Context-Free Grammars and Context-Free Languages

- Context-Free Grammars
  - Derivations
- Parse Trees
- Ambiguity

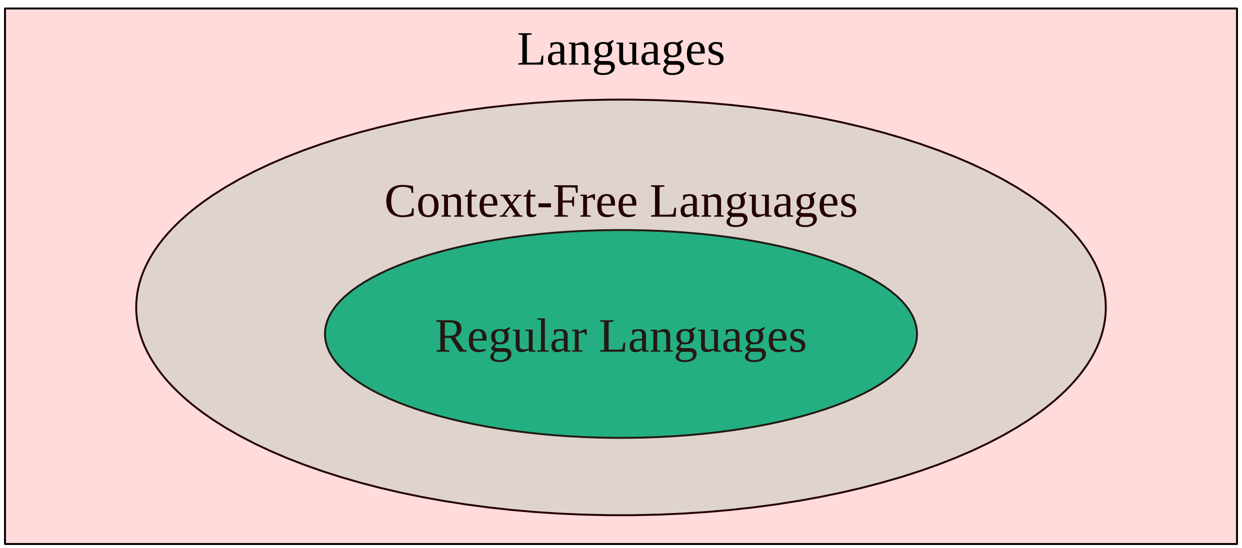
Context-Free Grammars

Context-Free Grammars

- We have seen that many languages cannot be regular.
  - We need to consider larger classes of languages.
  - **Context-Free Languages (CFLs)** is a larger class of languages (larger than regular languages).
  - Every regular language is a context-free language.
- Regular expressions are used to define regular languages.
- **Context-Free Grammars (CFGs)** are used to define Context-Free Languages (CFLs).
- **Pushdown Automatons** recognize CFLs.
- Context-Free Grammars (CFGs) played a central role in natural language processing, and compilers.
  - The syntax of a programming language is described by a **CFG**.
  - A parser for a programming language is a **pushdown automaton**.

Context-Free Grammars and Context-Free Languages

- A **context-free grammar** is **a notation for describing languages**.
- CFGs are more powerful than REs, but they cannot still define all possible languages.
  - **CFGs** define **Context-Free Languages (CFLs).**
  - CFGs are useful to describe nested structures.
  - Since every regular language is a CFL, it can be defined by a CFG.
  - There are also languages that are not CFLs.



CFG – Example

- The language $\{ 0^n1^n \mid n \geq 0\}$ is not a regular language, but it is a CFL.
  - It can be defined by a CFG.
- A CFG for $\{ 0^n1^n \mid n \geq 0\}$ is:

$$S \rightarrow \varepsilon$$

$$S \rightarrow 0S1$$

- 0 and 1 are **terminals**. $\Sigma = \{0,1\}$ is the alphabet of the language.
- S is a **variable** (or **nonterminal**).
- S is also the **start symbol** of this CFG.
- $S \rightarrow \varepsilon$ and $S \rightarrow 0S1$ are **productions** (or **rules**)

CFG – Example

Basis:

- Production **S → ε** says that **ε** is in the language.

Induction:

- Production **S → 0S1** says that if **w** is in the language then is **0w1** is in the language.

- ε is in the language.
- since ε is in the language, 01 is in the language.
- since 01 is in the language, 0011 is in the language.
- …
- Thus, the language of this CFG is $\{ 0^n 1^n \mid n \geq 0 \}$

Formal Definition of CFGs

- A **context-free grammar** **G** is a quadruple

  G = (V, T, P, S)

  where

- **V** is a finite set of **variables** (**non-terminals**).
  - Each variable represents a language.
- **T** is a finite set of **terminals**.
  - T is the alphabet of the language defined by the CFG.
- **P** is a finite set of **productions** of the form $\mathbf{A} \rightarrow \alpha$, where **A** is a variable and $\alpha \in (V \cup T)^*$
  - The left side of a production is a variable and its right side is a string of variables and terminals.
- **S** is a designated variable called the **start symbol**.
  - The start symbol is the variable whose language is defined.

CFG – Example 2

- Consider the language of palindromes

  $L_{pal} = \{w \in \Sigma^* : \; w = w^R \}$

- Some members of L_{pal} : *abba bob ses tat*
- L_{pal} is NOT regular, but L_{pal} is a context-free language.

- Let $\Sigma = \{0,1\}$ be the alphabet for L_{pal}.
- In this case, ε,0,1,00,11,000,010,101,111,0110,… will be in L_{pal}.
- A CFG G_{pal} for L_{pal} is:

  $$\mathbf{G_{pal} = (\{S\}, \{0,1\}, \{ S \rightarrow \varepsilon, S \rightarrow 0, S \rightarrow 1, S \rightarrow 0S0, S \rightarrow 1S1 \}, S)}$$

- Sometimes, we use a shorthand for a list of productions with the same left side.

  $$\mathbf{S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1}$$

Derivation

- Initially, we have a string that only contains the **start symbol**.
- We expand the **start symbol** using one of its productions (i.e., using a production whose left side (head) is the start symbol).
  - i.e. we replace the start symbol with a string which appears on the right side of a production rule belongs to the start symbol.
- If the resulting string contains at least one variable, we further expand the resulting string by replacing one of its variables with the right side (body) of one of its productions.
  - We can continue these replacements until we derive a string consisting entirely of terminals.
- ***The language of the grammar is the set of all strings of terminals that we can be obtained in this way.***
- **Replacement of a variable (in a string) with the right side of one of its productions is called as derivation.**

Derivation $\Rightarrow$

- Suppose **G = (V, T, P, S)** is a CFG.
- Let $\alpha A\beta$ be a string of terminals and variables where **A** is a variable.
  - i.e. α and β are strings in $(V \cup T)^*$, and A is V.
- Let $A \rightarrow \gamma$ be a production of G.
- Then, we say that

$$\alpha A\beta \underset{G}{\Rightarrow} \alpha\gamma\beta \quad \text{is a } \textbf{derivation}$$

- If G is understood, we just say that

$$\alpha A\beta \Rightarrow \alpha\gamma\beta \quad \text{is a } \textbf{derivation}$$

- One derivation step can replace any variable in the string with the right side (body) of one of its productions.

Derivation Sequence $\stackrel{*}{\Rightarrow}$

- We can extend the derivation ($\Rightarrow$) relationship to represent zero or more derivation steps.
- We use symbol $\stackrel{*}{\Rightarrow}$ to denote zero or more steps of a **derivation sequence**.

Derivation Sequence:

Basis:

- For any string α of terminals and variables, we say $\alpha \stackrel{*}{\Rightarrow} \alpha$.
- That is, ***any string derives itself***.

Induction:

- If $\alpha \stackrel{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \stackrel{*}{\Rightarrow} \gamma$.
- That is, if α can become β by zero or more steps, and one more step takes β to γ, then α can become γ by a derivation sequence.

Derivation Sequence $\overset{*}{\Rightarrow}$

- In other words, $\alpha \overset{*}{\Rightarrow} \beta$ means that there is a sequence of strings $\gamma_1, \gamma_2, \ldots, \gamma_n$ for some $n \geq 1$ such that

1. $\alpha = \gamma_1$,
2. $\beta = \gamma_n$, and
3. for $i=1,2,\ldots,n-1$, we have $\gamma_i \Rightarrow \gamma_{i+1}$

Derivation Sequence – Example 1

- Let CFG **$G = (\{S\}, \{0,1\}, \{ S \rightarrow \varepsilon, S \rightarrow 0S1 \}, S)$**
- **$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$** is a **derivation sequence**.
  - **S derives 000111; or 000111 is derived from S.**

- That is, **$S \overset{*}{\Rightarrow} 000111$** and also
  - **$S \overset{*}{\Rightarrow} 000S111$**
  - **$S \overset{*}{\Rightarrow} 00S11$**
  - **$0S1 \overset{*}{\Rightarrow} 000S111$**
  - **$00S11 \overset{*}{\Rightarrow} 000111$**

- **$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$** is a **derivation sequence**.

Derivation Sequence – Example 2

A CFG:

$\mathbf{S \rightarrow ASB \mid c}$

$\mathbf{A \rightarrow \varepsilon \mid aA}$

$\mathbf{B \rightarrow \varepsilon \mid bB}$

Derivation Sequences of **acb** from **S.**

$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB \Rightarrow acbB \Rightarrow acb$

$S \Rightarrow ASB \Rightarrow ASbB \Rightarrow ASb \Rightarrow Acb \Rightarrow aAcb \Rightarrow acb$

$S \Rightarrow ASB \Rightarrow AcB \Rightarrow aAcB \Rightarrow aAcbB \Rightarrow acbB \Rightarrow acb$

- We may select any non-terminal (variable) of the string for the replacement in each derivation step.

Leftmost and Rightmost Derivations

- **Leftmost Derivation** always replaces the **leftmost variable** (in the string) with one of its rule-bodies. $\Rightarrow_{lm}$

$$S \Rightarrow_{lm} ASB \Rightarrow_{lm} aASB \Rightarrow_{lm} aSB \Rightarrow_{lm} acB \Rightarrow_{lm} acbB \Rightarrow_{lm} acb$$

- **Rightmost Derivation** always replaces the **righmost variable** (in the string) by one of its rule-bodies. $\Rightarrow_{rm}$

$$S \Rightarrow_{rm} ASB \Rightarrow_{rm} ASbB \Rightarrow_{rm} ASb \Rightarrow_{rm} Acb \Rightarrow_{rm} aAcb \Rightarrow_{rm} acb$$

Leftmost and Rightmost Derivations

S → ASB | c

A → ε | aA

B → ε | bB

Derivation Sequences of **acb** from **S.**

$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB \Rightarrow acbB \Rightarrow acb$ is a **leftmost derivation**

$S \Rightarrow ASB \Rightarrow ASbB \Rightarrow ASb \Rightarrow Acb \Rightarrow aAcb \Rightarrow acb$ is a **rightmost derivation**

$S \Rightarrow ASB \Rightarrow AcB \Rightarrow aAcB \Rightarrow aAcbB \Rightarrow acbB \Rightarrow acb$

is **NOT** a leftmost or rightmost derivation.

Sentential Forms

- Let G = (V, T, P, S) be a CFG, and $\alpha \in (V \cup T)^*$
- If $S \overset{*}{\Rightarrow} \alpha$, we say that α is a **sentential form**.

- If $S \overset{*}{\Rightarrow}_{lm} \alpha$, we say that α is a **left-sentential form**.
- If $S \overset{*}{\Rightarrow}_{rm} \alpha$, we say that α is a **right-sentential form**.

- **L(G) is those sentential forms that are in T*.**

The Language of a CFG

- If G = (V, T, P, S) is a CFG, then **the language of G** is

$$\mathbf{L(G) = \{ w \in T^* : S \overset{*}{\Rightarrow} w \}}$$

- i.e. **the set of strings of terminals (strings over T*) that are derivable from S**

- If we call L(G) as a **context-free language**.
  - Ex: $L(G_{pal})$ is a context-free language.

- **For each CFL, there is a CFG, and each CFG generates a CFL.**

- Every regular language is a CFL.
  - That is, regular languages are a proper subset of context-free languages

The Language of a CFG – A Proof Example

- G_{pal} = ({S}, {0,1}, { $S \rightarrow \varepsilon$, $S \rightarrow 0$, $S \rightarrow 1$, $S \rightarrow 0S0$, $S \rightarrow 1S1$ }, S)
- $L_{pal} = \{w \in \Sigma^* : w = w^R\}$

Theorem: $L(G_{pal}) = L_{pal}$

Proof:

In order to prove this equality,

($\supseteq$ Direction): We have to prove that every member of L_{pal} is also a member of $L(G_{pal})$.

($\subseteq$ Direction): We have to prove that every member of $L(G_{pal})$ is also a member of L_{pal}.

The Language of a CFG – A Proof Example

$\supseteq$ Direction

Proof: **($\supseteq$ Direction) If $w \in L_{pal}$ then $w \in L(G_{pal})$,** i.e. G_{pal} can generate w

- Suppose $w = w^R$ ($w \in L_{pal}$)
- We prove by induction on the ***length of w*** (**|w|**) that $w \in L(G_{pal})$

Basis:

- $|w|=0$, or $|w|=1$.
- Then, w is **ε**, **0**, or **1**
- Since **S→ε** , **S→0** and **S→1** are productions of G_{pal},

  we can conclude that $\mathbf{S} \stackrel{*}{\Rightarrow} \mathbf{w}$ in all base cases.
  - $\mathbf{S} \stackrel{*}{\Rightarrow} \boldsymbol{\varepsilon}$
  - $\mathbf{S} \stackrel{*}{\Rightarrow} \mathbf{0}$
  - $\mathbf{S} \stackrel{*}{\Rightarrow} \mathbf{1}$

The Language of a CFG – A Proof Example

$\supseteq$ Direction

IH: If $w \in L_{pal}$ and $|w| \le n$ then $w \in L(G_{pal})$ i.e. $S \overset{*}{\Rightarrow} w$

Induction:

- Suppose $|w| = n+1 \ge 2$
- Since $w = w^R$, we have $w = 0x0$, or $w = 1x1$, and $x = x^R$

Case1:

- If $w=0x0$, by IH we know that $\mathbf{S} \overset{*}{\Rightarrow} \mathbf{x}$
- Then, by the structure of the grammar $\mathbf{S} \Rightarrow \mathbf{0S0} \overset{*}{\Rightarrow} \mathbf{0x0}$ where $0x0=w$

Case2:

- If $w=1x1$, by IH we know that $\mathbf{S} \overset{*}{\Rightarrow} \mathbf{x}$
- Then, by the structure of the grammar $\mathbf{S} \Rightarrow \mathbf{1S1} \overset{*}{\Rightarrow} \mathbf{1x1}$ where $1x1=w$

The Language of a CFG – A Proof Example

$\subseteq$ Direction

Proof: ($\subseteq$ Direction)

- We assume that $w \in L(G_{pal})$ and we must show that $w=w^R$.
- Since $w \in L(G_{pal})$, we have $S \overset{*}{\Rightarrow} w$
- We prove by induction of **the length of** $\overset{*}{\Rightarrow}$ (the length of the derivation sequence)

Basis:

- The derivation $S \overset{*}{\Rightarrow} w$ is done in one step.
- Then w must be **ε**, **0**, or **1**, they are all palindromes.

The Language of a CFG – A Proof Example

$\subseteq$ Direction

IH: If $S \overset{*}{\Rightarrow} w$ with less than n derivation steps and 1<n then $w \in L_{pal}$

Induction:

- Let n≥2, i.e. $S \overset{*}{\Rightarrow} w$ derivation takes n steps
- Derivation must be
  - $\mathbf{S \Rightarrow 0S0 \overset{*}{\Rightarrow} 0x0 = w}$ or
  - $\mathbf{S \Rightarrow 1S1 \overset{*}{\Rightarrow} 1x1 = w}$
- Since n≥2, and the productions **S→0S0** and **S→1S1** are the only productions that allows additional steps of a derivation.
- Note that, in either case, $\mathbf{S \overset{*}{\Rightarrow} x}$ takes n-1 steps.
- By the inductive hypothesis, we know that x is a palindrome;
- But if so, then **0x0** and **lxl** are also palindromes.
- We conclude that **w** is a palindrome, which completes the proof.

Parse Trees

Parse Trees

- **Parse trees** are an alternative representation to derivations.

- If $w \in L(G)$, for some CFG, then w has a parse tree, which tells us the (syntactic) structure of w.
  - If G is unambiguous, w can have only one parse tree.
  - If G is ambiguous, w may have more than one parse tree.
  - Ideally there should be only one parse tree for each string in the language. This means that the grammar should be unambiguous.
    - We may remove the ambiguity from some of ambiguous grammars in order to obtain unambiguous grammars by making certain assumptions.
    - Unfortunately, some CFLs are inherently ambiguous and they can be only defined by ambiguous grammars.

Constructing Parse Trees

- Let G = (V, T, P, S) be a CFG.

- **A tree is a parse tree for G if:**

  1. **Each interior node is labeled by a variable in V.**
     - **The root must be labeled by the start symbol S.**
  2. **Each leaf is labeled by a symbol in T ∪ {ε}.**
     - **Any ε-labeled leaf is the only child of its parent.**
  3. **If an interior node is labeled by the variable A, and its children (from left to right) labeled $X_1, X_2, \ldots, X_k$ then $A \rightarrow X_1 X_2 \ldots X_k \in P$.**

Parse Tree - Example

Grammar:

$S \rightarrow ASB \mid c$

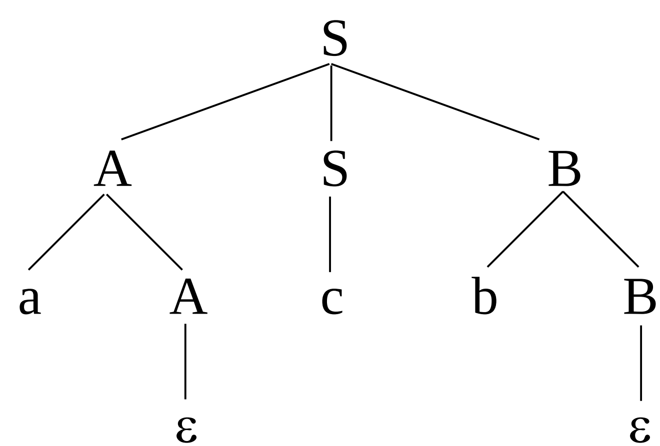
$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

A Derivation Sequence of **acb**

$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB \Rightarrow acbB \Rightarrow acb$

Parse tree of **acb**



Parse Tree & Derivation

Grammar:

$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

S

S

Parse Tree & Derivation

Grammar:

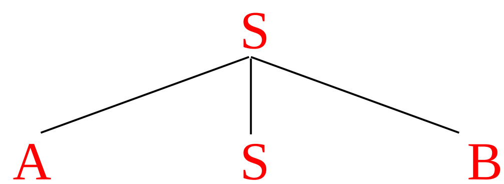
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$S \Rightarrow ASB$



Parse Tree & Derivation

Grammar:

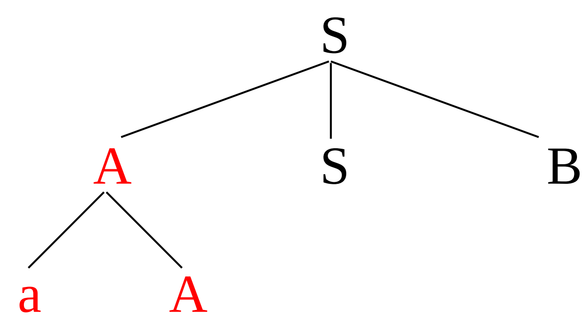
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$$S \Rightarrow ASB \Rightarrow aASB$$



Parse Tree & Derivation

Grammar:

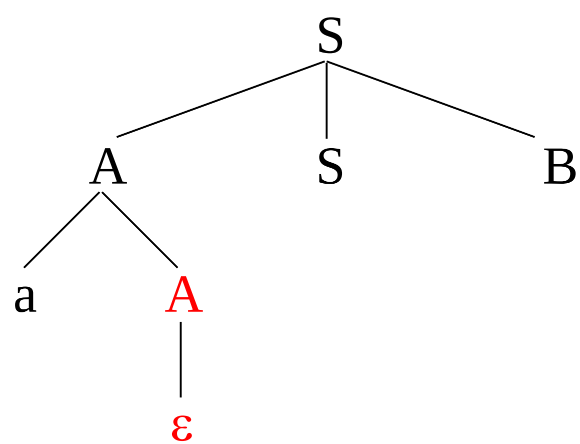
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB$



Parse Tree & Derivation

Grammar:

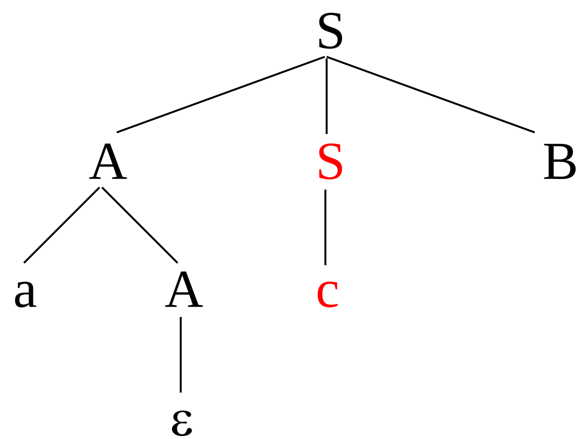
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB$$



Parse Tree & Derivation

Grammar:

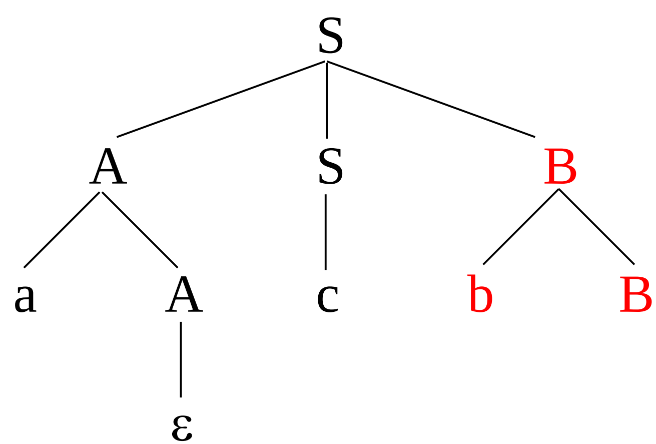
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB \Rightarrow acbB$$



Parse Tree & Derivation

Grammar:

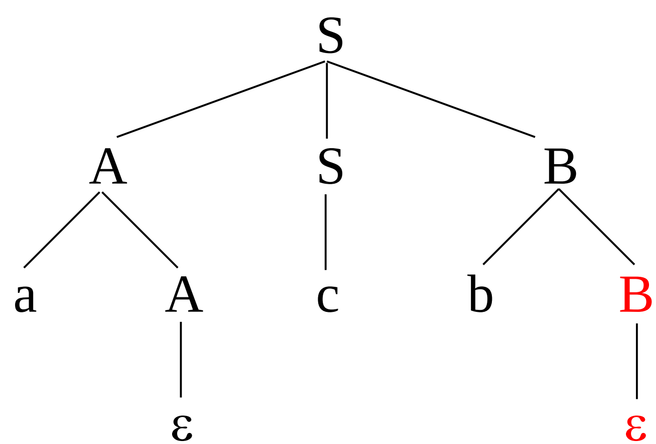
$S \rightarrow ASB \mid c$

$A \rightarrow \varepsilon \mid aA$

$B \rightarrow \varepsilon \mid bB$

- Each derivation step corresponds to the creation of an inner node (by creating its children).

$$S \Rightarrow ASB \Rightarrow aASB \Rightarrow aSB \Rightarrow acB \Rightarrow acbB \Rightarrow acb$$



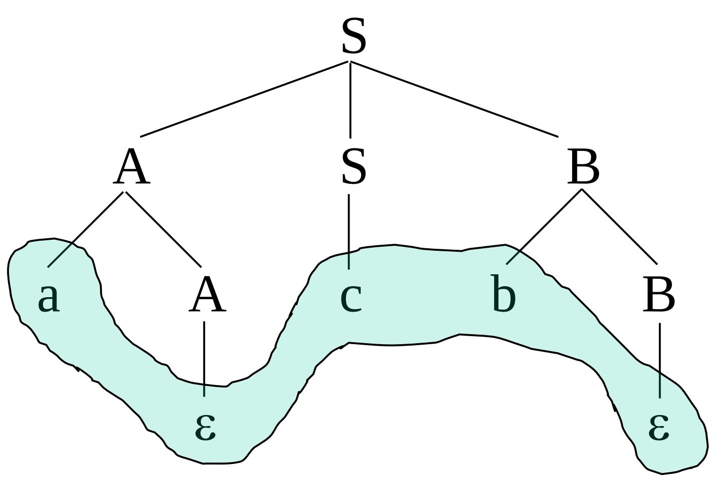
The Yield of a Parse Tree

- **The concatenation of the labels of the leaves in left-to-right order is called the** **yield of the parse tree.**

- The yield of the parse tree is a string of terminals.
  - The set of all yields of all parse trees of a CFG G is the language of G.

- Yield Example:

  a ε c b ε = acb



Parse Trees, Leftmost and Rightmost Derivations

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

- We will prove theorem for only leftmost derivations.

- We will prove:

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

  Part 2: If $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{w}$, then there is a parse tree with root A and yield w.

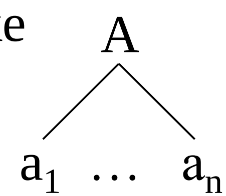
Proof – Part 1

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

Proof: Induction on the height of the tree.

– The height of a tree is the length of the longest path from the root to a leaf.

Basis: Height is 1. Tree looks like



– Its yield is $a_1 \ldots a_n$

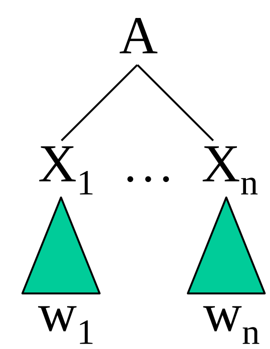
- $\mathbf{A \rightarrow a_1 \ldots a_n}$ must be a production.
- Thus, we have $\mathbf{A} \Rightarrow_{\mathrm{lm}} \mathbf{a_1 \ldots a_n}$
- $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{a_1 \ldots a_n}$

Proof – Part 1

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

IH: Part 1 holds for the trees with the height < h.

Induction: Take a tree whose height is h. Tree looks like



- Its yield is $w_1 \ldots w_n$
- The height of each subtree headed by X_i is less than h.

- $A \rightarrow X_1 \ldots X_n$ must be a production.
- $A \Rightarrow_{lm} X_1 \ldots X_n$ since $A \rightarrow X_1 \ldots X_n$ is a production.
- $X_i \overset{*}{\Rightarrow}_{lm} w_i$ holds for each X_i by IH.

- Thus, $\mathbf{A \Rightarrow_{lm} X_1 \ldots X_n \overset{*}{\Rightarrow}_{lm} w_1 X_2 \ldots X_n \overset{*}{\Rightarrow}_{lm} w_1 w_2 X_3 \ldots X_n \overset{*}{\Rightarrow}_{lm} \cdots \overset{*}{\Rightarrow}_{lm} w_1 w_2 \ldots w_n}$

Proof – Part 2

Part 2: If $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{w}$, then there is a parse tree with root A and yield w.

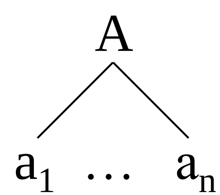
Proof:

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

Basis: The length of the derivation sequence $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{a_1 \ldots a_n}$ is 1.

– That is, the derivation sequence is $\mathbf{A} \Rightarrow_{lm} \mathbf{a_1 \ldots a_n}$

- $\mathbf{A} \rightarrow \mathbf{a_1 \ldots a_n}$ must be a production.
- Thus, there must be a parse tree looks like



– Its yield is $\mathbf{a_1 \ldots a_n}$

Proof – Part 2

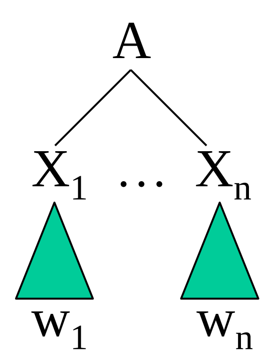
Part 2: If $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{w}$, then there is a parse tree with root A and yield w.

IH: Part 2 holds for the leftmost derivations with fewer steps than k.

Induction: Take a derivation sequence $\mathbf{A} \overset{*}{\Rightarrow}_{\mathbf{lm}} \mathbf{w}$ with k steps.

- The first step of the derivation sequence is $\mathbf{A} \Rightarrow_{lm} \mathbf{X_1 \ldots X_n}$

- w can be divided so the first portion w_1 is derived from X_1, the next w_2 is derived from X_2, and so on. If X_i is a terminal, then $w_i = X_i$.
- That is, each variable X_i has a derivation sequence $X_i \overset{*}{\Rightarrow}_{\mathbf{lm}} w_i$.
  - And the each derivation takes fewer steps 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.
  - Its yield is $\mathbf{w_1 w_2 \ldots w_n = w}$



Ambiguity

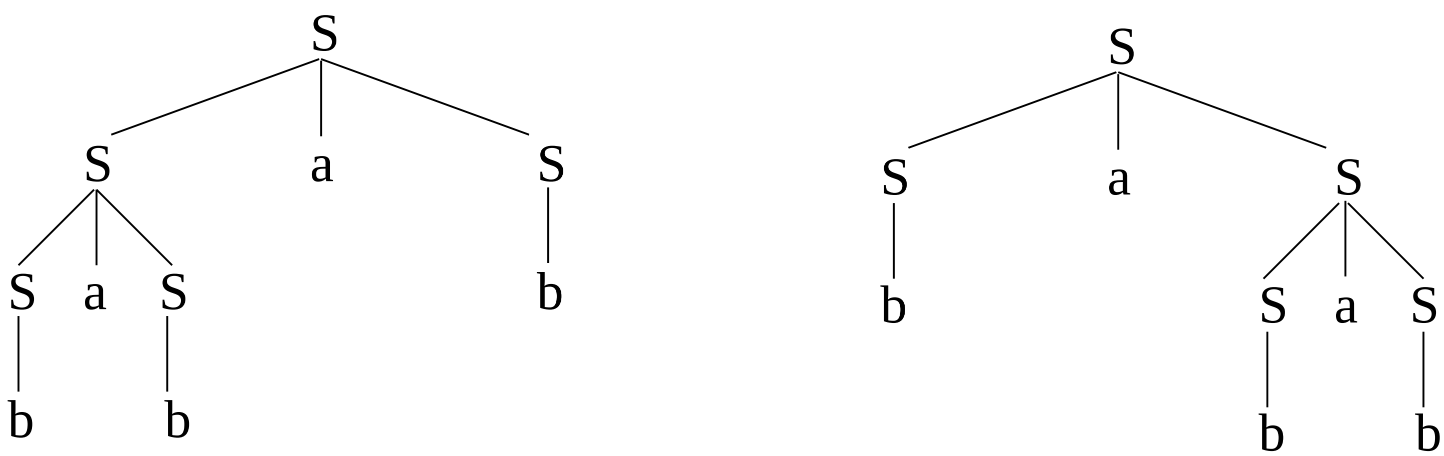
Ambiguous Grammars

- **A CFG is ambiguous if it produces more than one parse tree for a string in the language.**
  - **i.e. there is a string in the language that is the yield of two or more parse trees.**

Example:

$S \rightarrow SaS \mid b$ is an ambiguous grammar.

There are two parse trees for the string **babab**



Ambiguity, Leftmost and Rightmost Derivations

- If there are two different parse trees for a string in the language, they must produce two different **leftmost derivations** for that string.
  - Conversely, two different leftmost derivations of a string produce two different parse trees for that string.
- Likewise for rightmost derivations.

- Thus, equivalent definitions of **ambiguous grammar** are:

1. **A CFG is ambiguous if there is a string in the language that has two different leftmost derivations.**
2. **A CFG is ambiguous if there is a string in the language that has two different rightmost derivations.**

Ambiguity, Leftmost and Rightmost Derivations

$S \rightarrow SaS \mid b$

- There are two **leftmost** derivation sequences for the string **babab**

1. $\mathbf{S \Rightarrow_{lm} SaS \Rightarrow_{lm} SaSaS \Rightarrow_{lm} baSaS \Rightarrow_{lm} babaS \Rightarrow_{lm} babab}$
2. $\mathbf{S \Rightarrow_{lm} SaS \Rightarrow_{lm} baS \Rightarrow_{lm} baSaS \Rightarrow_{lm} babaS \Rightarrow_{lm} babab}$

- There are two **rightmost** derivation sequences for the string **babab**

1. $\mathbf{S \Rightarrow_{rm} SaS \Rightarrow_{rm} Sab \Rightarrow_{rm} SaSab \Rightarrow_{rm} Sabab \Rightarrow_{rm} babab}$
2. $\mathbf{S \Rightarrow_{rm} SaS \Rightarrow_{rm} SaSaS \Rightarrow_{rm} SaSab \Rightarrow_{rm} Sabab \Rightarrow_{rm} babab}$

Ambiguity

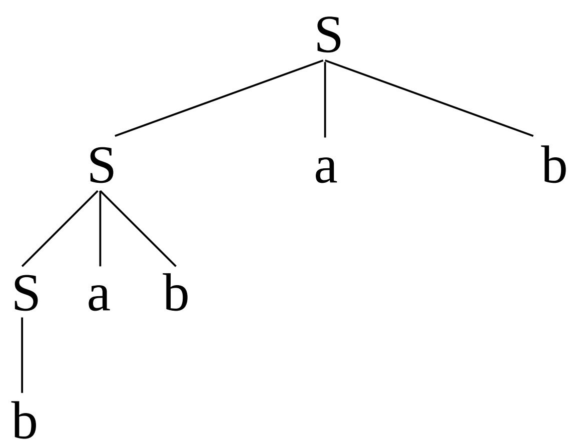
- We can create an equivalent CFG (which produces the same language) by eliminating the ambiguity from the following ambiguous CFG.

  $S \rightarrow SaS \mid b$

- In the following unambiguous CFG, we prefer left groupings.

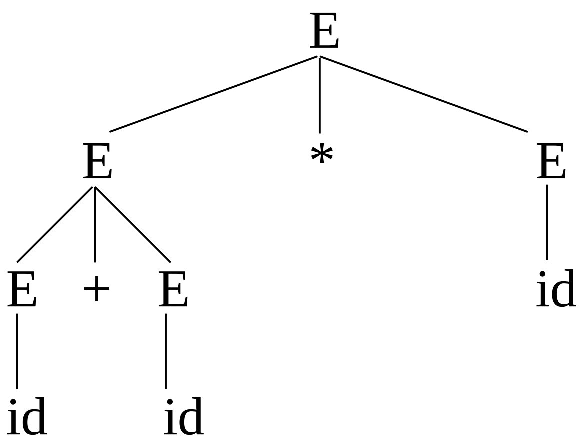
  $S \rightarrow Sab \mid b$

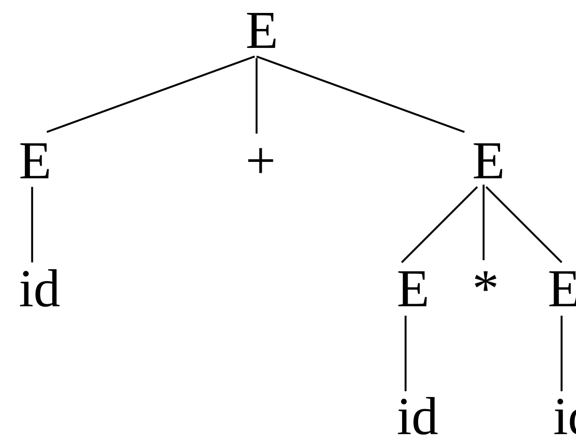
- Now, there is only one parse tree for the string **babab**



Ambiguity

- An ambiguous grammar for expressions: $\mathbf{E} \rightarrow \mathbf{E{+}E} \mid \mathbf{E{*}E} \mid \mathbf{E{\wedge}E} \mid \mathbf{id} \mid \mathbf{(E)}$
- 2 parse trees, 2 leftmost and 2 rightmost derivations for the expression **id+id*id**





$E \Rightarrow_{lm} E*E \Rightarrow_{lm} E+E*E \Rightarrow_{lm} id+E*E$
$\quad \Rightarrow_{lm} id+id*E \Rightarrow_{lm} id+id*id$

$E \Rightarrow_{lm} E+E \Rightarrow_{lm} id+E \Rightarrow_{lm} id+E*E$
$\quad \Rightarrow_{lm} id+id*E \Rightarrow_{lm} id+id*id$

$E \Rightarrow_{rm} E*E \Rightarrow_{rm} E*id \Rightarrow_{rm} E+E*id$
$\quad \Rightarrow_{rm} E+id*id \Rightarrow_{rm} id+id*id$

$E \Rightarrow_{rm} E+E \Rightarrow_{rm} E+E*E \Rightarrow_{rm} E+E*id$
$\quad \Rightarrow_{rm} E+id*id \Rightarrow_{rm} id+id*id$

Ambiguity – Operator Precedence

- Ambiguous grammars (because of ambiguous operators) can be disambiguated according to the precedence and associativity rules.

$$E \rightarrow E+E \mid E*E \mid E\text{^}E \mid id \mid (E)$$

- Disambiguate this grammar using the following precedence and associativity rules.

Precedence: ^ (right to left)
* (left to right)
\+ (left to right)

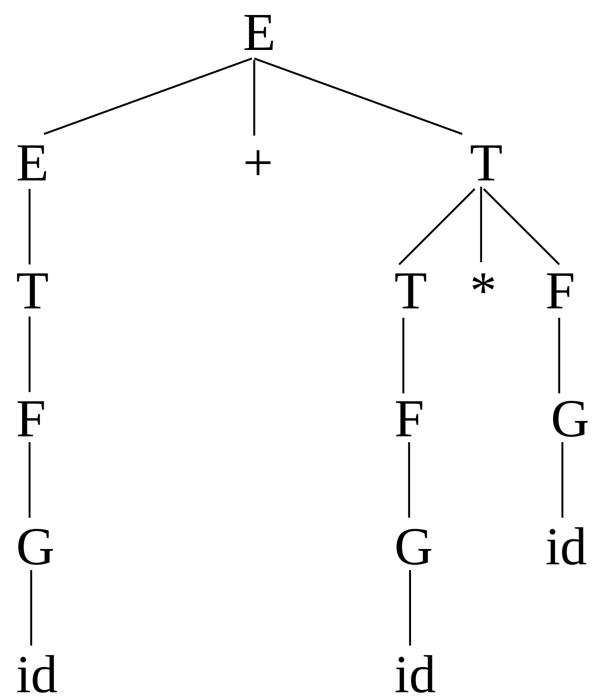
- Disambiguated grammar:

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow G\text{^}F \mid G$$
$$G \rightarrow id \mid (E)$$

Ambiguity – Operator Precedence

E → E+T | T
T → T*F | F
F → G^F | G
G → id | (E)

parse tree for **id+id*id**



Inherent Ambiguity

- Some CFLs may have both ambiguous grammars and unambiguous grammar.
  - In this case, we may disambiguate their ambiguous grammars.
- Unfortunately, there are some CFLs that do not have any unambiguous grammar.
- **A context free language L is said to be inherently ambiguous if all its grammars are ambiguous.**
- If even one grammar for L is unambiguous, then L is an unambiguous language.
  - Our expression language is an unambiguous language.
  - Even though the first grammar for expressions is ambiguous, there is another language for the expressions language is unambiguous.

Inherent Ambiguity

- **A context free language L is said to be inherently ambiguous if all its grammars are ambiguous.**

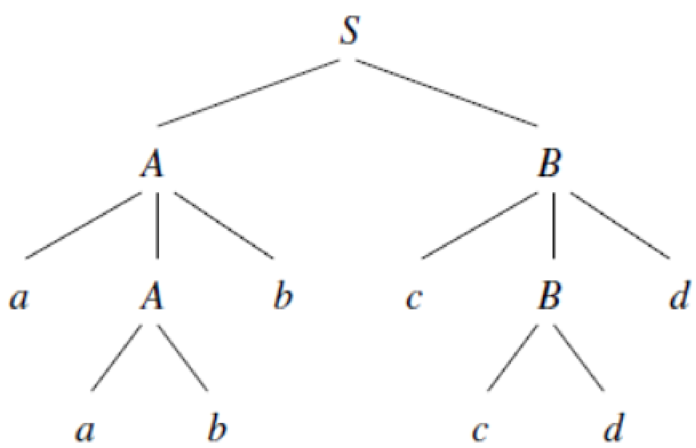
Example: Consider $L = \{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}$

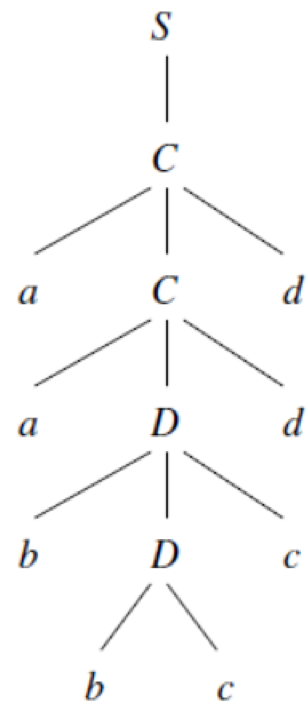
A grammar for L is

$$S \to AB \mid C$$
$$A \to aAb \mid ab$$
$$B \to cBd \mid cd$$
$$C \to aCd \mid aDd$$
$$D \to bDc \mid bc$$

Inherent Ambiguity

- The parse trees for the string **aabbccdd**.





$S \to AB \mid C$
$A \to aAb \mid ab$
$B \to cBd \mid cd$
$C \to aCd \mid aDd$
$A \to bDc \mid bc$

$S \Rightarrow_{lm} AB \Rightarrow_{lm} aAbB \Rightarrow_{lm} aabbB$
$\Rightarrow_{lm} aabbcBd \Rightarrow_{lm} aabbccdd$

$S \Rightarrow_{lm} C \Rightarrow_{lm} aCd \Rightarrow_{lm} aaCdd$
$\Rightarrow_{lm} aabDcdd \Rightarrow_{lm} aabbccdd$

- It can be shown that every grammar for L behaves like the one above. The language L is inherently ambiguous.

CFG – Questions

- Design context-free grammars for the following languages:

- $\{0^n1^m : n>m\geq 0\}$

$$S \rightarrow 0S1 \mid 0A$$
$$A \rightarrow \varepsilon \mid 0A$$

- The strings of 0's and 1's that contain equal number of 0's and 1's.

$$S \rightarrow 0S1S \mid 1S0S \mid \varepsilon$$

CFG – Questions

- Design context-free grammars for the following language:
- $\{0^n1^n : n \geq 0\} \cup \{1^n 0^n : n \geq 0\}$

  $S \rightarrow A \mid B$

  $A \rightarrow 0A1 \mid \varepsilon$

  $B \rightarrow 1B0 \mid \varepsilon$

- Is this grammar ambiguous?

  YES: Two leftmost derivations for **ε**

  $S \Rightarrow_{lm} A \Rightarrow_{lm} \varepsilon$ $\qquad\qquad$ $S \Rightarrow_{lm} B \Rightarrow_{lm} \varepsilon$

- Disambiguate this grammar.

  $S \rightarrow A \mid B \mid \varepsilon$

  $A \rightarrow 0A1 \mid 01$

  $B \rightarrow 1B0 \mid 10$

Is Every Regular Language a CFL?

- Every regular language is a CFL.

Create a CFG for a given regular language whose DFA is given:

- A DFA M = (Q,Σ,δ,q_0,F) is given, define the CFG **G = (V,T,P,S)** as follows:

  V = $\{ S_i \mid q_i \text{ is in } Q \}$

  T = Σ

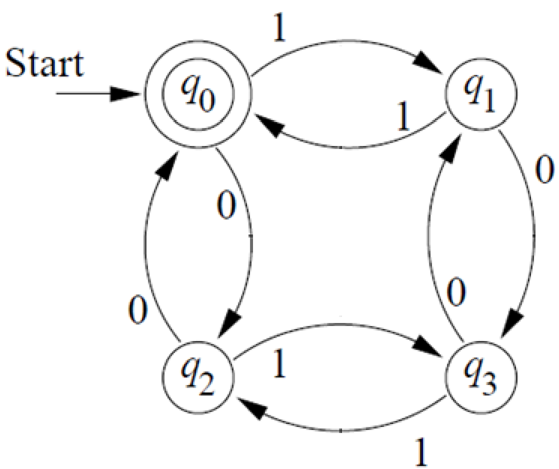
  P = $\{ S_i \rightarrow aS_j \mid \delta(q_i,a) = q_j \} \cup \{ S_i \rightarrow \varepsilon \mid q_i \text{ is in } F\}$

  S = S_0

Then prove the correctness.

Is Every Regular Language a CFL?

- Create a CFG for the following DFA:



$$S_0 \rightarrow 0S_2 \mid 1S_1 \mid \varepsilon$$
$$S_1 \rightarrow 0S_3 \mid 1S_0$$
$$S_2 \rightarrow 0S_0 \mid 1S_3$$
$$S_3 \rightarrow 0S_1 \mid 1S_2$$

- **Every regular language can be defined by a right linear grammar.**
- A **right linear grammar rule** must be in one of the following forms:

  A → ε

  A → a

  A → aB where **A,B** are variables and **a** is a terminal