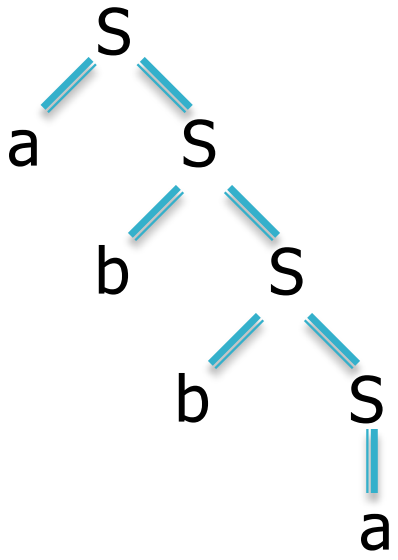
CFGs: Parse Trees and Ambiguity

Parse Trees

- Graphical means to illustrate a derivation of a string from a grammar
  - Root of the tree = start variable
  - Interior nodes = other variables
    - Children of nodes = application of a production rule
  - Leaf nodes = Terminal symbols

- Example: { wa | w ∈ {a,b}*}
  - S -> aS | bS | a
  - Suppose x = abba



A CFG for Algebraic Expressions

- Suppose we want to define a grammar that can generate algebraic expressions
  - We'll just use a single terminal, a, to represent any numeric value
  - E.g. a * (a + a*(a + a))

A CFG for Algebraic Expressions

- Defining the grammar for algebraic expressions:

- Terminals:
  - Let a be a numeric constant
  - Set of binary operators: {+, −, *, /}
  - Expressions can be parenthesized

A CFG for Algebraic Expressions

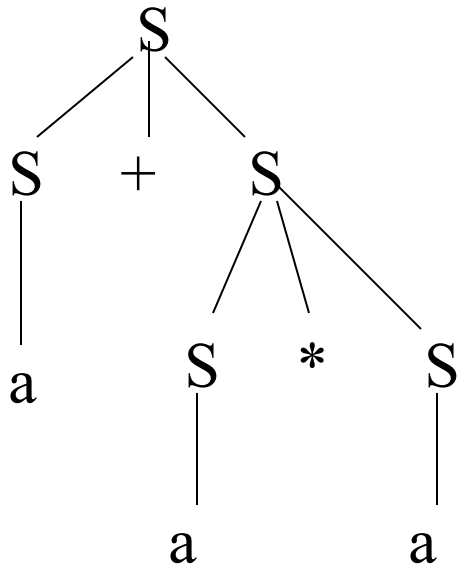
- Defining the grammar for algebraic expressions:
  - G = (V, Σ, R, S)
  - V = {S}
  - Σ = { a, −, +, *, /, (,) }
  - S = S
  - R = see next slide

A CFG for Algebraic Expressions

- Defining the grammar for algebraic expressions – Production rules
  - $S \rightarrow S + S$ (rule 1)
    $S \rightarrow S - S$ (rule 2)
    $S \rightarrow S * S$ (rule 3)
    $S \rightarrow S / S$ (rule 4)
    $S \rightarrow (S)$ (rule 5)
    $S \rightarrow a$ (rule 6)

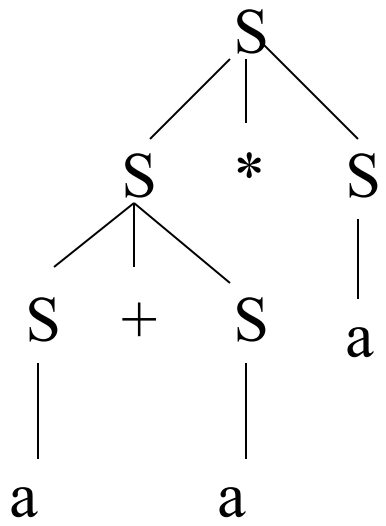
A CFG for Algebraic Expressions

- Show derivation for a + a * a
  - $S \Rightarrow S + S$ rule 1
    $S \Rightarrow^* a + S$ rule 6
    $S \Rightarrow^* a + S * S$ rule 3
    $S \Rightarrow^* a + a * S$ rule 6
    $S \Rightarrow^* a + a * a$ rule 6



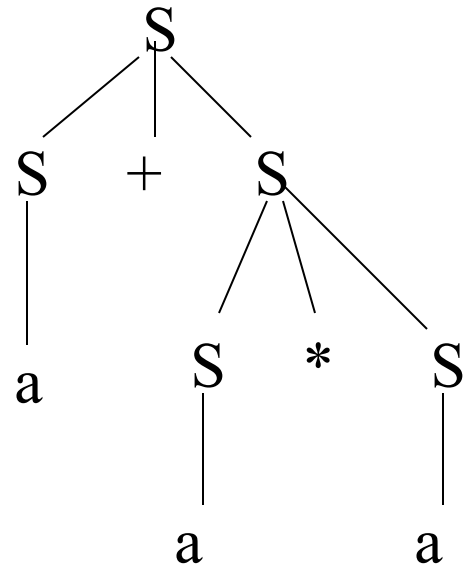
A Different Parse Tree

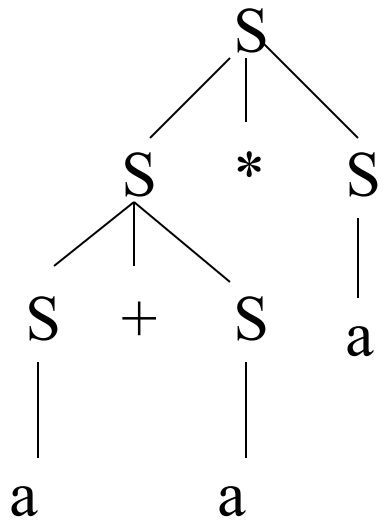
- Another derivation for a + a * a
  - $S \Rightarrow S * S$ rule 3
    $S \Rightarrow^* S * a$ rule 6
    $S \Rightarrow^* S + S * a$ rule 1
    $S \Rightarrow^* a + S * a$ rule 6
    $S \Rightarrow^* a + a * a$ rule 6



A String with 2 Parse Trees

- 1 string: a + a * a
- 2 different parse trees / derivations





Ambiguity

- A CFG is said to be ambiguous if there is at least 1 string in L(G) having two or more ***distinct*** derivations (i.e. different parse trees).

- In some applications, such as programming languages, this would be problematic
  - There needs to be a unique interpretation for each string

Ambiguity

- A CFG is said to be ambiguous if there is at least 1 string in L(G) having two or more ***distinct*** derivations (i.e. different parse trees).
- Some grammars are inherently ambiguous.
- Some grammars can have ambiguity removed without changing the language of the grammar.

Ambiguity

- To demonstrate that a particular grammar is ambiguous:
  - Find a string x in the L(G) that has two derivations
- To demonstrate that a particular grammar is not ambiguous
  - Can be difficult.
  - Need to argue that all strings have non-ambiguous derivation

Derivations

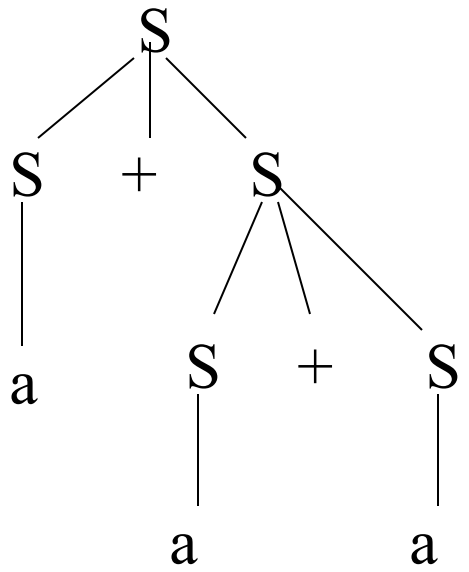
- Leftmost derivations
  - A leftmost derivation is one where the leftmost variable in the current string is always the first to get replaced via a production rule.
  - A rightmost derivation is one where the rightmost variable in the current string is always the first to get replaced via a production rule.

Ambiguity

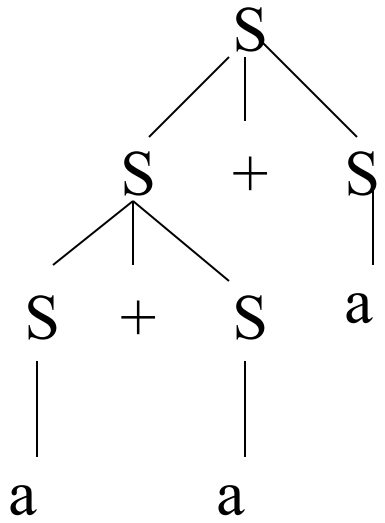
- As it turns out (we won't prove this)
  - In unambiguous grammars, leftmost derivations will always be unique.
  - In unambiguous grammars, rightmost derivations will always be unique.

Another Ambiguous String

- a + a + a



One possible leftmost derivation



Another possible leftmost derivation

Removing Ambiguities

- Some languages are inherently ambiguous
  - Removing ambiguities cannot always be done
- In fact,
  - We can/will show there is no “algorithm” for determining if a CFG is ambiguous
    - This is for later in the course
- However,
  - On a case by case basis, we may be able to remove ambiguities

Removing Algebraic Ambiguity

- Abbreviated grammar for algebraic expressions – Production rules
  - $S \rightarrow S + S$ (rule 1)
    $S \rightarrow S * S$ (rule 2)
    $S \rightarrow (S)$ (rule 3)
    $S \rightarrow a$ (rule 4)

(just ignore – and / to keep things simple)

Removing Algebraic Ambiguity

- This grammar has two problems:
  - Precedence of operators is not respected
    - a*a + a should be interpreted as (a*a) + a
  - Sequence of identical operators can be grouped either from the left or the right
    - a+a+a can be interpreted as either (a+a) + a or a + (a+a)

- We want to remove the ambiguity
  - Derive a CFG that generates the *same* language of algebraic expressions as before, but without any ambiguity

Example

- Solution
  - Introduce some new variables
    - Factor – expression that cannot be broken up by either * or +
      - a
      - (S)
    - Term – expression that cannot be broken up by +
      - All Factors
      - T * F
    - Expression – all possible expressions
      - All Terms
      - S + T

Example

- Our new grammar
  - S → S + T | T
  - T → T * F | F
  - F → (S) | a
- Note that
  - all recursion is leftmost
  - * has higher precedence than +
  - a + a + a + a * a is interpreted as
    - ((a+a) + a) + (a*a)

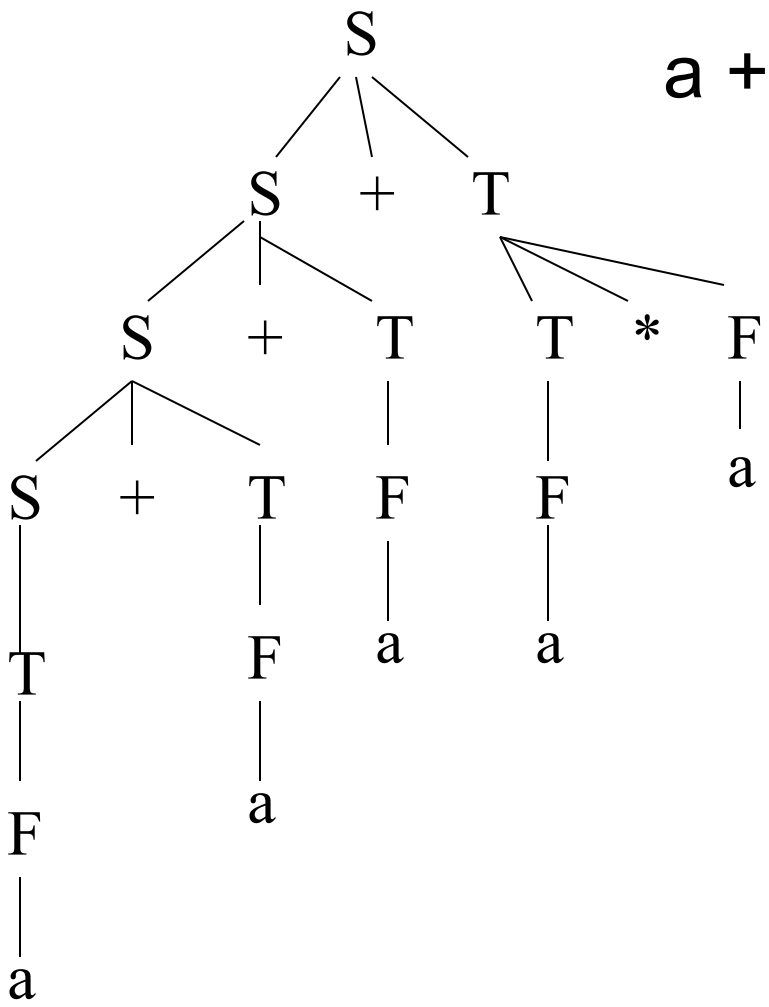
Example

a + a + a + a * a

Identify rightmost term

Identify next rightmost term

Identify next rightmost term



Identify rightmost factor

Removing Ambiguity

- Ambiguity will often come in two flavors:
  - Ambiguity over which rules to apply
    - Different choices of rules result in same derived string
  - Ambiguity over what order to apply a specific set of rules
  - General guide is to define the grammar (often through additional introduced variables) so that these ambiguities are no longer options
    - Controlling the choice of rules available along a given derivation path
    - Controlling the sequence of rules available along a given derivation path