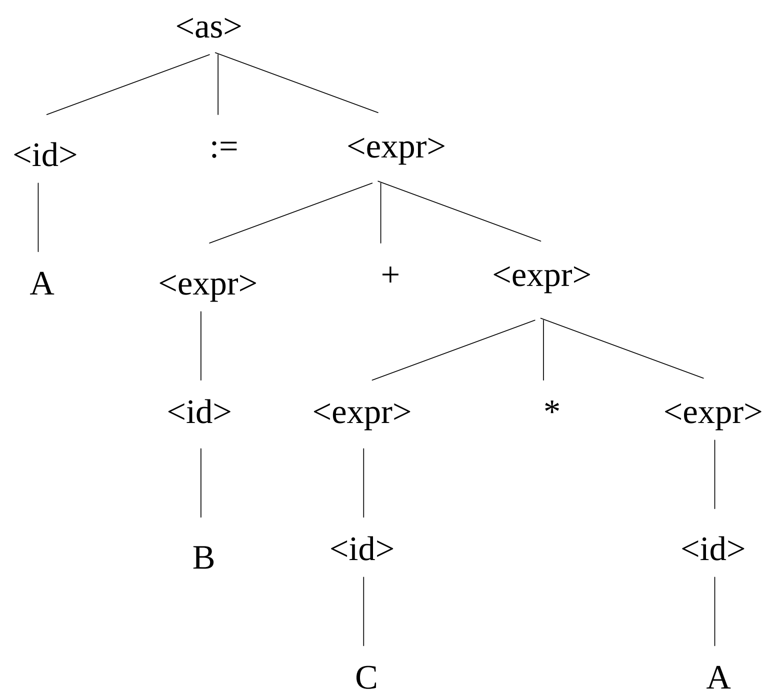
Ambiguity: Consider the following grammar for simple assignment statements:

<as> → <id> := <expr>

<id> → A | B | C

<expr> → <expr> + <expr>
| <expr> * <expr>
| (<expr>)
| <id>

We can parse A := B + C * A in two ways:



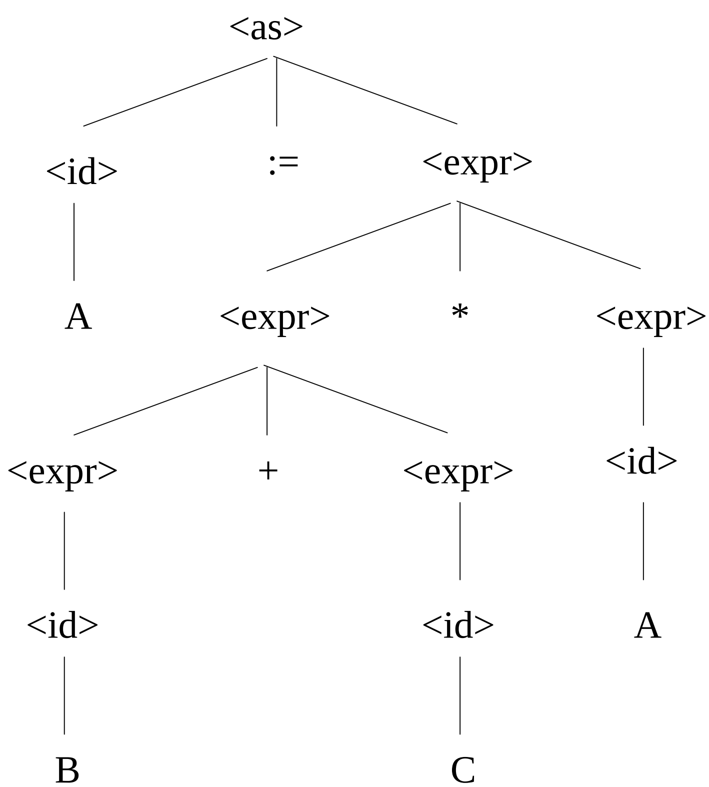
Suggests that B + (C * A) is computed

<as> → <id> := <expr>

<id> → A | B | C

<expr> → <expr> + <expr>
| <expr> * <expr>
| (<expr>)
| <id>

But we can also build the parse tree:



Suggests that (B + C) * A is computed

If any sentence in L(G) has more than one parse tree, then we say that G is *ambiguous*.

Can we give a grammar for this language of assignments that is **not** ambiguous?

If so, we prefer to correlate the parse tree with the intended meaning; in this case, operator precedence.

Here is a new grammar that defines exactly the same language as does the previous grammar.

<as> → <id> := <expr>

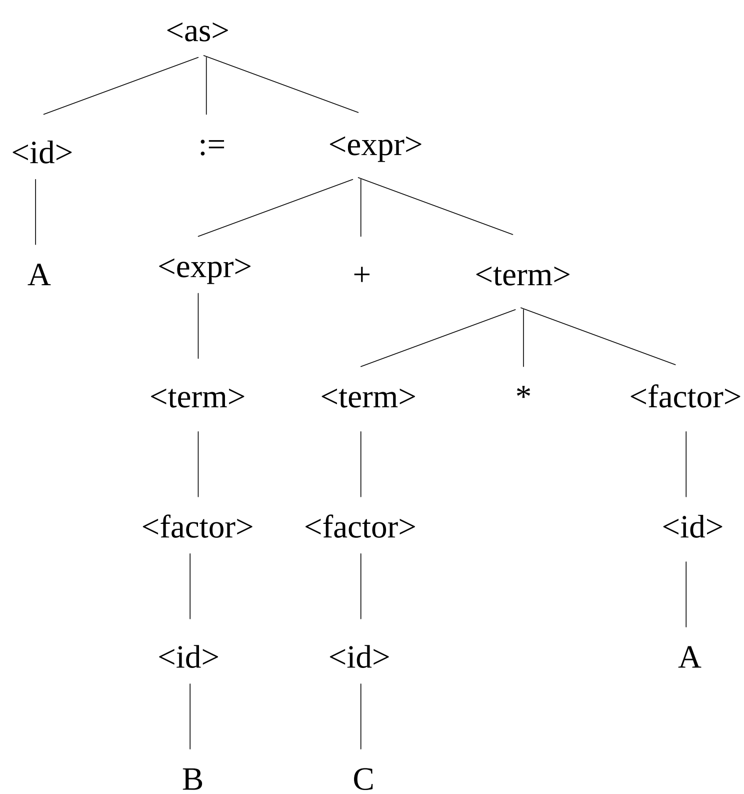
<id> → A | B | C

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

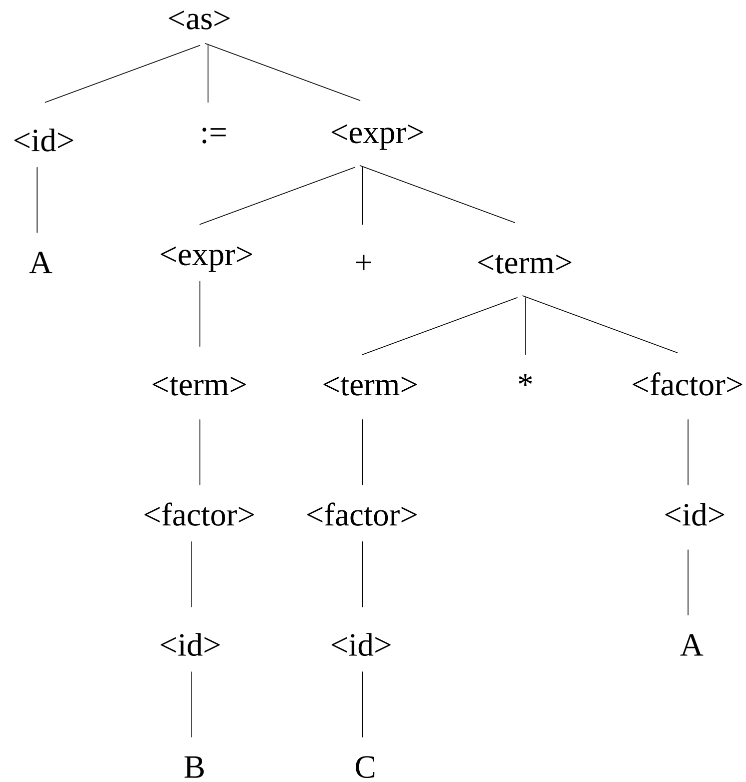
<factor> → (<expr>) | <id>

But there is only one parse tree for A := B+C*A



Note that * has precedence over +

The unique parse tree



corresponds to more than one derivation:

<as> $\Longrightarrow$ <id> := <expr>
$\Longrightarrow$ A := <expr>
$\Longrightarrow$ A := <expr> + <term>
$\Longrightarrow$ A := <term> + <term>
$\Longrightarrow$ A := <factor> + <term>
$\Longrightarrow$ A := <id> + <term>
$\Longrightarrow$ A := B + <term>
$\Longrightarrow$ A := B + <term> * <factor>
$\Longrightarrow$ A := B + <factor> * <factor>
$\Longrightarrow$ A := B + <id> * <factor>
$\Longrightarrow$ A := B + C * <factor>
$\Longrightarrow$ A := B + C * <id>
$\Longrightarrow$ A := B + C * A

Cannonical

<as> $\Longrightarrow$ <id> := <expr>
$\Longrightarrow$ A := <expr>
$\Longrightarrow$ A := <expr> + <term>
$\Longrightarrow$ A := <term> + <term>
$\Longrightarrow$ A := <factor> + <term>
$\Longrightarrow$ A := <id> + <term>
$\Longrightarrow$ A := B + <term>
$\Longrightarrow$ A := B + <term> * <factor>
$\Longrightarrow$ A := B + <term> * <id>
$\Longrightarrow$ A := B + <term> * A
$\Longrightarrow$ A := B + <factor> * A
$\Longrightarrow$ A := B + <id> * A
$\Longrightarrow$ A := B + C * A