Parse Trees

Definitions

Relationship to Left- and Rightmost Derivations

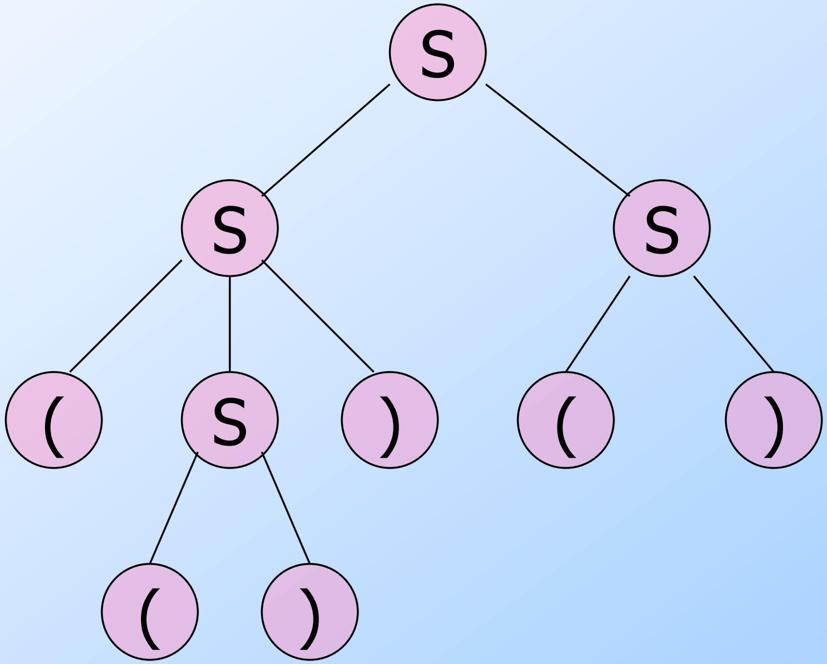
Ambiguity in Grammars

Parse Trees

- *Parse trees* are trees labeled by symbols of a particular CFG.
- Leaves: labeled by a terminal or ε.
- Interior nodes: labeled by a variable.
  - Children are labeled by the right side of a production for the parent.
- Root: must be labeled by the start symbol.

Example: Parse Tree

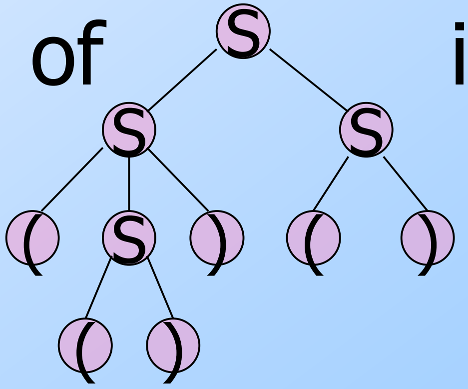
S -> SS | (S) | ()



Yield of a Parse Tree

- The concatenation of the labels of the leaves in left-to-right order
  - That is, in the order of a preorder traversal.

is called the *yield* of the parse tree.

- Example: yield of  is (())()

Parse Trees, Left- and Rightmost Derivations

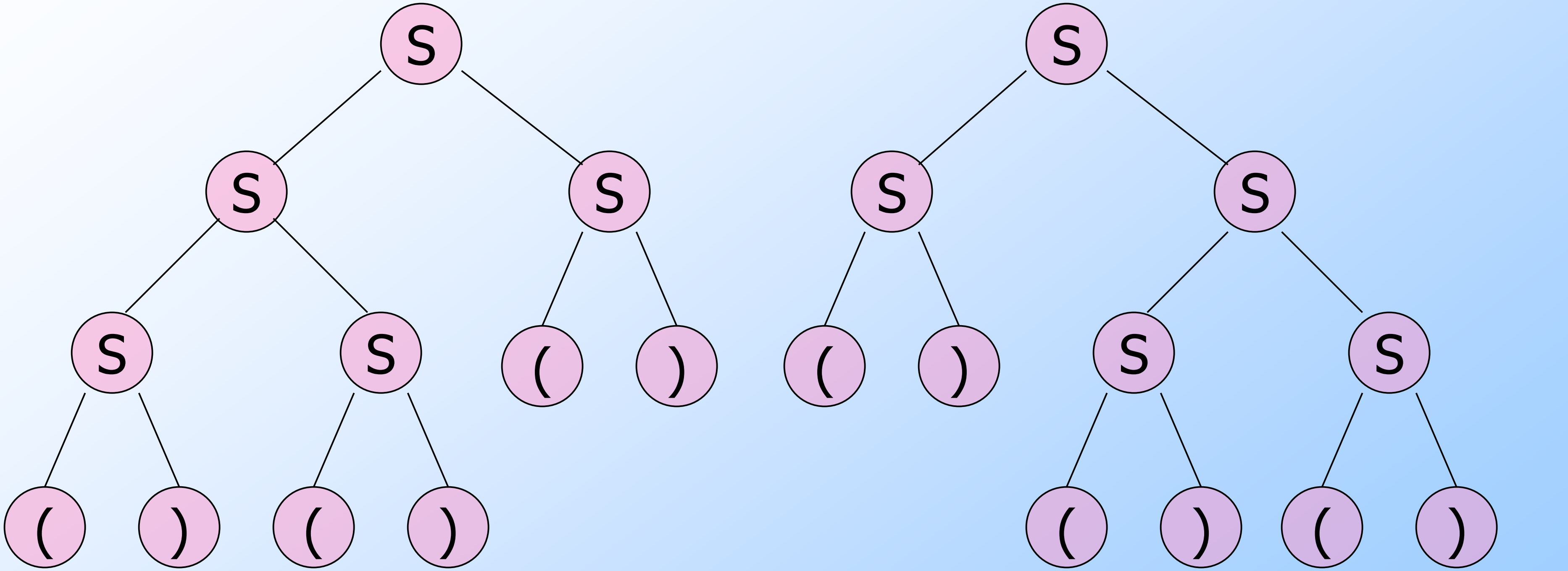
- For every parse tree, there is a unique leftmost, and a unique rightmost derivation.

1. If there is a parse tree with root labeled A and yield w, then $A \Rightarrow^*_{lm} w$.
2. If $A \Rightarrow^*_{lm} w$, then there is a parse tree with root A and yield w.

Ambiguous Grammars

- A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees.
- Example: S -> SS | (S) | ()
- Two parse trees for ()()() on next slide.

Example – Continued



Ambiguity, Left- and Rightmost Derivations

- If there are two different parse trees, they must produce two different leftmost derivations
- Conversely, two different leftmost derivations produce different parse trees
- Likewise for rightmost derivations.

Ambiguity, etc. – (2)

- Thus, equivalent definitions of "ambiguous grammar" are:
  1. There is a string in the language that has two different leftmost derivations.
  2. There is a string in the language that has two different rightmost derivations.

Ambiguity is a Property of Grammars, not Languages

- For the balanced-parentheses language, here is another CFG, which is unambiguous.

  B -> (RB | ε

  R ->) | (RR

B, the start symbol, derives balanced strings.

R generates strings that have one more right paren than left.

Example: Unambiguous Grammar

B -> (RB | ε R ->) | (RR

- Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.
  - If we need to expand B, then use B -> (RB if the next symbol is "(" and ε if at the end.
  - If we need to expand R, use R ->) if the next symbol is ")" and (RR if it is "(".

The Parsing Process

Remaining Input:

(())()

↑

Next symbol

Steps of leftmost derivation:

B

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

())()

Next symbol

Steps of leftmost derivation:

B

(RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

))()

↑ Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

B -> (RB | ϵ R ->) | (RR

The Parsing Process

Remaining Input:

)()

↑ Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

B -> (RB | ε   R ->) | (RR

The Parsing Process

Remaining Input:

()

↑

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

)

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

B -> (RB | ϵ   R ->) | (RR

The Parsing Process

Remaining Input:

Next symbol

Steps of leftmost derivation:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

(())()

B -> (RB | ϵ   R ->) | (RR

Inherent Ambiguity

- It would be nice if for every ambiguous grammar, there were some way to "fix" the ambiguity, as we did for the balanced-parentheses grammar.
- Unfortunately, certain CFL's are *inherently ambiguous*, meaning that every grammar for the language is ambiguous.

Example: Inherent Ambiguity

- The language $\{0^i1^j2^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous.
- Intuitively, at least some of the strings of the form $0^n1^n2^n$ must be generated by two different parse trees, one based on checking the 0's and 1's, the other based on checking the 1's and 2's.

One Possible Ambiguous Grammar

S -> AB | CD

A -> 0A1 | 01

B -> 2B | 2

C -> 0C | 0

D -> 1D2 | 12

A generates equal 0’s and 1’s

B generates any number of 2’s

C generates any number of 0’s

D generates equal 1’s and 2’s

And there are two derivations of every string with equal numbers of 0’s, 1’s, and 2’s. E.g.:

S => AB => 01B =>012

S => CD => 0D => 012