

week 2: **Context-free Grammars and Languages**

This Lecture Covers Chapter 5 of HMU: Context-free Grammars and Languages

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeek; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2025

Content of this Chapter

- (Context-free) Grammars
- (Leftmost and Rightmost) Derivations
- Parse Trees
- An Equivalence between Derivations and Parse Trees
- Ambiguity in Grammars

Additional Reading: Chapter 5 of HMU.

Grammars

Introduction to Grammars

- We have so far seen **machine-like** means (e.g., DFAs) and **declarative** means (e.g., regular expressions) of defining languages
- **Grammars** are a **generative** means of defining languages.
- Grammars can be used to create a strictly larger class of languages.
- They are especially useful in compiler and parser design; they can be used to check if:
 - parentheses are balanced in a program,
 - `else` occurrences have a matching `if`, etc.

Grammars: Formal Definition

- A **context-free** grammar (CFG) $G = (V, T, \mathcal{P}, S)$, where
 - V is a **finite** set whose elements are called **variables** or **non-terminal symbols**.
Notation: upper case letters, e.g., $A, B, \dots$
 - T is a **finite** set whose elements are called **terminal symbols**; T is precisely the alphabet of the language generated by the grammar G .
Notation: lower case letters, e.g., $s_1, s_2, \dots$
 - $\mathcal{P} \subseteq V \times (V \cup T)^*$ is a **finite** set of **production rules**.
 - Each production rule (A, α) is also written as $A \rightarrow \alpha$.
Terminology: A , α are called the **head** and **body** of the production rule, resp.
 - $S \in V$ is the unique variable/non-terminal that 'generates' the language.

Notation

- *Strings consisting of non-terminals and/or terminals will be denoted by greek symbols, e.g., $\alpha, \beta, \dots$*
- *Strings of terminals will be denoted by lower case letters, e.g., w, u, v*

Derivations and Languages

How do Grammars Generate Languages?

- A string $w \in T^*$ is in the language $L(G)$ generated by $G = (V, T, \mathcal{P}, S)$ iff we can **derive** w from S , i.e.,
 - start from S and use production rule(s) repeatedly to replace heads of the rules by their bodies until a string in T^* is obtained.

Example 5.2.1

Let $G = (\{S\}, \{0, 1\}, \mathcal{P}, S)$ be a CFG with $\mathcal{P}$ given by

$$(1) \left\{ \begin{array}{l} (S, \epsilon), (S, 0), (S, 1) \\ (S, 0S0), (S, 1S1) \end{array} \right\}$$

$$S \rightarrow 0$$

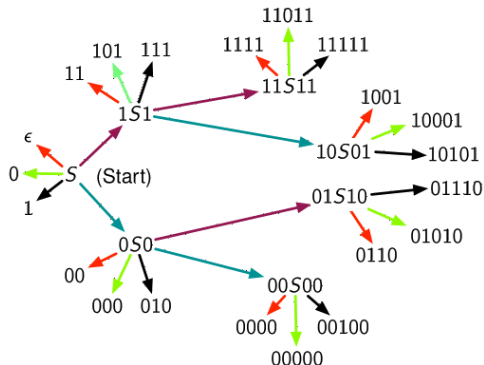
$$S \rightarrow 1$$

$$S \rightarrow 0$$

$$S \rightarrow 1.51$$

$$\delta \rightarrow \epsilon \mid 0 \mid$$

()



Derivation: Formal Definition

Definition

Given $G = (V, T, \mathcal{P}, S)$ and $\alpha, \beta \in (V \cup T)^*$, a **derivation** of β from α is a finite sequence of strings $\gamma_1 \xRightarrow{G} \gamma_2 \xRightarrow{G} \cdots \xRightarrow{G} \gamma_k$ for some $k \in \mathbb{N}$ where

1. $\gamma_1 = \alpha$ and $\gamma_k = \beta$;
2. $\gamma_1, \dots, \gamma_k \in (V \cup T)^*$
3. For each $i = 1, \dots, k - 1$, γ_{i+1} is obtained from γ_i by replacing the head of a production rule of $\mathcal{P}$ by its body.

The following phrases are used interchangeably.

β is derived from $\alpha \Leftrightarrow$ there exists a derivation of β from $\alpha \Leftrightarrow \alpha \xRightarrow{*}_G \beta$.

Example 5.2.2

For the grammar $G = (\{S\}, \{0, 1\}, \mathcal{P}, S)$ with $\mathcal{P}$ given by $S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$, the following is a derivation of 010111010 from S

$$\begin{array}{ccccccc}
 S & \xRightarrow{G} & \textcolor{red}{0S0} & \xRightarrow{G} & \textcolor{red}{01S10} & \xRightarrow{G} & \textcolor{red}{010S010} & \xRightarrow{G} & \textcolor{red}{0101S1010} & \xRightarrow{G} & \textcolor{red}{010111010}. \\
 S \rightarrow 0S0 & & & S \rightarrow 1S1 & & & S \rightarrow 0S0 & & S \rightarrow 1S1 & & S \rightarrow 1
 \end{array}$$

Sentential Forms and Language Generated by a Grammar: Definitions

Definition

Given $G = (V, T, \mathcal{P}, S)$, any string in $(V \cup T)^*$ derived from S is a sentential form.

- The set of all sentential forms of G (denoted by $\text{SF}(G)$) is defined inductively:
 - › Basis: $S \in \text{SF}(G)$
 - › Induction: if $\alpha A \gamma \in \text{SF}(G)$ for some $\alpha, \gamma \in (V \cup T)^*$ and $A \in V$, and $A \rightarrow \beta$ is a production rule, then $\alpha \beta \gamma \in \text{SF}(G)$.
 - › Only those strings that are generated by the above induction are sentential forms.

Definition

Given CFG $G = (V, T, \mathcal{P}, S)$, the language $L(G)$ generated by G is the set of sentential forms that are also in T^* , i.e., $L(G) = \text{SF}(G) \cap T^*$.

Example 5.2.3

For the CFG $G = (\{S\}, \{0, 1\}, \mathcal{P}, S)$ with $\mathcal{P}$ given by $S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$,

- (1) $S, \epsilon, 0, 1, 0S0, 00, 000, 010, 1S1, 11, 101, 111, \dots$ are all sentential forms.
- (2) $S, \epsilon, 0, 1, 0S0, 00, 000, 010, 1S1, 11, 101, 111, \dots$ are in $L(G)$.

Other Sentential Forms

- At each step of a derivation, one can replace any variable by a suitable production.
- If at each non-trivial step of the derivation the **leftmost** (or **rightmost**) variable is replaced by a production rule, then the derivation is said to be a **leftmost** (or **rightmost**) derivation, respectively. We let $\alpha \xRightarrow[LM]{*} \beta$ (or $\alpha \xRightarrow[RM]{*} \beta$) to denote the existence of a leftmost (or rightmost) derivation of β from α , respectively.
- Sentential forms derived via **leftmost** (or **rightmost**) derivations are known as **leftmost** (or **rightmost**) sentential forms, respectively.

Balanced Parentheses Example

Consider the CFG $G = (\{S\}, \{ (,) \}, \mathcal{P}, S)$ with $\mathcal{P}$ given by $S \rightarrow SS \mid (S) \mid ()$.

[Derivation]	$S \xRightarrow[G]{*} SS \xRightarrow[G]{*} (S)S \xRightarrow[G]{*} (S)() \xRightarrow[G]{*} (())()$
[Leftmost Derivation]	$S \xRightarrow[G]{*} SS \xRightarrow[G]{*} (S)S \xRightarrow[G]{*} (())S \xRightarrow[G]{*} (())()$
[Rightmost Derivation]	$S \xRightarrow[G]{*} SS \xRightarrow[G]{*} S() \xRightarrow[G]{*} (S)() \xRightarrow[G]{*} (())()$

In the above, $\uparrow$ indicates the variable that is replaced in the following step

Context-free Languages

Definition: Context-free Languages

We call a language context-free if it can be described by a context-free grammar.

Remark: There are alternative definitions, as we will see later.

Parse Trees

Parse Trees

- Parse trees are a graphical method of representing derivations.
- They are used in compilers to represent the source program.

Definition

Given a CFG $G = (V, T, \mathcal{P}, S)$, a **parse tree** for G is any directed labelled tree that meets the following three conditions:

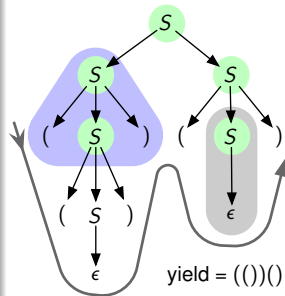
- every interior node is labelled by a non-terminal (i.e., variable);
- every leaf node is labelled by a non-terminal, or a terminal or ϵ ; however if it is labelled by ϵ , it is the sole child of its parent.
- if an interior node is labelled by $A \in V$, and its children are labelled $s_1, \dots, s_k \in V \cup T \cup \{\epsilon\}$, then $A \rightarrow s_1 \dots s_k$ is a production rule in $\mathcal{P}$.

The **yield** of a parse tree is the string formed from the labels of the tree leaves read from left to right.

Note: The yield is not necessarily a string of terminals.

$$G = (\{S\}, \{(\,, \,)\}, \mathcal{P}, S)$$

$$\mathcal{P} : S \rightarrow SS|(S)|\epsilon$$



An Equivalence between Parse Trees and Derivations

Derivations and Parse Trees

- Parse trees, derivations, leftmost derivations, and rightmost derivations are equivalent means of generating words of the language $L(G)$ of a CFG G .
- The proof for equivalence of rightmost derivations mirrors that of leftmost derivations. (So we'll not delve into rightmost derivations).

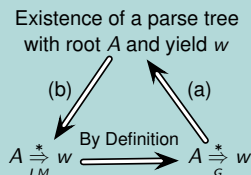
Theorem 5.5.1

Let CFG $G = (V, T, \mathcal{P}, S)$ be given. Let $A \in V$ and $w \in T^*$. Then,

$$A \xRightarrow[G]{*} w \Leftrightarrow A \xRightarrow{LM}{*} w \Leftrightarrow \text{there exists a parse tree with root } A \text{ and yield } w \Leftrightarrow A \xRightarrow{RM}{*} w.$$

Proof Idea

We'll show the following implications.



Part (a) of Proof of Theorem 5.5.1: $A \xRightarrow[G]{*} w \Rightarrow \exists \text{ Parse Tree}$

- We use induction on the (length of the) derivation.

Lemma 5.5.2

Let CFG $G = (V, T, \mathcal{P}, S)$ be given. Let $A \in V$ and $\alpha \in SF(G)$. If $A \xRightarrow[G]{} \alpha$, then there exists a parse tree with root A and yield α .*

Proof of Lemma 5.5.2 (Induction on the length of derivation)

- Suppose $A \xRightarrow[G]{*} \alpha$ is a derivation of length 0.
- Then A is a parse tree with root A and yield A .

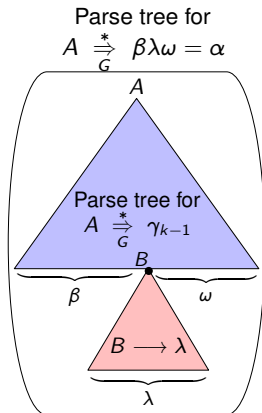
Part (a) of Proof of Theorem 5.5.1: $A \xRightarrow[G]{*} w \Rightarrow \exists$ Parse Tree

Proof of Lemma 5.5.2 (Induction on derivations)

- > Hypothesis: the claim is true for all derivations of length $k - 1$ or lesser for some $k \geq 1$.
- > Suppose a derivation of α from A in k steps exists.

$$A = \gamma_1 \xRightarrow[G]{*} \gamma_2 \xRightarrow[G]{*} \gamma_3 \xRightarrow[G]{*} \cdots \xRightarrow[G]{*} \gamma_{k-1} \xRightarrow[G]{*} \gamma_k = \alpha$$

- > We know that $\gamma_{k-1} = \beta B \omega$ and $\alpha = \beta \lambda \omega$ where (a) $\beta, \omega \in (V \cup T)^*$, (b) $B \in V$, and (b) $B \rightarrow \lambda$ is a production rule.
- > Extend the parse tree from the first $k - 1$ steps by:
 - If $\lambda = X_1 \dots X_n$ with $X_1, \dots, X_n \in V \cup T$, add children $X_1, \dots, X_n$ to node B .



Part (b) of Proof of Theorem 5.5.1: Parse Tree $\Rightarrow A \xRightarrow[LM]{*} w$

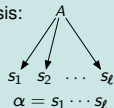
Proof of Theorem 5.5.1 (Induction on the height of the tree)

- Base case: the parse tree has height 0
- Then A is a leftmost derivation in zero steps.
- Induction: Let the claim be true for all parse trees of up to height $\ell - 1$.
- Consider the root and its (say k) children. This corresponds to a production rule $A \rightarrow X_1 \cdots X_k$.
 - If X_i is a leaf, then the yield of the sub-tree rooted at X_i is $w_i = X_i$ itself. Then trivially $X_i \xRightarrow[LM]{*} w_i$.
 - If X_i is not a leaf, let w_i be the yield of the parse (sub-)tree rooted at X_i of depth $\ell - 1$ or less. Then, by induction hypothesis, $X_i \xRightarrow[LM]{*} w_i$.

Then, the following is a leftmost derivation for α from A

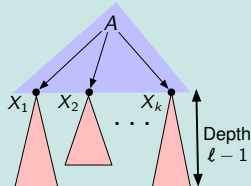
$$A \xRightarrow[G]{*} X_1 X_2 \cdots X_k \xRightarrow[LM]{*} w_1 X_2 \cdots X_k \xRightarrow[LM]{*} w_1 w_2 X_3 \cdots X_k \xRightarrow[LM]{*} \cdots \xRightarrow[LM]{*} w_1 \cdots w_k$$

Basis:



$$(A, \alpha) \equiv (A \rightarrow \alpha) \in \mathcal{P}$$

Induction:



Ambiguous Grammars

Ambiguity in CFGs

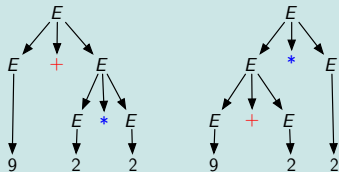
Definition

A given CFG G is **ambiguous** if a string $w \in L(G)$ is the yield of two **different** parse trees. Equivalently, a CFG G is ambiguous if a string $w \in L(G)$ has two **different** leftmost (or rightmost) derivations.

- › Ambiguity is a property of a grammar, and **not** the language it generates.

An Example

- › CFG $G = (\{E\}, \{0, 1, \dots, 9, +, *\}, \mathcal{P}, E)$ with $\mathcal{P} : E \longrightarrow E + E \mid E * E \mid 0 \mid 1 \mid \dots \mid 9$
- › Consider the parse trees for $9 + 2 * 2$.
- › Since there are two distinct parse trees, a compiler will not know to reduce this to 13 or to 22.



- › This ambiguity is addressed by precedence rules for operators.

Ambiguity in CFGs

- Some languages are generated by unambiguous as well as ambiguous grammars.

Balanced Parentheses Example

- CFG $G_1 = (\{S\}, \{(\,,\,)\}, \mathcal{P}, S)$ with $\mathcal{P} : S \longrightarrow SS|(S)|()$
- G_1 is ambiguous for there are two leftmost derivations for $()()()$.

$$S \xRightarrow{LM} SS \xRightarrow{LM} ()S \xRightarrow{LM} ()SS \xRightarrow{LM} ()()S \xRightarrow{LM} ()()()$$

$$S \xRightarrow{LM} SS \xRightarrow{LM} SSS \xRightarrow{LM} ()SS \xRightarrow{LM} ()()S \xRightarrow{LM}^* ()()()$$

- CFG $G_2 = (\{B, R\}, \{(\,,\,)\}, \mathcal{Q}, B)$ with $\mathcal{Q} : B \longrightarrow (RB|\epsilon$ and $R \longrightarrow)(RR$
- G_2 is **not** ambiguous, since there is precisely only one rule at any stage of derivation.

$$B \xRightarrow{LM}^* (RB \xRightarrow{LM} ()B \xRightarrow{LM} ()(RB \xRightarrow{LM} ()()B \xRightarrow{LM} ()()()B \xRightarrow{LM} ()()()()\epsilon$$

- Some languages are intrinsically ambiguous, e.g., $\{0^i1^j2^k : i = j \text{ or } j = k\}$. All grammars for such languages are ambiguous.
- In general, there is **no** way to tell if a grammar is ambiguous.