CFGs + CFLs

Miguel Ballesteros

Algorithms for NLP Course.
7-11

Today

- Introduction about Context Free Grammars (CFGs)
- Introduction about Context Free Languages (CFLs)

- This is essential material to understand parsing and to develop parsers.

Grammar

- What is a grammar?

Grammar

Def-1: the whole system and structure of a language or of languages in general, usually taken as consisting of syntax and morphology (including inflections) and sometimes also phonology and semantics.

Def-2: the study of the way the sentences of a language are constructed; morphology and syntax.

Def-3: the set of rules that explain how words are used in a language

Context Free Grammar

- a formal grammar in which every production rule is of the form

  $V \rightarrow w$

  - where V is a single nonterminal symbol, and w is a string of terminals and/or nonterminals (w can be empty).

- A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal.

  (A.K.A phrase-structure grammar)

Notation + Formal Definition

- CFG (**Phrase structure grammar**)
  - G = (V, T, P, S)

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - V: a finite set of variables, non-terminal symbols.

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - V : a finite set of variables, non-terminal symbols.
      - Elements of V are denoted by **A, B, C, …, S**

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - T : a finite set of terminal symbols (equiv. To Σ in FSAs)

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - T : a finite set of terminal symbols (equiv. To Σ in FSAs)
      - Terminals are denoted by **a, b, c...**
      - Strings over **T*** are denoted by **x, y, z, w**

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - P: a set of context free production rules, each of the form
      - $A \rightarrow \alpha$, where $A \varepsilon V$, $\alpha \varepsilon (V \cup T)^*$

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - P: a set of context free production rules, each of the form
      - $A \rightarrow \alpha$, where $A \varepsilon V$, $\alpha \epsilon (V \cup T)^*$
      - Strings over $(V \cup T)^*$ are denoted by $\boldsymbol{\alpha, \beta, \gamma}$

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - S: a start non-terminal S ϵ V

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - V: a finite set of variables, non-terminal symbols.
    - T: a finite set of terminal symbols (equiv. To Σ in FSAs)
    - P: a set of context free production rules, each of the form
      - $A \rightarrow \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$
    - S: a start non-terminal $S \in V$

Notation + Formal Definition

- CFG
  - G = (V, T, P, S)
    - V: a finite set of variables, non-terminal symbols.
    - T: a finite set of terminal symbols (equiv. To Σ in FSAs)
    - P: a set of context free production rules, each of the form
      - $A \rightarrow \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$
    - S: a start non-terminal $S \in V$
      - Single variables or terminals are denoted by **X, Y, Z**
- **Using all of this, a grammar can be specified simply by its production rules.**

Context Free Languages

- Languages generated by context free grammars are called “Context free languages”

Let's see an example.

CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

N → students

N → language

N → instructors

V → study

N → study

CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

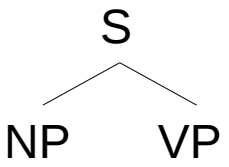
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

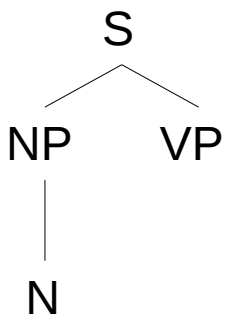
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

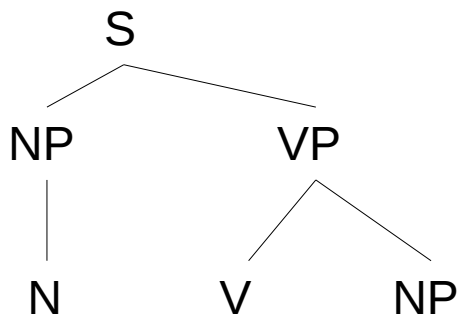
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

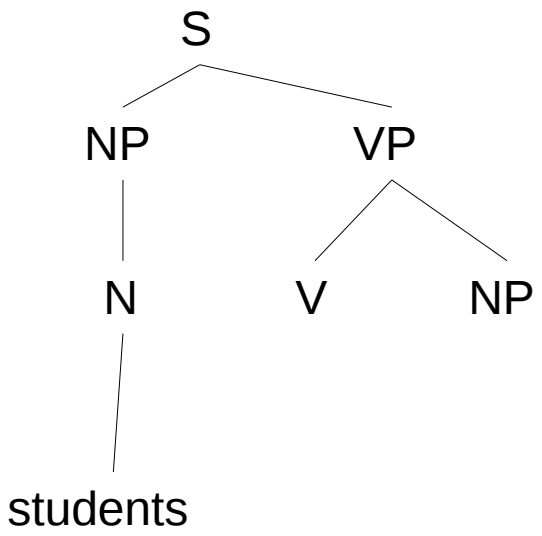
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

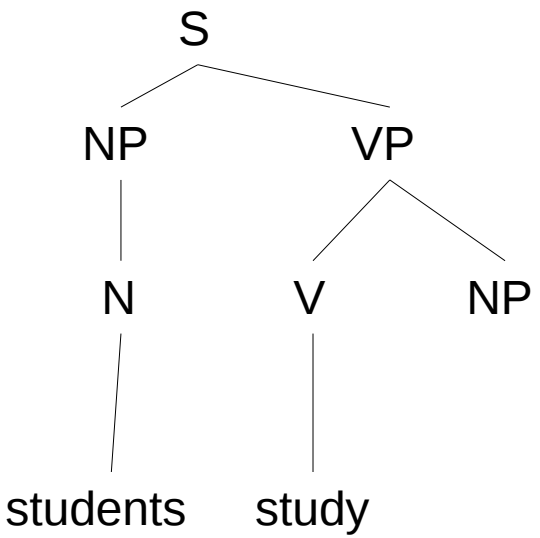
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

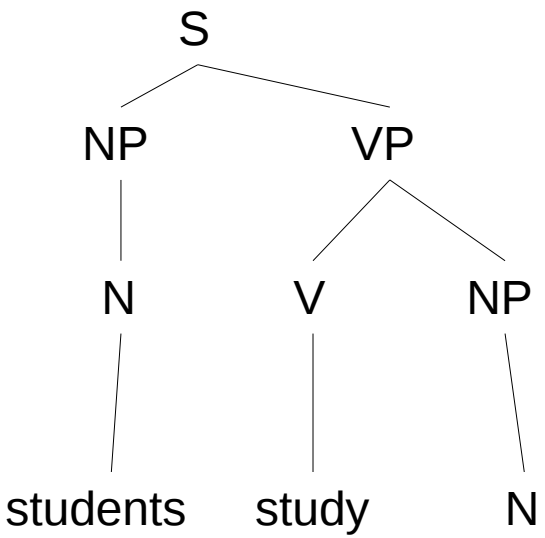
N → students

N → language

N → instructors

V → study

N → study



CFG

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → ?

PP → P NP

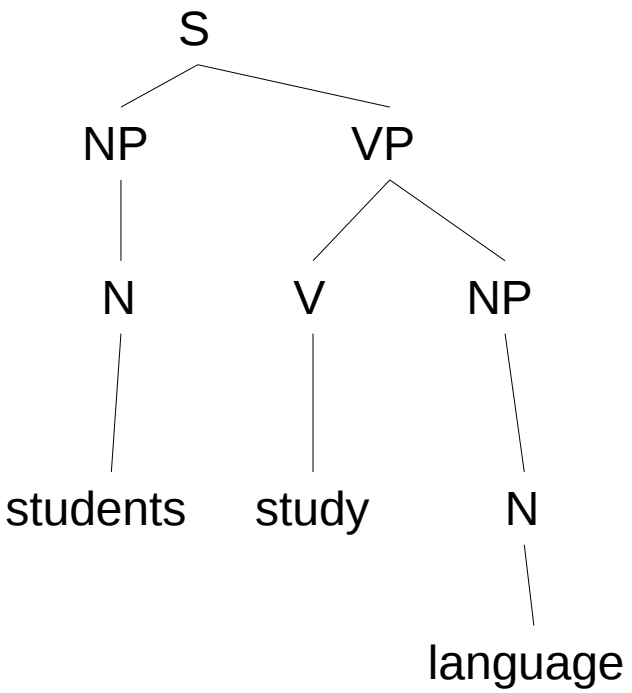
N → students

N → language

N → instructors

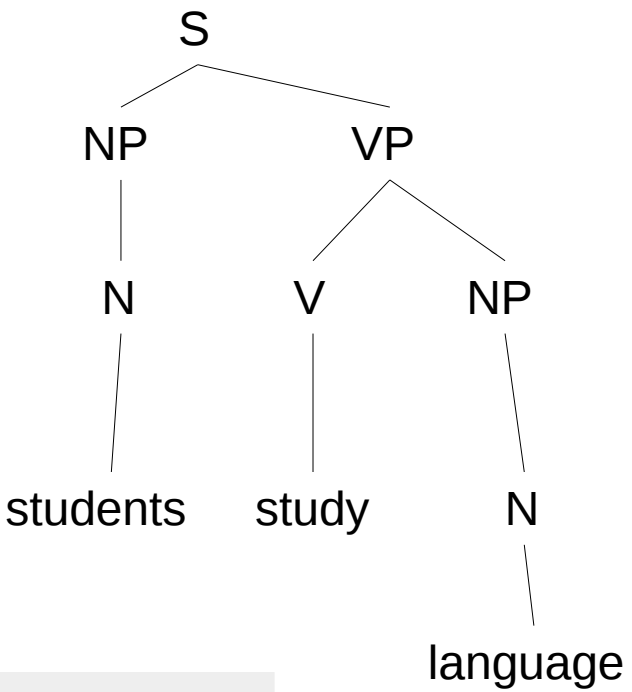
V → study

N → study



CFG

```
S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → ?
PP → P NP
N → students
N → language
N → instructors
V → study
N → study
```



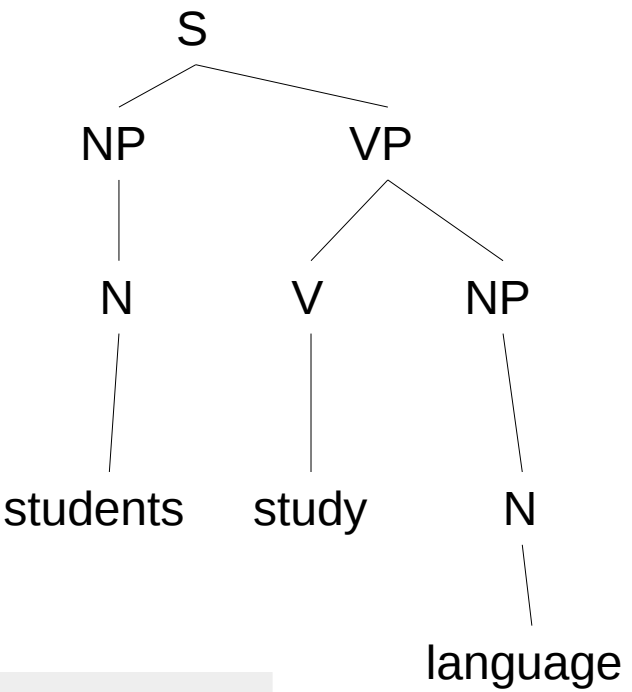
Other examples for this grammar:

- Students study language
- Instructors study students
- Students study study

...

CFG

```
S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → ?
PP → P NP
N → students
N → language
N → instructors
V → study
N → study
```



Other examples for this grammar:

- Students study language
- Instructors study students
- Students study study

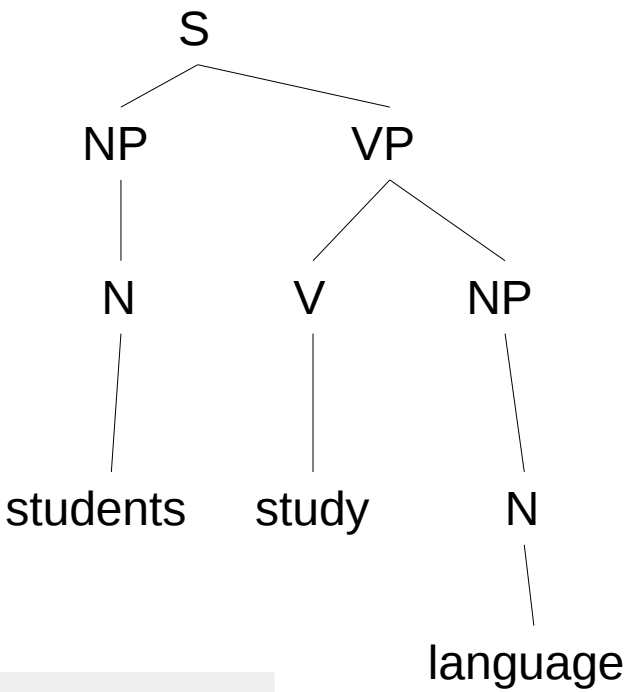
...

This would be the CFL

CFG

It seems like an ambiguous grammar!

```
S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → ?
PP → P NP
N → students
N → language
N → instructors
V → study
N → study
```



Other examples for this grammar:

- Students study language
- Instructors study students
- Students study study

...

Applications of CFG

- Parsers:
  - Many aspects of a programming language have a structure that may be described by regular expressions. But they are not enough...
  - Typical languages use parentheses and/or brackets in a nested fashion. We should be able to match left brackets with right brackets →
  - Example: (( ), ( )( ), (( ) ( ))

    This grammar will do that:
    - B → BB | (B) | ϵ

Applications of CFG

- YACC: yet another compiler compiler
  - Parser generator:
    - Input: a CFG.
    - Yacc turns such a specification into a subroutine that handles the input process.
    - It could be used to specify C and get parse trees of C language.

Applications of CFG

- Markup languages such as XML or *HTML*

<BODY>

<P> THIS IS A PARAGRAPH </P>

<OL>

<LI> LIST ITEM</LI>

<LI> ANOTHER LIST ITEM </LI>

</OL>

</BODY>

Element → Text | *<P> Element </p>* | *<OL> Listitem </OL>*

Listitem → <LI> Element </LI> | *<LI> Element </LI> ListItem*

CFG – Derivations (1)

1- Let $\varphi_1, \varphi_2 \in (V \cup T)^*$,

- φ_1 **derictly derives** φ_2 for a given G ($\varphi_1 \Rightarrow_G \varphi_2$)

  if $\varphi_1 = \alpha A \beta$, $\varphi_2 = \alpha \gamma \beta$,

  - for some $\alpha, \gamma, \beta \in (V \cup T)^*$
  - $A \in V$ and $A \rightarrow \gamma$ is a rule in P_G

CFG – Derivations (2)

2 - φ_1 **derives** φ_2 for a given G ($\varphi_1 =^*>_G \varphi_2$)

- If there exists a finite sequence of direct derivations such that

$$\varphi_1 =>_G \varphi'_1 =>_G \varphi'_2 =>_G \varphi'_3 =>_G \ldots =>_G \varphi'_k = \varphi_2$$

CFG – Derivations (3)

3 - $\varphi_1 \Rightarrow_G^i \varphi_2$

Means that φ_1 **derives** φ_2 in exactly i steps.

CFL of a CFG

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

CFL of a CFG (1)

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

A language L is <u>context-free</u> if there exists a grammar G, such that

$$L=L(G)$$

CFL of a CFG (2)

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

The set of all such languages is called CFLs

CFL of a CFG

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

CFL of a CFG (3)

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

A string $\alpha \in (V \cup T)^*$ is a sentencial form if

$S =^*>_G \alpha$

CFL of a CFG (4)

- The language of a CFG = L(G) is defined as:
  - $L(G) = \{ w \in T^* \mid S =^*>_G w \}$

A string $\alpha \in (V \cup T)^*$ is a sentencial form if

$S =^*>_G \alpha$

Example:

G:

$S \rightarrow a\ S\ b$

$S \rightarrow a\ b$

$L(G) = \{a^n b^n \mid n>=1\}$

Parse (Derivation) Trees

- A graphical representation of a derivation of a (terminal) string according to the grammar.

- Captures the grammatical structure of the (terminal) string.

Formal definition

- Let G be a CFG. G=(V, T, P, S)
- A tree is a parse tree if the following conditions hold:

  1- Each node (vertex) in the tree is labeled by a symbol x ϵ V U T U{S}

Formal definition

- Let G be a CFG. G=(V, T, P, S)
- A tree is a parse tree if the following conditions hold:

  1- Each node (vertex) in the tree is labeled by a symbol x ϵ V U T U{**S**}

  2- The root is labeled by **S**

Formal definition

- Let G be a CFG. G=(V, T, P, S)
- A tree is a parse tree if the following conditions hold:

  1- Each node (vertex) in the tree is labeled by a symbol x ϵ V U T U{**S**}

  2- The root is labeled by **S**

  3- Every internal node is labeled by a symbol A ϵ V

Formal definition

- Let G be a CFG. G=(V, T, P, S)
- A tree is a parse tree if the following conditions hold:

  1- Each node (vertex) in the tree is labeled by a symbol x ϵ V U T U{**S**}

  2- The root is labeled by **S**

  3- Every internal node is labeled by a symbol A ϵ V

  4- If a node n is internal, labeled by A and its children are $x_1, x_2, \ldots, x_k$ then $A \rightarrow x_1 x_2 \ldots x_k \in P_G$

Formal definition

- Let G be a CFG. G=(V, T, P, S)
- A tree is a parse tree if the following conditions hold:

  1- Each node (vertex) in the tree is labeled by a symbol x ϵ V U T U{**S**}

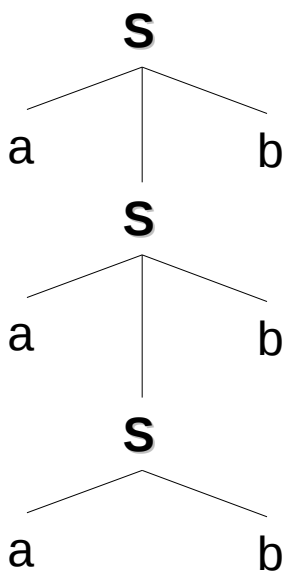
  2- The root is labeled by **S**

  3- Every internal node is labeled by a symbol A ϵ V

  4- If a node n is internal, labeled by A and its children are $x_1, x_2, \ldots, x_k$ then $A \rightarrow x_1 x_2 \ldots x_k \in P_G$

  5- If a node n is labeled **Є**, it is a leaf, and its the only son of its parent node.

Example



S → aSb

- Note that the order of the children in a tree is important.
- The grammar rule implies the left to right ordering of sibling nodes in a parse tree.

Simplifying a CFG

- It is often interesting/convenient to simplify a CFG.
- We can restrict the format of productions in P without reducing the generative power of the grammar:
  - If L(G) is not empty, we can make sure that

    1. Every variable and terminal are used in the derivation of some w ϵ L(G)
    2. There are no productions of the form A → B

    => No useless symbols in G

Normal forms of CFGs

- If Є ∈ L(G), then G can be simplified into either of the following two forms:
  - Chomsky Normal Form: CNF
  - Greibach Normal Form: GNF

Chomsky Normal Form

- Each production is of the form:

  (i) $A \rightarrow BC$

  (ii) $A \rightarrow a$

  - Where a is a terminal, A, B, C are nonterminals and B,C may not be start variable.

Conversion to CNF

- Given a G=(V, T, P, S) we do the following steps:

  1- Construct and equivalent grammar G_1 that contains no unit productions and no ϵ-rules.

  2- Rules of the form $A \rightarrow a$ are not modified.

  3- For any rule $A \rightarrow x_1x_2 \ldots x_m$ m >= 2, where some x_i is a terminal a, add a variable Ca and a rule
  Ca → a, and replace X_i with Ca in the original rule.

  4- For any rule $A \rightarrow B_1B_2 \ldots B_m$, m >= 3 we replace the rule with

  $A \rightarrow B_1 D_1$

  $D_1 \rightarrow B_2 D_2$

  …

  $D_{m-2} \rightarrow B_{m-1} D_m$

  Where $D_1, D_2, \ldots D_{m-2}$ are new variables (no terminals)

  The resulting grammar is in CNF.

Chomsky Normal Form

- Any context-free language is generated by a context-free grammar in CNF.
- Proof:
  - Conversion procedure has several stages where the rules that violate CNF conditions are replaced with equivalent rules that satisfy these conditions.

  (see previous slide).

Conversion Example

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid aS \mid a$

$B \rightarrow aBB \mid bS \mid b$

$S \rightarrow C_bA \mid C_aB$

$A \rightarrow C_bAA \mid C_aS \mid a$

$B \rightarrow C_aBB \mid C_bS \mid b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$S \rightarrow C_bA \mid C_aB$

$A \rightarrow C_bD1 \mid C_aS \mid a$

$D1 \rightarrow AA$

$B \rightarrow C_aD2 \mid C_bS \mid b$

$D2 \rightarrow BB$

$C_a \rightarrow a$

$C_b \rightarrow b$

Greibach Normal Form

- Each production is of the form

  $A \rightarrow a\ \alpha$, such that $\alpha \in V^*$

  Note that this could be

  - $A \rightarrow a\ A_1\ A_2\ A_3 ...$
  - $A \rightarrow a$

Summary

- **Context-free grammars**: A CFG is a way of describing languages by recursive rules called productions.
    - $B \rightarrow BB \mid (B) \mid \epsilon$
    - This one describes the language of parentheses or brackets, with words such as ( ) ( ) ( )

Summary

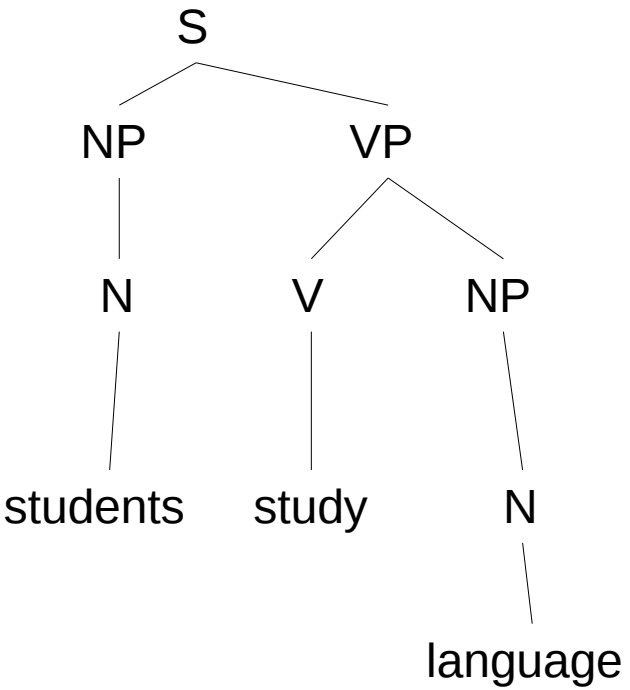
- **Derivations and languages**: Beginning with the start symbol, we derive terminal strings by replacing a variable by the body of some production.

  The language of the CFG is the set of terminal strings we can derive from the start symbol.

  Such a language is called **CFL: context-free-language.**

Summary

- **Parse trees**: a parse tree is a tree that shows the essentials of a derivation. Internal nodes are variables and the leaves are the terminal symbols.



Summary

- **Ambiguous grammar**: For some CFGs it is possible to find a terminal string with more than one parse tree. Such a grammar is an ambiguous grammar.

Further reading

- See Introduction to Automata Theory, Languages, and Computation, J.E, Hopcroft and J.D Ullman, Addison-Wesley 1979.