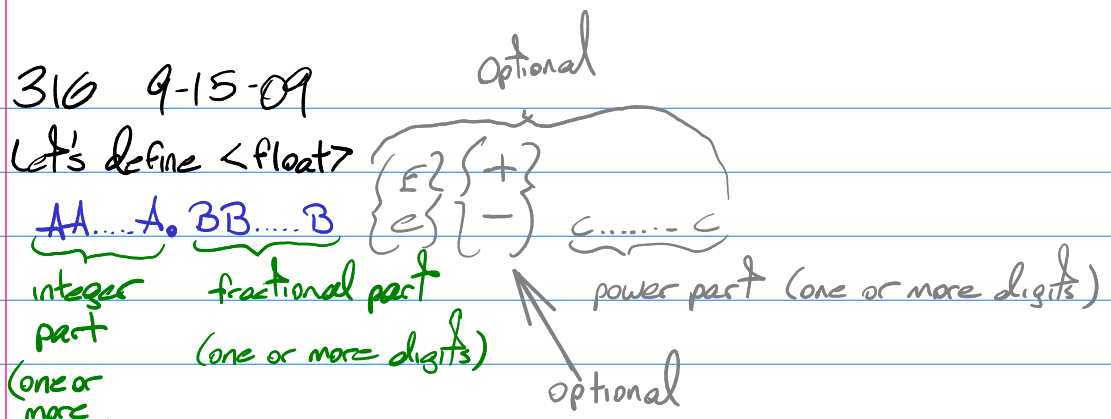
316 9-15-09

- Let's define <float>



$23.45e-5 = 23.45 \times 10^{-5}$

<float> → <int> "." <int> | <int> "." <int> <exponent>

<exponent> → <esign> <int> | <esign> <sign> <int>

<esign> → e | E

<sign> → + | -

One-step derivations from last class ($1.2E-5 = 1.2 \times 10^{5}$)

<float> ⇒ <int>.<int><exponent>

⇒ ...

⇒ 1.<int><exponent>

⇒ ...

⇒ 1.2<exponent>

⇒ 1.2 <esign><sign><int>

⇒ 1.2E <sign><int>

⇒ ...

⇒ 1.2E-5

Arithmetic expressions (example of non-token category)

<int>, <id>, <float>, +, -, *, /, (,)

(variable names)

<E> → <int> | <float> | <id> | <E> + <E> | <E> - <E> | <E> * <E> | <E> / <E> | "(" <E> ")"

ex = Derive A+B/C

$\langle E \rangle \Rightarrow \langle E \rangle + \langle E \rangle$

$\Rightarrow \langle id \rangle + \langle E \rangle$

$\Rightarrow \ldots$ (skipping derivation of A)

$\Rightarrow A + \langle E \rangle / \langle E \rangle$

$\Rightarrow A + \langle id \rangle / \langle E \rangle$

$\Rightarrow \ldots$ (skipping derivation of B)

$\Rightarrow A + B / \langle E \rangle$

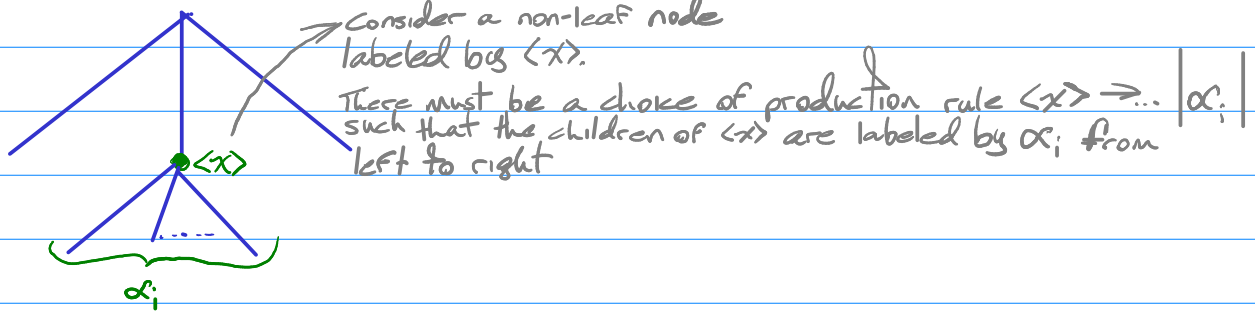
$\Rightarrow A + B / \langle id \rangle$

$\Rightarrow \ldots$ (skipping derivation of C)

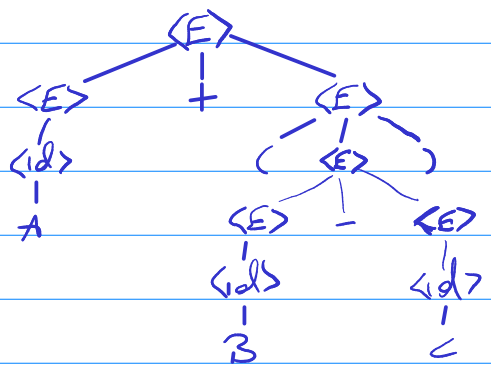
$\Rightarrow A + B / C$

Parse Trees

- Show derivations in tree form (can readily identify syntactic structure)
- Constructed by the compiler's syntactic analyzer (parser)
- Constructed for non-token categories
- Each non-leaf node is labeled by a syntactic category
- Each leaf node is labeled by a token

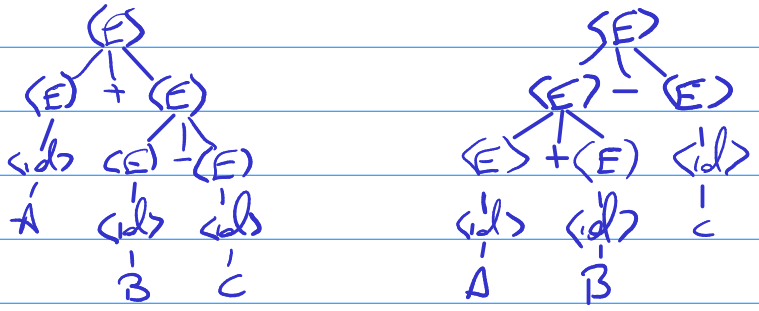


Construct a parse tree for A+(B-C)



(2)

Construct all the parse trees for A+B-C

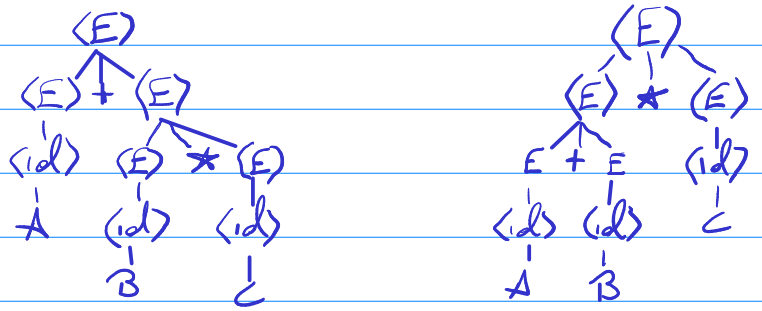


Both expressions are mathematically equivalent.

Because subtraction occurs first, this expression won't return an overflow error.

[Because addition occurs first] This expression may result in an overflow error (if A and B are too large)

Construct Parse Trees for A+B*C



Not mathematically equivalent

Ambiguity Problem

- A syntactic category <X> in a BNF grammar is said to be ambiguous if there exists more than one parse tree, for at least one string of terminals belonging to <X>

BNF Grammars for a Programming Language

- Syntax must not be ambiguous.
  - Use completely unambiguous grammars (preferred)
  - In some exceptional cases, use ambiguous BNF and attach a disambiguous comment in English.

Boolean expression

statements

<condition> → if "(" <B> ")" <S> | if "(" <B> ")" <S> else <S>

<S> → ..... <condition>

<B> → .....

consider: if (B_1) if (B_2) S_1 else S_2