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

<number> ::= <digit> | <number> <digit>

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

- "::=" means "is defined as"
  - some variants use ":=" instead
- "|" 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

<expression> ::= <expression> + <term>
| <expression> - <term>
| <term>
<term> ::= <term> * <factor>
| <term> / <factor>
| <factor>
<factor> ::= <primary> ^ <factor>
| <primary>
<primary> ::= <primary>
| <element>
<element> ::= (<expression>)
| <variable>
| <number>

<variable> ::= <letter> | < variable > <letter>
| < variable > <number>
<letter> ::= a | b | … | z | A | B | … | Z

<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 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

  <expr> ::= <digits>

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

  <id> ::= <letter> | <id><letter> | <id><digit>

- EBNF

  <expr> ::= <digit>+

  <id> ::= <letter> (<letter> | <digit>)*

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

<s_expression> ::= <atomic_symbol>
| "(" <s_expression> "." <s_expression> ")"
| <list>
<list> ::= "(" <s_expression>* ")"
<atomic_symbol> ::= <letter> <atom_part>
<atom_part> ::= ε
| <letter> <atom_part>
| <number> <atom_part>
<letter> ::= "a" | "b" | ... | "z"
<number> ::= "1" | "2" | ... | "9"

Extended BNF

- [ ] --- optional parts

  <if_stmt> ::= ‘if’ <cond> ‘then’ <stmt> [‘else’ <stmt>]

- ( ) --- alternative parts

  <exp> ::= <exp> (‘+’ | ‘-’ | ‘*’ | ‘/’) <exp>

- { } --- repetitions

  <id> ::= <letter> { <letter> | <digit> }

  <letter> ::= ‘a’ | ‘b’ | ‘c’ | … | ‘z’

  <digit> ::= ‘0’ | ‘1’ | ‘2’ | … | ‘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 | B
    - ⇒ A ::= a { a } B
  - (ii) Look for common string that can be factored out
    - A ::= a B | a
    - ⇒ A := a [B]
- EBNF to BNF
  - (i) Options: []
    - A ::= a [B] C
    - ⇒ A' ::= a N C N ::= B | ε
  - (ii) Repetition: {}
    - A ::= a { B1 B2 ... Bn } C
    - ⇒ A' ::= a N C N ::= B1 B2 ... Bn N | ε