

# Computational Linguistics: Syntax I

RAFFAELLA BERNARDI

KRDB, FREE UNIVERSITY OF BOZEN-BOLZANO

P.ZZA DOMENICANI, ROOM: 2.28, E-MAIL: BERNARDI@INF.UNIBZ.IT

# Contents

1	Reminder	5
2	Syntax	7
3	Dependency	8
4	Long-distance Dependencies	9
	4.1 Relative Pronouns	10
	4.2 Coordination	11
5	Formal Approaches	12
6	Sentence Structures: English	13
	6.1 Exercises	14
7	Syntax Recognizer	15
	7.1 NLS are not RL: Example I	16
	7.2 NLS are not RL: Example II	17
8	FSA for syntactic analysis	19
9	Formal Grammar: Terminology	20
	9.1 Formal Grammars: Definition	21
	9.2 Derivations	22
	9.3 Formal Languages and FG	23

9.4	FG and Regular Languages	24
9.5	FSA and RG	25
10	Context Free Grammars	26
11	CFG: Formal Language	27
12	CFG: More derivations	28
12.1	CFG: Language Generated	29
13	FG for Natural Languages	30
14	PSG: English Toy Fragment	32
15	English Toy Fragment: Strings	33
16	English Toy Fragment: Phrase Structure Trees	34
17	Extending our grammar	35
18	Recursion	36
19	Summing up (I)	37
20	Summing up (II)	38
21	Overgeneration and Undergeneration	39
22	Undergeneration	41
22.1	Undergeneration (Cont'd)	42
23	Trasformational Grammar	43
23.1	Trasformational Grammars: Relative Clauses	44

24	Next Steps .....	45
----	------------------	----

# 1. Reminder

Main issues of last lecture:

- ▶ Different levels of Natural Language
  1. Phonology
  2. Morphology
  3. Syntax
  4. Semantics
  5. Discourse
  6. Pragmatics
  
- ▶ Linguistically motivated computational models. For any topic:
  1. Linguistic Theory
  2. Formal Analysis
  3. Implementation

▶ Linguistic Theories

1. Morphology: Stems vs. Affixes; Inflectional and derivational forms.
2. PoS: classes (categories) of words

▶ Natural Language as Formal Language

1. Morphology can be formalized by means of Regular Languages and as such modeled by FSA.
2. FSA cannot handle “Dependency” ( $a^n b^n$ , cannot count)

▶ Implementation

## 2. Syntax

- ▶ **Syntax:** “setting out things together”, in our case things are words. The main question addressed here is “*How do words compose together to form a grammatical sentence (s) (or fragments of it)?*”
- ▶ **Constituents:** Groups of categories may form a single *unit or phrase* called constituents. The main phrases are noun phrases (*np*), verb phrases (*vp*), prepositional phrases (*pp*). Noun phrases for instance are: “she”; “Michael”; “Rajeev Goré”; “the house”; “a young two-year child”.

Tests like substitution help decide whether words form constituents.

Another possible test is coordination.

### 3. Dependency

**Dependency:** Categories are interdependent, for example

Ryanair **services** [Pescara]<sub>np</sub>      Ryanair **flies** [to Pescara]<sub>pp</sub>  
\*Ryanair **services** [to Pescara]<sub>pp</sub>      \*Ryanair **flies** [Pescara]<sub>np</sub>

the verbs **services** and **flies** determine which category can/must be juxtaposed. If their constraints are not satisfied the structure is **ungrammatical**.



## 4. Long-distance Dependencies

Interdependent constituents need not be juxtaposed, but may form long-distance dependencies, manifested by **gaps**

- ▶ **What cities** does Ryanair **service** [...]?

The constituent **what cities** depends on the verb **service**, but is at the front of the sentence rather than at the **object position**.

Such distance can be large,

- ▶ **Which flight** do you want me to **book** [...]?
- ▶ **Which flight** do you want me to have the travel agent **book** [...]?
- ▶ **Which flight** do you want me to have the travel agent nearby my office **book** [...]?

## 4.1. Relative Pronouns

**Relative Pronoun** (eg. who, which): they function as e.g. the **subject** or **object** of the **verb** embedded in the relative clause (*rc*),

- ▶ [[the [student [who [...] knows Sara]<sub>rc</sub>]<sub>n</sub>]<sub>np</sub> [left]<sub>v</sub>]<sub>s</sub>.
- ▶ [[the [book [which Sara wrote [...] ]<sub>rc</sub>]<sub>n</sub>]<sub>np</sub> [is interesting]<sub>v</sub>]<sub>s</sub>.

Can you think of another relative pronoun?

## 4.2. Coordination

**Coordination:** Expressions of the **same** syntactic category can be coordinated via “and”, “or”, “but” to form more **complex phrases** of the **same category**. For instance, a **coordinated verb phrase** can consist of two other verb phrases separated by a conjunction:

- ▶ There are no flights [[leaving Denver]<sub>vp</sub> and [arriving in San Francisco]<sub>vp</sub>]<sub>vp</sub>

The conjuncted expressions belong to traditional constituent classes, *vp*. However, we could also have

- ▶ I [[[want to try to write [...]] and [hope to see produced [...]]] [the movie]<sub>np</sub>]<sub>vp</sub>”

Again, the interdependent constituents are disconnected from each other.

**Long-distance dependencies** are **challenging phenomena** for formal approaches to natural language analysis. We will study them in the second part of the course (May).

## 5. Formal Approaches

To examine how the syntax of a sentence can be computed, you must consider two things:

1. **The grammar**: A formal specification of the structures allowable in the language. [Data structures]
2. **The parsing technique**: The method of analyzing a sentence to determine its structure according to the grammar. [Algorithm]

## 6. Sentence Structures: English

The structure of a sentences can be represented in several ways, the most common are the following notations: (i) brackets or (ii) trees. For instance, “John ate the cat” is a sentence (s) consisting of noun phrase (np) and a verb phrase (vp). The noun phrase is composed of a verb (v) “ate” and an np, which consists of an article (art) “the” and a common noun (n) “cat”.

$$[\text{John}_{np} [\text{ate}_v [\text{the}_{art} \text{cat}_n]_{np}]_{vp}]_s$$

Give the tree representation of this structure.

## 6.1. Exercises

Now represent in the format you prefer the sentences below:

I like a red shirt

I will leave Boston in the morning.

John saw the man with the telescope.

John thinks someone left.

## 7. Syntax Recognizer

In lecture 1, we have used FSA to recognize/generate natural language morphology, and FSA can be used to concatenate words, i.e. at the syntactic level.

We have said that FSA recognize/generate “Regular Languages”. But it has been shown that at the **syntactic level NLS are not regular**.

## 7.1. NLS are not RL: Example I

1. The cat died.
2. The cat the dog chased died.
3. The cat the dog the rat bit chased died.
4. ...

Let, determiner+noun be in the set  $A : \{ \text{the cat, the dog, ...} \}$ , and the transitive verbs in  $B : \{ \text{chased, bit, ...} \}$ . Thus the strings illustrated above are all of the form:

$x^n y^{n-1}$  died, where  $x \in A$  and  $y \in B$ , which can be proved to be not a RL.



## 7.2. NLS are not RL: Example II

Another evidence was provided by Chomsky in 1956. Let  $S_1, S_2, \dots, S_n$  be declarative sentences, the following syntactic structures are grammatical English sentences:

- ▶ If  $S_1$ , then  $S_2$
- ▶ Either  $S_3$ , or  $S_4$
- ▶ The man who said  $S_5$  is arriving today

In each case there is a lexical dependency between one part of each structure and another. “If” must be followed by “then” “either” must be followed by “or”.

Moreover, these sentences can be embedded in English one in another.

**If either** the man who said  $S_5$  is arriving today **or** the man who said  $S_5$  is arriving tomorrow, **then** the man who said  $S_6$  is arriving the day after.  
Let

if	$\rightarrow a$
then	$\rightarrow a$
either	$\rightarrow b$
or	$\rightarrow b$
<b>other words</b>	$\rightarrow \epsilon$

The sentence above would be represented as  $abba$ .

This structure of nested dependencies can be represented more generally by a language like  $xx^R$  with  $x \in \{a, b\}^*$  and  $R$  denoting the reversal of the string  $x$ . (Eg.  $abbababba$ ) We can prove via the Pumping Lemma that this language is not in a regular language.

Again, this is an example of open and closed balanced parentheses (or **nested dependencies**) that are not in RL.

## 8. FSA for syntactic analysis

Finite state methods have been applied to syntactic analysis too. Although they are not expressive enough if a full syntactic analysis is required, there are many applications where a partial syntactic analysis of the input is sufficient.

Such partial analyses can be constructed with cascades of finite state automata (or rather **transducers**) where one machine is applied to the output of another.

Anyway, in order to deal with syntactic analysis of natural language we need **a more powerful device than FSA** (and of their corresponding formal grammars, namely regular (or right linear) grammar (RG).)

## 9. Formal Grammar: Terminology

Formal Grammars are string **re-write systems**. The re-write rules say that a certain sequence of symbols may be substituted by another sequence of symbols. These symbols are divided into two classes:

- ▶ **terminal**: symbols that will appear in the string of the language generated by the grammar.
- ▶ **non-terminal**: symbols that will be used only in the re-write process.

## 9.1. Formal Grammars: Definition

A Formal Grammar (FG) is a formalism to give a finite representation of a Language.

A Grammar,  $G$ , is a tuple:  $G = (V_T, V_N, S, P)$ , such that:

- ▶  $V_T$  is the finite set of Terminal Symbols.
- ▶  $V_N$  is the finite set of Non-Terminal Symbols.
- ▶ Terminal and Non-Terminal symbols give rise to the alphabet:  $V = V_T \cup V_N$ .
- ▶ Terminal and Non-Terminal symbols are disjoint sets:  $V_T \cap V_N = \{\}$ .
- ▶  $S$  is the start symbol (Scope) of the Language, and  $S \in V_N$ .
- ▶  $P$  is the finite set of Productions,  $P = \{\alpha \rightarrow \beta \mid \alpha \in V^+ \wedge \beta \in V^*\}$ .

## 9.2. Derivations

To characterize a Language starting from a Grammar we need to introduce the notion of Derivation.

- ▶ The notion of Derivation uses Productions to generate a string starting from the Start symbol  $S$ .
- ▶ Direct Derivation (in symbols  $\Rightarrow$ ). If  $\alpha \rightarrow \beta \in P$  and  $\gamma, \delta \in V^*$ , then  $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ .
- ▶ Derivation (in symbols  $\Rightarrow^*$ ). If  $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \dots, \alpha_{n-1} \Rightarrow \alpha_n$ , then  $\alpha_1 \Rightarrow^* \alpha_n$ .

## 9.3. Formal Languages and FG

A string belongs to a Language if and only if:

1. The string is made only of Terminal Symbols;
2. The string can be Derived from the Start Symbol,  $S$ , of the Language.

**Generative Definition of a Language** We say that a Language  $L$  is **generated** by the Grammar  $G$ , in symbols  $L(G)$ , if:

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

## 9.4. FG and Regular Languages

We have said that the languages generated/recognized by a FSA are called “Regular Languages”. The formal grammars that generate/recognize these languages are known as “Regular Grammar” (RG) or Right Linear Grammars. (or Left Linear Grammar).

Regular Grammars have rules of the form:

$$\blacktriangleright A \rightarrow xB$$

$$\blacktriangleright A \rightarrow x$$

where  $A$  and  $B$  are non-terminal symbols and  $x$  is any string of terminals (possibly empty). Moreover, a rule of the form:  $S \rightarrow \epsilon$  is allowed if  $S$  does not appear on the right side of any rule.



## 9.5. FSA and RG

The association between FSA and RG is straight:

RG	FSA
$A \rightarrow xB$	from state $A$ to state $B$ reading $x$
$A \rightarrow x$	from state $A$ reading $x$ to a designed final state.
start symbol	initial state.

As in FSA, the string already generated/recognized by the grammar has no influence on the strings to be read in the future (no memory!).

See Artale's Course "Compiler" for more details.

## 10. Context Free Grammars

Formal Grammar **more powerful** than Regular Grammars are Context Free Grammars (CFG).

These grammars are called **context free** because all rules contain only one symbol on the left hand side — and wherever we see that symbol while doing a derivation, we are free to replace it with the stuff on the right hand side. That is, the ‘context’ in which a symbol on the left hand side of a rule occurs is unimportant — we can always use the rule to make the rewrite while doing a derivation.

There are **more expressive kinds of grammars**, with more than one symbol on the left hand side of the rewrite arrow, in which the symbols to the right and left have to be taken into account before making a rewrite. Such grammars are linguistically important, and we will study them in May.

A language is called context free if it is generated by some context free grammar.

Well known CFG are **Phrase Structure Grammars** (PSG) also known as Context Free Phrase Structure Grammars and they are based on **rewrite rules**. They can be used for both understanding and generating sentences.

## 11. CFG: Formal Language

Let's start by using simple grammars that generate formal languages. E.g., take the grammar below.

### Rules

Rule 1  $S \rightarrow A B$     Rule 2  $S \rightarrow A S B$

Rule 3  $A \rightarrow a$         Rule 4  $B \rightarrow b$

the above grammar let us rewrite 'S' to 'aabb'. Try it your self!

S

ASB Rule 2

aSB Rule 3

aSb Rule 4

aABb Rule 1

aaBb Rule 3

aabb Rule 4

Such a sequence is called a **derivation** of the symbols in the last row, in this case, i.e. a derivation of the string 'aabb' ( $S \Rightarrow^* aabb$ ).

## 12. CFG: More derivations

Note that there may be many derivations of the same string. For example,

S

ASB Rule 2

ASb Rule 4

aSb Rule 3

aABb Rule 1

aAbb Rule 4

aabb Rule 3

is another derivation of 'aabb'.

## 12.1. CFG: Language Generated

The above grammar generates the language  $a^n b^n - \epsilon$  (the language consisting of all strings consisting of a block of  $a$ 's followed by a block of  $b$ 's of equal length, except the empty string).

If we added the rule  $S \rightarrow \epsilon$  to this grammar we would generate the language  $a^n b^n$ . Therefore, these two languages **are context free**.

On the other hand,  $a^n b^n c^n$  **is not**. That is, no matter how hard you try to find CFG rules that generate this language, you won't succeed. No CFG can do the job. The same holds for, e.g.  $a^n b^m c^n d^m$ .

Again, there are formal ways to prove whether a language is or is not context free.

## 13. FG for Natural Languages

Now we will move to see how CFG have been applied to natural language. To this end, it is convenient to distinguish rules from non-terminal to terminal symbols which define the lexical entries (or lexicon).

- ▶ Terminal: The terminal symbols are **words** (e.g. sara, dress ...).
- ▶ Non-terminal: The non-terminal symbols are syntactic **categories** (CAT) (e.g. *np*, *vp*, ...).
- ▶ Start symbol: The start symbol is the *s* and stands for **sentence**.

The production rules are divided into:

- ▶ Lexicon: e.g.  $np \rightarrow \text{sara}$ . They form the set LEX
- ▶ Grammatical Rules: They are of the type  $s \rightarrow np vp$ .

**Alternative notation** The lexicon and the derivation can also be written as below:

$np \rightarrow \text{sara}$  is also written as  $\langle \text{sara}, np \rangle$

A derivation of a sequence of words  $w_1, \dots, w_n$  from the start symbol will be represented as,

$\langle w_1 \dots w_n, s \rangle$

## 14. PSG: English Toy Fragment

We consider a small fragment of English defined by the following grammar  $G = \langle \text{LEX}, \text{Rules} \rangle$ , with vocabulary (or alphabet)  $V$  and categories  $\text{CAT}$ .

- ▶  $\text{LEX} = V \times \text{CAT}$ 
  - ▷  $V = \{\text{Sara}, \text{dress}, \text{wears}, \text{the}, \text{new}\}$ ,
  - ▷  $\text{CAT} = \{\text{det}, n, np, s, v, vp, \text{adj}\}$ ,
  - ▷  $\text{LEX} = \{\langle \text{Sara}, np \rangle, \langle \text{the}, \text{det} \rangle, \langle \text{dress}, n \rangle, \langle \text{new}, \text{adj} \rangle, \langle \text{wears}, v \rangle\}$
- ▶  $\text{Rules} = \{s \rightarrow np\ vp, np \rightarrow \text{det}\ n, vp \rightarrow v\ np, n \rightarrow \text{adj}\ n\}$

Among the elements of the [language recognized](#) by the grammar,  $L(G)$ , are

- ▶  $\langle \text{the}, \text{det} \rangle$  because this is in the lexicon, and
- ▶  $\langle \text{Sara wears the new dress}, s \rangle$  which is in the language by repeated applications of rules.



## 15. English Toy Fragment: Strings

$\langle \text{Sara wears the new dress, } s \rangle$  is in the language. Try to prove it your self.

- (1)  $\langle \text{new dress, } n \rangle \in L(G)$  because  
 $n \rightarrow \text{adj } n \in \text{Rules}$ ,  
 $\langle \text{new, adj} \rangle \in L(G)$  (LEX), and  
 $\langle \text{dress, } n \rangle \in L(G)$  (LEX)
- (2)  $\langle \text{the new dress, } np \rangle \in L(G)$  because  
 $np \rightarrow \text{det } n \in \text{Rules}$ ,  
 $\langle \text{the, det} \rangle \in L(G)$  (LEX), and  
 $\langle \text{new dress, } n \rangle \in L(G)$  (1)
- (3)  $\langle \text{wears the new dress, } vp \rangle \in L(G)$  because  
 $vp \rightarrow v \ np \in \text{Rules}$ ,  
 $\langle \text{wears, } v \rangle \in L(G)$  (LEX), and  
 $\langle \text{the new dress, } np \rangle \in L(G)$  (2)
- (4)  $\langle \text{Sara wears the new dress, } s \rangle \in L(G)$  because  
 $s \rightarrow np \ vp \in \text{Rules}$ ,  
 $\langle \text{Sara, } np \rangle \in L(G)$  (LEX), and  
 $\langle \text{wears the new dress, } vp \rangle \in L(G)$  (3)

Now try to build the structure of the parsed string.

## 16. English Toy Fragment: Phrase Structure Trees

$\langle \text{new}, \text{adj} \rangle$

adj  
|  
new

$\langle \langle \text{new}, \text{adj} \rangle, \langle \text{dress}, \text{n} \rangle, \text{n} \rangle$

n  
/ \  
adj n  
| |  
new dress

$\langle \text{dress}, \text{n} \rangle$

n  
|  
dress

$\text{n} \rightarrow \text{adj n}$

n  
/ \  
adj n

s  
/ \  
np vp  
| / \  
Sara v np  
| | / \  
wears det n  
| | / \  
the adj n  
| |  
new dress

## 17. Extending our grammar

Try to extend your grammar so to deal with the sentence you have analyzed before and which are repeated below.

She likes a red shirt.

I will leave Boston in the morning.

John saw the man with the telescope.

John gave Mary a red shirt.

## 18. Recursion

Which rule do we need to generate the phrases below?

1. the flight to Boston
2. the flight to Boston from Miami
3. the flight to Boston from Miami in February
4. the flight to Boston from Miami in February on Friday

np --> np pp

I.e. a recursive rule, it contains the same category on the left and the right side.

## 19. Summing up (I)

We have seen that

- ▶ There is a close correspondence between **parse trees** and **derivations**: every derivation corresponds to a parse tree, and every parse tree corresponds to (maybe many) derivations.
- ▶ PSG, besides deciding whether a **string** belongs to a given language, deals with **phrase structures** represented as **trees**.
- ▶ An important difference between strings and phrase structures is that whereas **string concatenation** is assumed to be **associative**, **trees** are **bracketed structures**.
- ▶ Thus trees **preserve** aspects of the **compositional** (constituent) structure or derivation which is lost in the string representations.

## 20. Summing up (II)

- ▶ The **language generated** by a grammar consists of all the strings that the grammar classifies as grammatical.
- ▶ A CFG **recognizer** is a program that correctly tells us whether or not a string belongs to the language generated by a PSG.
- ▶ A CFG **parser** is a program that correctly decides whether a string belongs to the language generated by a CFG and also tells us what its structure is.
- ▶ A **Context Free Language** is a language that can be generated by a CFG.

## 21. Overgeneration and Undergeneration

We would like the Formal Grammar we have built to be able to recognize/generate **all and only** the grammatical sentences.

- ▶ **Overgeneration:** If the FG generates as grammatical also sentences which are not grammatical, we say that it overgenerates.
- ▶ **Undergeneration:** If the FG does not generate some sentences which are actually grammatical, we say that it undergenerates.

For instance, can the CFG we have built distinguish the sentences below?

1. She likes a red shirt
2. \*She like a red shirt
3. I like him
4. \*I like he

In the next two hours, we will see how to enrich CFG so to deal with such differences, namely how to deal with **agreements** (overgeneration).



## 22. Undergeneration

Context free rules work **locally**. For example, the rule

$$s \rightarrow np\ vp$$

tells us how an  $s$  can be decomposed into two parts, an  $np$  and a  $vp$ .

But we have seen that certain aspects of natural language seem to work in a non-local, **long-distance way**. Indeed, for a long time it was thought that such phenomena meant that grammar-based analyses had to be replaced by very powerful new mechanisms

## 22.1. Undergeneration (Cont'd)

Consider these two English np. First, an np with an object relative clause:

“The witch who Harry likes”.

Next, an np with a subject relative clause:

“Harry, who likes the witch.”

What is their syntax? That is, how do we build them?

Today we will briefly see a fairly traditional explanation in terms of movement, gaps, extraction, and so on. In the second part of the course, we will look into more modern approaches.





## 24. Next Steps

We said that to examine how the syntax of a sentence can be computed, we must consider two things:

1. **The grammar**: A formal specification of the structures allowable in the language. [Data structures]
2. **The parsing technique**: The method of analyzing a sentence to determine its structure according to the grammar. [Algorithm]

We have seen 1. today, we will look at 2. next Thursday.