In general, to describe a language, there are two possible approaches:

1) Recognition: describe rules (or a mechanism) to determine whether or not a certain string belongs to a language

   e.g. an automaton is such a mechanism

2) Generation: define rules to generate all strings of a language

A grammar is a formalism for defining a language in terms of rules that generate all strings of the language.

Since 1920, various formal methods based on the notion of rewriting or derivation have been proposed by Axel Thue, Emil Post, R.A. Muench.

In the mid 1950s, the linguist Noam Chomsky introduced the notion of formal grammar with the aim of formalizing natural language. Formal grammars are in fact too simplistic to capture natural language, but they were adopted as the main formal tool to define syntactic properties of artificial languages (e.g. programming languages).
Definition: Given an alphabet $\Sigma$, a (formal) grammar $G$ is a quadruple $G = (V_N, V_T, P, S)$ where
- $V_T \subseteq \Sigma$ is a finite nonempty set of symbols called terminals.
- $V_N$ is a finite nonempty set of symbols such that $V_N \cap \Sigma = \emptyset$, called variables or nonterminals, or syntactic categories.
- Each variable represents a language.
- $S \in V_N$ is called start symbol or axiom, and represents the language being defined by $G$.
- $P$ is a binary relation over
  
  \[(V_N U V_T)^* \cdot V_N \cdot (V_N U V_T)^* \cdot \sum (V_N U V_T)^* \cdot \]

  Each element $(\alpha, \beta) \in P$ is called a production or rule, and is generally written as $\alpha \rightarrow \beta$.

  Note: $\alpha$... sequence of terminals and nonterminals with at least one nonterminal

  $\beta$... sequence of terminals and nonterminals

Definition: The language $L(G)$ generated by a grammar $G$ is the set of strings of terminals only that can be generated starting from the axiom by a finite sequence of rule applications.

Each application of a rule $\alpha \rightarrow \beta$ consists in replacing an occurrence of $\alpha$ with $\beta$. 
Example: Palindromes:

A palindrome is a word that reads the same both forwards and backwards. (ALLATITALIA, AMORAMA)

$L_{\text{pal}} = \{ w \in \{0, 1\}^* \mid w^R = w \}$

Grammar $G_{\text{pal}} = (V_N, V_T, P, S)$, where $P$ consists of:

1) $S \rightarrow \varepsilon$  
   \text{basis: } \varepsilon, 0, 1 \text{ are palindromes}

2) $S \rightarrow 0$

3) $S \rightarrow 1$

4) $S \rightarrow 0 S O$ \text{ induction: if } S \text{ is a palindrome, so are OSO and 1S1}

5) $S \rightarrow 1 S 1$

Example of derivation:

0110 : $S \rightarrow 0 S O \rightarrow 0 S 1 S O \rightarrow 0 1 S 1 0 \rightarrow 0110$

11011 : $S \rightarrow 1 S 1 \rightarrow 1 S S 1 S 1 \rightarrow 11011$

Exercise E5.1: Prove that the above grammar generates all and only palindromes over \{0, 1\}.

Hint: use induction on the length of the derivation.

Example: Natural language generation

Sentence $\rightarrow$ NounPhrase VerbPhrase

NounPhrase $\rightarrow$ Adjective NounPhrase

NounPhrase $\rightarrow$ Noun

Noun $\rightarrow$ car

Noun $\rightarrow$ train

Adjective $\rightarrow$ big

Adjective $\rightarrow$ broken
Notation:

1) To denote the set of productions

\[ \alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \ldots, \alpha \rightarrow \beta_n \]

we use

\[ \alpha \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n \]

2) We use \( V = V_N \cup V_T \)

A production of the form \( \alpha \rightarrow \epsilon \), with \( \epsilon \in V^* = V_N \cup V_T \)

is called \( \epsilon \)-production.

Example:

\[ L_{eq} = \{ w \in \{0,1\}^* \mid w \text{ has equal number of 0's and 1's} \} \]

We have already seen that this language is not regular.

Idea is to define \( G_{eq} \) s.t. \( L(G_{eq}) = L_{eq} \) by induction

Base: \( \epsilon \in w \in L_{eq} \)

induction: if \( w_A \) has one more 1 than 0, then \( 0w_A \in L_{eq} \)

if \( w_B \) has one more 0 than 1, then \( 1w_B \in L_{eq} \)

Characterize also languages for \( w_A \) and \( w_B \) inductively.

Grammar \( G_{eq} = (\{S, A, B\}, \{0,1\}, \{\epsilon\}, \rho, S) \) with \( \rho \)

\[ S \rightarrow \epsilon | 0A | 1B \]

\[ A \rightarrow 1S | 0AA \]

\[ B \rightarrow 0S | 1BB \]

(A generates strings with one more 1 than 0.)

(B generates strings with one more 0 than 1.)

Exercise E5.2: Prove that \( L(G_{eq}) = L_{eq} \) (by induction)
Definition: Given $G$, the direct derivation for $G$ is the binary relation on $(V^* \cdot V^n \cdot V^*) \equiv V^*$ defined as follows:

$$(\psi \rightarrow \psi')$$ is in the relation if there are

$$\alpha \in V^*, \ \beta, \gamma, \delta \in V^*$$

such that $\psi = \beta \gamma \delta$, $\psi' = \beta \gamma \delta$ and $\alpha \rightarrow \beta \in P$.

We write $\psi \Rightarrow \psi'$ and say that $\psi'$ directly derives from $\psi$ by $G$.

Definition: We call derivation the reflexive, transitive closure of direct derivation. In other words: $\psi'$ derives from $\psi$ by $G$, written $\psi \Rightarrow^\ast \psi'$ if

a) $\psi = \psi'$, or

b) there are $\psi_1, \ldots, \psi_n \in V^*$ such that

$\psi_1 = \psi$, $\psi_n = \psi'$, and $\psi_i \Rightarrow \psi_{i+1}$, $\forall i$, $1 \leq i < n$

Definition: Given a grammar $G$, the language generated by $G$ is

$L(G) = \{w \in V_T^* \mid S \Rightarrow^\ast w\}$

Notice: words in $L(G)$ are constituted by terminals only.

Terminology:

- sentence: any word $w \in V_T^*$ s.t. $S \Rightarrow^\ast w$, i.e. $w \in L(G)$
- sentential form: any $\alpha \in V^* = (V_T \cup V_N)^*$ s.t. $S \Rightarrow \alpha$

Notation: terminals: $a, b, c, \ldots$

nonterminals: $A, B, C, \ldots$

strings of terminals: $w, x, y, z, k, j, 2$,

symbols of $V = V_T \cup V_N$: $K, Y, 2$, 

sentential forms: $\alpha, \beta, \gamma, \delta, \ldots$. 

Example: Productions for G_{eq}:

\[ S \rightarrow E \mid OA \mid 1B \]
\[ A \rightarrow 1S \mid 0AA \]
\[ B \rightarrow 0S \mid 1BB \]

derivation:

1) \( 001SA \Rightarrow 001AS \) \hspace{1cm} (using \( A \rightarrow 1S \))
2) \( 001SA \Rightarrow 001S \) \hspace{1cm} (using \( S \rightarrow E \))
3) \( 001SA \Rightarrow 001SA \) \hspace{1cm} (using (1) and (2))
4) \( S \Rightarrow 001110 \)

Example: Grammar for \( L_{eq} = \{ a^n b^m c^n \mid m \geq 0 \} \)

\( G_{eq} = ( \{ A, B, C, S \}, \{ a, b, c \}, P, S ) \)

with \( P \):

1) \( S \rightarrow aSBC \)
2) \( S \rightarrow aBC \)
3) \( CB \rightarrow BC \)
4) \( aB \rightarrow ab \)
5) \( bB \rightarrow bb \)
6) \( bC \rightarrow bc \)
7) \( CC \rightarrow cc \)

\( aaaa\underbrace{bbb\underbrace{ccc}}_{\text{m}} \)

delete the rightmost \( C \) since \( S \rightarrow aSBC \)

\[ S \Rightarrow aSBC \Rightarrow aSBCBC \Rightarrow aSCBCBC \]

Example of derivation of \( aaaa\overbrace{bbb\underbrace{ccc}}^{m} \):

\[ S \Rightarrow aSBC \Rightarrow aSBCBC \Rightarrow aSCBCBC \]
\[ B \Rightarrow b \]
\[ C \Rightarrow c \]

Note: we cannot simply have \( B \Rightarrow b \), \( C \Rightarrow c \) because this would generate

Example of derivation of \( aaaa\overbrace{bbb\underbrace{ccc}}^{m} \):

\[ S \Rightarrow aSBC \Rightarrow aSBCBC \Rightarrow aSCBCBC \]
\[ B \Rightarrow b \]
\[ C \Rightarrow c \]

because this would generate

Example of derivation of \( aaaa\overbrace{bbb\underbrace{ccc}}^{m} \):

\[ S \Rightarrow aSBC \Rightarrow aSBCBC \Rightarrow aSCBCBC \]
\[ B \Rightarrow b \]
\[ C \Rightarrow c \]

because this would generate
Note: not each sequence of direct derivations leads to a sentence in $L(G_{3n})$

E.g. with the previous grammar we could generate

$$S \Rightarrow \varepsilon SBC \Rightarrow \varepsilon \varepsilon SBCBC \Rightarrow \varepsilon \varepsilon \varepsilon BCBCBC$$
$$\Rightarrow \varepsilon \varepsilon \varepsilon BCBBCC \Rightarrow \varepsilon \varepsilon \varepsilon BCBBC$$
$$\Rightarrow \varepsilon \varepsilon \varepsilon BCBCC$$
we cannot apply any other production

Also, the application of productions could go on forever

(eg. rule 1 in the previous example)

Classification of Chomsky grammars into 4 groups, depending on the form of the productions:

- grammars of type 0 : no limitations
- ---- 1 : context-sensitive
- ---- 2 : context-free
- ---- 3 : regular (or right linear)

Definition: grammars of type 0.
Productions have the most general form $\alpha \rightarrow \beta$, with

$$\alpha \in V \cup V^* \cup V^* \quad \beta \in V^*$$

Grammars of type 0 allow for derivations that shorten the sentential form:

A language generated by a grammar of type 0 is called of type 0.
Definition: grammar of type 1, or context-sensitive
Productions have the form \( A \rightarrow \beta \), with
\[ \alpha \in V^* \cdot V_N \cdot V^* \quad \beta \in V^+ \quad |\alpha| \leq |\beta| \]
These productions cannot shorten the length of the sentential form to which they are applied.
A language generated by a grammar of type 1 is called of type 1, or context-sensitive.

Example: \( G_{3m} \) is context-sensitive. Obviously, it is also of type 0.

Definition: grammar of type 2, or context-free
Productions have the form \( A \rightarrow \beta \), with \( A \in V_N \), \( \beta \in V^+ \).
These productions are productions of type 1, with the additional requirement that on the left there is a single nonterminal.
A language generated by a grammar of type 2 is called of type 2, or context-free.

Example \( L_{3m} = \{ a^n b^n \mid n > 1 \} \) is of type 1, since the following grammar \( G_{3m} \) generates \( L_{3m} \):
\[
S \rightarrow aB \mid SAB \\
B \bar{A} \rightarrow AB \\
aA \rightarrow aA \\
\alphaB \rightarrow \alpha \bar{B} \\
bB \rightarrow bb
\]
\( L_{3m} \) is also of type 2, since it is generated by
\[
S \rightarrow aSB \mid \epsilon \bar{b}
\]
We said that grammars of type 1 are also called context-sensitive (in contrast to context-free grammars). This is justified by the original definition by Chomsky for context-sensitive grammars:

**Definition:** Chomsky CS-grammar

Productions have the form \( \gamma_1 A \gamma_2 \rightarrow \gamma_0 \beta \gamma_2 \)

with \( \gamma_1, \gamma_2 \in V^*, \ \alpha \in V_N, \ \beta \in V^+ \)

Intuitively, \( A \) is replaced by \( \beta \) only if it appears "in the context" of \( \gamma_1 \) and \( \gamma_2 \)

**Theorem:** Grammars of type 1 and Chomsky CS grammars generate the same class of languages

**Proof:** We show that for every language \( L \):

There is a type-1 grammar \( G_1 \) s.t. \( L = L(G_1) \) iff there is a Chomsky CS grammar \( G_C \) s.t. \( L = L(G_C) \)

"if" immediate, since each Chomsky CS grammar is of type 1 (in \( \gamma_1 A \gamma_2 \rightarrow \gamma_0 \beta \gamma_2 \), we have \( \beta \in V^+ \) and hence \( |\gamma_1 A \gamma_2| \leq |\gamma_0 \beta \gamma_2| \))

"only-if" let \( G_1 \) be a type-1 grammar for \( L \).

We construct from \( G_1 \) a Chomsky CS grammar \( G_C \) as follows:

1) for each \( \alpha \in V_1 \), add a new nonterminal \( N_\alpha \);
2) replace in each production of \( G_1 \), each \( \alpha \in V_1 \) by \( N_\alpha \)

Now all productions have the form

\[ A_1 A_2 \ldots A_m \rightarrow B_1 B_2 \ldots B_n \] with \( m \leq n \)

and all \( A_i, B_j \in V_N \).
3) For each such production $A_1 \cdots A_m \rightarrow B_1 \cdots B_n$, introduce a new nonterminal $N$, and replace the production by the following ones:

$$A_1 A_2 \cdots A_m \rightarrow NA_2 \cdots A_m$$
$$NA_2 \cdots A_m \rightarrow NB_1 A_3 \cdots A_m$$
$$NB_2 A_3 \cdots A_m \rightarrow NB_2 B_3 A_4 \cdots A_m$$

\[ \vdots \]
$$NB_{n-1} A_m \rightarrow NB_2 \cdots B_{n-2} B_m \cdots B_n$$
$$NB_2 \cdots B_m \rightarrow B_1 B_2 \cdots B_n$$

Observe that all such productions are of the form $\gamma_1 A \gamma_2 \rightarrow \gamma_1 \beta \gamma_2$ with $\gamma_1, \gamma_2 \in V^*$, $\alpha \in V_N$, $\beta \in V_T$

4) For each $\alpha \in V_T$, add the production

$$N_\alpha \rightarrow \alpha$$

(where $N_\alpha$ is the new nonterminal associated to $\alpha$)

It is not difficult to see that $L(G_1) = L(G_\alpha)$

(the proof is by induction on the length of the derivation of a string $w \in L(G_1)$ (resp., $L(G_\alpha)$))
Definition: grammar of type 3, or regular, or right linear

Productions have the form \( A \rightarrow S \) with \( A \in V_N \) and \( S \in V_T \cup (V_T \circ V_N) \)

(i.e., \( A \rightarrow aB \) or \( A \rightarrow a \), with \( A, B \in V_N \), \( a \in V_T \))

A language generated by a grammar of type 3 is called of type 3 or regular.

Example: \( \{ a^m b^m | m \geq 0 \} \) is of type 3, since it is generated by the grammar

\[
S \rightarrow aS \\
S \rightarrow b
\]

Note: a grammar of type 3 is called linear, because on the right hand side of a production there is at most one non-terminal. It is called right-linear because the non-terminal is on the right of the terminal.

Exercise: E5.3:
Show that grammars of type 3 generate the class of regular languages that do not contain \( \varepsilon \).

Hint: given \( G = (V_N, V_T, P, S) \), construct an NFA

\( A_G = (V_N \cup \{ F \}, V_T, \delta, S, \{ F \}) \) with

\[
\delta(A, a) = B \quad \text{if} \quad A \rightarrow aB \quad \text{and} \\
\delta(A, a) = F \quad \text{if} \quad A \rightarrow a
\]

Show by induction on \( |w| \) that \( w \in S(A_G) \) iff \( w \in L(G) \).

Conversely, given an NFA \( A \), construct a grammar \( G_A \)
by having again non-terminals correspond to states of \( A \).
Note on $E$-productions (for grammars of type 1, 2, 3)

As we have defined them, grammars of type 1 (resp. 2, 3) cannot generate the empty string $E$.

We could extend the definition by allowing also the generation of $E$:

- If the start symbol $S$ does not appear on the right-hand side of productions, we allow also for a production $S \rightarrow E$ (E-production).

- If the start symbol $S$ appears on the right-hand side of productions, we introduce a new non-terminal $S_{new}$, make it the new start symbol, add a production $S_{new} \rightarrow S$, and allow for $S_{new} \rightarrow E$.

Hence, an $E$-production used just to generate $E$ is harmless.

Note that, allowing for $E$-productions for every non-terminal is not that harmless.

**Exercise:** E5.4: Show that, for every language $L$ of type 0 there is a grammar of type 1 extended with $E$-productions on arbitrary non-terminals that generates $L$.

*Hint:* Introduce a new non-terminal $N_{E}$ that is eliminated through an $E$-production $N_{E} \rightarrow E$, and use $N_{E}$ to make the right-hand side of productions as long as the left-hand side.
In a CFG, the productions have the form $A \rightarrow \beta$ with $A \in V_N$, $\beta \in V^*$. (Note: we allow for $\epsilon$-productions.)

**Example:** CFG for arithmetic expressions over variables $i$

$$G = (\{E, T, F\}, \{i, +, *, (, )\}, P, E),$$

where $P$ is

- $E \rightarrow T \mid E + T$
- $T \rightarrow F \mid T * F$
- $F \rightarrow i \mid (E)$

This grammar generates, e.g., $i + i * i$

$$E \rightarrow E + T \rightarrow T + T \rightarrow F + T \rightarrow i + T \rightarrow i + T * F \rightarrow i + i * E \rightarrow i + i * i$$

We can also represent a derivation of a string by a CFG by means of a tree, called parse tree.

As a tree whose nodes are labeled by elements of $V_0 \cup \{E, \Sigma\}$ satisfying:

1) each interior node is labeled by a non-terminal
2) each leaf is labeled by a non-terminal, a terminal, or $E$. If it is labeled by $E$, then it is the only child of its parent.
3) If an interior node is labeled $A$, and its children from left to right are labeled $X_1, X_2, \ldots, X_n$, then there is a production $A \rightarrow X_1 X_2 \cdots X_n \in P$.

**Example:** Parse tree for $i + i * i$
We call A-tree a subtree of the parse tree rooted at non-terminal A.

Yield (or frontier) of a tree:

is the sequence of labels of the leaves from left to right.

Example:

```
  E
 /   \
+-----+
|     |
| T x F |
```

Theorem: \( \alpha \in V^* \) is the yield of an A-tree \( \Rightarrow A \Rightarrow^* \alpha \)

Proof: by induction on the height of the tree

(see textbook)

Note: a parse tree does not specify a unique way to derive \( \alpha \) from A. (The order in which non-terminals are expanded is not specified).

The parse tree specifies, however, which rule is applied for each non-terminal.

Specific derivation orders:
- leftmost derivation: obtained by traversing the tree depth-first, by first going to the left subtree, and then to the right one.

  \[ E \Rightarrow^* E + T \Rightarrow^* T + T \Rightarrow E + T \Rightarrow i + T \Rightarrow \ldots \]

- rightmost derivation: defined similarly.
  \[ E \Rightarrow^* E + T \Rightarrow E + T \times F \]
Theorem: The following are all equivalent statements for a CFG $G = (V, T, P, S)$ and a string $w \in T^*$:

1. $w \in L(G)$ (or $S \Rightarrow^* w$)
2. $S \Rightarrow^* w$
3. $S \Rightarrow^*_m w$
4. There exists an $S$-tree with yield $w$.

Proof: The equivalence of (1) and (4) follows from the previous theorem. The other equivalences are obvious.

Thus, we could always use $\text{lm}$-derivation as a canonical way to derive any $w \in L(G)$; i.e., as a canonical way to interpret a parse tree for $w$.

Ambiguous grammars:

A $w \in L(G)$ could have two distinct parse trees, and hence two distinct $\text{lm}$-derivations.

Example: another grammar for arithmetic expressions

$$E \rightarrow i \mid (E) \mid E + E \mid E \times E$$

$$w = i + i \times i$$

These parse trees correspond to two different $\text{lm}$-derivations, and also to two ways of interpreting $w$. 
Definition: A CFG $G$ is ambiguous if for some $w \in L(G)$ there exist two distinct parse trees.

Ambiguity has to be avoided in compilers, since it corresponds to different ways of interpreting string.

Sometimes grammar can be redesigned to remove ambiguity. (e.g., for arithmetic expressions)

This is not always possible:

Definition: A CF language is (inherently) ambiguous if all its grammars are ambiguous.

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

$L$ is CF (show for exercise)

Consider strings of the form $a^n b^n c^n d^n$. We cannot tell whether they come from first or second types of strings in $L$, and any CFG must allow for both possibilities.