Parse Trees

Definitions

Relationship to Left- and Rightmost Derivations

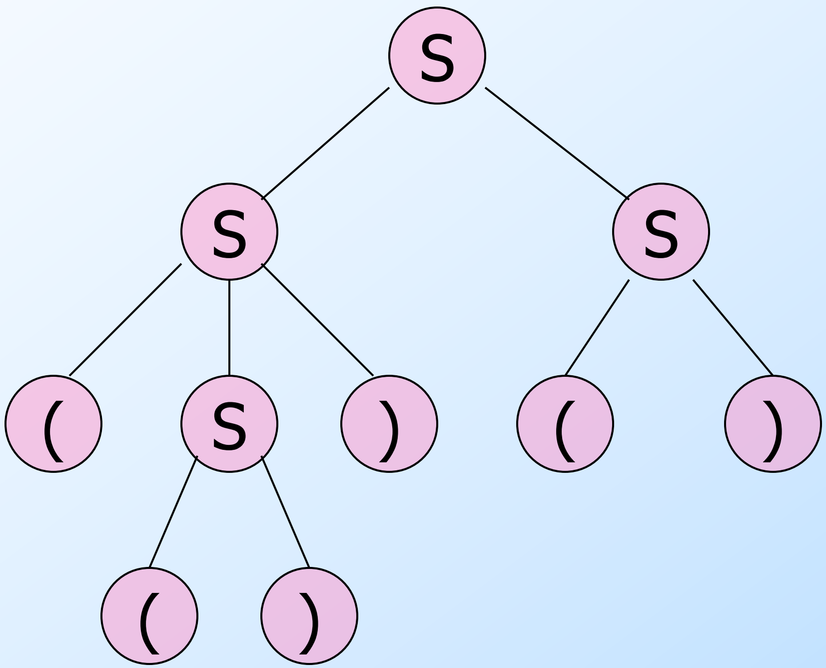
Ambiguity in Grammars

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 body of a production for the parent.
- Root: must be labeled by the start symbol.

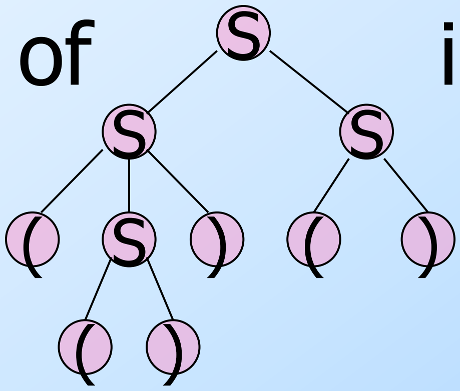
Example: Parse Tree

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 (())()

Generalization of Parse Trees

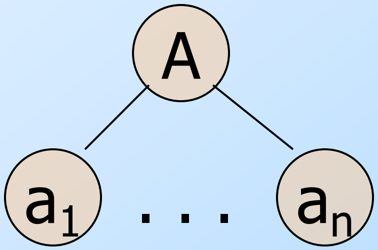
- We sometimes talk about trees that are not exactly parse trees, but only because the root is labeled by some variable A that is not the start symbol.
- Call these *parse trees with root A*.

Parse Trees, Leftmost and Rightmost Derivations

- Trees, leftmost, and rightmost derivations correspond.
- We'll prove:
  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.

Proof – Part 1

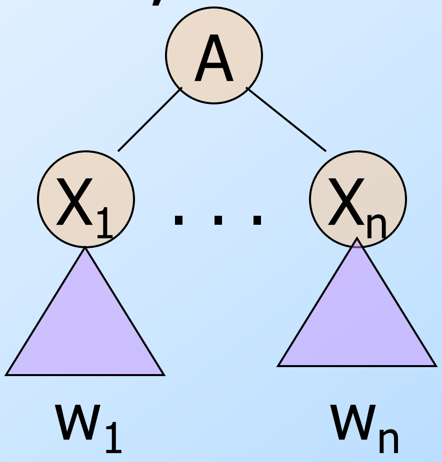
- Induction on the *height* (length of the longest path from the root) of the tree.
- Basis: height 1. Tree looks like



- $A \rightarrow a_1 \ldots a_n$ must be a production.
- Thus, $A \Rightarrow^*_{lm} a_1 \ldots a_n$.

Part 1 – Induction

- Assume (1) for trees of height < h, and let this tree have height h:



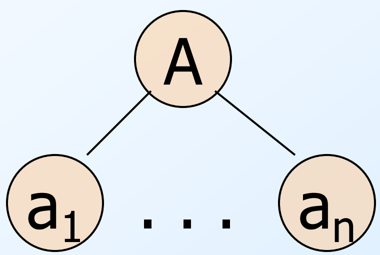
- By IH, $X_i \Rightarrow^*_{lm} w_i$.
  - Note: if X_i is a terminal, then $X_i = w_i$.
- Thus, $A \Rightarrow_{lm} X_1 \ldots X_n \Rightarrow^*_{lm} w_1X_2\ldots X_n \Rightarrow^*_{lm} w_1w_2X_3\ldots X_n \Rightarrow^*_{lm} \ldots \Rightarrow^*_{lm} w_1\ldots w_n$.

Proof: Part 2

- Given a leftmost derivation of a terminal string, we need to prove the existence of a parse tree.
- The proof is an induction on the length of the derivation.

Part 2 – Basis

- If $A \Rightarrow^*_{lm} a_1 \ldots a_n$ by a one-step derivation, then there must be a parse tree

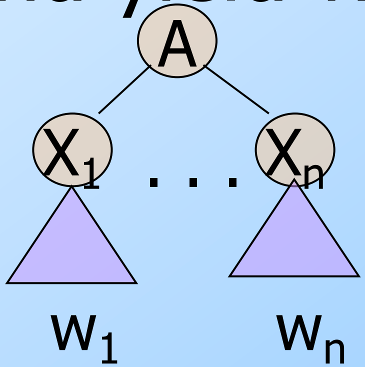


Part 2 – Induction

- Assume (2) for derivations of fewer than $k > 1$ steps, and let $A =>^*_{lm} w$ be a k-step derivation.
- First step is $A =>_{lm} X_1 \ldots X_n$.
- Key point: w can be divided so the first portion is derived from X_1, the next is derived from X_2, and so on.
  - If X_i is a terminal, then $w_i = X_i$.

Induction – (2)

- That is, $X_i =>*_{lm} w_i$ for all i such that X_i is a variable.
  - And the derivation takes fewer than k steps.
- By the IH, if X_i is a variable, then there is a parse tree with root X_i and yield w_i.
- Thus, there is a parse tree



Parse Trees and Rightmost Derivations

- The ideas are essentially the mirror image of the proof for leftmost derivations.
- Left to the imagination.

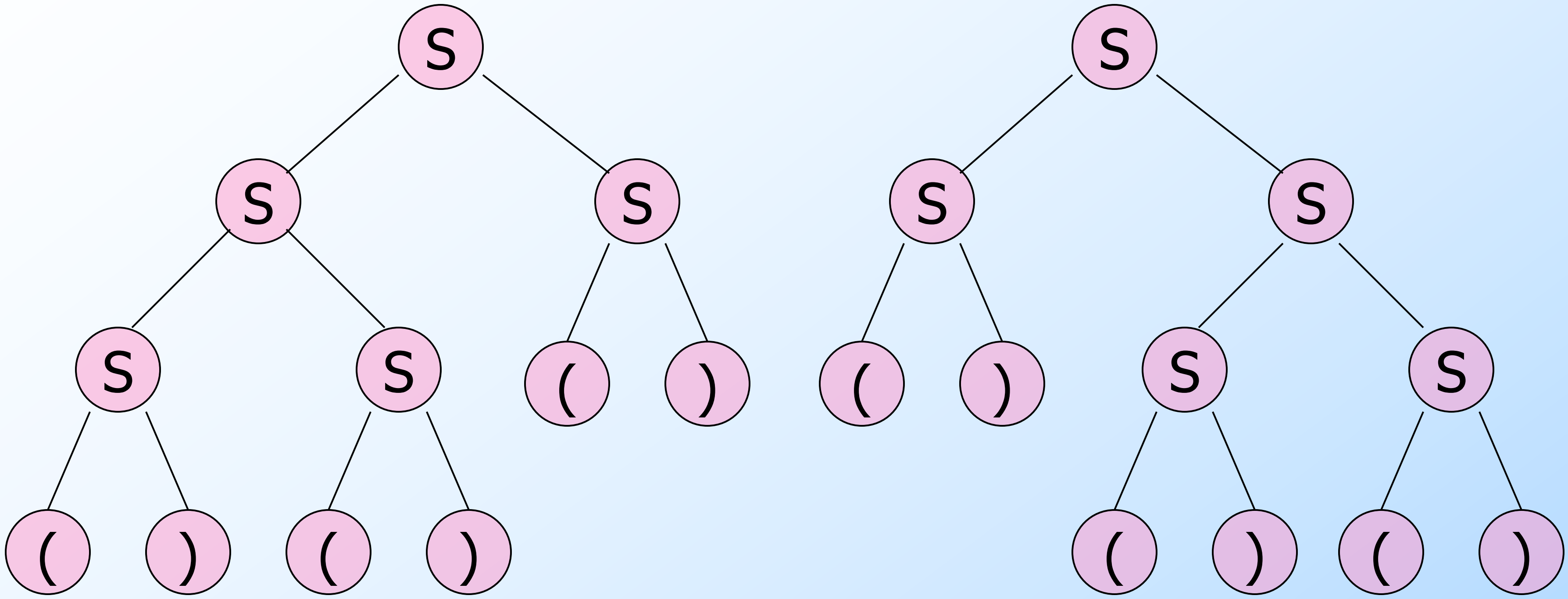
Parse Trees and Any Derivation

- The proof that you can obtain a parse tree from a leftmost derivation doesn't really depend on "leftmost."
- First step still has to be $A \Rightarrow X_1 \ldots X_n$.
- And w still can be divided so the first portion is derived from X_1, the next is derived from X_2, and so on.

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 by the construction given in the proof.
- Conversely, two different leftmost derivations produce different parse trees by the other part of the proof.
- 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 certain 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 "("; use 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

LL(1) Grammars

- As an aside, a grammar such B -> (RB | ϵ R ->) | (RR, where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol is called LL(1).
  - "Leftmost derivation, left-to-right scan, one symbol of lookahead."

LL(1) Grammars – (2)

- Most programming languages have LL(1) grammars.
- LL(1) grammars are never ambiguous.

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 — A generates equal 0's and 1's

B -> 2B | 2 — B generates any number of 2's

C -> 0C | 0 — C generates any number of 0's

D -> 1D2 | 12 — 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