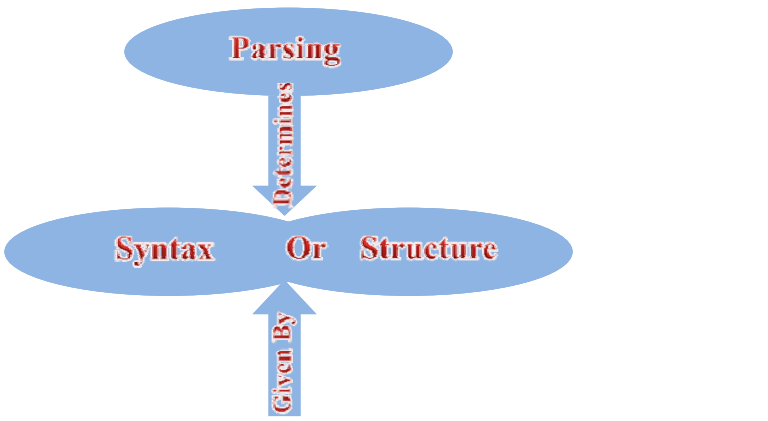
Syntax Analysis

1

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Syntax Analysis



2

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Chomsky Hierarchy of Grammars

- Type 0: Unrestricted Grammar (Equivalent to Turing Machines).
- Type 1: Context Sensitive Grammar (Linear Bound Automata).
- Type 2: Context Free Grammar (Push down Automata).
- Type 3: Regular Grammar (RE).

3

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Context Free Grammars (CFG)

Provides the syntactic structure:
A grammar is quadruple (V_T, V_N, S, R)

- A set of finite terminals "V_T": Basic symbols from which sentences are formed.
- A set of finite non-terminals "V_N": Syntactic variables denoting sets of sentences.
- A set of productions "R": Rules specifying how the terminals and non-terminals can be combined to form sentences. (Å → Æ)
- A unique start symbol "S": A distinguished non-terminal denoting the language (S € N).

4

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Must Conditions for CFG

Three conditions are to be full filled:

1. $V_T \cap V_N = \emptyset$; V_T, V_N are not allowed to have symbol in common. Meaning we must be able to tell terminals and non-terminals apart.
2. $S \in V_N$; S is an element of non-terminal.
3. $R \subseteq \{(N, \alpha) \mid N \in V_N , \alpha \in (V_N \cup V_T)^*\}$ Which means the left side of each production must be non-terminal and right hand side may consists of both (terminals and non-terminals) and is not allowed to include any other symbol.

5

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example

Grammar Rules:

exp ⟶ exp op exp
op ⟶ + | - | * | /

First rule defines an expression structure (with name exp) consists of expression followed by an operator and another expression.
Second rule defines an operator (name op) consists of add, subtract, multiply and division.

(This notation was given by John Backus and adopted by Peter Naur, so called *Backus Naur form*, BNF)

Terminals: id, +, -, *, /
Non-terminals: exp, op
Start symbol: exp

6

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Specifications for CFG

- CFG uses similar naming conventions and operations as RE, the difference is; rules are recursive, so no symbol is required for repetition.

7

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Specifications for CFG

- Given an alphabet a CFG rule in BNF consists of a string of symbols, first symbol is the name of the structure, followed by meta symbol $\longrightarrow$.
- After the meta symbol, there are symbols either from alphabet or structure name or meta-symbol $|$.

8

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Specifications for CFG

- Informally, a BNF rule defines a structure whose name is on the left side of the arrow.
- Structure consists of one of the choices separated by |.
- The sequence of symbols and structures within each choice defines the layout of the structure.

9

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Specifications for CFG

- Meta-symbol alternatives: (No universal Standard)

  “→“ , “=“, “:”, “::=“

- For text files structure names are written in angle brackets.
- Example: <exp> ::= <exp> <op> <exp>

10

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Derivations

$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid id$

- A derivation step is an application of a production as a rewriting rule. (E drives in -E)

$$E \Rightarrow -E$$

- A sequence of derivation steps

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(id)$$

is called a derivation of -(id) from E.

- The symbol $\Rightarrow^*$ denotes, derives in zero or more steps and symbol $\Rightarrow^+$ denotes, derives in one or more steps.

11

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Language Determined by Grammar Rules

How ?

- Grammar rules determine the legal strings of token symbols by means of derivation.
- Derivation is a sequence of replacements of structure name by choice on the right hand side of the grammar rule.
- Derivation begins with a single structure name and end with a string of token symbols.

String: (43 - 63) * 100 [exp → exp op exp | (exp) | number]
[op → + | - | *]

exp ⇒ exp op exp
⇒ exp * number
⇒ (exp op exp) * number
⇒ (exp - number) * number
⇒ exp op number
⇒ (exp) * number
⇒ (exp op number) * number
⇒ (number - number) * number

12

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example of CFG

String: (())((()))()()

Rule: S → SS | (S) | ()

S ⇒ SS
⇒ SSS
⇒ (S)SS
⇒ (())SS
⇒ (())(S)S
⇒ (())(SS)S
⇒ (())((S)S)S
⇒ (())((())S)S
⇒ (())(()())()S
⇒ (())(()())()()

13

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: ((a))

L(G) = {a, (a), ((a)), (((a))), ...}

E → (E) | a

E → (E)
(E) → ((E))
((E)) → ((a))

L(G) = {a, a+a, a+a+a, ...}

E → E + a | a

E → E + a
E + a → E + a + a
E + a + a → E + a + a + a
E + ... → .....

E → (E)
L(G) = { }

14

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Non-Context Free Grammar

String: xxxxbbbbcccc

Rules:

S ⟶ xSBC
S ⟶ xbC
CB ⟶ BC
bB ⟶ bb
bC ⟶ bc
cC ⟶ cc

S ⟶ xSBC
xxSBCBC
xxxSBCBCBC
xxxxbCBCBCBC
xxxxbBCCBCBC
xxxxbbBCCCBC
xxxxbbbCBCCC
xxxxbbbbCCCC
xxxxbbbbcCCC
xxxxbbbbccCC
xxxxbbbbcccC
xxxxbbbbcccc

xxxxbbCCBCBC
xxxxbbbCCCBC
xxxxbbbBCCCC
xxxxbbbbccCC

xxxxbbCBCCBC
xxxxbbbCCBCC
xxxxbbbbCCCC
xxxxbbbbcccC

15

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Leftmost & Rightmost Derivation

A leftmost derivation always chooses the leftmost non-terminal to rewrite:

Rules:

E ⟶ E + E
E ⟶ E - E
E ⟶ E * E
E ⟶ E / E
E ⟶ (E)
E ⟶ id

String: (x + y)/(x - y)

Leftmost:

E ⇒ E / E
⇒ (E) / E
⇒ (E + E) / E
⇒ (id + E) / E
⇒ (id + id) / E
⇒ (id + id) / (E)
⇒ (id + id) / (E - E)
⇒ (id + id) / (id - E)
⇒ (id + id) / (id - id)

Rightmost:

E ⇒ E / E
⇒ E / (E)
⇒ E / (E - E)
⇒ E / (E - id)
⇒ E / (id - id)
⇒ (E) / (id - id)
⇒ (E + E) / (id - id)
⇒ (E + id) / (id - id)
⇒ (id + id) / (id - id)

16

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: ["(34-3)*42"]

LMD

(1) exp	⇒ exp op exp	[exp → exp op exp]
(2)	⇒ (exp) op exp	[exp → (exp)]
(3)	⇒ (exp op exp) op exp	[exp → exp op exp]
(4)	⇒ (number op exp) op exp	[exp → number]
(5)	⇒ (number - exp) op exp	[op → -]
(6)	⇒ (number - number) op exp	[exp → number]
(7)	⇒ (number - number) * exp	[op → *]
(8)	⇒ (number - number) * number	[exp → number]

RMD

(1) exp	⇒ exp op exp	[exp → exp op exp]
(2)	⇒ exp op number	[exp → number]
(3)	⇒ exp * number	[op → *]
(4)	⇒ (exp) * number	[exp → (exp)]
(5)	⇒ (exp op exp) * number	[exp → exp op exp]
(6)	⇒ (exp op number) * number	[exp → number]
(7)	⇒ (exp - number) * number	[op → - [
(8)	⇒ (number - number)*number	[exp → number]

17

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Difference

- A leftmost derivation corresponds to a pre-order traversal of the parse tree.
- A rightmost derivation corresponds to a post-order traversal of the parse tree in reverse order.

  - Both of these construct different types of parsers.

    - LMD: Top-down Parser
    - RMD: Bottom-up Parser

Top-down parsers construct leftmost derivations.

- Left-to-right traversal of input, constructing a Leftmost derivation

Bottom-up parsers construct rightmost derivations.

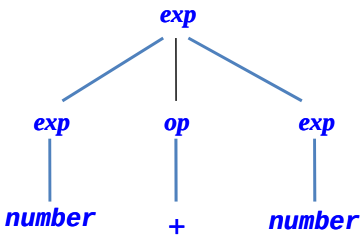
- Left-to-right traversal of input, constructing a Rightmost derivation

18

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Parse Tree

exp ⇒ exp op exp
⇒ number op exp
⇒ number + exp
⇒ number + number

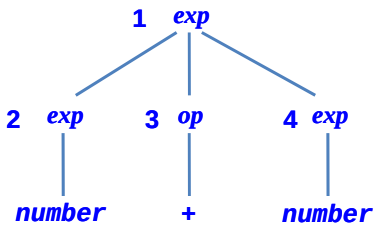


19

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Parse Tree (pre-order numbering)

(1) exp ⇒ exp op exp
(2) ⇒ number op exp
(3) ⇒ number + exp
(4) ⇒ number + number

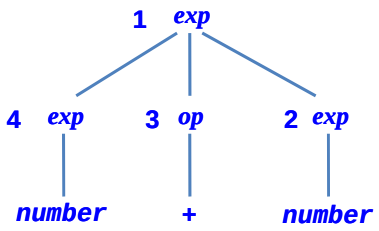


20

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Parse Tree (post-order numbering)

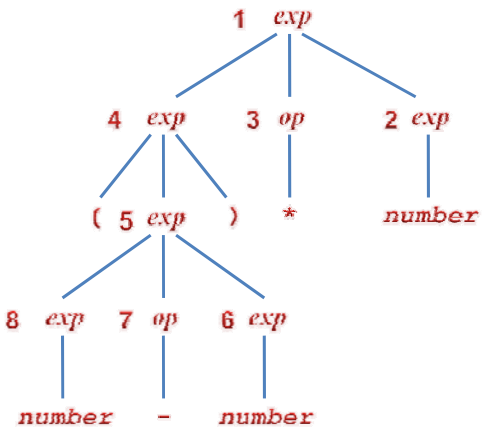
(1) exp ⇒ exp op exp
(2) ⇒ exp op number
(3) ⇒ exp + number
(4) ⇒ number + number



21

Dr. D. M. Akbar Hussain
Department of Electronic Systems

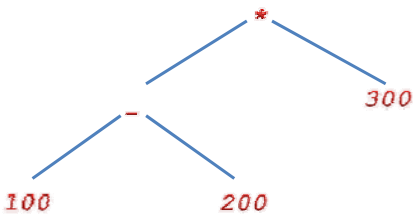
Parse Tree (100 – 200) * 300



22

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Abstract Syntax Tree (100 – 200) * 300



23

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example

statement → *if-stmt* | `other`
if-stmt → `if (` *exp* `)` *statement*
| `if (` *exp* `)` *statement* `else` *statement*
exp → `0` | `1`

Possible strings:

```
other
if(0) other
if(1) other
if(0) other else other
if(1) other else other
if(0) if(0) other
if(0) if(1) other else other
if(1) other else if(0) other else other
.....
```

24

Dr. D. M. Akbar Hussain
Department of Electronic Systems

ε- Productions

- *Grammar generating sequences of one or more statements separated by a semicolon.*
- *stmt-seq → stmt ; stmt-seq | stmt*
  stmt → s
- *L(G) = {s, s;s, s;s;s, ……..}*

To include ε

- *stmt-seq → stmt ; stmt-seq | ε*
  stmt → s
- *L(G) = {ε, s;, s;s;, s;s;s;, ……..}*

In this case ";" has become the statement terminator instead of statement separator.

(zero or more stmts terminated by a ";")

To fix the problem:

- *stmt-seq → non-stmt-seq | ε*
  non-stmt-seq → stmt; non-stmt-seq | stmt
  stmt → s

25

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Dangling else Problem

statement → if-stmt | other
if-stmt → if (exp) statement
| if (exp)statement else statement
exp → 0 | 1

Consider the following string:

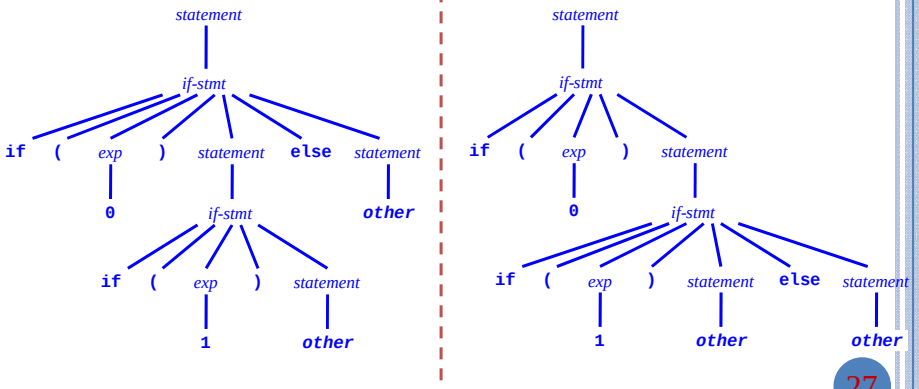
if(0) if(1) other else other

It will produce the following two trees:

26

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Parse trees for Dangling Else Problem



27

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Solutions for Dangling Problems

- ***Most-closely nested rule*** are easy to state, but hard to put into the grammar itself. Two Possibilities to deal with dangling:
  - **Always associate else part** with the nearest if-statement that does not yet have an associated else-part.
  - Use a "**Bracketing Keyword**" to remove the ambiguity:

Bracketing keyword

if-stmt → `if ( exp ) stmt end`
| `if ( exp ) stmt else stmt end`

28

Dr. D. M. Akbar Hussain
Department of Electronic Systems

EBNF

- Standard Backus-Naur Form (BNF)
  - Meta-symbols are | → ε
- Extended BNF (EBNF):
  - New meta-symbols [...] and {...}
  - ε largely eliminated by these new symbols
- Brackets [...] mean optional like "?"
  - *exp* → *term* '|' *exp* | *term* becomes: *exp* → *term* ['|' *exp*]
  - *if-stmt* → `if` (*exp*) *stmt* | `if` (*exp*) *stmt* `else` *stmt* becomes: *if-stmt* → `if` (*exp*) *stmt* [`else` *stmt*]

29

Dr. D. M. Akbar Hussain
Department of Electronic Systems

EBNF continued

- Braces {...} mean repetition
  - *exp* → *exp* + *term* | *term* becomes: *exp* → *term* { + *term* }
- Choices:
  - *exp* → *exp* + *term* | *exp* − *term* | *term*

    are they same ?

    exp → *term* { + *term* } | *term* { − *term* }

30

Dr. D. M. Akbar Hussain
Department of Electronic Systems

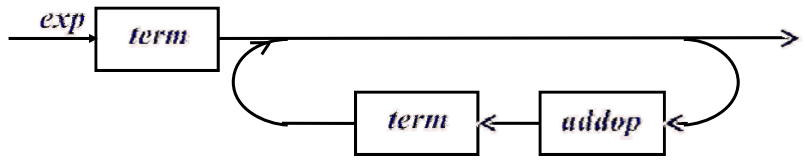
EBNF expression example

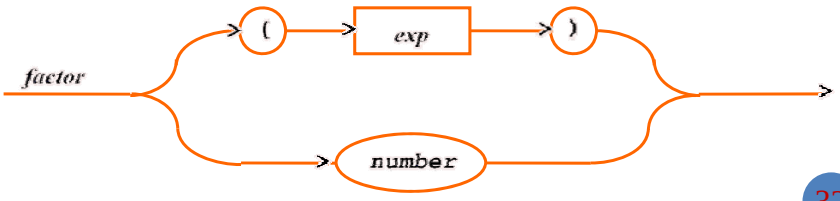
exp → *term* { *addop term* }
addop → + | -
term → *factor* { *mulop factor* }
mulop → *
factor → (*exp*) | number

31

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Syntax Diagram for EBNF





32

Dr. D. M. Akbar Hussain
Department of Electronic Systems