Context Free Grammars and Ambiguity

Introduction to CFGs and Ambiguity

1. More to a language than just words
    - Examine the grammatical concerns of programming languages

Parsing Definition Quiz

1. What is the role of the parser in compilers?
    - Lexical analysis of the program to generate tokens
    - Syntactic analysis of the program using the grammar of the programming language (true)
    - Breaking down an expression in the program into subexpressions as per operator precedences to put a structute on the same determining order of its evaluation (true)
    - Check if a variable is declared before it is used

Context Free Grammars

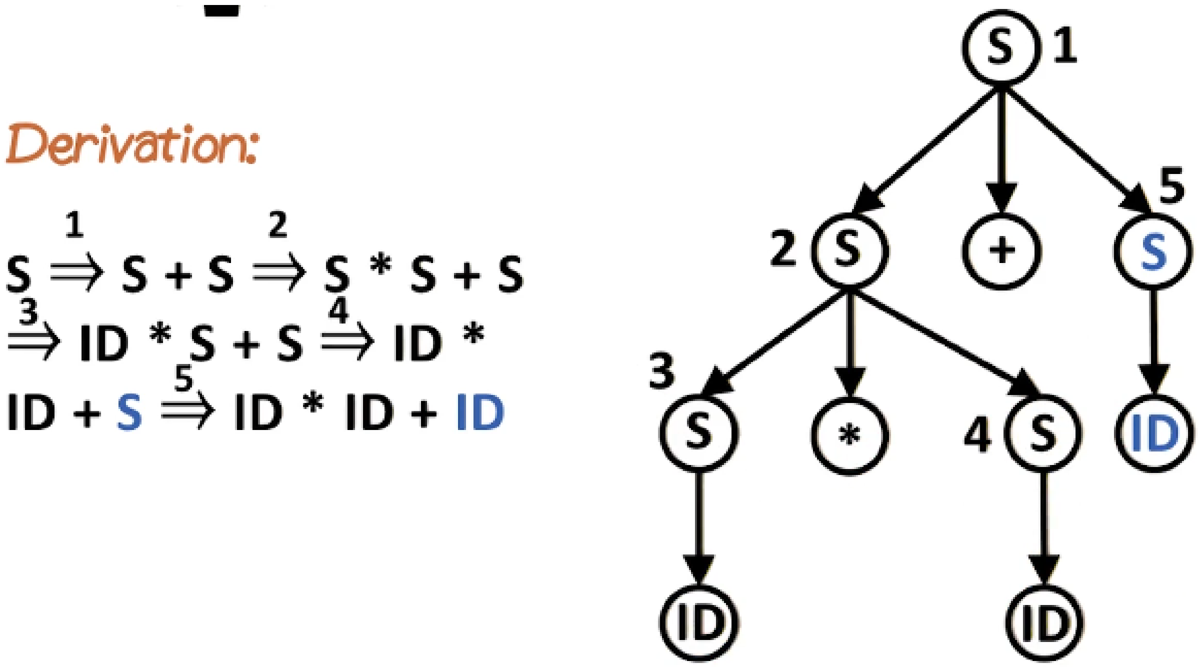
1. Generative aspect of CFG
    - It is easy to derive strings w in L(G) from a CFG G
2. Analytical aspect of CFG
    - Given a CFG G and string w, decide if w in L(G) and if so how do you determine the derivation tree or the sequence of rules that produce w?
    - This is the problem of parsing

Derivation Example Part 1

1. Grammar
    - S -> S+S | S*S | (S) | ID
2. String:
    - ID * ID + ID

Derivation Example Part 2

1. Derivation:
    - S => S + S => S * S + S => ID * S + S => ID * ID + S => ID * ID + ID
    - Choose the leftmost symbol which is unexpanded and apply the rule of expansion to it



Derivation

Derivation Example Left and Right

1. Left-most Derivations:
   - At each step, replace the leftmost non-terminal
2. Right-most Derivations:
   - At each step, replace the rightmost non-terminal
3. Both produce the same tree, but through two different processes

Defining a Parse Tree

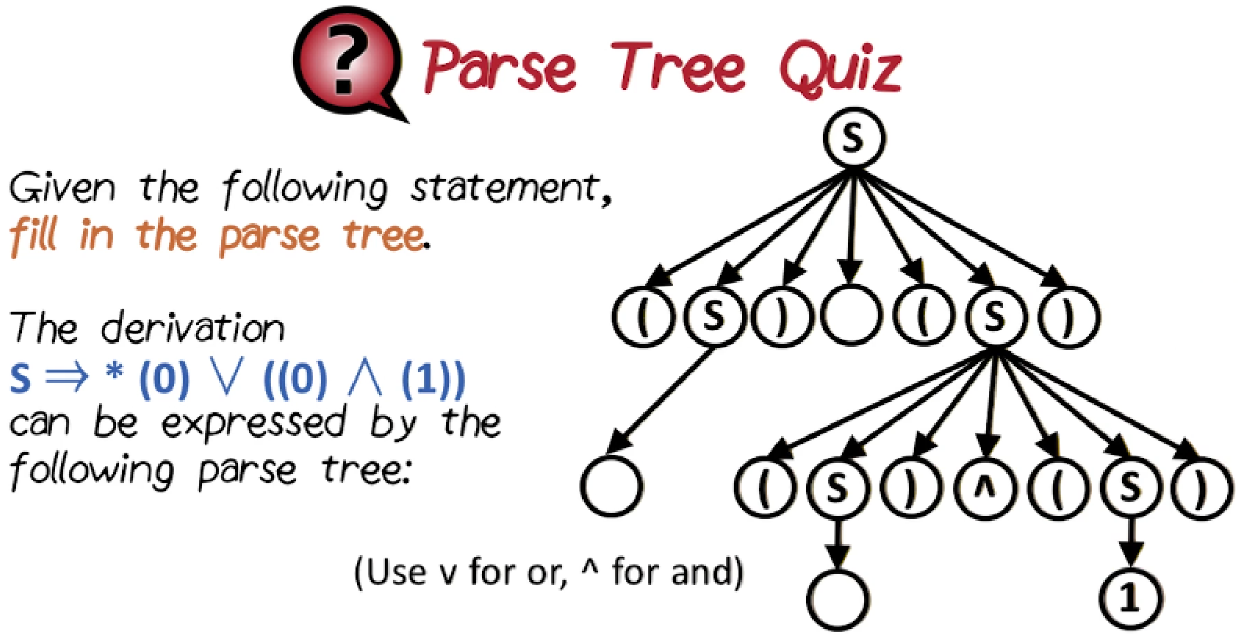
1. For a CFG G = (V, E, R, S) a derivation tree has the following properties:
   - V, E, R, S = Terminals, non-terminals, rules, and a start symbol
   - The root is labeled S
   - Each leaf is from E U {e}
   - Each interior node is in V
   - If node has label A in V and its children a1…an (from L to R), then P must have the rule A -> a1…an (with aj in V U T U {l}) A leaf labeled e is a single child (has no siblings)
   - Let G be a CFG. We have w in L(G) if and only if there exists a derivation tree of G that yields w.

Derivation Example 2

1. Consider CFG S -> 0 | 1 | !(S) | (S) | (S) | (S) & (S) for terminals "0", "1", "!", "|", "&", "(", and ")"
   - Derivations of "(0) | ((0) & (1))"
   - Leftmost: S => (S) | (S) => (0) | (S) => (0) | ((S) & (S)) =>
  (0) ((0) & (S)) => (0) | ((0) & (1))
   - Rightmost: S => (S) | (S) => (S) | ((S) & (S)) => (S) | ((S) & (1)) =>
  (S) ((0) & (1)) => (0) | ((0) & (1))
   - Could also expand in a random order, but uncommon
     - Most compilers match leftmost because tokens are being matched from left to right

Parse Tree Quiz

1. Given the following statement, fill in the parse tree.
2. The derivation S => * (0) | ((0) & (1)) can be expressed by the following parse tree:
    - Use v for or, ˆ for and



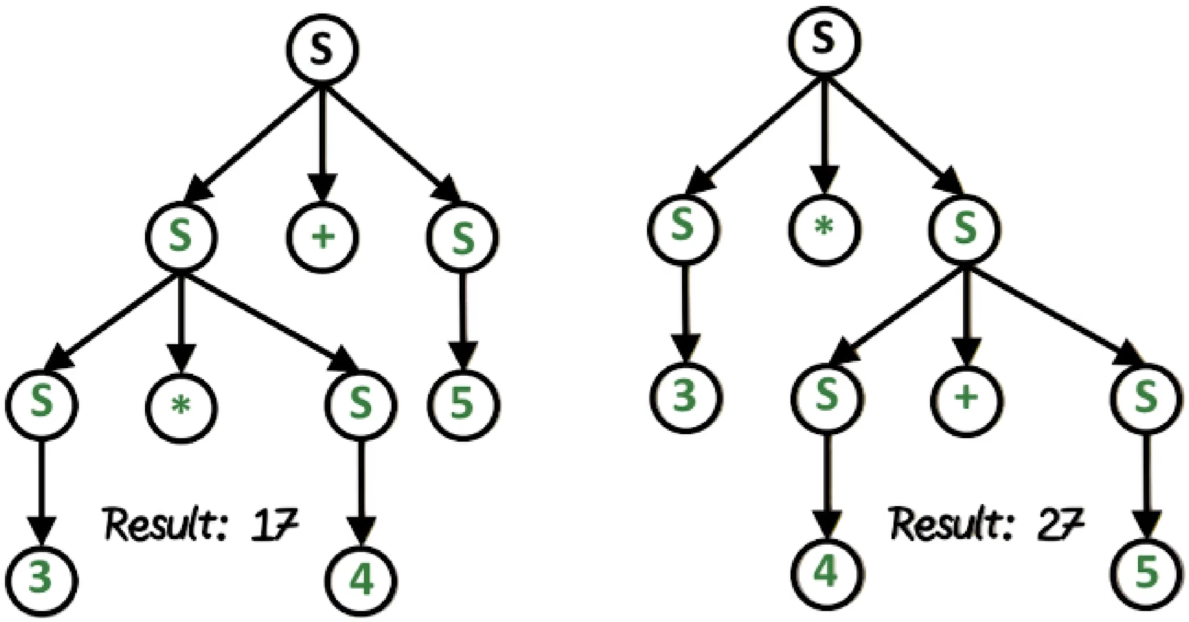
Quiz 1

Ambiguity

1. A string w in L(G) is derived ambiguously if it has more than one derivation tree (or equivalently: if it has more than one leftmost derivation (or rightmost)).
    - Derivation tree must be unique, regardless of how we parse it
2. A grammar is ambiguous if some strings are derived ambiguously
    - Example: Rule S -> 0 | 1 | S+S | S*S
    - These two derivations lead to the same string but are different
        - S => S+S => S*S+S | 0*S+S => 0*1+S => 0*1+1
        - S => S*S => 0*S | 0*S+S => 0*1+S => 0*1+1

Ambiguity and Parse Trees Quiz

1. Fill in the two different parse trees for the expression: 3*4+5



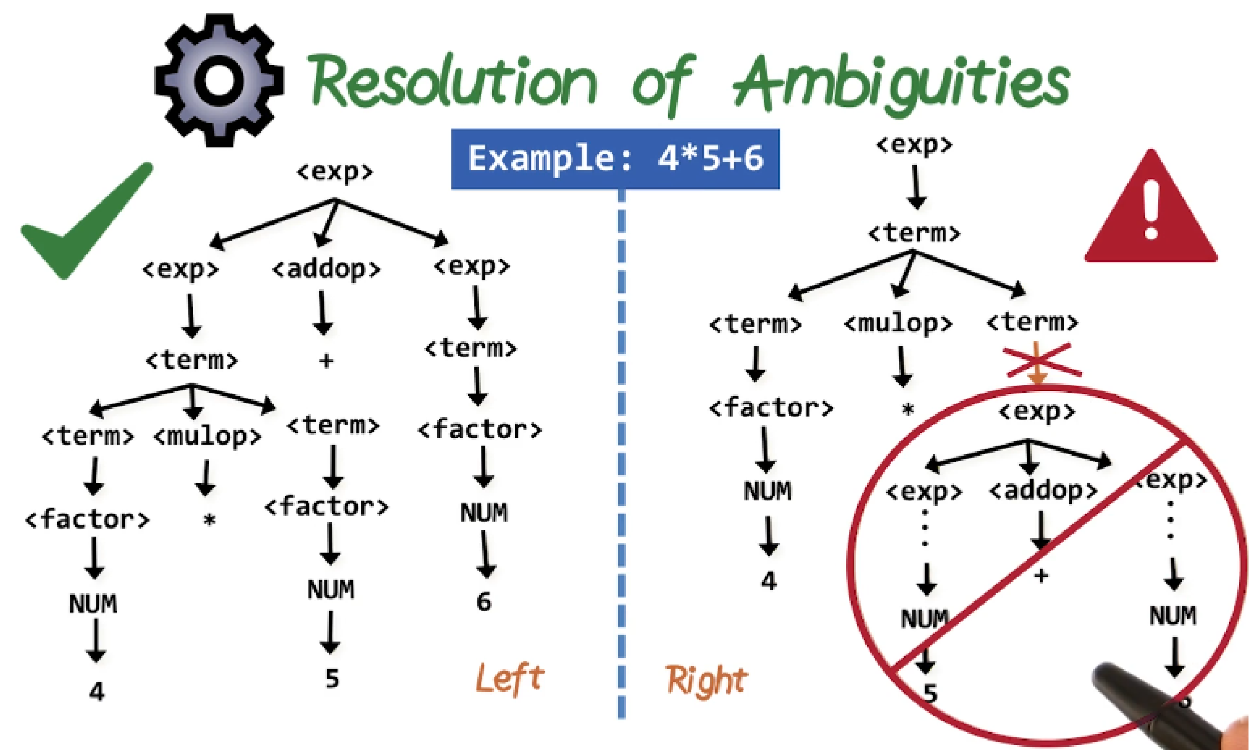
Quiz 2

Resolution of Ambiguities: Introduction

1. Some ambiguities are inessential but some others must be resolve
    - The following grammar is ambiguous:
        - exp -> exp op exp | (exp) | number op -> +|-|*
        - Sampel ambiguous strings: 1+2*3 and 1-2-3
    - Resolution of ambiguity:
        - Precedence: * has higher precedence than + and -
        - Left-association: Performs ops from left to right
        - Full parenthesization
2. Precedence: Group operators into different groups and make operations with lower precedence closer to the root
    - exp -> exp addop exp | term
    - addop -> + | -
    - term -> term mulop term | factor
    - mulop -> *
    - factor -> (exp) | number

Resolution of Ambiguities: Example

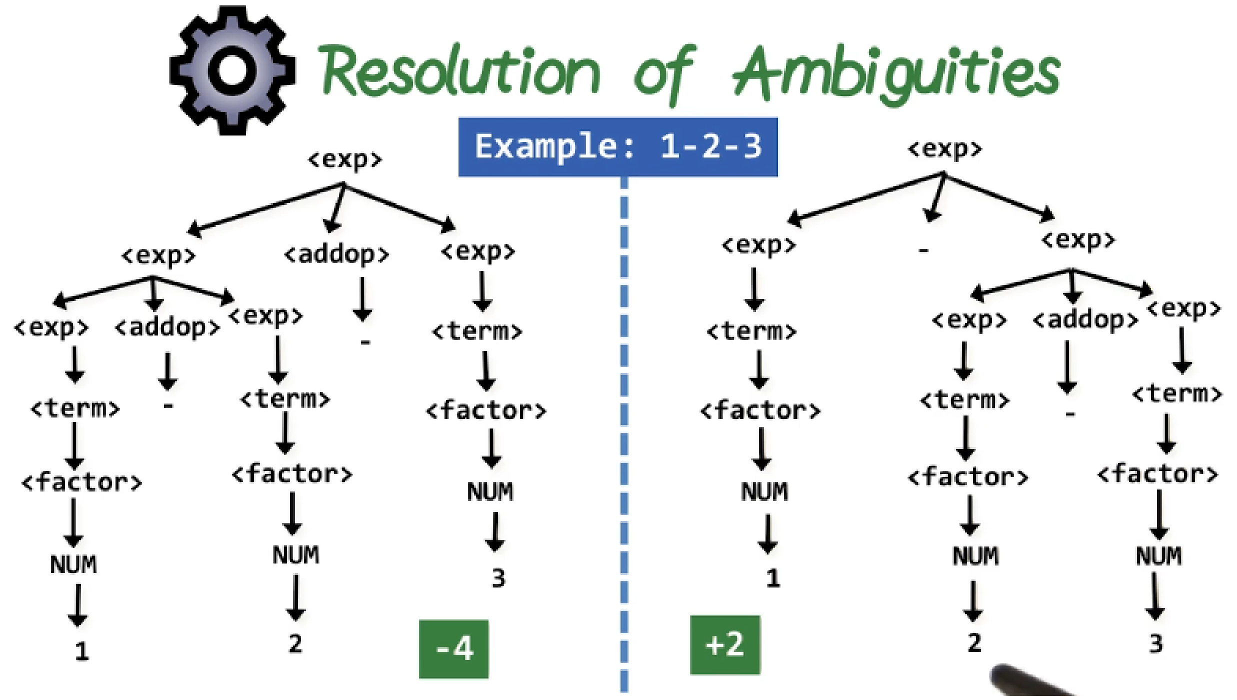
1. Assume the parser can make the right decision when first choosing addop
    - Because the term operator is below addop, we can't turn a term into and addop, so the example on the right doesn't work
    - Only one legal parse tree shown on left
    - Parse tree on right not possible. <term> can not derive <exp> and therefore <addop>
    - Will study how the parser makes decisions in the future



Resolving Ambiguity

Resolution of Ambiguities: Associativity

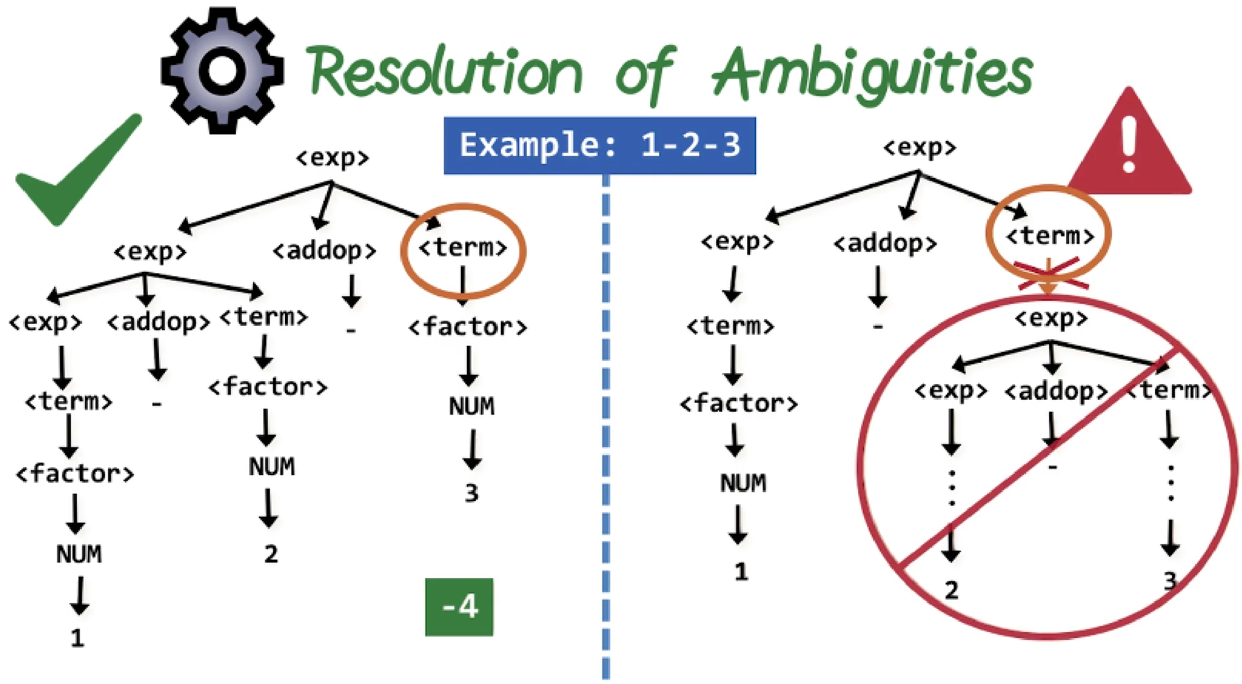
1. Associativity: Allow recursion only on left
   - exp -> exp addop term | term
   - term -> term mulop factor | factor
   - Only allows recursion on left implement associativity
   - exp causes the problem
2. Ambiguity is still present, left associativity not enforced
   - Results are different



Associativity

Resolution of Ambiguities: Example 2

- Left associative operator: minus
- Only legal parse tree on the left side



Resolving Ambiguity

Dangling Else Statement

1. Dangling else statement
   - To which if do we associate the else?
2. Ambiguous grammar:
   - statement -> if-stmt | other
   - if-stmt -> if (exp) statement | if (exp) statement else statement
3. Example:

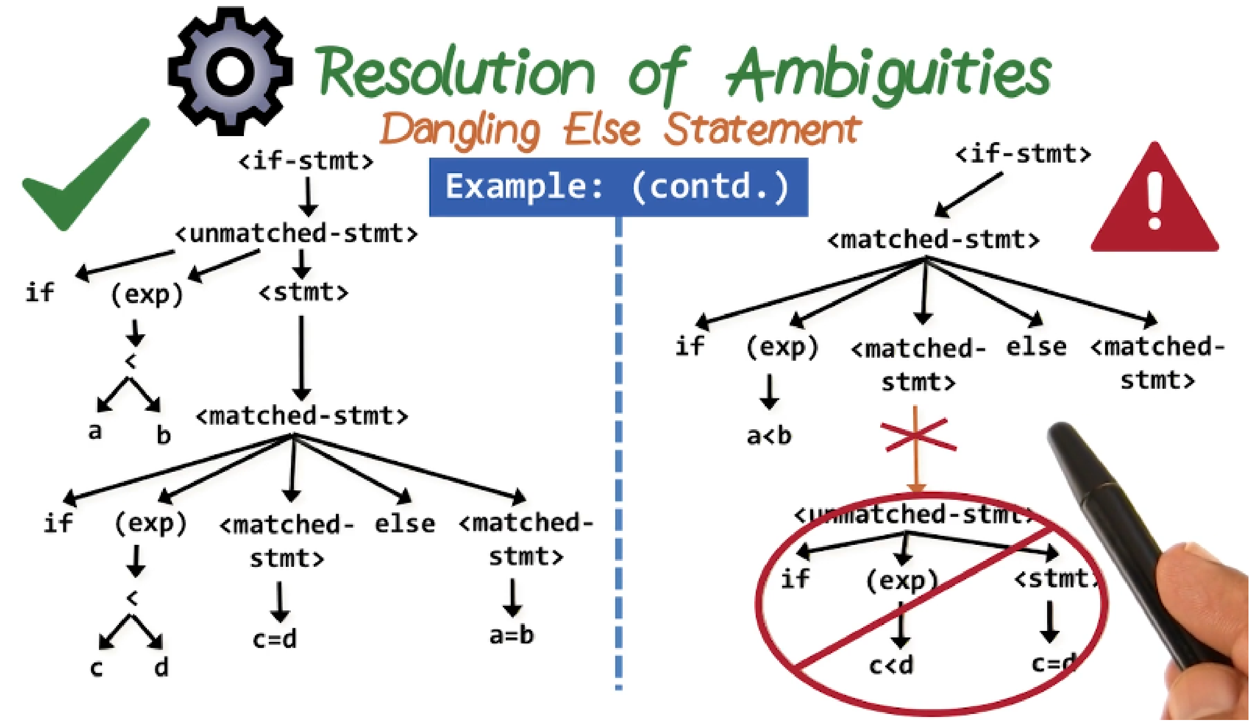
```
if (a < b)
    if (c < d)
        c = d;
    else
        a = b;
```

4. If we choose the if without else rule first, we match it correctly
   - However, if we choose with if with else rule first, we won't get the intended behavior
   - Else is associated with out if
     - c = d evaluates if both (a<b) and (c<d) are true
     - a = b evalues if (a<b) is false regardless of (c<d)
     - Semantics of program changed

Dangling Else Statement Resolution

1. Resolution
   - Bracketing with endif (e.g., shell script)
   - Revise the grammar
     - statement -> matched-stmt | unmatched-stmt
     - matched-stmt -> if (exp) matched-stmt else matched-stmt | other
     - unmatched-stmt -> if (exp) statement | if (exp) matched-stmt else unmatched-stmt

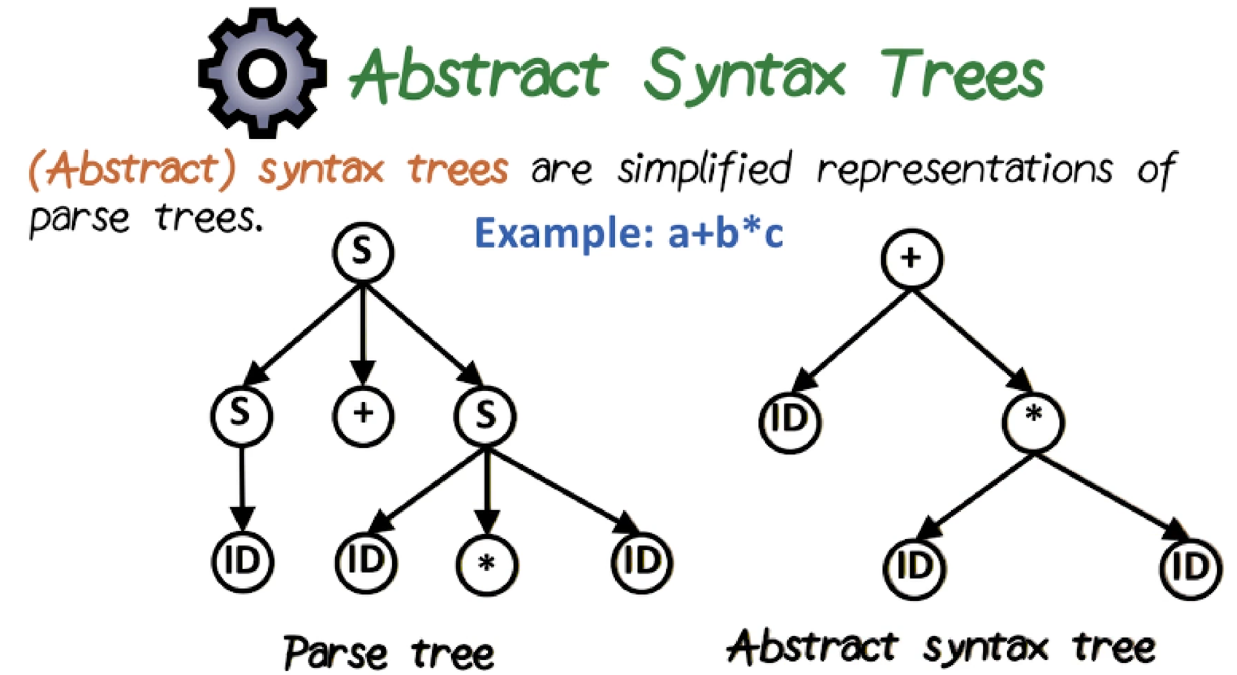
- Second derivation is not possible
  - Unambiguous grammar



Resolving Ambiguity

Abstract Syntax Trees

1. (Abstract) syntax trees are simplified representations of parse trees
   - Example: a + b * c



Abstract Syntax Tree