CS3005D Compiler Design

Winter 2024

Lecture #8

Parsing - Introduction

Saleena N

CSED NIT Calicut

January 2024

Compiler - Analysis

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis

Lexical Analysis

- Lexical Analysis
  - Program (as character stream) converted to a stream of tokens
  - Requires specification of tokens - patterns describing the lexemes corresponding to each token

Syntax Analysis or Parsing

- Syntax Analysis
  - Token stream to Parse tree (or some intermediate representation of the program)
  - Grammar rules - prescribe the syntactic structure of well-formed programs

Syntax Specification

- Backus-Naur Form (BNF)
- Context Free Grammars (CFG)

Syntax Descriptions - origins

- Noam Chomsky - CFG formalism as part of study on natural languages
- John Backus, Peter Naur - Algol 60 - BNF (Backus-Naur Form)

BNF

$$
\begin{aligned}
\langle \text{expression} \rangle ::= \quad & \langle \text{expression} \rangle + \langle \text{expression} \rangle \\
& \mid \langle \text{id} \rangle \\
& \mid \langle \text{num} \rangle
\end{aligned}
$$

CFG

$$\begin{aligned} \mathit{expression} \rightarrow \quad & \mathit{expression} + \mathit{expression} \\ & \mid \textbf{id} \\ & \mid \textbf{num} \end{aligned}$$

CFG

$$
\begin{aligned}
E \rightarrow \quad & E + E \\
& \mid \textbf{id} \\
& \mid \textbf{num}
\end{aligned}
$$

Parsing

Given Grammar G and a string w test if $w \in L(G)$[1]

[1] L(G) - Language generated by G

Parser- types

- Universal
  - Parse any grammar - e.g. CYK Algorithm - Cocke, Younger and Kasami - $O(n^3)$
- Compilers commonly use
  - Top-Down Parsing - parse tree built from top(root) to bottom(leaves)
  - Bottom-Up Parsing - parse tree building starts from the leaves

Parsing

Input: Stream of Tokens, CFG

- Checks if the token stream can be generated by the grammar
- Generates a parse tree (or some intermediate representation of the program)
- Error reporting/recovery
- Create/Update Symbol Table entries, type checking, intermediate code generation...

Context Free Grammar

$$G = (V, T, P, S)$$

Context Free Grammar

$$S \rightarrow \textbf{if}\ (E)\ S\ \textbf{else}\ S \mid \textbf{id} = E$$
$$E \rightarrow E + E \mid \textbf{id}$$

V = ?

T = ?

Context Free Grammar

- Variables / Nonterminals - S, E
  each denotes a set of strings
- Terminals - **if**, **else**, **id**

CFG - alternate notation

$$stmt \rightarrow \textbf{if}\ (expression)\ stmt\ \textbf{else}\ stmt \mid \textbf{id} = expression$$

$$expression \rightarrow expression + expression \mid \textbf{id}$$

$$V = ?$$
$$T = ?$$

Derivation

input: **id** $=$ **id** $+$ **id**

$$S \Rightarrow id = E \Rightarrow id = E + E \Rightarrow id = id + E \Rightarrow id = id + id$$

Derivation

$$S \Rightarrow id = E \Rightarrow id = E + E \Rightarrow id = id + E \Rightarrow id = id + id$$

Derivation step: replaces a nonterminal A by a string of grammar symbols α, where $A \rightarrow \alpha$ is a production [2]

$$E \rightarrow E + E$$

[2] α, β, γ ... denote strings of grammar symbols

Derivation

$$S \Rightarrow id = E \Rightarrow id = E + E \Rightarrow id = id + E \Rightarrow id = id + id$$

$$S \stackrel{*}{\Rightarrow} id = id + id$$

$\stackrel{*}{\Rightarrow}$: derives in zero or more steps

Sentential Form

If $S \overset{*}{\Rightarrow} \alpha$, we say α is a *sentential form*

Sentence

A sentence is a sentential form with no nonterminals.

Language

L(G) - Language generated by grammar G - set of sentences derivable from the start symbol of G.

Language

A string of terminals $w \in L(G)$ if and only if $S \stackrel{*}{\Rightarrow} w$ [3]

[3] $u,\ v,\ w\ldots$ to denote strings of terminals

Derivations

$$S \Rightarrow id_1 = E \Rightarrow id_1 = E + E \Rightarrow id_1 = id_2 + E \Rightarrow id_1 = id_2 + id_3$$

$$S \Rightarrow id_1 = E \Rightarrow id_1 = E + E \Rightarrow id_1 = E + id_3 \Rightarrow id_1 = id_2 + id_3$$

Derivations

Leftmost derivation

$$S \Rightarrow id_1 = E \Rightarrow id_1 = E + E \Rightarrow id_1 = id_2 + E \Rightarrow id_1 = id_2 + id_3$$

Rightmost derivation

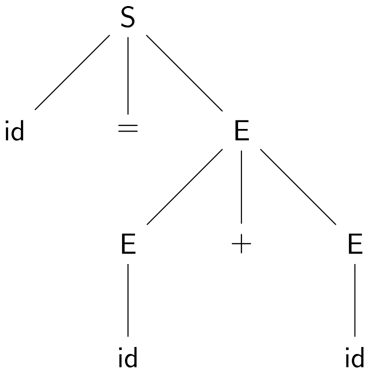
$$S \Rightarrow id_1 = E \Rightarrow id_1 = E + E \Rightarrow id_1 = E + id_3 \Rightarrow id_1 = id_2 + id_3$$

Derivations

Leftmost derivation: Each step replaces the leftmost nonterminal in the sentential form

Rightmost derivation: Each step replaces the rightmost nonterminal in the sentential form.

Parse Tree



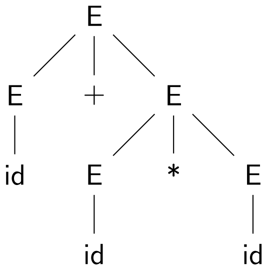
Expression Grammar

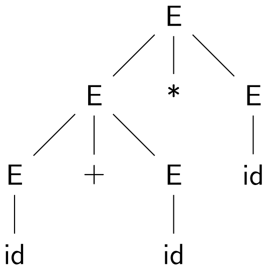
$$E \rightarrow E + E \mid E * E \mid id$$

Derive: $id + id * id$

Parse Tree ?

Two different Parse Trees





Ambiguous Grammar

More than one parse tree for some sentence.

More than one leftmost derivations?

Top-Down Parsing

- Construct parse tree, starting from the root and creating nodes in preorder
- Tree Nodes
    - Nonterminal - choose production, add children
    - Terminal - match with the next input token
- Equivalently finding a Left-most derivation

Bottom-Up Parsing

- Construct parse tree bottom-up, starting from the leaves
- Parent node A added to a set of nodes matching the body α of production $A \rightarrow \alpha$