$
$$E \to 1 | 2 | 3$$

and the string $1+2\times 3$ generated by it. There are two distinct parse trees that yield this string.



It is not difficult to see that every parse tree corresponds to a unique "leftmost derivation" and vice versa. Here:

<u>Def</u> A derivation is said to be a leftmost derivation if at every step, the leftmost variable is substituted for using a grammar rule.

The leftmost derivations corresponding to the parse trees above are, respectively:

$$\underline{E} \rightarrow \underline{E}+E \rightarrow 1+\underline{E} \rightarrow 1+\underline{E}\times E \rightarrow 1+2\times \underline{E} \rightarrow 1+2\times 3$$

$$\underline{E} \rightarrow \underline{E}\times E \rightarrow \underline{E}+E\times E \rightarrow 1+\underline{E}\times E \rightarrow 1+2\times \underline{E} \rightarrow 1+2\times 3$$

The two derivations are "intrinsically different." If semantics are attached to the terminals $+$, $\times$ then these amount to whether in the expression $1+2\times3$, the addition is evaluated first or the multiplication. The grammar is said to be ambiguous.

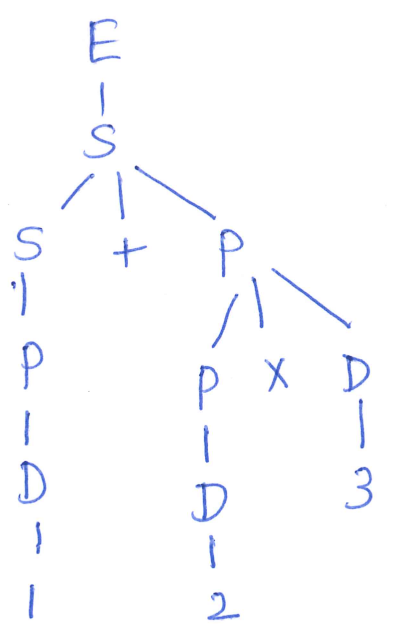
<u>Def</u>. A grammar G is ambiguous if there is <u>some</u> string $w \in L(G)$ that has two (or more) different leftmost derivations (or equivalently parse trees).

It is possible to rewrite this grammar as below so that the new grammar is unambiguous.

$$
\begin{aligned}
G'_{\text{Add mult}} :\quad & E \rightarrow S \\
& S \rightarrow S + P \mid P \\
& P \rightarrow P \times D \mid D \\
& D \rightarrow 1 \mid 2 \mid 3
\end{aligned}
$$

'S' = 'Sum'
'P' = 'Product'
'D' = 'Digit'

The interpretation is that the overall arithmetic expression is viewed as "sum of products". Now the string $1+2\times3$ has only one parse tree and a leftmost derivation:



$$
\begin{aligned}
& \underline{E} \\
\downarrow\ & \underline{S} \\
\downarrow\ & \underline{S}+P \\
\downarrow\ & \underline{P}+P \\
\downarrow\ & \underline{D}+P \\
\downarrow\ & 1+\underline{P} \\
\downarrow\ & 1+\underline{P}\times D \\
\downarrow\ & 1+\underline{D}\times D \\
\downarrow\ & 1+2\times\underline{D} \rightarrow 1+2\times3
\end{aligned}
$$

I.e. the parse tree on the left (before) is "preferred".

Exercise Convince yourself that

- $L(G'_{Add\text{-}Mult}) = L(G)$ and
- $G'_{Add\text{-}Mult}$ is unambiguous.

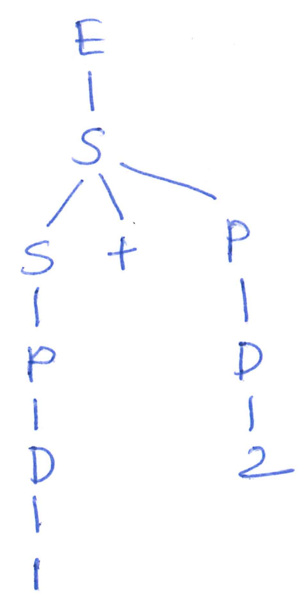
Note We emphasize that $G'_{Add\text{-}Mult}$ may yield two different derivations for the same string, e.g. for the string $1+2$

$$\underline{E} \to \underline{S} \to \underline{S}+P \to \underline{P}+P \to D+\underline{P} \to D+\underline{D} \to \underline{D}+2 \to 1+2$$

$$\underline{E} \to \underline{S} \to S+\underline{P} \to S+\underline{D} \to \underline{S}+2 \to \underline{P}+2 \to \underline{D}+2 \to 1+2$$

However, there is only one leftmost derivation and a parse tree, i.e.

$$\underline{E} \downarrow \underline{S} \downarrow \underline{S}+P \downarrow \underline{P}+P \downarrow \underline{D}+P \downarrow 1+\underline{P} \downarrow 1+\underline{D} \downarrow 1+2$$



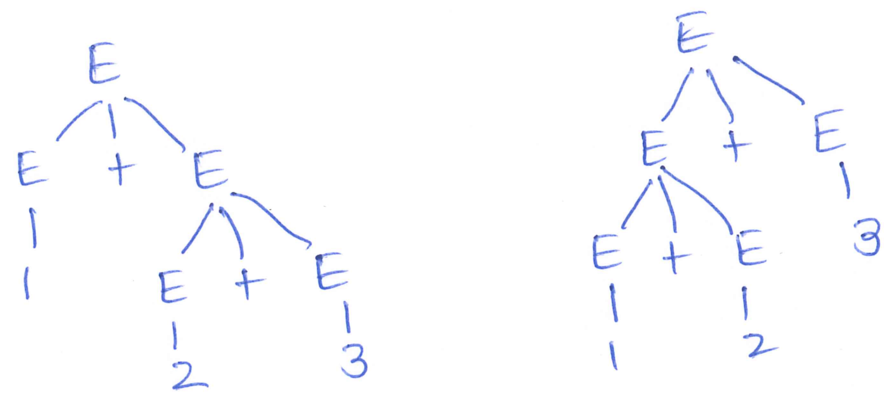
This is why we insist on leftmost derivations (or parse trees).

__Example__ This grammar is ambiguous.

G_{Add} : $E \to E + E$

$E \to 1 | 2 | 3$.

Since $1+2+3$ has two parse trees.



It is easy to rewrite an equivalent, unambiguous grammar.

G'_{Add} : $E \to E + D \mid D$

$D \to 1 | 2 | 3$

__Example__ If-then-else grammar as below is ambiguous.

S → if E then S

S → if E then S else S

S → other

Here if, then, else, other are terminals. 'S' stands for Statement, 'other' stands for other statement, 'E' stands for expression (or logical condition) that we don't really care about.

Consider the following statement.

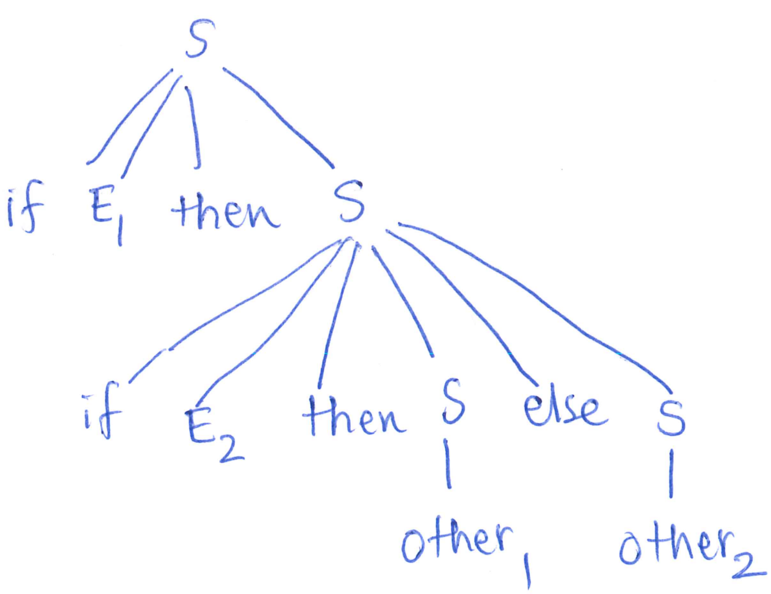
if E_1 then if E_2 then $other_1$ else $other_2$

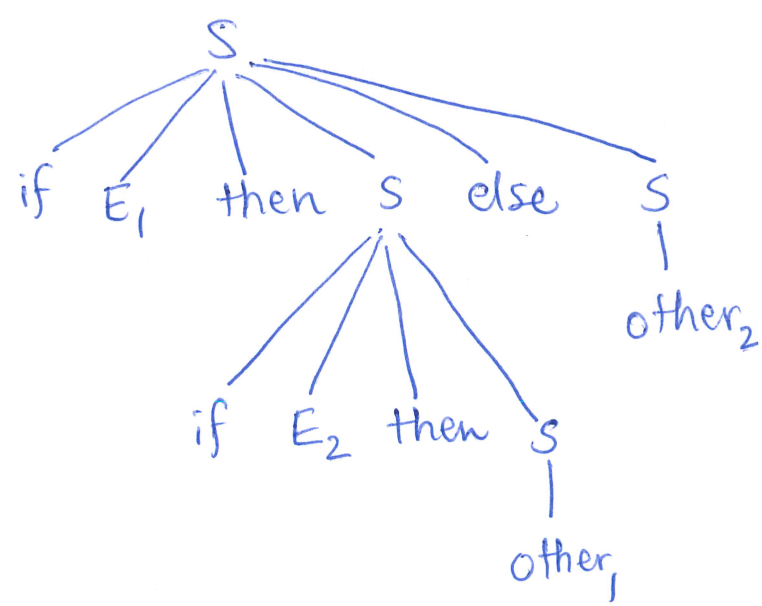
There are two ways to (semantically) interpret the statement:

if E_1 then { if E_2 then $other_1$ else $other_2$ }

if E_1 then { if E_2 then $other_1$ } else $other_2$

These lead to two different parse trees:





Hence the grammar is ambiguous.

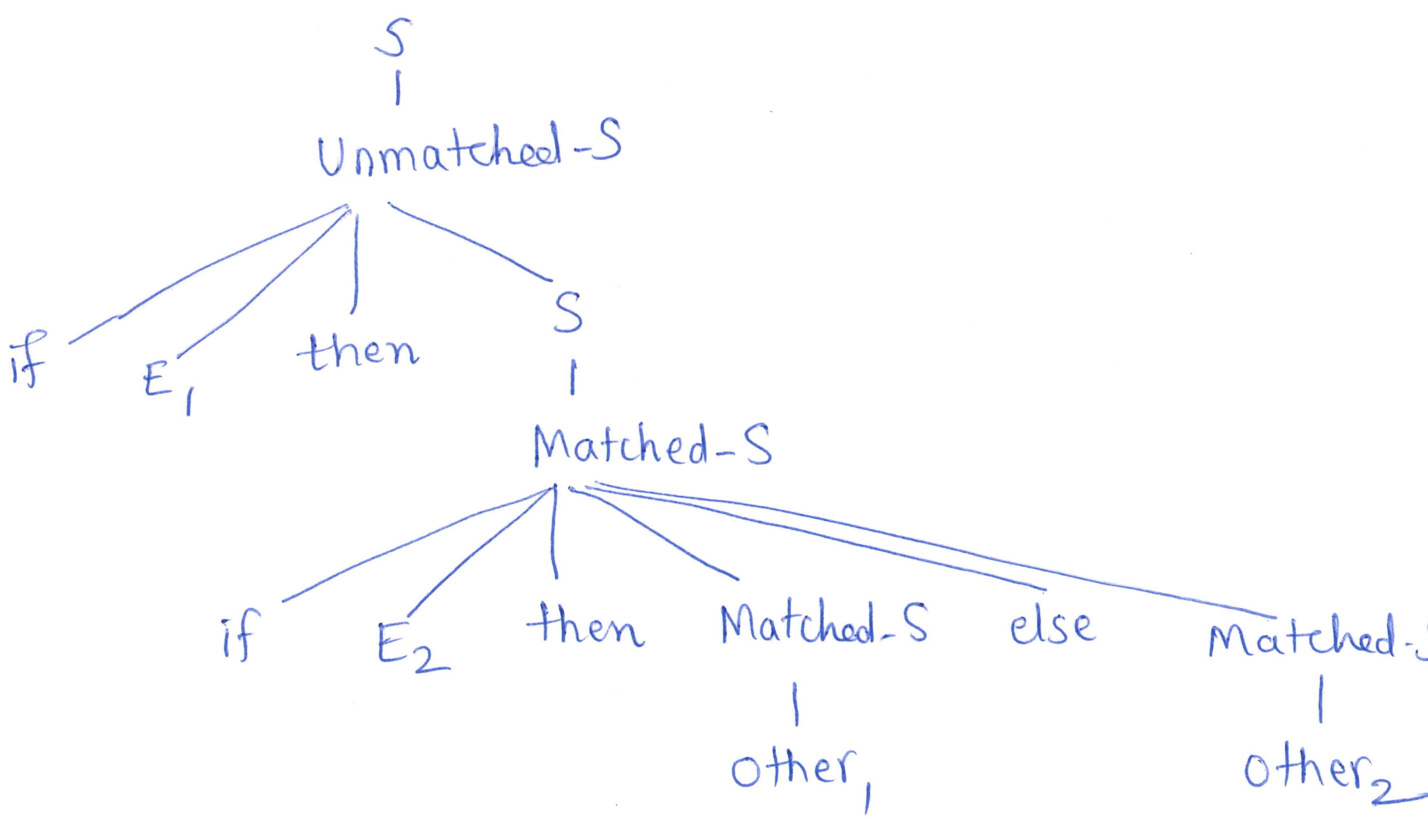
There is a text-book manner to rewrite an equivalent unambiguous grammar.

S → Matched-S | Unmatched-S

Matched-S → if E then Matched-S else Matched-S
| other

Unmatched-S → if E then S |
if E then Matched-S else Unmatched-S

The statement before has the parse tree:



I.e. the left parse tree (before) is "preferred"

Sometimes it is impossible to write a grammar for a CFL that is unambiguous.

Def A CFL is called inherently ambiguous if every grammar for it is ambiguous.

Example $L = \{a^i b^j c^k \mid i=j \text{ or } j=k,\ i,j,k \geq 0\}$ is inherently ambiguous. We'll not prove this in this class.

Note $L_1 = \{a^i b^j c^k \mid i=j\}$
$L_2 = \{a^i b^j c^k \mid j=k\}$ are both context free.

However, as we prove later,
$L_1 \cap L_2 = \{a^i b^j c^k \mid i=j=k\}$ is not context free. Thus the class of CFLs is not closed under intersection and hence also not closed under complements (why? $\overline{\overline{A} \cup \overline{B}} = A \cap B$). In this regard, CFLs differ from regular languages.