**Context-Free Grammars.**

### Context-Free Grammars

A \*\*Context-Free Grammar (CFG)\*\* is a type of formal grammar that is used to define the syntax of programming languages and artificial languages. A CFG consists of a set of production rules that describe how to form strings from the language's alphabet that are valid according to the language's syntax. Formally, a CFG is defined as a 4-tuple \( G = (V, T, P, S) \) where:

- \( V \) is a finite set of variables (non-terminals).

- \( T \) is a finite set of terminals (alphabet).

- \( P \) is a finite set of production rules of the form \( A \to \alpha \), where \( A \) is a non-terminal and \( \alpha \) is a string consisting of terminals and/or non-terminals.

- \( S \) is the start symbol, a special non-terminal from which production starts.

### Derivations Using a Grammar

A \*\*derivation\*\* in a CFG is a sequence of production rules applied to transform the start symbol \( S \) into a string of terminals. Each step in the derivation replaces a non-terminal with the right-hand side of a production rule. For example, given a CFG with the rules:

- \( S \to ASB \ | \ c \)

- \( A \to \epsilon \ | \ aA \)

- \( B \to \epsilon \ | \ bB \)

The string "acb" can be derived as follows:

1. \( S \to ASB \)

2. \( ASB \to aASB \)

3. \( aASB \to aSB \)

4. \( aSB \to acB \)

5. \( acB \to acbB \)

6. \( acbB \to acb \)

### Leftmost and Rightmost Derivations

\*\*Leftmost derivation\*\* always replaces the leftmost non-terminal first. For example:

- \( S \to\_{\text{lm}} ASB \to\_{\text{lm}} aASB \to\_{\text{lm}} aSB \to\_{\text{lm}} acB \to\_{\text{lm}} acbB \to\_{\text{lm}} acb \)

\*\*Rightmost derivation\*\* always replaces the rightmost non-terminal first. For example:

- \( S \to\_{\text{rm}} ASB \to\_{\text{rm}} ASbB \to\_{\text{rm}} ASb \to\_{\text{rm}} Acb \to\_{\text{rm}} aAcb \to\_{\text{rm}} acb \)

### The Language of a Grammar

The language generated by a CFG \( G \), denoted as \( L(G) \), is the set of all strings composed of terminal symbols that can be derived from the start symbol \( S \). Formally, \( L(G) = \{ w \in T^\* \ : \ S \to^\* w \} \).

### Sentential Forms

A \*\*sentential form\*\* is any string derived from the start symbol \( S \) that contains both terminals and non-terminals. If \( S \to^\* \alpha \), then \( \alpha \) is a sentential form. \*\*Left-sentential forms\*\* are derived using only leftmost derivations, and \*\*right-sentential forms\*\* are derived using only rightmost derivations.

### Parse Trees

A \*\*parse tree\*\* is a tree representation of the syntactic structure of a string according to a CFG. Each internal node is labeled with a non-terminal, and each leaf node is labeled with a terminal or \( \epsilon \). The root is the start symbol \( S \), and the children of an internal node correspond to the symbols in the production rule for that non-terminal.

### Ambiguity in Grammars and Languages

A CFG is \*\*ambiguous\*\* if there exists at least one string that can be generated by the grammar in more than one way (i.e., it has more than one distinct parse tree or derivation sequence). For example, the grammar:

- \( S \to SaS \ | \ b \)

is ambiguous because the string "babab" can have multiple parse trees.

Ambiguity can sometimes be resolved by rewriting the grammar to enforce precedence and associativity rules, but some languages are inherently ambiguous, meaning that any grammar that generates the language will be ambiguous. For instance, the language \( L = \{a^n b^n c^m d^m \ | \ n \ge 1, m \ge 1 \} \cup \{a^n b^m c^m d^n \ | \ n \ge 1, m \ge 1 \} \) is inherently ambiguous.

### Example and Parse Tree

Given the CFG:

- \( S \to ASB \ | \ c \)

- \( A \to \epsilon \ | \ aA \)

- \( B \to \epsilon \ | \ bB \)

The parse tree for "acb" derived from \( S \) is:

```

S

/ | \

A S B

\ | \

ε S B

| \

c B

\

b

```

### Applications of Context-Free Grammars

CFGs are widely used in the design and implementation of programming languages, natural language processing, and the development of parsers and compilers. They provide a framework for specifying the syntactic structure of languages, enabling the automation of syntax checking and translation processes.