Wednesday
04/11/2020

Ex: Consider the grammar: $S \to SS \mid aSb \mid bSa \mid \lambda$
string $w = aabb$

Sol^n Step 1: Look at all the productions of the form $S \to x$
if none of this result in string w then go to the next round

✓ 1) $S \to SS$
✓ 2) $S \to aSb$ $w = aabb$
✗ 3) $S \to bSa$ none of the string are w
✗ 4) $S \to \lambda$

~~Step 2~~ 3 and 4 are to be deleted because it will never give the string w

Step 2: Apply all applicable productions to the leftmost variable of every x.

1) $S \to SS \to SSS$ ✓
$S \to SS \to aSbS$ ✓
$S \to SS \to bSaS$ ✗
$S \to SS \to S$ ✓

2) $S \to aSb \to aSSb$ ✓
$S \to aSb \to aaSbb$ ✓
$S \to aSb \to abSab$ ✗
$S \to aSb \to ab$

Step 3: $S \to aSb \to aaSbb \to aaSSbb$
$S \to aSb \to aaSbb \to aaaSbbb$
$S \to aSb \to aaSbb \to aabSabb$
$S \to aSb \to aaSbb \to \underline{aabb}$ (w)

try for all the remaining possibilities

Date 04/11/2020
Page

This method is not efficient parsing.

While the method always parses a $w \in L(G)$ it is possible that it never terminates for strings not in $L(G)$

Ex: $w = abb$

the method will go on producing trial sentential forms indefinitely unless we build into it some way of stopping.

If we eliminate two types of productions, those of the form $A \rightarrow \lambda$ and of the form $A \rightarrow B$ then the algorithm can be terminated.

Theorem 5.2

Suppose that $G = (V, T, S, P)$ is a context-free grammar that does not have any rules of the form $A \rightarrow \lambda$ or $A \rightarrow B$, where $A, B \in V$. Then the exhaustive search parsing method can be made into an algorithm which, for any $w \in Σ^*$, either produces a parsing of w or tells us that no parsing is possible $2 \cdot |w|$ rounds.

After $|w|$ rounds, we either have produced the string w, or we cannot generate the string w i.e. w does not belong to $L(G)$.

**

Definition A context-free grammar $G = (V, T, S, P)$ is said to be a simple grammar or s-grammar if all its productions are of the form:

$$A \rightarrow ax$$

where $A \in V$, $a \in T$, $x \in V^*$ and any pair (A, a) occurs at most once in P

Ex: The grammar $S \rightarrow aS \mid bSS \mid c$ is an s-grammar.

The grammar $S \rightarrow aS \mid bSS \mid aSS \mid c$ is not s-grammar

Date 04/11/2020

because the pair (S, a) occurs in the two productions $S \rightarrow aS$ and $S \rightarrow aSS$.

Theorem:

If G is an s-grammar then any string w in $L(G)$ can be parsed with an effort proportional to $|w|$. In the exhaustive search algorithm the parsing can be done in no more than $|w|$ steps.

Ambiguity in Grammars and Languages

A no. of different derivation trees may exist for a given string.

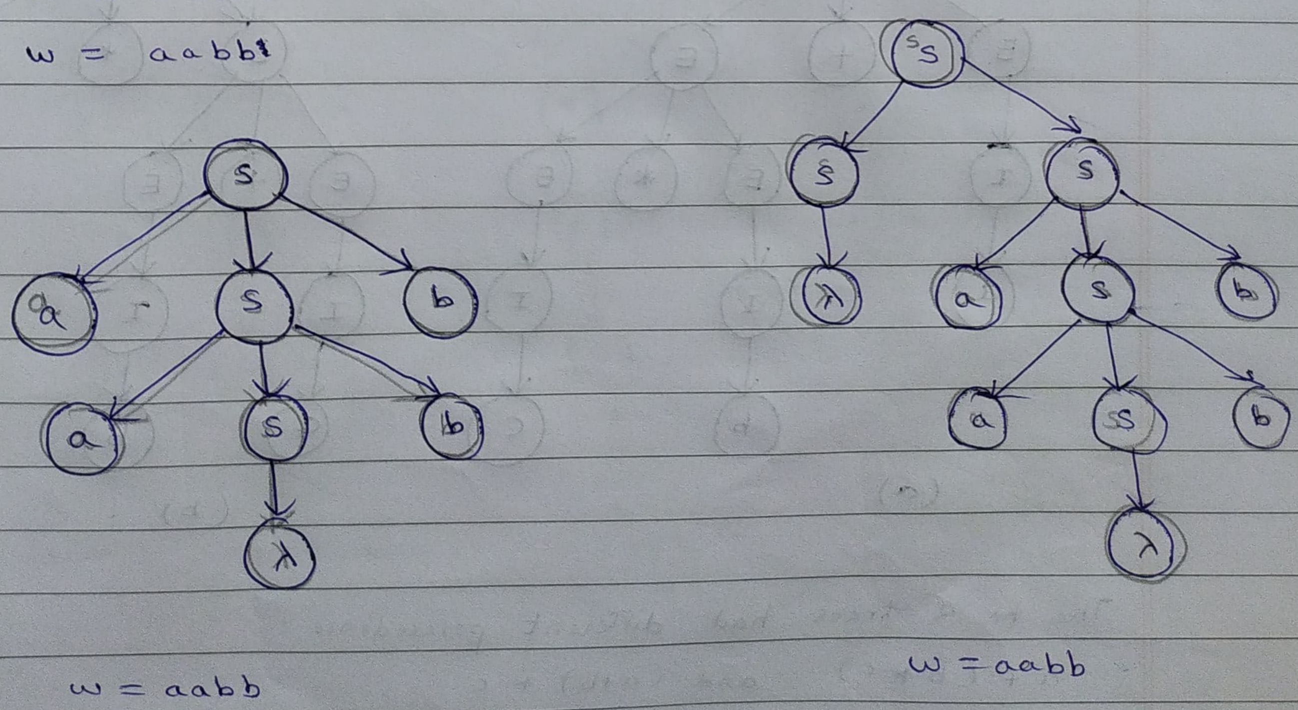
Definition A context-free grammar G is said to be ambiguous if there exists some $w \in L(G)$ that has atleast two distinct derivation trees.

→ Ambiguity implies existence of two or more leftmost or rightmost derivations.

Example:- The grammar with productions: $S \rightarrow aSb \mid SS \mid \lambda$ is ambiguous.

For $w = aabb$

$w = aabb$



classmate

In programming languages where there should be only one interpreting of each statement, ambiguity must be removed when possible.

This can be achieved by rewritting the grammar in equivalent, unambiguous form.

<u>Examples</u> Consider the grammar

$V = \{ E, I \}$

$T = \{ a, b, c +, *, (,) \},$

and productions

$E \rightarrow I$

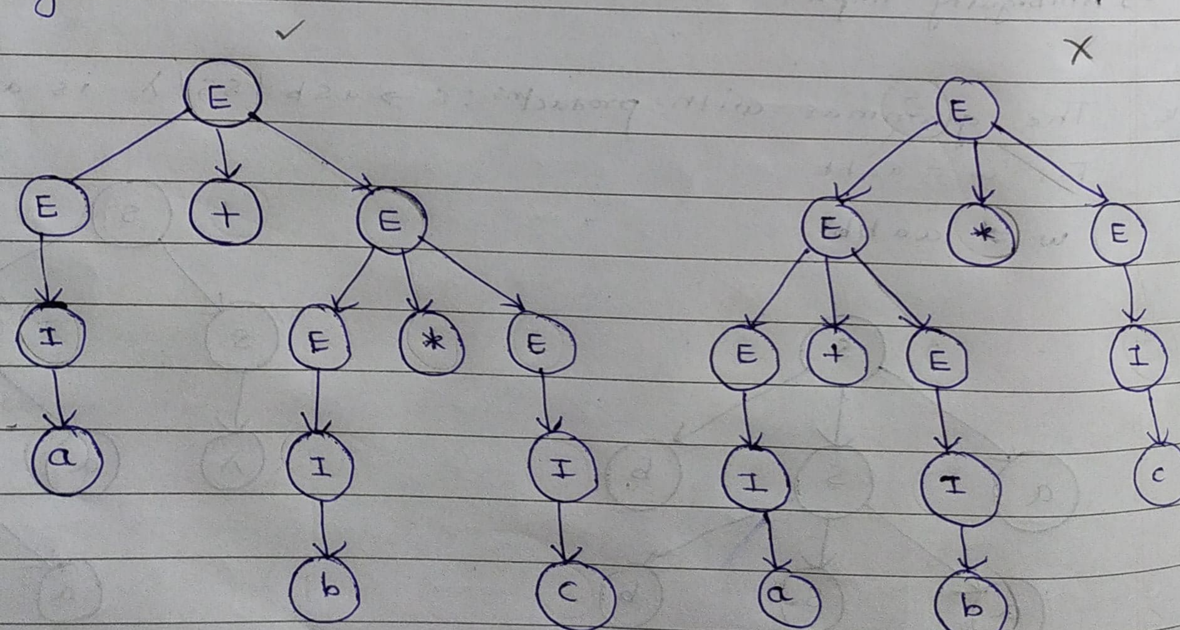
$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow E$

$I \rightarrow a \mid b \mid c$

string $w = a + b * c$



(a) (b)

The 2 trees had different precedence:

$a + (b * c)$ and $(a+b) * c$

To resolve the ambiguity we use precedence rules.

(a) as the correct parsing a + (b*c)

To show precedence in the grammar, we need to rewrite the grammar so that only one parsing is possible.

$V = \{E, T, F, I\}$

$E \rightarrow T$

$T \rightarrow F$

$F \rightarrow I$

$E \rightarrow E + T$

$T \rightarrow T * F$

$F \rightarrow (E)$

$I \rightarrow a | b | c$.

This grammar is unambiguous.

$w = a + b * c$