Compilers: Week 3 - Parsing and Top-Down parsing

Ian Quah (itq)

June 13, 2017

Topic 1

Parsing - Introduction

Regular Languages

1. Weakest formal languages - most languages aren't regular

   $\{(^i)^i \mid i \geq 0\}$, the set of all balanced parens, which can't be represented as a regular language

2. What **CAN** a regular language express?

   (a) count mod k

   can't calculate to some arbitrarily high value, like in balanced parens problem

3.

Phase	Input	Output
Lexer	String of Chars	String of Tokens
Parser	String of tokens (output of lexer)	Parse Tree

Topic 2

Context Free Grammars

1. **The Problem**

   (a) Not all strings of tokens are programs

   Thus, need a language for describing valid strings of tokens and method to distinguish invalid and valid strings of tokens

   Note: CFG $\supset$ RegExp

   (b) Programming Languages have a recursive structure, and CFGs are a natural notation for describing these structures.

2. **What IS a CFG?**

   (a) A set of terminals, T

   (b) A set of non-terminals, N

   (c) A start symbol, S (S $\in$ N)

   (d) A set of productions: (A symbol, followed by an arrow then a list of symbols)

   $X \rightarrow Y_1, \ldots Y_n$

   X $\in$ N and $Y_i \in N \cup T \cup \{\epsilon\}$

3. **CFG example: Balanced Parens Problem**

   Productions / Rules

$S \rightarrow (S)$	S, our start state then becomes another state with two parens around it
$S \rightarrow \epsilon$	

   Our States

   N = {S}

   T = {(,)}

   S = start symbol in general the first production will specify the start symbol for that CFG

Topic 2 continued on next page...

4. **Productions as rules:**

   (a) Begin with a String that only has start symbol, S

   (b) Replace any non-terminal X in start by RHS of some production

   (c) Repeat (2) until there are no non-terminals

   If we consider a single step as α, then a program can be thought of as

   $\alpha_0 \to \alpha_1 \to ... \to \alpha_n$ equiv $\alpha_0 \to^* \alpha_n$

5. **CFG formal definition**

   Let G be a CFG with start symbol S, then L(G) of G is:

$$\{a_1, ..., a_n | \forall i, a_i \in T, S \to^* a_1, ..., a_n\}$$

Topic 3

Derivations Part 1

1. **Derivation:** a sequence of productions

2. **Representations**

   - A derivation can be drawn as a tree instead of a linear path

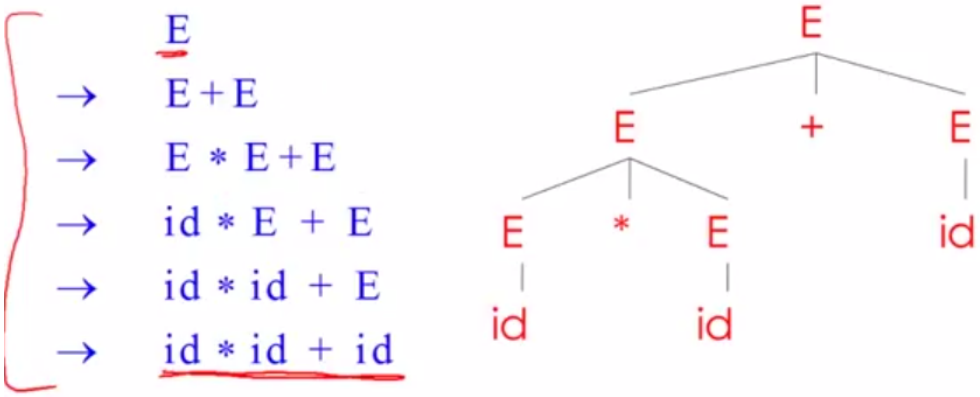
   - A derivation can be drawn as a tree

   (a) Start symbol is the root

   (b) For a production, add the children

   Example:

   Grammar: E → E + E | E * E | (E) | id



   LHS: Derivation intermediates (Left-most derivation)

   RHS: Parse tree of input string

3. **Parse Trees**

   (a) Terminals at the leaves

(b) Non-terminals at interiors

(c) in-order traversal of tree is original input

(d) Parse tree shows association of operations, input string does not (in tree we see multiplication is more tight than addition)

(e) Left-most and right-most derivation produces same parse tree, but the main difference is the intermediate step

Topic 4

Ambiguity in CFGs

1. **Ambiguity**: if there is more than one parse tree for some string (more than 1 left-most or right-most derivation)

   This is bad, because it means that the decision is left to the compiler

2. Determining if it's ambiguous

   Construct a string, then work backwards and see if there is more than 1 way to get the tree

3. **Removing ambiguity**

   id * id + id

   E $\rightarrow$ E' + E — E'

   E' $\rightarrow$ id * E' — id — (E) * E' — (E)

   rewrite to enforce precedence of * over +, by separating multiplication and + s.t * is handled by E'