Context-free Grammars and Languages

COMP 455 – 002, Spring 2019

Context-free Grammars

Context-free grammars provide another way to specify languages.

Example: A context-free grammar for mathematical expressions:

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow \mathbf{i}$$

Show that a string is in the language using a *derivation*:

$$E \Rightarrow E + E$$
$$\Rightarrow (E) + E$$
$$\Rightarrow (E) + E * E$$
$$\Rightarrow (\mathbf{i}) + E * E$$
$$\Rightarrow (\mathbf{i}) + \mathbf{i} * E$$
$$\Rightarrow (\mathbf{i}) + \mathbf{i} * \mathbf{i}$$

Formal Definition of CFGs

- A context-free grammar (CFG) is denoted using a 4-tuple $G = (V, T, P, S)$, where:
  - V is a finite set of *variables*
  - T is a finite set of *terminals*
  - P is a finite set of productions of the form $\boxed{variable} \rightarrow \boxed{string}$
  - S is the *start symbol*. (S is a variable in V)

"body"

"head"

Formal CFG Definition: Example

To define our example grammar using this tuple notation:

- $V = \{E\}$
- $T = \{+, *, (,), \mathbf{i}\}$
- P is the set of rules defined previously:
- $S = E$

$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$E \to \mathbf{i}$$

More CFG Examples

In our discussion of the Pumping Lemma for Regular Languages, we discussed the following language:

$$L = \{x \mid (x = x^R) \wedge x \in (\mathbf{0} + \mathbf{1})^*\}$$

Can we show this language is context-free?

Yes:

$V = \{R\}$

$T = \{0, 1\}$

$S = R$

$$P = \{ \\ R \to 0R0, \\ R \to 1R1, \\ R \to 0, \\ R \to 1, \\ R \to \varepsilon, \\ \}$$

More CFG Examples

What about the language L consisting of all strings containing an equal number of 0s and 1s?

- $V = \{R\}$
- $T = \{0, 1\}$
- $S = R$
- $P =$
  - $R \rightarrow 0R1R$
  - $R \rightarrow 1R0R$
  - $R \rightarrow \varepsilon$

A Historical Note

We are talking about context-*free* languages, but what about a language that is not context-free?

- These languages exist and are called *context-sensitive*.
  - Context-sensitive languages allow production rules with strings, e.g. $1S0 \rightarrow 110$.
- Context-sensitive languages were used in the study of natural languages, but ended up with few practical applications.

Derivations

- We will be following the notational conventions from page 178 of the textbook (Section 5.1.4)
- We say that string α_1 *directly derives* α_2 if and only if:
  - $\alpha_1 = \alpha A \gamma$,
  - $\alpha_2 = \alpha \beta \gamma$, and
  - $A \rightarrow \beta$ is a production rule in P.
- This can be denoted $\alpha A \gamma \underset{G}{\Rightarrow} \alpha \beta \gamma$

Derivations

- We will be following the notational conventions from page 178 of the textbook (Section 5.1.4)
- We say that string α_1 *directly derives* α_2 if and only if:
  - $\boxed{\alpha}_1 = \alpha A \gamma$,
  - $\alpha_2 = \alpha\beta\gamma$, and
  - $\boxed{A} \to \beta$ is a production rule in P.
- This can be denoted $\alpha A \gamma \boxed{\underset{G}{\Rightarrow}} \alpha\beta\gamma$

Lowercase Greek letters: strings (including variables and terminals)

Uppercase letters near the start of the alphabet: variables

A derivation using a single invocation of a production rule in the grammar G. (We can omit the G if the grammar we're talking about is obvious.)

Derivations (continued)

- $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$ means α_1 *derives* α_m (in 0 or more steps).
  - i.e., $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$
- $\alpha \overset{i}{\Rightarrow} \beta$ means α derives β in *exactly i* steps.
- α is a *sentential form* if and only if $S \overset{*}{\Rightarrow} \alpha$.

Leftmost and Rightmost Derivations

- It can be useful to restrict a derivation to only replace the leftmost variables in a string. This is called a *leftmost derivation*.
  - Steps in a leftmost derivation are indicated using $\underset{lm}{\Rightarrow}$ for a single step or $\overset{*}{\underset{lm}{\Rightarrow}}$ for many steps.
- A string encountered during a leftmost derivation is called a *left sentential form*.
  - i.e., α is a left-sentential form if and only if $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$.

Leftmost and Rightmost Derivations

- Similarly to a leftmost derivation, a *rightmost derivation* only replaces the rightmost variable in each step.
  - Steps in a rightmost derivation are indicated using $\underset{rm}{\Rightarrow}$ or $\underset{rm}{\overset{*}{\Rightarrow}}$.
- A *right-sentential form* is a string encountered during a rightmost derivation from the start symbol.

Leftmost and Rightmost Derivations

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow \mathbf{i}$

Example using the grammar from before:

First example	Leftmost	Rightmost
$E \Rightarrow E + E$	$E \Rightarrow E + E$	$E \Rightarrow E + E$
$\Rightarrow (E) + E$	$\Rightarrow (E) + E$	$\Rightarrow E + E * E$
$\Rightarrow (E) + E * E$	$\Rightarrow (\mathbf{i}) + E$	$\Rightarrow E + E * \mathbf{i}$
$\Rightarrow (\mathbf{i}) + E * E$	$\Rightarrow (\mathbf{i}) + E * E$	$\Rightarrow E + \mathbf{i} * \mathbf{i}$
$\Rightarrow (\mathbf{i}) + \mathbf{i} * E$	$\Rightarrow (\mathbf{i}) + \mathbf{i} * E$	$\Rightarrow (E) + \mathbf{i} * \mathbf{i}$
$\Rightarrow (\mathbf{i}) + \mathbf{i} * \mathbf{i}$	$\Rightarrow (\mathbf{i}) + \mathbf{i} * \mathbf{i}$	$\Rightarrow (\mathbf{i}) + \mathbf{i} * \mathbf{i}$

The Language of a CFG

- For a CFG G, $L(G) \equiv \left\{ w \mid \boxed{w \in T^*} \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$

w consists only of terminal symbols

- L is a *context-free language* if and only if $L = L(G)$ for some CFG G.
- Grammars G_1 and G_2 are *equivalent* if and only if $L(G_1) = L(G_2)$.

Showing Membership in a CFG

Demonstrating that a string is in the language of a CFG can be accomplished two ways:

- **Top-down**: Give a derivation of the string. *i.e.*, Begin with the start symbol and use production rules to create the string.
- **Bottom-up**: Start with the string, and try to apply production rules “backwards” to end up with a single start symbol.
- We will now consider a technique called *recursive inference*, which is basically a bottom-up approach.

Recursive Inference

- Define a language $L(X)$ for each variable X. $L(X)$ contains *all strings that can be derived from* X.
  - If $V \rightarrow X_1X_2 \ldots X_n$ is a production rule, then all strings $x_1x_2 \ldots x_n$ are in $L(V)$, where:
    - If X_i is a terminal symbol, then $x_i = X_i$,
    - If X_i is a variable, then x_i is in $L(X_i)$.
- Productions with only terminal symbols in the body give us the *base case*. (So, we basically end up applying productions backwards.)
- A string x is in $L(G)$ if and only if it is in $\boxed{L(S)}$.

Strings that can be derived from the start symbol S.

Recursive Inference

- The goal of recursive inference is to look at successively larger substrings of some string x to determine if x is in $L(S)$.

Recursive Inference: Example

(This example is from Figure 5.3 in the book.)

We want to use recursive inference to show that $a * (a + b00)$ is in $L(G)$.

i. $a \in L(I)$, by Production rule 5
ii. $b \in L(I)$, by Production rule 6
iii. $b0 \in L(I)$, by Production rule 9 and *ii*
iv. $b00 \in L(I)$, by Production rule 9 and *iii*
v. $a \in L(E)$, by Production rule 1 and *i*
vi. $b00 \in L(E)$, by Production rule 1 and *iv*
vii. $a + b00 \in L(E)$, by Production rule 2 and *v* and *vi*
viii. $(a + b00) \in L(E)$, by Production rule 4 and *vii*
ix. $a * (a + b00) \in L(E)$, by Production rule 3 and *v* and *viii*.

Grammar G for simple expressions:

- $V = \{E, I\}$
- $T = \{a, b, 0, a, +, *, (,)\}$
- E is the start symbol

Production rules:

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Parse Trees

- Parse trees show how symbols of a string are grouped into substrings, and the variables and productions used.
- In *general*, the root is S, internal nodes are variables, and leaves are variables or terminals.

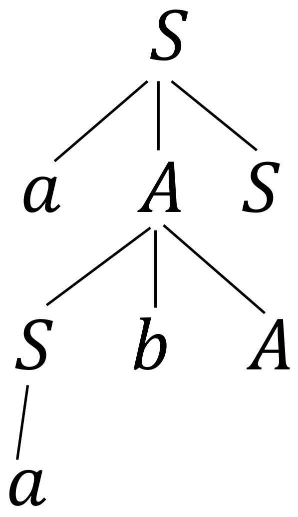
If $\begin{array}{c} A \\ / \quad \backslash \\ X_1 \cdots X_n \end{array}$, then $A \rightarrow X_1 \dots X_n$

Parse Tree Example

Example grammar:

Note: new notation

- $S \to \boxed{aAS \mid a}$
- $A \to \boxed{SbA \mid SS \mid ba}$



An example parse tree

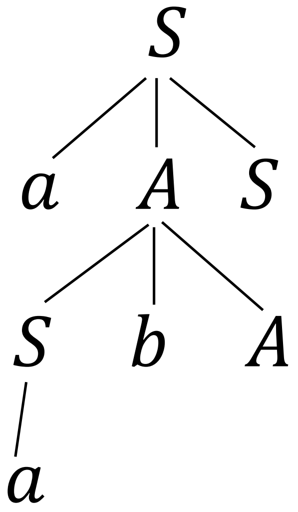
A Parse Tree's "Yield"

Example grammar, again: $S \rightarrow aAS \mid a,\ A \rightarrow SbA \mid SS \mid ba.$

- The *yield* of a parse tree is the string obtained from reading its leaves left-to-right.

The yield of this tree is $aabAS$.

Note that the yield of a parse tree is a sentential form.



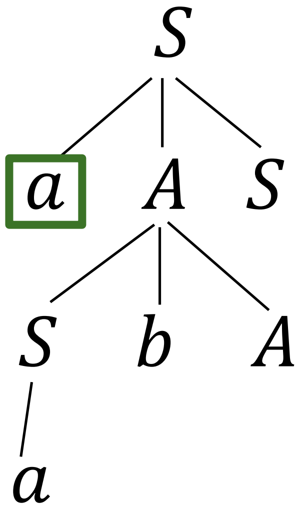
A Parse Tree's "Yield"

Example grammar, again: $S \to aAS \mid a,\ A \to SbA \mid SS \mid ba.$

- The *yield* of a parse tree is the string obtained from reading its leaves left-to-right.

The yield of this tree is $aabAS$.

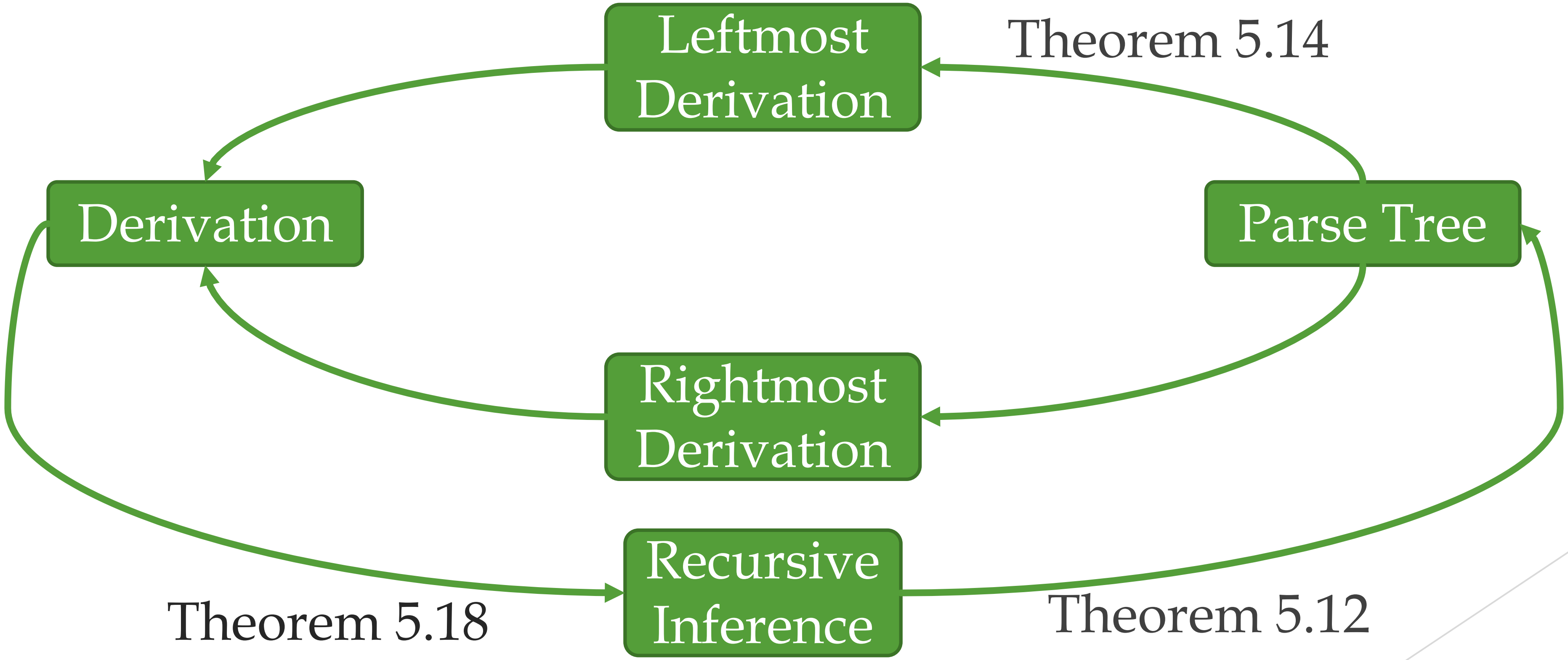
Note that the yield of a parse tree is a sentential form.



$aabAS$

Inference, Derivation, and Parse Trees

We will show that all of these are equivalent ways for showing that a string is in a CFL. Specifically, we show:

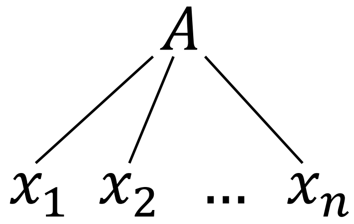


From Recursive Inference to Parse Tree

Theorem 5.12: Let $G = (V, T, P, S)$ be a CFG. If recursive inference tells us that string $w \in T^*$ is in the language of variable $A \in V$, then a parse tree exists with root A and yield w.

We will prove this by induction on the number of steps in the recursive inference.

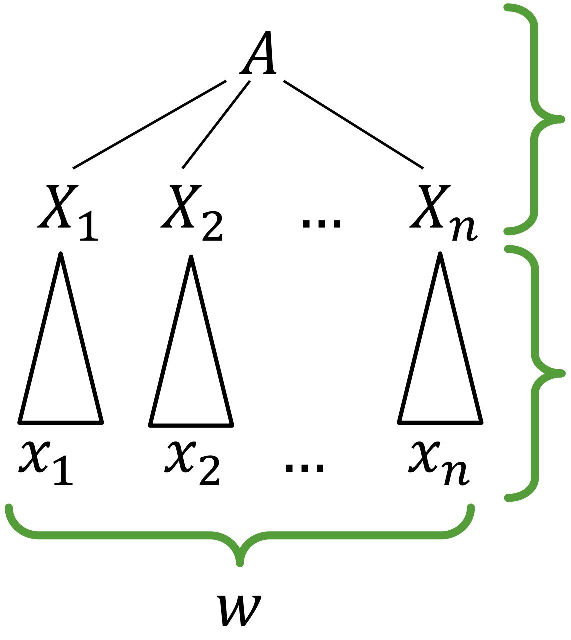
Base case: One step. This means that there is a production rule $A \to w$. The tree for this is:



Where $w = x_1 x_2 \dots x_n$.

From Recursive Inference to Parse Tree

Inductive step: Assume that the last inference step looked at the production $A \rightarrow X_1 X_2 \ldots X_n$, and previous inference steps verified that $x_i \in L(X_i)$, for each x_i in $w = x_1 x_2 \ldots x_n$. The tree for this is:



The tree from A to $X_1 X_2 \ldots X_n$.

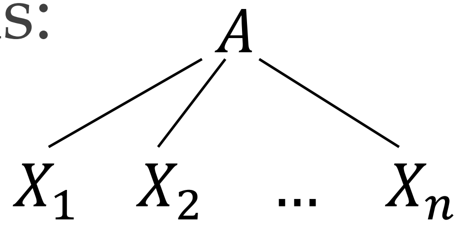
The inductive hypothesis lets us assume we already have trees yielding the terminal strings.

From Parse Tree to Derivation

Theorem 5.14: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with a root of variable A with yield $w \in T^*$. Then there is a leftmost derivation $A \underset{lm}{\overset{*}{\Rightarrow}} w$ in G.

We will prove this by induction on tree height.

Base case: The tree's height is one. The tree looks like this:



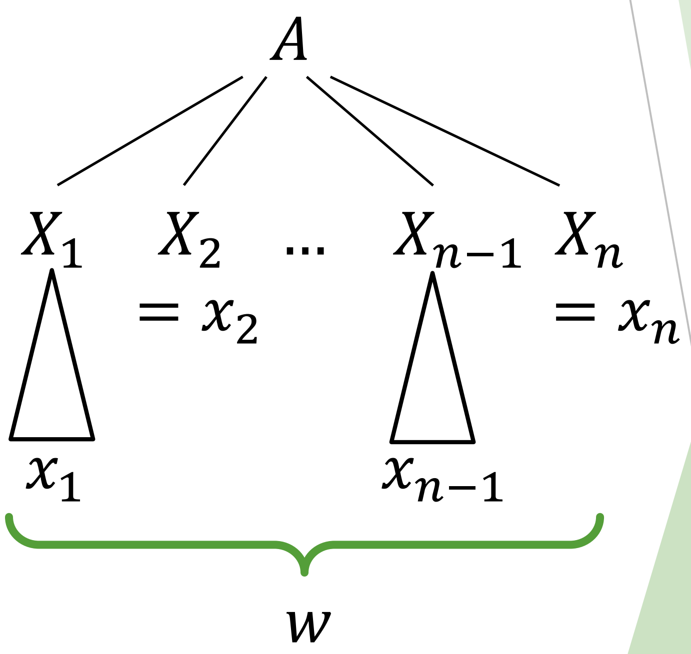
So, there must be a production $A \rightarrow X_1X_2 \ldots X_n$ in G, where $w = X_1X_2 \ldots X_n$.

From Parse Tree to Derivation

Inductive step:

The tree's height exceeds 1, so the tree looks like this:

Note that A may produce some terminal strings (like x_2 and x_n) and other strings containing variables (like X_1 and X_{n-1}).

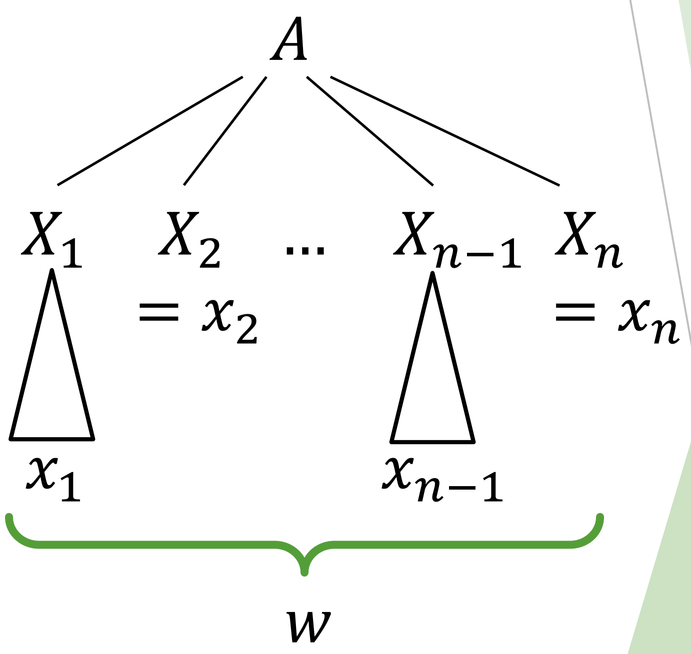


From Parse Tree to Derivation

Inductive step:

The tree's height exceeds 1, so the tree looks like this:

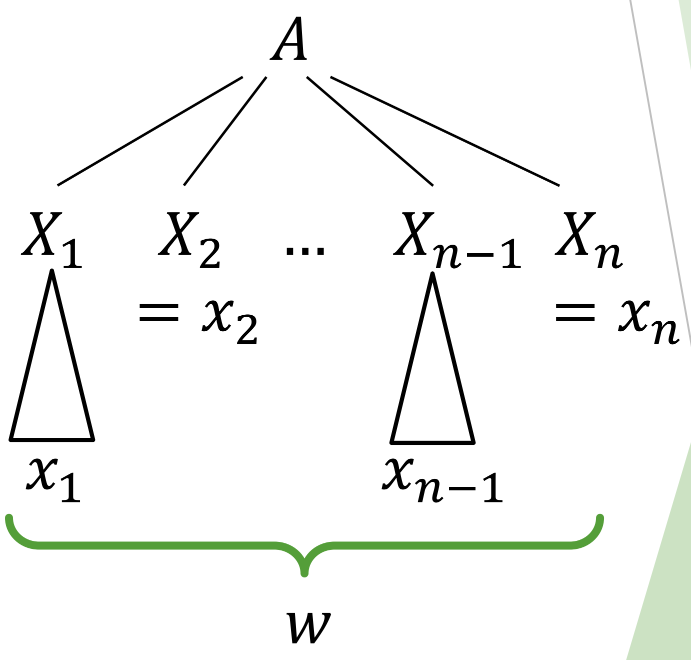
Note that A may produce some terminal strings (like x_2 and x_n) and other strings containing variables (like X_1 and X_{n-1}).



From Parse Tree to Derivation

Inductive step (continued):

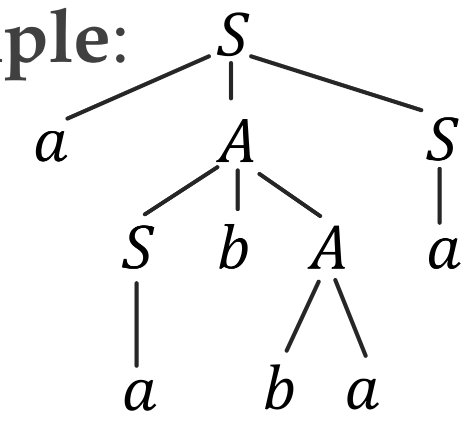
- By the inductive hypothesis, $X_1 \underset{lm}{\overset{*}{\Rightarrow}} x_1$, $X_{n-1} \underset{lm}{\overset{*}{\Rightarrow}} x_{n-1}$, etc.
- Trivially, $X_2 \underset{lm}{\overset{*}{\Rightarrow}} x_2$, $X_n \underset{lm}{\overset{*}{\Rightarrow}} x_n$, etc., because they are terminals only.
- Since $A \Rightarrow X_1 X_2 \dots X_{n-1} X_n$, and $w = x_1 x_2 \dots x_{n-1} x_n$, we know that $A \underset{lm}{\overset{*}{\Rightarrow}} w$.



Notes on Derivations From Parse Trees

- The leftmost derivation corresponding to a parse tree will be unique.
- We can prove the same conversion is possible for rightmost derivations.
  - Such a rightmost derivation will also be unique.

Example:



Leftmost derivation: $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa.$

Rightmost derivation: $S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa.$

From Derivation to Recursive Inference

Theorem 5.18: Let $G = (V, T, P, S)$ be a CFG, $w \in T^*$, and $A \in V$. If a derivation $A \overset{*}{\Rightarrow} w$ exists in grammar G, then $w \in L(A)$ can be inferred via recursive inference.

We will prove this by induction on the length of the derivation.

Base case: The derivation is one step. This means that $A \rightarrow w$ is a production, so clearly $w \in L(A)$ can be inferred.

From Derivation to Recursive Inference

Inductive step: There is more than one step in the derivation. We can write the derivation as

$$A \Rightarrow X_1 X_2 \dots X_n \overset{*}{\Rightarrow} x_1 x_2 \dots x_n = w$$

By the inductive hypothesis, we can infer that $x_i \in L(X_i)$ for every i. Next, since $A \rightarrow X_1 X_2 \dots X_n$ is clearly a production, we can infer that $w \in L(A)$.

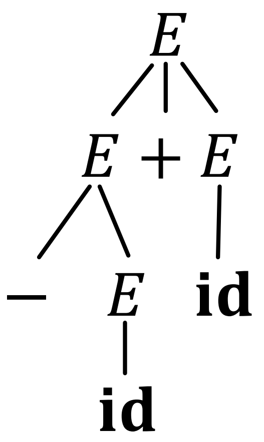
Ambiguity

- A grammar is *ambiguous* if some word in it has multiple parse trees.
  - Recall: This is equivalent to saying that some word has more than one leftmost or rightmost derivation.
- Ambiguity is important to know about, because parsers (i.e. for a programming language compiler) need to determine a program's structure from source code. This is complicated if multiple parse trees are possible.

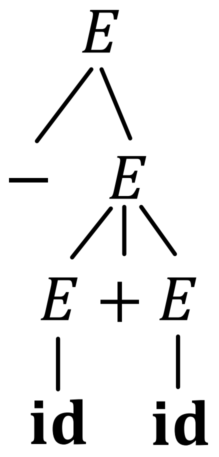
Ambiguous Grammar: Example

- $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow -E + E \\
&\Rightarrow -\mathbf{id} + E \\
&\Rightarrow -\mathbf{id} + \mathbf{id}
\end{aligned}
$$



$$
\begin{aligned}
E &\Rightarrow -E \\
&\Rightarrow -E + E \\
&\Rightarrow -\mathbf{id} + E \\
&\Rightarrow -\mathbf{id} + \mathbf{id}
\end{aligned}
$$



Resolving Ambiguity

- $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$
- Ambiguity in this grammar is caused by the lack of *operator precedence*.
- This can be resolved by introducing more variables.
  - For example, $E \rightarrow E + E \mid -E \mid \mathbf{id}$, the part of our grammar causing the ambiguity, can be made unambiguous by adding a variable F:
    $E \rightarrow F + F,\ F \rightarrow -E \mid \mathbf{id}$.
- Section 5.4 of the book discusses this in more depth.

Inherent Ambiguity

- A context-free language for which all possible CFGs are ambiguous is called *inherently ambiguous*.
- One example (from the book) is: $L = \{a^n b^n c^m d^m \mid m, n \geq 1\} \cup \{a^n b^m c^m d^n \mid m, n \geq 1\}$.
- Proving that languages are inherently ambiguous can be quite difficult.
- These languages are encountered quite rarely, so this has little practical impact.