# CMSC330: Context Free Grammars

Chris Kauffman

*Last Updated:*
*Mon Oct 9 11:12:55 PM EDT 2023*

# Logistics

## Assignments

- No online lecture quiz this week due to Exam 1
- Project 4 is up, OCaml basics, due Sun 15-Oct

## Reading

Chapter 5 "Context Free Grammars" from Automata Theory, Languages, and Computation by Hopcroft, Motwani, Pullman

## Goals

Context Free Grammars, notation, terminology, usage, limits

# Limits of Regexs

- Recall that **Regular Languages** (recognized by Regexs/FSMs) had a limit to their power to recognize/accept
- Could not derive a Regex for the following two languages:
  1. Equal-ABs $= \{a^n b^n | n > 0\}$
     Examples of: Equal-ABs $=$ `{ab, aabb, aaabbb, aaaabbbb, ...}`
  2. Balanced-Paren $= \{(^n)^n | n > 0\}$
     Examples of Balanced-Paren $=$ `{(), (()), ((())), ...}`
- Clearly the latter has applications in processing programming languages
- Rather than a Regular language, these are examples of a **Context Free Languages**

# Context Free Grammars (CFGs)

- **Non-terminal** symbols (capitals) which will be replaced by other symbols
- **Terminal** symbols (lowercase) are letters of the alphabet
- **Production Rules**: a single Non-terminal on a left-hand side produces right-hand side combination of Terminals/Non-terminals
- Usual convention is the first Non-terminal with a Production Rule listed is the **Start Symbol**

## Example: CFG for Equal-ABs

$$X \rightarrow aXb$$
$$X \rightarrow \epsilon$$
$$or$$
$$X \rightarrow aXb|\epsilon$$

- Terminals: Only $a, b$ (not counting empty string $\epsilon$
- Non-Terminals: $X$
- Productions: 2, both for $X$
- Start Symbol: $X$

# Deriving/Producing Strings from a CFG

## Example: CFG for Equal-ABs

$$X \rightarrow aXb \quad (1)$$
$$X \rightarrow \epsilon \quad (2)$$

Useful to number each of the production rules so they can be easily referenced during derivations

## Derive: $aabb$

$X \Rightarrow_1 aXb$ Use production 1
$\Rightarrow_1 aaXbb$ Use production 1
$\Rightarrow_2 aabb$ Use production 2

## Derive: Others

- $aaabbb$:
  $X \Rightarrow_1 aXb \Rightarrow_1 aaXbb \Rightarrow_1 aaaXbbb \Rightarrow_2 aaabbb$
- $ab$: $X \Rightarrow_1 aXb \Rightarrow_2 ab$

**Notation**: In lecture examples will try to use

- $\rightarrow$ (single arrow) for CFG production rules
- $\Rightarrow$ (double arrow) for derivations

# A more Interesting CFG

## CFG for PlusMinus

$$
\begin{aligned}
A &\to A + A && (1)\\
A &\to A * A && (2)\\
A &\to N && (3)\\
N &\to DN && (4)\\
N &\to D && (5)\\
D &\to 0|1|2|3|..|9 && (6)
\end{aligned}
$$

- Terminals: $+, *$ and numbers like $5, 124$
- Non-Terminals: $A, N, D$
- Productions: 3 for $A$, 2 for $N$, "1" for $D$, 5 total

## Sample Derivation

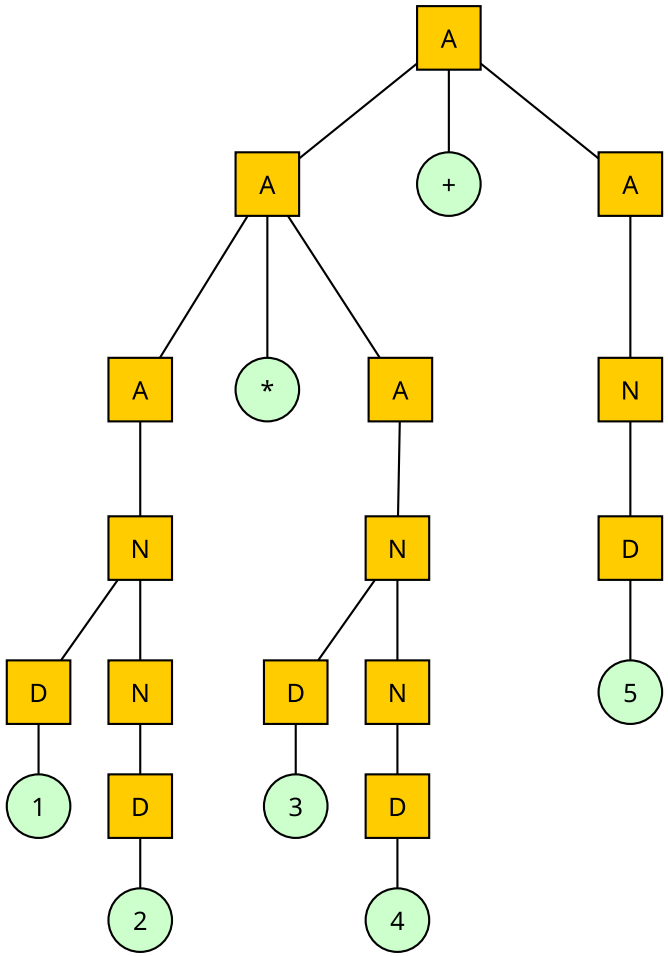
$$
\begin{aligned}
\underline{A} &\Rightarrow_1 \underline{A} + A\\
&\Rightarrow_2 \underline{A} * A + A\\
&\Rightarrow_3 N * \underline{A} + A\\
&\Rightarrow_3 N * N + \underline{A}\\
&\Rightarrow_3 \underline{N} * N + N\\
&\Rightarrow_4 D\underline{N} * N + N\\
&\Rightarrow_5 DD * \underline{N} + N\\
&\Rightarrow_4 DD * D\underline{N} + N\\
&\Rightarrow_5 DD * DD + \underline{N}\\
&\Rightarrow_5 \underline{DD} * \underline{DD} + \underline{D}\\
&\Rightarrow_6 12 * 34 + 5
\end{aligned}
$$

The Non-terminal (variable) to which production rules are applied are underlined

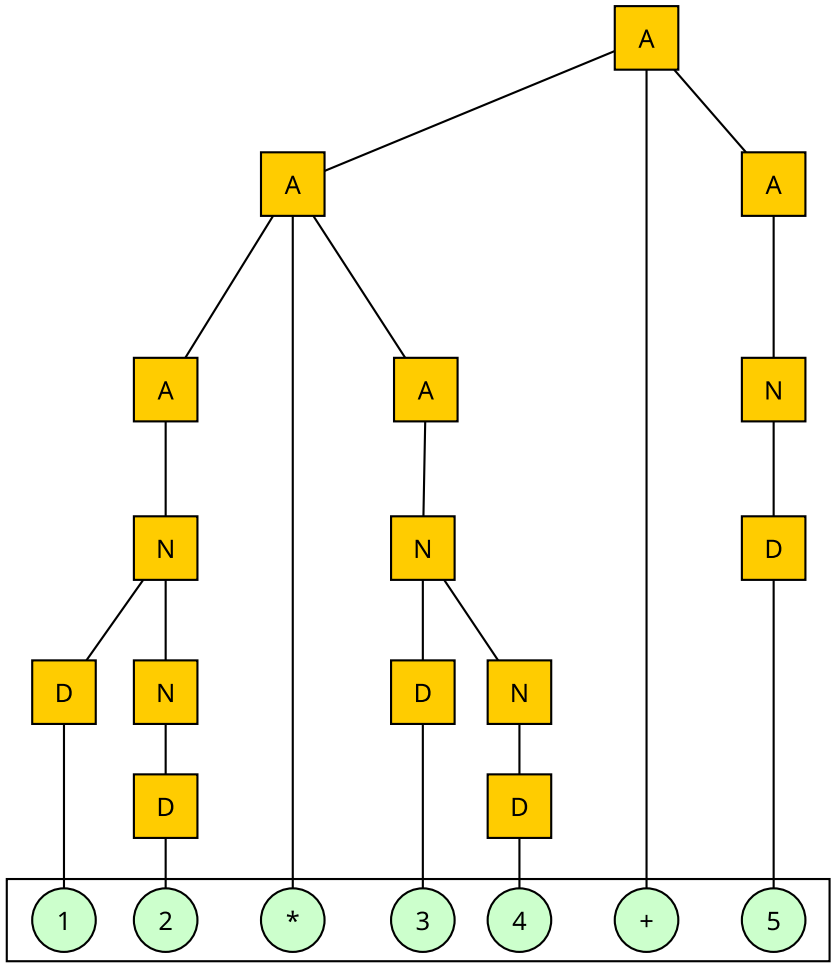
# CFG Derivations and Parse Trees

**Parse Trees** are a graphical representation of a string derived/produced from a CFG, examples below

**TYPICAL PARSE TREE with terminals at differing heights**




**TERMINALS LEVEL to show the Derived Expression**




Yield of the tree

# Leftmost and Rightmost Derivations

- Notice **choices** in earlier derivation: pick a Non-terminal and apply a production rule
- Often want to eliminate choices so enforce an ordering to derivations
- **Leftmost / Left-hand** derivation always applies a production to the leftmost non-terminal
- **Rightmost / Right-hand** derivation does likewise for rightmost non-terminal

## Leftmost Derivation

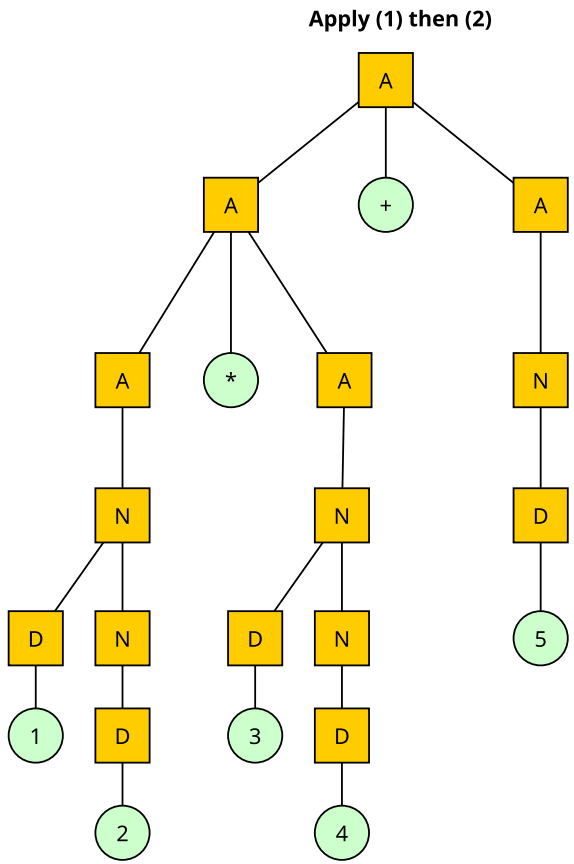
$$
\begin{aligned}
\underline{A} &\Rightarrow_1 \underline{A} + A \\
&\Rightarrow_2 \underline{A} * A + A \\
&\Rightarrow_3 \underline{N} * A + A \\
&\Rightarrow_4 \underline{D}N * A + A \\
&\Rightarrow_6 1\underline{N} * A + A \\
&\Rightarrow_5 1\underline{D} * A + A \\
&\Rightarrow_6 12 * \underline{A} + A \\
&\Rightarrow_3 12 * \underline{N} + A \\
&\Rightarrow_4 12 * \underline{D}N + A \\
&\Rightarrow_6 12 * 3\underline{N} + A \\
&\Rightarrow_5 12 * 3\underline{D} + A \\
&\Rightarrow_6 12 * 34 + \underline{A} \\
&\Rightarrow_4 12 * 34 + \underline{N} \\
&\Rightarrow_5 12 * 34 + \underline{D} \\
&\Rightarrow_6 12 * 34 + 5
\end{aligned}
$$

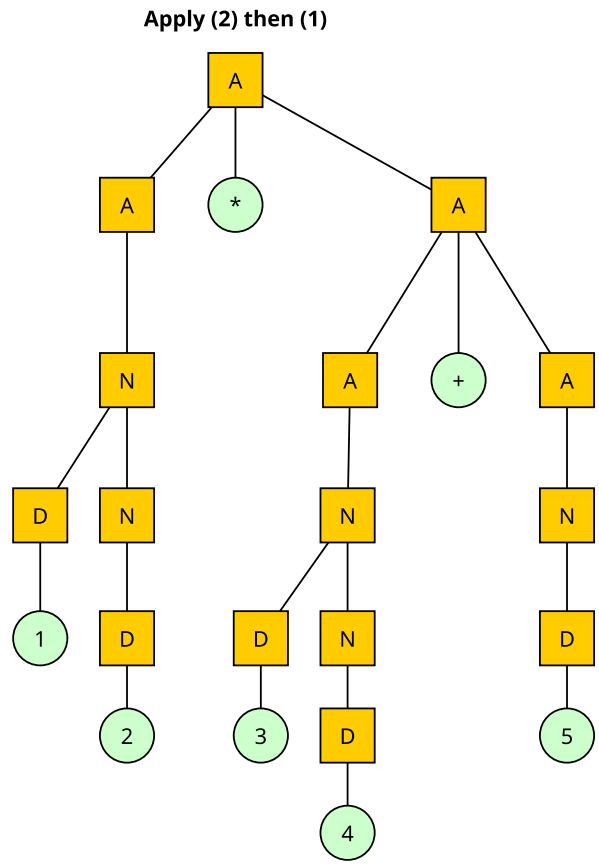
# Exercise: Ambiguity in CFGs

This grammar has another choice: when deriving $12 * 34 + 5$, which order to apply

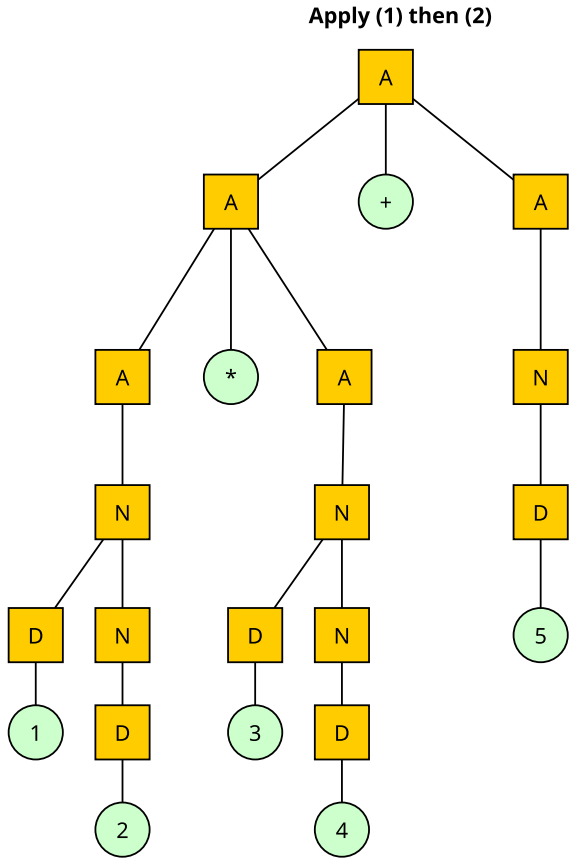
- (1) $A \to A + A$ then (2) $A \to A * A$ OR
- (2) $A \to A * A$ then (1) $A \to A + A$

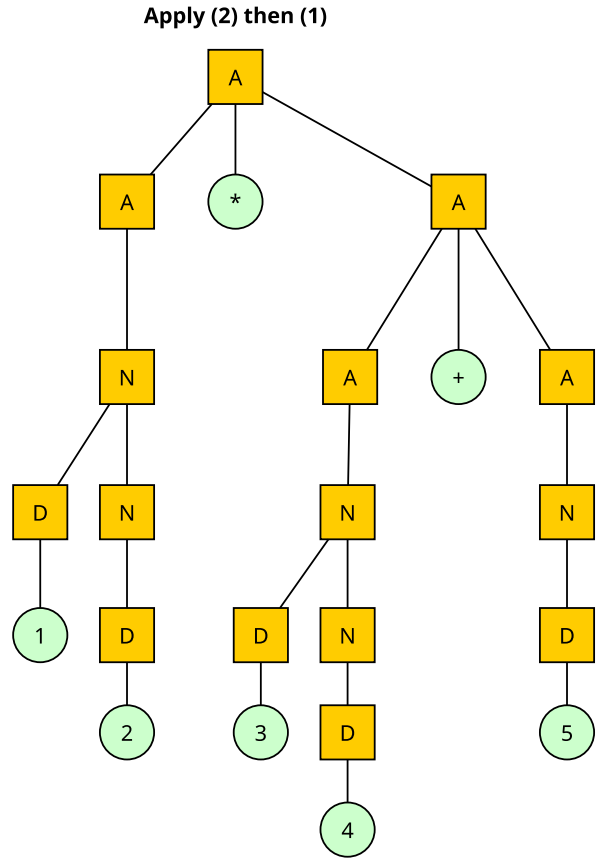
What is the difference shown in the two Parse Trees?



# **Answers**: Ambiguity in CFGs




- $(12 * 34) + 5$ for (1) then (2)
- $12 * (34 + 5)$ for (2) then (1)

Low-precedence operator is higher in the tree, typically want higher-precedence for $*$ over $+$, resolve this momentarily via an adjustment to the grammar

# Exercise: Proving a Grammar is Ambiguous

To prove a Grammar is ambiguous

- Using only leftmost derivations. . .
- Derive the same string via two different choices of productions
- Creates two different parse trees for the same string

Show the below grammar is ambiguous

$$X \to aX \qquad (1)$$
$$X \to Xb \qquad (2)$$
$$X \to Y \qquad (3)$$
$$Y \to b \qquad (4)$$
$$Y \to \epsilon \qquad (5)$$

# **Answers**: Proving a Grammar is Ambiguous

To prove a Grammar is ambiguous

- Using only leftmost derivations. . .
- Derive the same string via two different choices of productions
- Creates two different parse trees for the same string

Show the below grammar is ambiguous

$$X \to aX \quad (1)$$
$$X \to Xb \quad (2)$$
$$X \to Y \quad (3)$$
$$Y \to b \quad (4)$$
$$Y \to \epsilon \quad (5)$$

$ab$ has several different leftmost derivations; two are

$X \Rightarrow_1 aX \Rightarrow_2 aXb \Rightarrow_3 aYb \Rightarrow_5 ab$

AND

$X \Rightarrow_1 aX \Rightarrow_3 aY \Rightarrow_4 ab$

Drawing Parse Trees for these is a good idea

# ========== END PART 1 ==========

# Resolving Ambiguities in CFGs

# Operator Precedence in CFGs

# Chomsky Normal Form

# Parse Trees and their Yield

# Parse Trees vs Abstract Syntax Trees (ASTs)

# Expressing Precedence in Grammars

# Machine Implementation of CFGs

# Recursive Descent Parsers

# Pushdown Automata

# Beyond CFGs: The Chomsky Hierarchy