1.3 CONTEXT-FREE GRAMMARS

- Developed by Noam Chomsky in the mid-1950s
- Language generators, meant to describe the syntax of natural languages
- Define a class of languages called context-free Languages.

A rule has a left-hand side (LHS) and a right-hand side (RHS), and consists of terminal and nonterminal symbols

BNF- Backus-Naur Form (1959) – Invented by John Backus to describe the syntax of Algol 58 – BNF is equivalent to context-free grammars

- An abstraction (or nonterminal symbol) can have more than one RHS.
- Abstractions are used to represent classes of syntactic structures.
- They act like syntactic variables (also called non-terminal symbols, or just non-terminals)
- Terminals are lexemes or tokens
- A rule has a left-hand side (LHS), which is a nonterminal, and a right-hand side (RHS), which is a string of terminals and/or non-terminals

Examples of BNF rules:

<ident_list> → identifier | identifier, <ident_list>

<if_stmt> → if <logic_expr> then <stmt>

BNF

A grammar is a finite nonempty set of rules. An abstraction (or nonterminal symbol) can have more than one RHS

<Stmt> -> <single_stmt>

| begin <stmt_list> end

Syntactic lists are described in BNF using recursion

<ident_list> -> ident

| ident, <ident_list>

Derivation

A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)

An Example Grammar

<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → a | b | c | d

<expr> → <term> + <term> | <term> - <term>

<term> → <var> | const

An Example Derivation

<program> => <stmts>

=> <stmt>

=> <var> = <expr>

=> a = <expr>

=> a = <term> + <term>

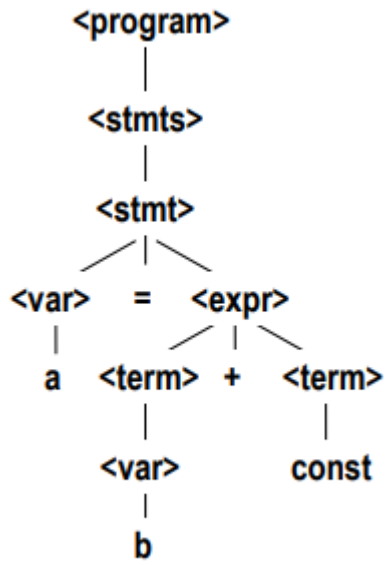
=> a = <var> + <term>

=> a = b + <term>

=> a = b + const

- Every string of symbols in a derivation is a sentential form
- A sentence is a sentential form that has only terminal symbols
- A leftmost derivation is one in which the leftmost nonterminal in each sentential form is the one that is expanded
- A derivation may be neither leftmost nor Rightmost

A hierarchical representation of a derivation



a = b + const

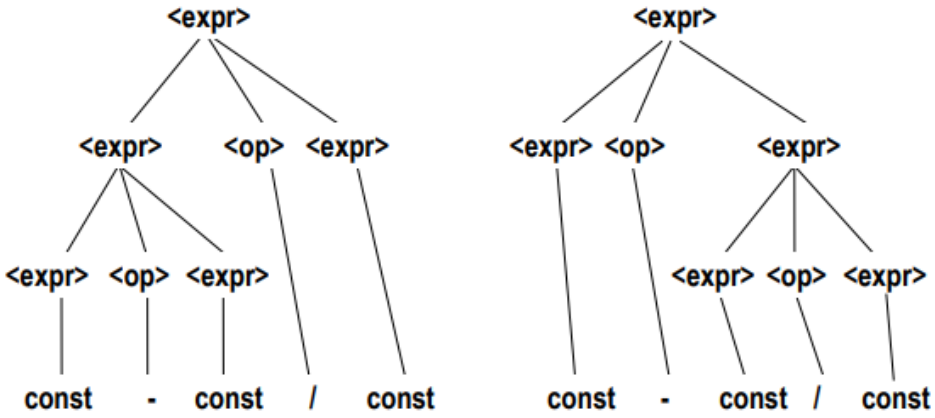
Ambiguity in Grammars

A grammar is ambiguous if and only if it generates a sentential form that has two or more distinct parse trees

An Ambiguous Expression Grammar

<expr> → <expr> <op> <expr> | const

<op> → / | -

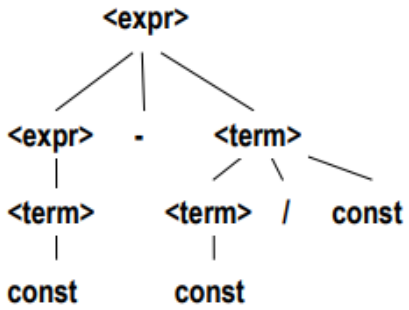


An Unambiguous Expression Grammar

•If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity

<expr> → <expr> - <term> | <term>

<term> → <term> / const| const



Operator Precedence

If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

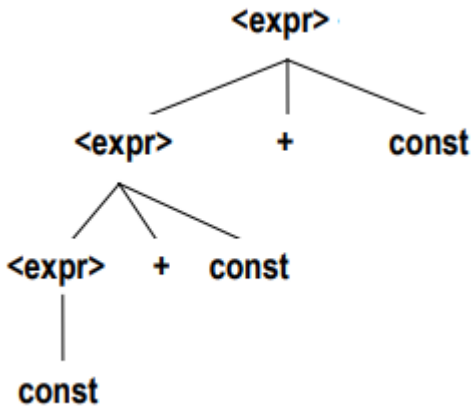
<factor> → (<expr>) | <id>

Associativity of Operators

Operator associativity can also be indicated by a grammar.

<expr> -> <expr> + <expr> | const (ambiguous)

<expr> -> <expr> + const | const (unambiguous)



Extended BNF

- Optional parts are placed in brackets [ ]

  <proc_call> → ident [(<expr_list>)]

- Alternative parts of RHSs are fplaced inside parentheses and separated via vertical bars

  <term> → <term> (+|-) const

- Repetitions (0 or more) are placed inside braces { }

  <ident_list> → <identifier> {, <identifier>}

BNF

<expr> → <term> |

<expr> + <term> |

<expr> - <term>

<term> → <factor> |

<term> * <factor> |

<term> / <factor>

EBNF

<expr> → <term> {(+ | -) <term>}

<term> → <factor> {(* | /) <factor>}

Recent Variations in EBNF

- Alternative RHSs are put on separate lines
- Use of a colon instead of =>
- Use of **opt** for optional parts
- Use of **oneof** for choices