Context Free Grammar

Beulah A.
AP/CSE

Grammars

- Inherently recursive structures of a programming language are defined by a context-free grammar.
- Context-free grammar is a 4-tuple $G = (N, T, P, S)$ where
  - T is a finite set of tokens (*terminal* symbols)
  - N is a finite set of *nonterminals*
  - P is a finite set of *productions* of the form
    $\alpha \rightarrow \beta$
    where $\alpha \in N$ and $\beta \in (N \cup T)^*$
  - $S \in N$ is a designated *start symbol*

Example

- $G = (\{E\}, \{+, -, *, /, (,), id\}, P, E)$,

  where P consists of

  $E \rightarrow E + E$

  $E \rightarrow E - E$

  $E \rightarrow E * E$

  $E \rightarrow E / E$

  $E \rightarrow - E$

  $E \rightarrow (E)$

  $E \rightarrow id$

Notational Conventions Used

- Terminals

  $a, b, c, \ldots \in T$

  specific terminals: **0**, **1**, **id**, **+**
- Nonterminals

  $A, B, C, \ldots \in N$

  specific nonterminals: *expr*, *term*, *stmt*
- Grammar symbols

  $X, Y, Z \in (N \cup T)$
- Strings of terminals

  $u, v, w, x, y, z \in T^*$
- Strings of grammar symbols

  $\alpha, \beta, \gamma \in (N \cup T)^*$

Context Free Language

- The language generated by CFG G is defined as :
  - $L(G) = \{w \mid w \text{ is in } T^{+} \text{ and } S \overset{*}{\Rightarrow} w\}$. That is a string is in L(G) if
    - The string consists of terminals only
    - The string has to be derived from S only
- L is a Context Free Language (CFL), if it is L(G) for some CFG G.

Applications of Context Free Grammar

- To design a Parser, a CFG is needed.
- The DTD (Document type Definitions) is a CFG whose language is a class of related documents.

Derivations

$E \Rightarrow E+E$

- E+E derives from E
  - we can replace E by E+E
  - to able to do this, we have to have a production rule $E \rightarrow E+E$ in our grammar.

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow id+E \\
&\Rightarrow id+id
\end{aligned}
$$

- A sequence of replacements of non-terminal symbols is called a **derivation** of id+id from E.

Derivations Cont…

- $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$ (α_n derives from α_1 or α_1 derives α_n)
- $\Rightarrow$: derives in one step
- $\overset{*}{\Rightarrow}$: derives in zero or more steps
- $\overset{+}{\Rightarrow}$: derives in one or more steps

Derivation Cont…

- A **left-most derivation** of a sentential form is one in which rules transforming the left-most non-terminal are always applied.
- $\overset{lm}{\Rightarrow}$: leftmost derivation

- A **right-most derivation** of a sentential form is one in which rules transforming the right-most non-terminal are always applied
- $\overset{rm}{\Rightarrow}$: rightmost derivation

Left-Most and Right-Most Derivations

Left-Most Derivation

$$E \overset{lm}{\Rightarrow} -E \overset{lm}{\Rightarrow} -(E) \overset{lm}{\Rightarrow} -(E+E) \overset{lm}{\Rightarrow} -(id+E) \overset{lm}{\Rightarrow} -(id+id)$$

Right-Most Derivation

$$E \overset{rm}{\Rightarrow} -E \overset{rm}{\Rightarrow} -(E) \overset{rm}{\Rightarrow} -(E+E) \overset{rm}{\Rightarrow} -(E+id) \overset{rm}{\Rightarrow} -(id+id)$$

Derivation Tree/ Parse Tree

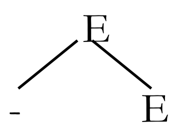
- Let G=(N, T, P, S) be a CFG. A tree is a derivation (or) parse tree for G if :
  - Every vertex has a label which is a non-terminal (or) terminal (or) ε, (i.e.) N ∪ T ∪ (ε).
  - The label of the root is S(Start symbol)
  - The internal vertices must be in N (non - terminal) labeled as A.
  - If A is a label for a node and nodes n_1, n_2,........ n_k are the sons of node n, in order from the left, then $A \rightarrow n_1 n_2 \ldots n_k$ must be a production in P.
  - If a vertex has label ε, then the vertex is a leaf and is the only son of its father.

Derivation Tree/ Parse Tree

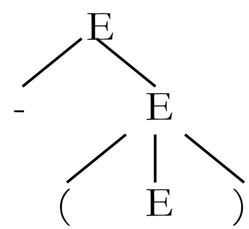
$E \Rightarrow -E$
$\Rightarrow -(E)$
$\Rightarrow -(E+E)$
$\Rightarrow -(id+E)$
$\Rightarrow -(id+id)$

- Inner nodes of a parse tree are non-terminal symbols.
- The leaves of a parse tree are terminal symbols.
- A parse tree can be seen as a graphical representation of a derivation.

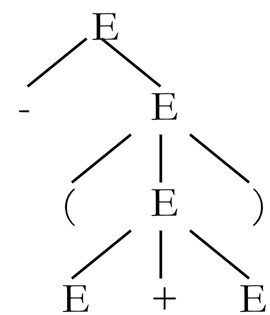
$E \Rightarrow -E$



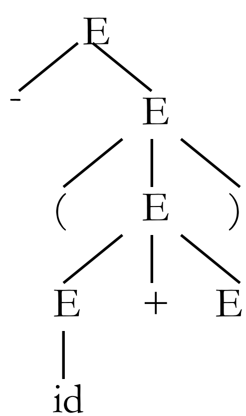
$\Rightarrow -(E)$



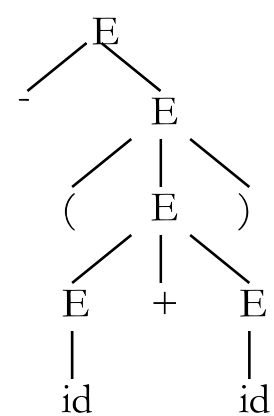
$\Rightarrow -(E+E)$



$\Rightarrow -(id+E)$



$\Rightarrow -(id+id)$

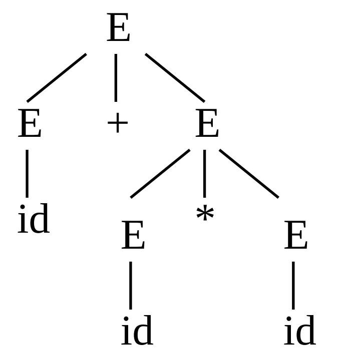


Ambiguous Grammar

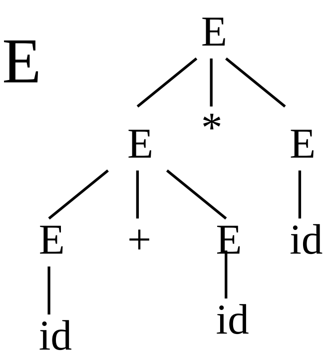
- A grammar G is ambiguous if there is a word $w \in L(G)$ having at least two different parse trees
- CFG is ambiguous $\Leftrightarrow$ any of following equivalent statements:
  - $\exists$ string w with more than one derivation trees.
  - $\exists$ string w with more than one leftmost derivations.
  - $\exists$ string w with more than one rightmost derivations.

Ambiguous Grammar

$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+E*E$
$\Rightarrow id+id*E \Rightarrow id+id*id$



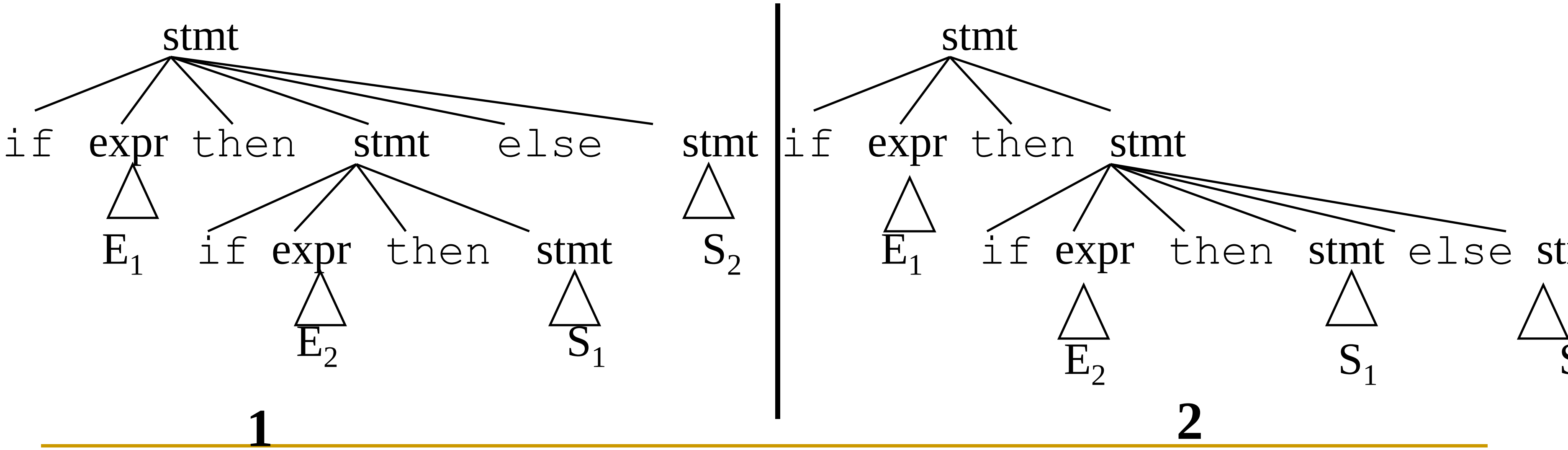
$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow id+E*E$
$\Rightarrow id+id*E \Rightarrow id+id*id$



Ambiguous Grammar

stmt → `if` expr `then` stmt |
`if` expr `then` stmt `else` stmt | otherstmts

`if` E_1 `then` `if` E_2 `then` S_1 `else` S_2



Summary

- Discussion about context free grammar
- Language of CFG
- Derivations from a grammar for a string/word
- Parse tree for a string/word
- Ambiguous grammar

Test Your Knowledge

- What the does the given CFG defines?
  S→aSbS|bSaS|e and w denotes terminal
  a) wwr
  b) wSw
  c) Equal number of a's and b's
  d) None of the mentioned
- A grammar G=(V, T, P, S) is ___________ if every production taken one of the two forms:
  B→aC
  B→a
  a) Ambiguous
  b) Regular
  c) Non Regular
  d) None of the mentioned

Reference

- Hopcroft J.E., Motwani R. and Ullman J.D, "Introduction to Automata Theory, Languages and Computations", Second Edition, Pearson Education, 2008