AMBIGOUS GRAMMAR

* For a given string and grammar, two distinct parse tree exists then grammar known as ambiguous grammar.
* Two parse tree exist for string a+b\*c

Eliminating ambiguity

* An ambiguous grammar should be rewritten to eliminate ambiguity.
* The grammar must be rewritten such that reduction of ‘\*’ precedes the reduction of ‘+’ in string a+b\*c
* The normal method of achieving this is to use a hierarchy of NTs in the grammar and to associate the derivation or reduction of an operator with an appropriate NT.

OPERATOR GRAMMAR(OG)

An Operator grammar is a grammar none of whose productions contain two or more consecutives NTs in any RHS alternatives.

GRAMMAR TYPES:

TYPE 0: **(Phrase structure grammar)**

TYPE 1: **(Context sensitive grammar)**

TYPE 2: **(Context free grammar)**

TYPE 3: **regular grammar/ linear grammar)**

TYPE 0 POINTS:

α=β (strings of Ts and NTs)

-Permits arbitrary substitutions of strings

-No limitation on production rules: at least one nonterminal on LHS.

-not relevant to specification of PLs.

-Not used much

TYPE 1 POINTS:

αA β = α Πβ

-not relevant to specification of PLs.

TYPE 2 POINTS:

* A = Π
* Limit production rules to have exactly one nonterminal on LHS, but anything on RHS.
* suited for programming language specification.

TYPE 3 POINTS:

A= tB|t or Bt|t

<id> = l|<id>l|<id>|d

-Limit production rules to have exactly one nonterminal on LHS and at most one nonterminal and terminal on RHS:

- restricted to the specification of lexical units

- nesting of construct or matching parenthesis can not be specified

RECURSION

**Recursive Specification**

* **A grammar is in recursive specification, if NT being defining in a production, itself occurs in a RHS string of the production, e.g. X :: =AXB**
* The RHS alternative employing recursion is called recursive rules.

**Parse tree**

* A parse tree is used to depict syntactic structure of a valid string as it emerges during a sequence of derivations or reductions.

Recursive Specification

* + Two types of recursive rules
  + Left recursive rule 🡪 NT appears on the extreme left in the recursive rule
  + Right recursive rule 🡪 NT appears on the extreme right in the recursive rule

Reductions

The reductions operation helps to recognize valid strings.

Derivation

A grammar G is used for two purpose

To generate valid strings of LG

To recognized valid strings of LG

The derivation operations helps to **generate valid strings.**

Indirect recursion

Occurs when two or more NTs are defined in terms of one another.

Such recursion is useful for specifying nested constructs in a language

Linkers and loaders

1. Steps of exe

->Translation by translator

-> Linking and relocation by linker

-> Loader-loading

What is the reason of linked origin and loader origin differ?

* + **Same set of translated addresses may have been used by different object modules of the program. This results to conflict in memory allocation.**
  + **OS may require that a program should execute from specific area of memory. This may require change in its origin, thus changing execution start address and symbol addresses.**

DEFINITIONS

* **Def: Linking**: Linking is the process of binding an external reference to correct link time address.

Linker computes both:

Segment Base Address

Offset of External Symbol

* **BINARY PROGRAMS**: Is a machine language program comprising set of program units SP such that for all Pi Є SP,
  + 1. Pi relocated at link origin.
  + 2. Linking is performed for each external reference in Pi.

Grammar (G)

A grammar G of a language LG is a Quadruple (Σ, SNT, S, P) where

Σ = is the set of Ts

SNT= is the set of NTs

S = is the distinguished symbols /starting symbol

P= is the set of productions

Productions

A productions also called a rewriting rule, is a rule of of grammar.

A production has the form

A Nonterminal symbol = String of Ts and NTs