316 9.22.09

# Extended BNF (EBNF)

- "syntactic sugars" for BNF, providing concise notations
- provide no additional power compared to original BNF
  - any grammar defined by EBNF can always be defined by equivalent, original BNF
- mandatory multiple choice $(\alpha_1 | \ldots | \alpha_n)$ denotes a mandatory choice of exactly one $\alpha_i$
- optional multiple choice $[\alpha_1 | \ldots | \alpha_n]$ denotes an optional choice of at most one $\alpha_i$
- zero or more iterations $\{\alpha_i\}$ denotes zero or more iterations of $\alpha$, i.e. $\epsilon, \alpha, \alpha\alpha, \alpha\alpha\alpha, \ldots \alpha^i, \ldots$
- one or more iterations $\{\alpha\}^+$ denotes one or more iterations of $\alpha$, i.e. $\alpha, \alpha\alpha, \alpha\alpha\alpha, \ldots, \alpha^i, \ldots$

$\{(\alpha_1 | \ldots | \alpha_n)\}$ and $\{(\alpha_1 | \ldots | \alpha_n)\}^+$ may be abbreviated to, respectively, $\{\alpha_1 | \ldots | \alpha_n\}$ and $\{\alpha_1 | \ldots | \alpha_n\}^+$

<int> → {<digit>}+

<id> → <letter>{<digit> | <digit>}

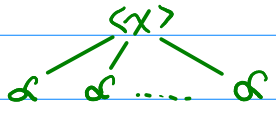
<float> → {<digit>}+ "." {<digit>}+ [(E|e)[+|-]{<digit>}+]

<E> → <term>{(+|-)<term>}

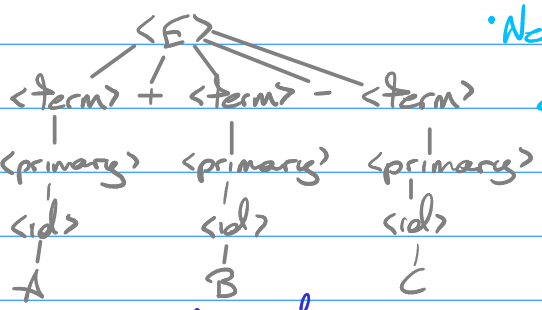
<term> → <primary>{*|/)<primary>}

<primary> → <id>|<int>|<float>|"("<E>")"

Parse trees for $\langle X \rangle \rightarrow \{\alpha\}$, $\langle X \rangle \rightarrow \{\alpha\}^+$




example: A+B-C




- Associativity information is not specified
- Need a comment about the internal associativity of operators
- In general, EBNF may introduce ambiguity, which must be resolved by concise comments

statements category

terminal symbols

<S> → <id> = <E>;

if "(" <B> ")" <S> [else <S>] |

while "(" <B> ")" <S> |

do <S> while "(" <B> ")" |

"{" <S List> "}"

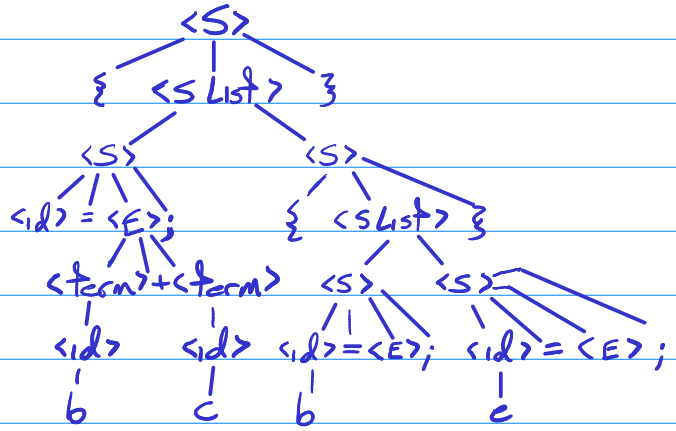
<S List> → {<S>}+

ambiguity!

comment: Each occurrence of else matches the closest preceding unmatched if.

Construct a parse tree for:

```
{ a = b + c;
  { b = c / d;
    e = b;
  }
}
```




## Metasymbols of BNF/EBNF

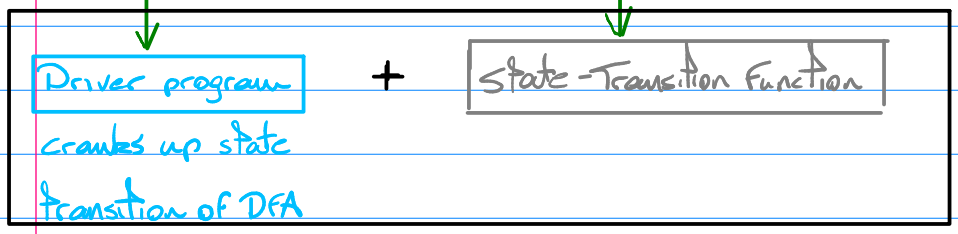
|, <, >, (, ), [, ], {, }

If any of these needs to be used as a terminal symbol, enclose it by "..." (example: "<", "|", "{", etc..)

## Methods of Lexical Analysis

- BNF/EBNF for tokens can be done by a small subset of BNF/EBNF corresponding to regular expressions (no need for any recursive production rules, but you need $\{\alpha\}$ and $\{\alpha\}^+$

Construct a Deterministic Finite Automaton (DFA)




Lexical Analyzer

## Review of DFA's

- A DFA consists of:
  - a finite set I of symbols
  - a finite set S of states
  - a special state $s \in S$, called the initial (start) state
  - a subset $F \subseteq S$, called the final state
  - a state-transition function $\delta : S \times I \to S$

Let $\delta(s, i) = S'$ In diagram:  Use  for final states

- <plus> → +
- <minus> → −
- <times> → *
-  → /
- <LParen> → "("

<RParen> → ")"

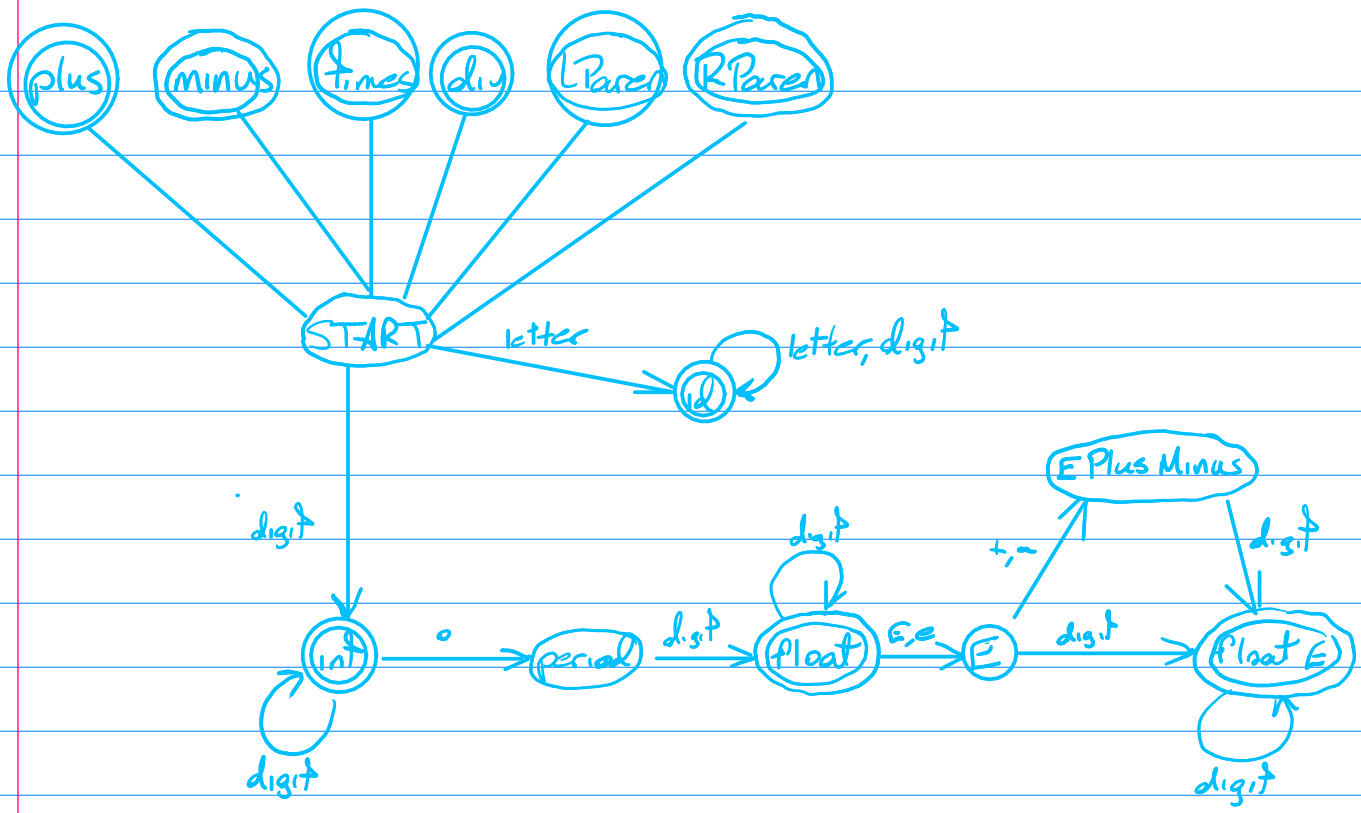
<int> → {<digit>}$^+$

<id> → <letter> {<letter> | <digit>}

<float> → {<digit>}$^+$ "." {<digit>}$^+$

<float E> → <float> (E|e) [+|-] {<digit>}$^+$

<digit> → 0|1|...|9| <letter> → a|b|...|z|A|B|...|Z


plus
minus
times
div
LParen
RParen
START
letter
letter, digit
id
digit
int
digit
.
digit
float
E,e
E
+,-
E Plus Minus
digit
digit
float E
digit