Programming Languages Recitation

Grammars, Ada walk through

Arpit Jain

Computer Science Department
Courant Institute of Mathematical Sciences
New York University

arpit.jain@cs.nyu.edu

September 4, 2014

Overview

1 Introduction

2 Grammars
- Context Free Grammar (CFG)
- Regular Grammar
- Parse Tree

3 Ada

Syntax and Semantics

Syntax

- Refers to external representation
- Given some text, is it well formed ?

Semantics

- Refers to the meaning
- Given some well formed text, what does it mean ?

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens
2. Parser: Tokens $\rightarrow$ Parse Tree

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens
2. Parser: Tokens $\rightarrow$ Parse Tree
3. Semantic Analyzer

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens
2. Parser: Tokens $\rightarrow$ Parse Tree
3. Semantic Analyzer
4. Intermediate Code Generation

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens
2. Parser: Tokens $\rightarrow$ Parse Tree
3. Semantic Analyzer
4. Intermediate Code Generation
5. Optimization (Machine independent and dependent)

Compilation

Phases of compiler:

1. Lexer : Text $\rightarrow$ Tokens
2. Parser: Tokens $\rightarrow$ Parse Tree
3. Semantic Analyzer
4. Intermediate Code Generation
5. Optimization (Machine independent and dependent)
6. Target Code Generation

Grammars

A grammar G is a tuple $(\sum, N, S, \delta)$

- $\sum$ is set of Terminal symbols
- N is set of Non-Terminal symbols
- S is a Distinguished Non-Terminal symbol
- δ is a set of rewrite rules of form- ABC... ::= XYZ, where ABCXYZ $\in (\sum \cup N \cup S)$

Backus-Naur Form(BNF) for CFG

General Form

- N ::= XYZ
- One non-terminal on left and mixture on right

Backus-Naur Form(BNF) for CFG

Common patterns

- Alternation: Symbol ::= Letter | Digit
- Repetition:
  - Identifier ::= Letter{Symbol}
  - Identifier ::= Letter Symbol* (Kleene Star- Zero or more repetitions)
  - Identifier ::= Letter Symbol$^+$ (One or more repetitions)
- Optional: Number::= ['+'|'-'] Digit

Backus-Naur Form(BNF) for CFG

Some more patterns

- ID{2,5} - anywhere between 2-5 IDs
- ID{5,} - 5 or more IDs
- ID{4} - Exactly 4 times ID
- ID Digit - Expression ‘ID’ followed by expression ‘Digit’ (Concatenation)

Regular Grammar

General Form

- N::= TN
- One non-terminal on left and at most one on right

Common patterns

- a - matches the character ‘a’
- [abc] - matches ‘a’ or ‘b’ or ‘c’
- [a-z] - matches any character between ‘a’ through ‘z’
- ^a - matches everything except ‘a’
- [A-Za-z] - matches all the alphabets (Uppercase and Lowercase)
- Digit ::= 0|1|2|3|4|5|6|7|8|9
- $R^? \equiv \in |R$

Parse Tree

Given grammar-

- Var_Decl ::= TYPE ID {‘,’ ID } ‘;’
- TYPE ::= ‘int’|‘string’
- ID ::= Letter {Letter | Digit}
- Letter ::= ‘a’|‘b’|‘c’|. . . . . . |‘y’|‘z’
- Digit ::= ‘1’|‘2’|‘3’. . . . . . |‘9’|‘0’

We can rewrite the above clauses as follows-

- Var_Decl ::= TYPE ID S2 ‘;’
- S2 ::= ‘,’ ID S2 | ∈
- ID ::= Letter S3
- S3 ::= Letter S3 | Digit S3 | ∈

Parse Tree

Parse tree for

Input : string pl2,p3,a;

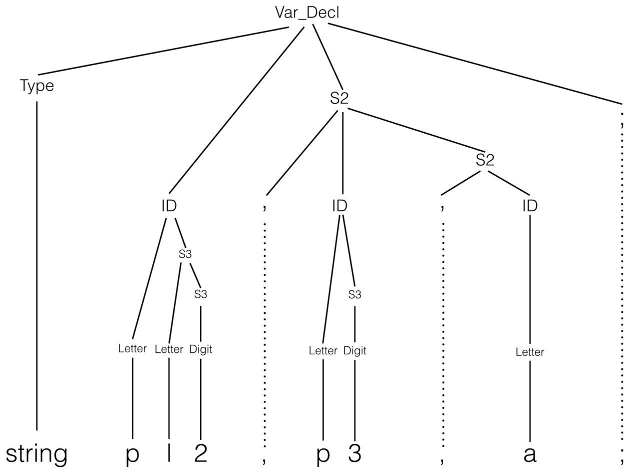

Figure: Parse Tree

Ada Installation and Example

- Follow link on NYU Classes for installation
- Compile the examples