

# BNF, EBNF

국민대학교 컴퓨터공학부

강 승 식

# BNF: Backus-Naur Form

- BNF originally stood for “Backus Normal Form”
  - In 1964, Donald Knuth wrote a letter published in Communications of the ACM in which he suggests it stand for Backus-Naur form instead
- This was for two reasons:
  - To recognize Naur’s contribution
  - BNF is not technically a “normal form”; this would imply that there would be only one correct way of writing a grammar

# BNF example

$\langle \text{number} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{number} \rangle \langle \text{digit} \rangle$

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

- “ $::=$ ” means “is defined as”
  - some variants use “ $:=$ ” instead
- “ $\mid$ ” means “or”
- Angle brackets mean a **nonterminal symbols** and without angle brackets are **terminal symbols**.

# More BNF Examples

<while loop> ::= 'while' ( <condition> ) <statement>  
<assignment statement> ::= <variable> = <expression>  
<statement list> ::= <statement>  
                  | <statement list> <statement>  
<unsigned integer> ::= <digit>  
                  | <unsigned integer><digit>

<condition> ::= ...  
<statement> ::= ...  
<variable> ::= ...  
<expression> ::= ...  
<digit> ::= ...

# BNF for Expressions

$\langle \text{expression} \rangle ::= \langle \text{expression} \rangle + \langle \text{term} \rangle$   
                   $| \langle \text{expression} \rangle - \langle \text{term} \rangle$   
                   $| \langle \text{term} \rangle$

$\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$   
                   $| \langle \text{term} \rangle / \langle \text{factor} \rangle$   
                   $| \langle \text{factor} \rangle$

$\langle \text{factor} \rangle ::= \langle \text{primary} \rangle ^ \langle \text{factor} \rangle$   
                   $| \langle \text{primary} \rangle$

$\langle \text{primary} \rangle ::= \langle \text{primary} \rangle$   
                   $| \langle \text{element} \rangle$

$\langle \text{element} \rangle ::= ( \langle \text{expression} \rangle )$   
                   $| \langle \text{variable} \rangle$   
                   $| \langle \text{number} \rangle$

$\langle \text{variable} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{variable} \rangle \langle \text{letter} \rangle$   
                                   $| \langle \text{variable} \rangle \langle \text{number} \rangle$

$\langle \text{letter} \rangle ::= a \mid b \mid \dots \mid z \mid A \mid B \mid \dots \mid Z$

$\langle \text{number} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{number} \rangle \langle \text{digit} \rangle$

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

# What is EBNF?

- It makes expressing grammars more convenient
- EBNF is no more “powerful” than BNF
  - anything that can be expressed in EBNF can also be expressed in BNF
- EBNF is widely used as the de facto standard to define programming languages

# What are the Extensions?

- Derived from regular expression syntax
- “\*” (The Kleene Star): 0 or more occurrences
- “+” (The Kleene Cross): 1 or more occurrences
- “?”: 0 or 1 occurrences
  - sometimes “[ ... ]” is used instead
- Use of parentheses for grouping

# BNF vs. EBNF

- BNF

$\langle \text{expr} \rangle ::= \langle \text{digits} \rangle$

$\langle \text{digits} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{digits} \rangle$

$\langle \text{id} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{id} \rangle \langle \text{letter} \rangle \mid \langle \text{id} \rangle \langle \text{digit} \rangle$

- EBNF

$\langle \text{expr} \rangle ::= \langle \text{digit} \rangle^+$

$\langle \text{id} \rangle ::= \langle \text{letter} \rangle (\langle \text{letter} \rangle \mid \langle \text{digit} \rangle)^*$



# BNF vs. EBNF

- BNF

`<expr> ::= '-' <num> | <num>`

`<num> ::= <digits> | <digits> '.' <digits>`

`<digits> ::= <digit> | <digit> <digits>`

`<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'`

- EBNF

`<expr> ::= '-' ? <digit>+ ('.' <digit>+)?`

`<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'`

# EBNF for Lisp

$\langle s\_expression \rangle ::= \langle atomic\_symbol \rangle$   
                   $| "(" \langle s\_expression \rangle "." \langle s\_expression \rangle ")"$   
                   $| \langle list \rangle$

$\langle list \rangle ::= "(" \langle s\_expression \rangle^* ")"$

$\langle atomic\_symbol \rangle ::= \langle letter \rangle \langle atom\_part \rangle$

$\langle atom\_part \rangle ::= \epsilon$   
                   $| \langle letter \rangle \langle atom\_part \rangle$   
                   $| \langle number \rangle \langle atom\_part \rangle$

*empty string*

$\langle letter \rangle ::= "a" | "b" | \dots | "z"$

$\langle number \rangle ::= "1" | "2" | \dots | "9"$

# Extended BNF

- [ ] --- optional parts

$\langle \text{if\_stmt} \rangle ::= \text{'if'} \langle \text{cond} \rangle \text{'then'} \langle \text{stmt} \rangle [ \text{'else'} \langle \text{stmt} \rangle ]$

- ( ) --- alternative parts

$\langle \text{exp} \rangle ::= \langle \text{exp} \rangle ( \text{'+'} \mid \text{'-'} \mid \text{'*'} \mid \text{'/'} ) \langle \text{exp} \rangle$

- { } --- repetitions

$\langle \text{id} \rangle ::= \langle \text{letter} \rangle \{ \langle \text{letter} \rangle \mid \langle \text{digit} \rangle \}$

$\langle \text{letter} \rangle ::= \text{'a'} \mid \text{'b'} \mid \text{'c'} \mid \dots \mid \text{'z'}$

$\langle \text{digit} \rangle ::= \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \dots \mid \text{'9'}$

# BNF and EBNF

- BNF

```
<expr> → <expr> '+' <term>
        | <expr> '-' <term>
        | <term>
<term>  → <term> '*' <factor>
        | <term> '/' <factor>
        | <factor>
```

- EBNF

```
<expr> → <term> { ('+' | '-') <term> }
<term> → <factor> { ('*' | '/') <factor> }
```

# Conversion from EBNF to BNF and Vice Versa

- BNF to EBNF

- (i) Look for recursion in grammar

- $A ::= a A \mid B$

- $\Rightarrow A ::= a \{ a \} B$

- (ii) Look for common string that can be factored out

- $A ::= a B \mid a$

- $\Rightarrow A ::= a [B]$

- EBNF to BNF

- (i) Options:  $[]$

- $A ::= a [B] C$

- $\Rightarrow A' ::= a N C \quad N ::= B \mid \epsilon$

- (ii) Repetition:  $\{\}$

- $A ::= a \{ B_1 B_2 \dots B_n \} C$

- $\Rightarrow A' ::= a N C \quad N ::= B_1 B_2 \dots B_n N \mid \epsilon$