The syntactic Specification of Programming Languages

Introduction

- For the syntactic specification of a programming language a notation called context free grammar is used
- It has a significant number of advantages:
  1. A grammar gives a precise, easy to understand, syntactic specification for the programs of a particular language
  2. An efficient parser can be constructed automatically from a properly designed grammar
  3. A grammar imparts a structure to a program that is useful for its translation into an object code and for detecting errors

Context Free Grammars

- Regular languages and its limitations
- $\{(^i)^i \mid i>0\}$
- If if if then then then
- If if if then then then fi fi fi
- Some languages cannot be generated using a Regular grammar but is possible using a Context free grammar
- $G=(V_n,\Sigma,P,S)$
- $P=\{A->\alpha \mid A \in V_n \text{ and } \alpha \in \{V_n \cup \Sigma\}^*$

Context free grammars

- If s1 and s2 are statements and E is an expression, then

"**if** E **then** s1 **else** s2" is a statement

- If s1,s2,s3,…,sn are statements, then

"**begin** s1; s2;…;sn **end**"

Statement-> **if** expression **then** statement **else** statement

Statement->**begin** statement; statement;…;statement **end**

- The use of (…) would create problems during translation

CFGs

- Statement->**begin** statement; statement-list **end**
- Statement-list->statement | statement; statement-list

Derivations

- Productions may be applied repeatedly to replace the nonterminals in a string of nonterminals and terminals until all the strings are a sequence of terminals
- E-> E + E |E * E| € |-E |**id**
- Derive the string **-(id + id)**

Leftmost derivation and Rightmost derivation

- Applying the production rules on the leftmost nonterminals. (from left to right)
- Applying the production rules on the rightmost nonterminals. (from right to left)
- Derive the string using leftmost, rightmost and the parse tree
  - (id + id * id)