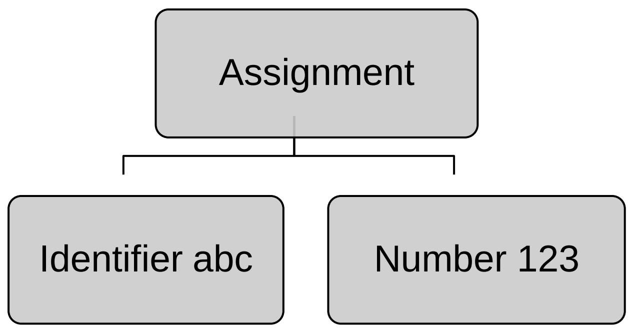
# Context Free Grammars

# What we want...

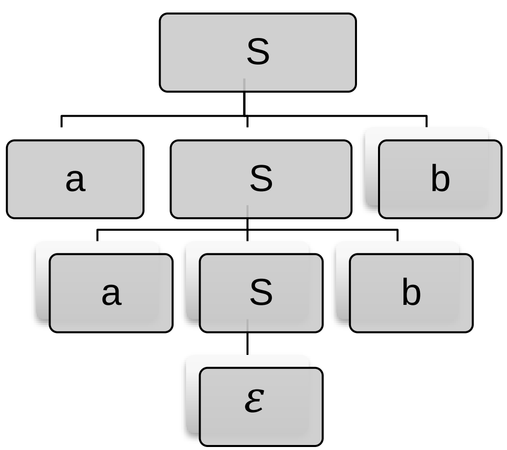
- We have seen how to create a lexer that transforms a source code over an alphabet $\Omega$ into a sequence of tokens, for example
  $\text{abc} = 123 \quad \rightarrow \quad$ *Identifier, AssignmentOperator, Number*
  (if we ignore the white spaces)
- In our next step, we want turn this sequence of tokens into a hierarchical structure called the syntax tree




- In the same way we used REs to describe valid sequences of characters (=lexems) and to build a lexer, we will now use *Context Free Grammars* to describe valid sequences of tokens and build a parser

# An Example

- The non-regular language over the alphabet $\{a, b\}$
$$\{\, a^n b^n \;\; | \, n \in \mathbb{N}\}$$
can be described by the following Context Free Grammar (CFG):
$$S \rightarrow a\; S\; b \,|\;\; \varepsilon$$
- How it works:
  1. Start with $S$ ($S$ is called a *non-terminal* symbol)
  2. $S$ can be derived to either $\varepsilon$ (= $a^0 b^0$) or to $a\; S\; b$
  3. $a\; S\; b$ can be derived to $a\; \varepsilon\; b$ (= $a^1 b^1 = ab$) or to $a\; a\; S\; b\; b$
  4. $a\; a\; S\; b\; b$ can be derived to $a\; a\; \varepsilon\; b\; b$ ($= a^2 b^2$) or to ...
- The derivation of $a\; a\; b\; b$ can be represented as a syntax tree



# Definition

- A CFG $G =< \Sigma, N, P, S >$ is defined by
  - An alphabet $\Sigma$ of terminal symbols
  - A set $N$ of non-terminal symbols (disjoint from $\Sigma$)
  - A set $P$ of production rules of the form $A \to \alpha$ with
    - $A \in N$
    - $\alpha \in X^*$ for $X = N \cup \Sigma$
  - A start symbol $S \in N$
- In our example $S \to a\ S\ b \mid \varepsilon$ , our CFG has
  - Terminal symbols $\Sigma = \{a, b\}$
  - Non-terminal symbol $N = \{S\}$
  - Two rules:

    $S \to a\ S\ b$

    $S \to \varepsilon$
  - Start symbol $S$

# Context Free Languages

- Derivation $\alpha \Rightarrow \beta$ for a CFG $G =< \Sigma, N, P, S >$:
  - A sequence of terminal and non-terminal symbols
$$\alpha = \alpha_1 A \alpha_2$$
    (where $A$ is a non-terminal symbol) can be derived to
$$\beta = \alpha_1 \gamma \alpha_2$$
    if there is a rule $A \rightarrow \gamma$. We write
$$\alpha \Rightarrow \beta$$
  - If $\alpha_1 \in \Sigma^*$, we write $\alpha \Rightarrow_l \beta$ (leftmost derivation)
  - If $\alpha_2 \in \Sigma^*$, we write $\alpha \Rightarrow_r \beta$ (rightmost derivation)
- The language $L(G)$ generated by CFG $G$ is given by:
$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$
  i.e., all $w \in L(G)$ can be obtained by starting at the start symbol $S$ and applying rules until the result only consists of terminal symbols
- We say that a language $L$ is context free if there is a CFG that generates it
- Note: $\{w \in \Sigma^* \mid S \Rightarrow^* w\} = \{w \in \Sigma^* \mid S \Rightarrow_l^* w\} = \{w \in \Sigma^* \mid S \Rightarrow_r^* w\}$

# Analysis

- The sequence of rules $r_1, r_2, \ldots$ that we apply to achieve $S \Rightarrow^* w$ for a given grammar is called the analysis of $w$
- $S \Rightarrow_l^* w$ gives the leftmost analysis
- $S \Rightarrow_r^* w$ gives the rightmost analysis

- In the following, we will, for a more compact representation, give each rule a number and write the analysis as a sequence of numbers
  - Example for $S \rightarrow a\ S\ b \mid \varepsilon$

    Rule 1: $S \rightarrow a\ S\ b$

    Rule 2: $S \rightarrow \varepsilon$

  Analysis of $aabb$: rule 1, rule 1, rule 2

# Bigger example

- A CFG for arithmetic expressions

$$E \to E + T \mid T \qquad \text{(rule 1 and rule 2)}$$
$$T \to T * F \mid F \qquad \text{(rules 3 and 4)}$$
$$F \to (E) \mid Number \mid Identifier \qquad \text{(rules 5, 6, and 7)}$$

- Leftmost derivation of $(89) * x$:

$$E \underset{l}{\overset{2}{\Rightarrow}} T \underset{l}{\overset{3}{\Rightarrow}} T * F \underset{l}{\overset{4}{\Rightarrow}} F * F \underset{l}{\overset{5}{\Rightarrow}} (E) * F \underset{l}{\overset{2}{\Rightarrow}} (T) * F$$
$$\underset{l}{\overset{4}{\Rightarrow}} (F) * F \underset{l}{\overset{6}{\Rightarrow}} (Number) * F \underset{l}{\overset{7}{\Rightarrow}} (Number) * Identifier$$

- Rightmost derivation of $(89) * x$:

$$E \underset{l}{\overset{2}{\Rightarrow}} T \underset{l}{\overset{3}{\Rightarrow}} T * F \underset{l}{\overset{7}{\Rightarrow}} T * Identifier \underset{l}{\overset{4}{\Rightarrow}} F * Identifier$$
$$\Rightarrow \cdots \Rightarrow (Number) * Identifier$$

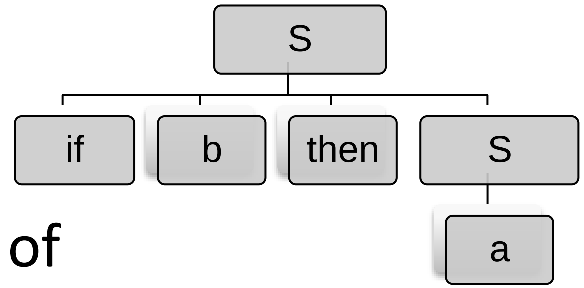
# Ambiguity

- Example: a programming language with if-then and if-then-else

$$S \rightarrow a \mid if\ \ b\ then\ S \mid if\ \ b\ then\ S\ else\ S$$

- Exercise: What is the syntax tree of

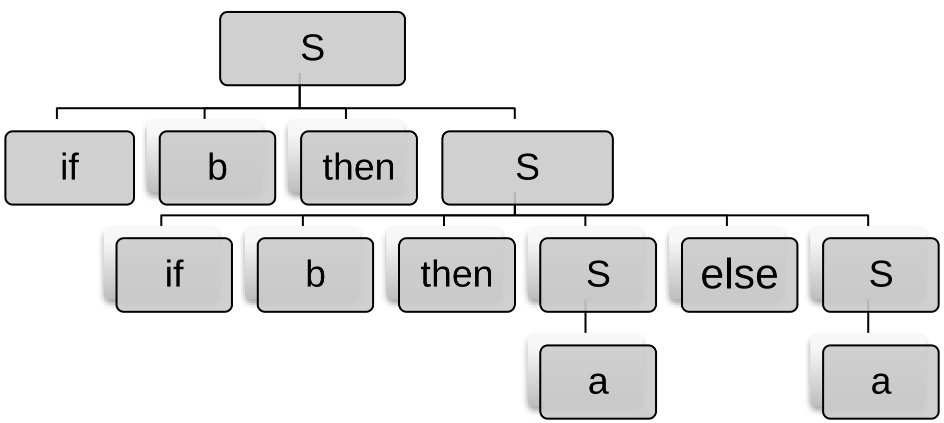
$$if\ \ b\ then\ a$$

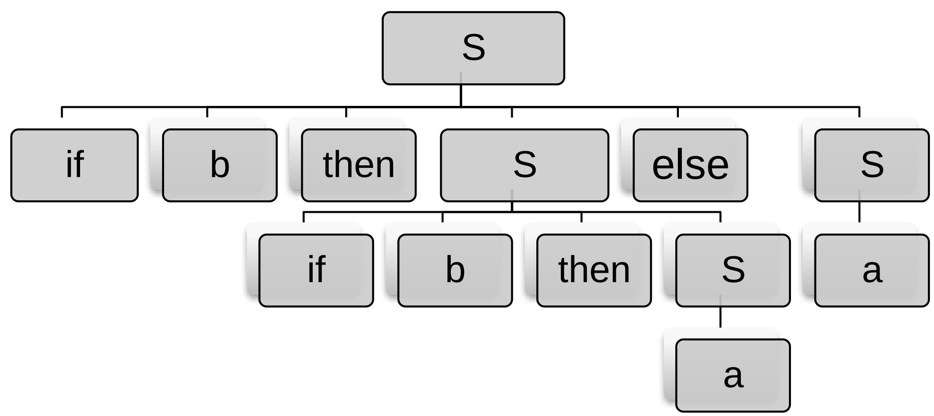



- Now, more difficult: What is the syntax tree of

$$if\ \ b\ then\ if\ \ b\ then\ a\ else\ a$$

- Two possible trees:




- To which "if" does the "else" belong? (*dangling else problem*)

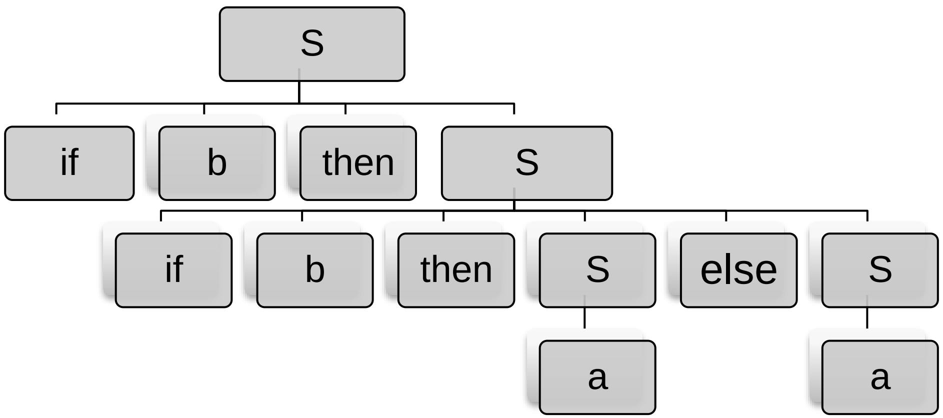
# Ambiguous vs Unambiguous Grammar

- Every syntax tree represents exactly one $w \in L(G)$
- Every syntax tree corresponds to exactly one leftmost derivation $S \Rightarrow_l^* w$ and vice versa
- Every syntax tree corresponds to exactly one rightmost derivation $S \Rightarrow_r^* w$ and vice versa
- But as shown on the previous slide, a $w \in L(G)$ can have several derivations and, therefore, several syntax trees
- A CFG $G$ is *unambiguous* if every $w \in L(G)$ has exactly one syntax tree. It is *ambiguous* otherwise.
- A language $L$ is *inherently ambiguous* if every $G$ with $L(G) = L$ is ambiguous

- In general, it is <u>undecidable</u> whether a CFG is ambiguous or not!
- However, given a CFG $G$ and $w \in \Sigma^*$, the problem $w \stackrel{?}{\in} L(G)$ is <u>decidable</u>

# Remark: Concrete Syntax Tree vs Abstract Syntax Tree

- In practice, a parser will return a “cleaned up” version of the syntax tree, called the Abstract Syntax Tree (AST)
- Usually, the AST is created by hand-written code during parsing
- Our if-then-else example:
  - Concrete Syntax Tree




  - “Clean” version (AST):