Lecture 28 Mar 2017 CSE396 ①

Parsing and Ambiguity (partly overlaps CSE305)

Def: A *parse tree* for a string $x \in L(G)$

or a *sentential form* $X \in (\Sigma \cup V)^*$

But for "L_A" on the next page we'd want the root label to be A not S.

is a tree with *internal* nodes labelled by variables

(*) $\boxed{\text{root label is S}}$ *leaves* labelled by terminals or ε

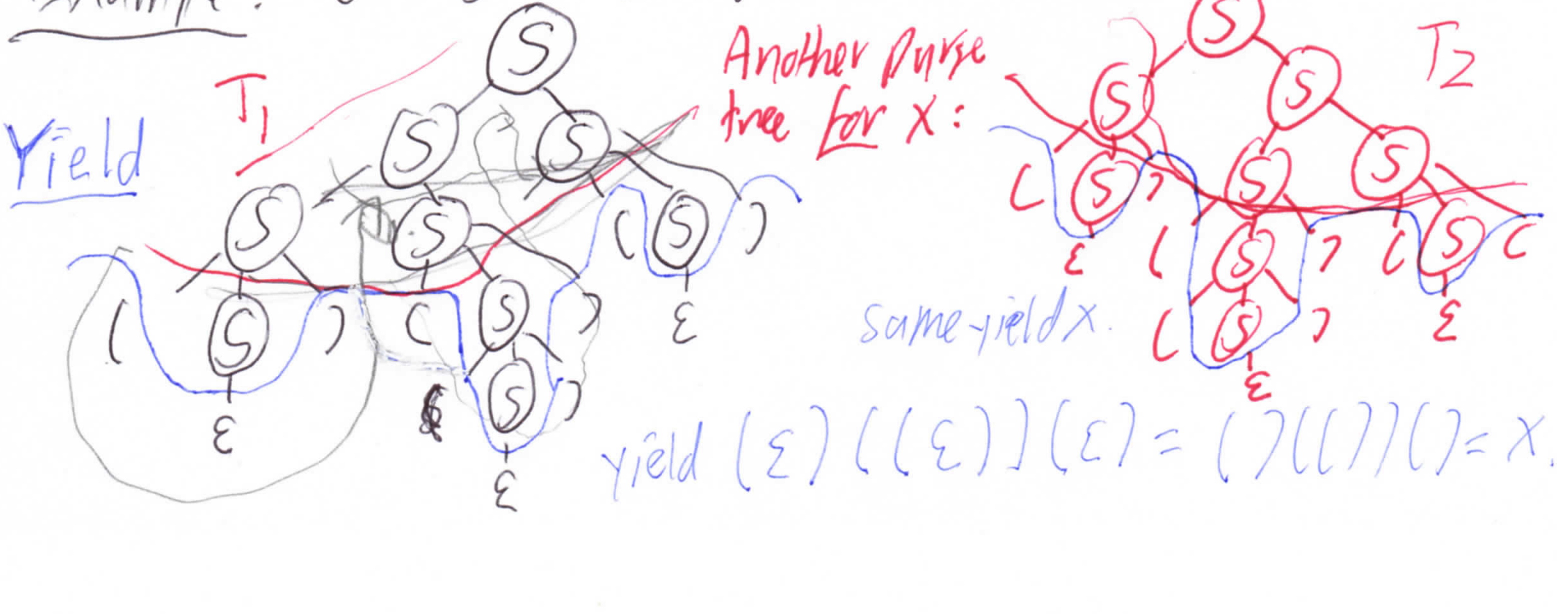
such that for each internal node with label $A \in V$,

its children have labels $u_1, \dots, u_m$ such that

$A \to u_1 \cdots u_m$ is a rule in the CFG G

and finally the leaves in left-to-right order form x (or X).

Example: $G = S \to SS \mid (S) \mid \varepsilon$ $\quad x = \overset{S}{()} \cdot \overset{S}{(())} \cdot \overset{S}{()}$



same-yield X.

Yield (ε)((ε))(ε) = ()(())() = X.

②

Def^n: A string x (or X) is *ambiguous* (in G) if it has two or more distinct parse trees. Else it is *unambiguous* (in G). Here, ε, $()$, are unambiguous, ditto (S) and SS, but SSS is ambiguous.

Observation: Every parse tree gives a *unique* $\left\{\begin{matrix}\text{leftmost}\\ \text{rightmost}\end{matrix}\right.$ derivation in which every step expands the $\left\{\begin{matrix}\text{leftmost}\\ \text{rightmost}\end{matrix}\right.$ variable. For this X:

T_1: $S \Rightarrow \underline{S}S \Rightarrow \underline{S}SS \Rightarrow (S)SS \Rightarrow ()\underline{S}S \Rightarrow ()(S)S \Rightarrow ()((\underline{S}))S$
$\Rightarrow ()(())\underline{S} \Rightarrow ()(())(S) \Rightarrow ()(())() = X.$

LM deriv using T_2: $S \Rightarrow \underline{S}S \Rightarrow (\underline{S})S \Rightarrow ()\underline{S} \Rightarrow ()\underline{S}S$
$\Rightarrow ()(\underline{S})S \Rightarrow ()((S))S \Rightarrow ()(())S \Rightarrow^2 ()(())() = X.$

Fact: X is ambiguous in G if and only if X has two distinct *leftmost* derivations ($\Leftrightarrow$ X has two distinct *RM* deriv^{n}s).

I.e. Parse Trees are in 1-1 correspondence with *LM* derivations (and in 1-1 correspondence with *RM* deriv^{n}s)
And Every derivation gives a unique parse tree.

Final Def^{n}s: A CFG G is *ambiguous* if there is some ambiguous $x \in L(G)$ — [or ambiguous sentential form $X \in (\Sigma \cup V)^*$]

Def^n: For every $A \in V$, we can define $L_A = \{x \in \Sigma^* : A \Rightarrow^* x\}$ (*) see note on p1. A is *live* if $L_A \neq \emptyset$, else A is *deadwood*. If no variables are dead, then these two def^{n}s of ambiguity are equivalent. Finally, a language B is *inherently ambiguous* if every CFG G s.t. $L(G) = B$ is ambiguous.

A Typical Expression Grammar

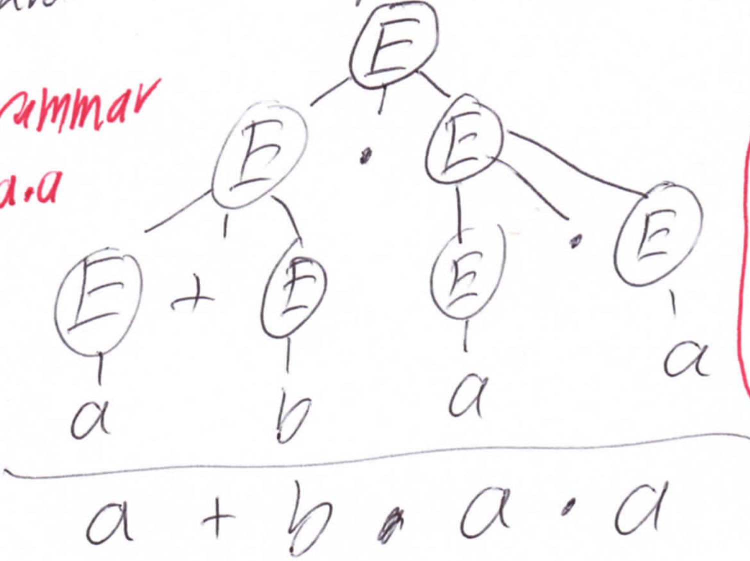
Let $\Sigma = \{a, b\}$ but let $T = \{a, b, +, \circ, {}^*, \underset{\sim}{\varepsilon}, \underset{\sim}{\emptyset}, (,)\}$

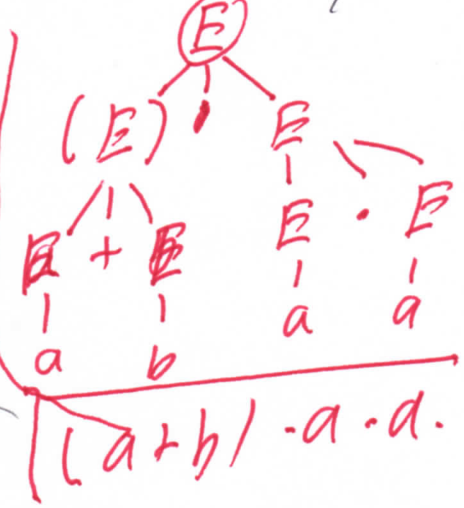
Define $V = \{E\}$, $S = E$, and the rules

$$\underline{G_1}: \quad E \to (E) \mid E+E \mid E \circ E \mid E^* \mid a \mid b \mid \underset{\sim}{\varepsilon} \mid \underset{\sim}{\emptyset}$$

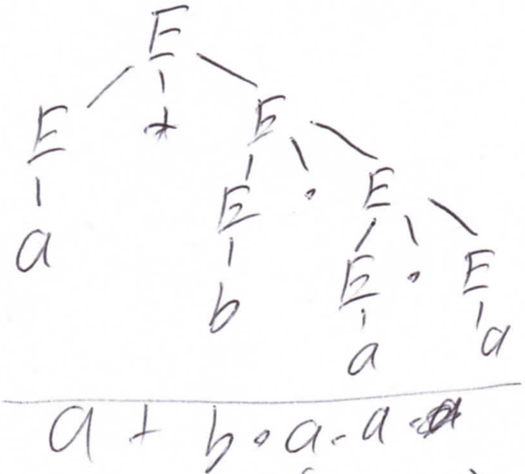
$L(G_1) = \{$"liberal" textual representations of regular expressions over $\{a,b\}\}$.

note: the grammar allows $(a+b)\cdot a \cdot a$ but this is a different string $x' \in L(G)$





We can also get x by a parse tree that respects the **actual** reading:



read as $a + (b a a)$.

Hence G_1 is ambiguous (many times over), but happily, $L(G_1)$ is <u>not</u> inherently ambiguous...

An "overkill" fix: CFG $\underline{G_2}$

$$E \to (E+E) \mid (E \circ E) \mid (E^*) \mid a \mid b \mid \underset{\sim}{\varepsilon} \mid \underset{\sim}{\emptyset}$$

$L(G_2) = \{$<u>fully parenthesized</u> regexps$\} \neq L(G_1)$. e.g.: $x'' = (a+((b \circ a) \cdot a))$

An unambiguous grammar^{G_3} for regexps the way we write them, using three "categories":

G_3 Expression $E \to T \mid E+T$ ~~or T+E~~

Term $T \to F \mid T \cdot F$

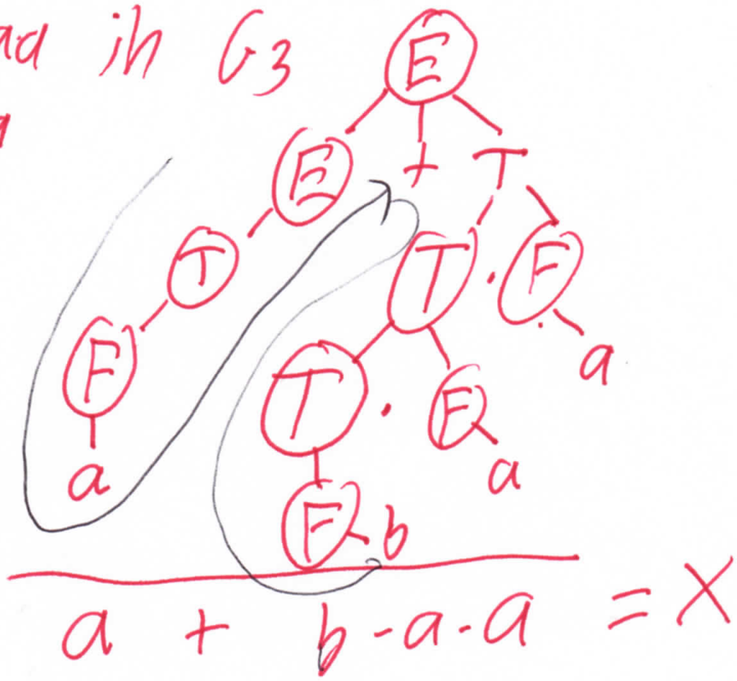
Factor $F \to a \mid b \mid \varepsilon \mid \emptyset \mid (E) \mid F^*$

Let us try to derive $a + b.aa$ in G_3

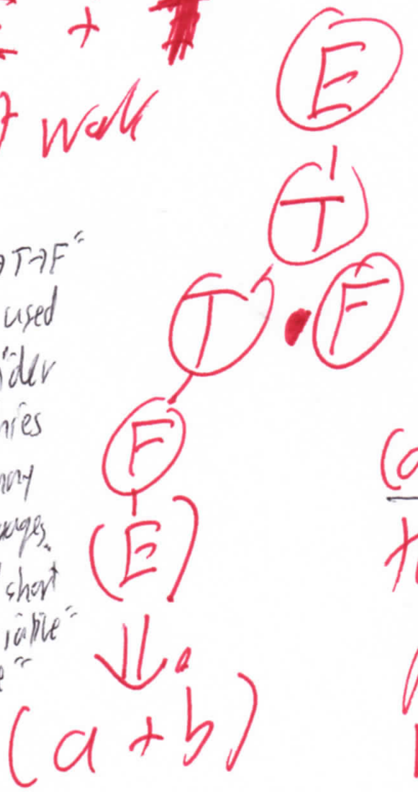
Now try to group $(a+b)\cdot a\cdot a$

Can we do it without parens?

doesn't work



a + b·a·a = X



Added: The "E→T→F" design pattern is used to define much wider precedence hierarchies of operators in many programming languages. My lecture stopped short of the "left-associative" vs "right-associative" issue, which comes into play when − and / (etc.) are operators and which you should see in CSE305. The big "gotcha" is that a/b*c parses as (a/b)*c when you probably intended a/(b*c) as if writing $\frac{a}{bc}$.

conclusion: to group (a+b)aa the grammar will force the parentheses. That's intuitively why it is unambiguous but gives the same language as G_1..