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 Syntax 4
2 Dependency 5
3 Long-distance Dependencies 6
3.1 Relative Pronouns 7
3.2 Coordination 8
4 Formal Approaches 9
5 Sentence Structures: English 10
5.1 Exercises 11
6 Syntax Recognizer 12
6.1 NLs are not RL: Example I 13
6.2 NLs are not RL: Example II 14
7 FSA for syntactic analysis 16
8 Formal Grammars: Definition 17
8.1 Formal Grammar: Terminology 18
8.2 Derivations 19
8.3 Formal Languages and FG 20
8.4 FG and Regular Languages 21

8.5 FSA and RG 22
9 Context Free Grammars 23
10 CFG: Formal Language 24
11 CFG: More derivations 25
11.1 CFG: Language Generated 26
12 FG for Natural Languages 27
13 PSG: English Toy Fragment 29
14 English Toy Fragment: Strings 30
15 English Toy Fragment: Phrase Structure Trees 31
16 Extending our grammar 32
17 Summing up (I) 33
18 Summing up (II) 34
19 Overgeneration and Undergeneration 35
20 Undergeneration 37
20.1 Undergeneration (Cont'd) 38
21 Trasformational Grammar 39
21.1 Trasformational Grammars: Relative Clauses 40
22 Next Steps 41

1. 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.

2. Dependency

Dependency: Categories are interdependent, for example

Ryanair services [Pescara]$_{np}$	Ryanair flies [to Pescara]$_{pp}$
*Ryanair services [to Pescara]$_{pp}$	*Ryanair flies [Pescara]$_{np}$

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

3. 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 [. . .]?

3.1. 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]$_{rc}$]$_{n}$]$_{np}$ [left]$_{v}$]$_{s}$.
- [[the [book [which Sara wrote [. . .]]$_{rc}$]$_{n}$]$_{np}$ [is interesting]$_{v}$]$_{s}$.

Can you think of another relative pronoun?

3.2. 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]$_{vp}$ and [arriving in San Francisco]$_{vp}$]$_{vp}$

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]$_{np}$]$_{vp}$"

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 after the winter break.

4. 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]

5. 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.

5.1. Exercises

Now represent in the format you prefer the sentences below:

I like a read shirt

I will leave Boston in the morning.

John saw the man with the telescope.

John thinks someone left.

6. Syntax Recognizer

In lecture 1, we have used FSA to recognize/generate natural language morphology, and in Lab 1 you will do an exercise using FSA 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**.

6.1. 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 : { the cat, the dog, ...}, and the transitive verbs in B : { 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.

6.2. NLs are not RL: Example II

Another evidence was provided by Chomsky in 1956. Let $S_1, S_2, \ldots S_n$ 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	$\rightarrow b$
other words	$\rightarrow \epsilon$

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. $abbabbabba$) 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.

7. 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).)

8. 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, $P = \{\alpha \rightarrow \beta \mid \alpha \in V^+ \wedge \beta \in V^*\}$.

8.1. Formal Grammar: Terminology

In other words, Formal Grammars are string rewrite 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.

8.2. 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 $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3$, $\ldots, \alpha_{n-1} \Rightarrow \alpha_n$, then $\alpha_1 \Rightarrow^* \alpha_n$.

8.3. Formal Languages and FG

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

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

The above definition says that 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.

8.4. 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:

- $A \rightarrow xB$
- $A \rightarrow x$

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.

8.5. FSA and RG

The association between FSA and RG is straight:

RG	FSA
$A \to xB$	from state A to state B reading x
$A \to x$	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 on Principles of Compilers for more details.

9. 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 **after Christmas**.

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.

10. CFG: Formal Language

Let's start by using simple grammars that generate formal languages, rather than natural language examples, as the formal examples are typically shorter. 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 lets 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'.

11. 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’.

11.1. 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.

12. 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 stand for sentence.

The production rules are divided into:

- Lexicon: Instead of writing $np \rightarrow$ sara, we will write: $\langle \text{sara}, np \rangle$. They form the set LEX
- Grammatical Rules: They are of the type $s \rightarrow np\ vp$.

A derivation of a sequence of words $(w_1, \ldots w_n)$ from the start symbol will be represented as,
$\langle w_1 \ldots w_n, s \rangle$

13. PSG: English Toy Fragment

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

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

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

- $\langle \text{the}, det\rangle$ 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.

14. English Toy Fragment: Strings

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

(1) $\langle$new dress, $n\rangle \in L(G)$ because
$n \rightarrow adj\ n \in$ Rules,
$\langle$new, $adj\rangle \in L(G)$ (LEX), and
$\langle$dress, $n\rangle \in L(G)$ (LEX)

(2) $\langle$the new dress, $np\rangle \in L(G)$ because
$np \rightarrow det\ n \in$ Rules,
$\langle$the, $det\rangle \in L(G)$ (LEX), and
$\langle$new dress, $n\rangle \in L(G)$ (1)

(3) $\langle$wears the new dress, $vp\rangle \in L(G)$ because
$vp \rightarrow v\ np \in$ Rules,
$\langle$wears, $v\rangle \in L(G)$ (LEX), and
$\langle$the new dress, $np\rangle \in L(G)$ (2)

(4) $\langle$Sara wears the new dress, $s\rangle \in L(G)$ because
$s \rightarrow np\ vp \in$ Rules,
$\langle$Sara, $np\rangle \in L(G)$ (LEX), and
$\langle$wears the new dress, $vp\rangle \in L(G)$ (3)

Now try to build the structure of the parsed string.

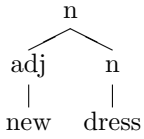
15. English Toy Fragment: Phrase Structure Trees

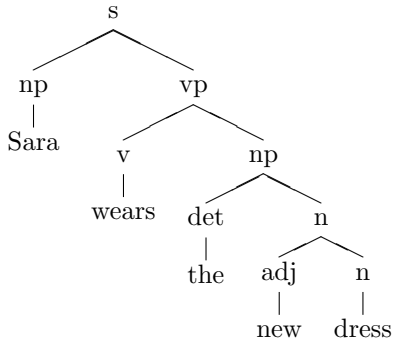
⟨ new, adj ⟩

⟨ dress, n ⟩

n → adj n

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





16. 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 read shirt.

I will leave Boston in the morning.

John saw the man with the telescope.

John gave Mary a read shirt.

17. 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.

18. 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.

19. 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.
- **Undergeration**: 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 read shirt
2. *She like a read shirt
3. I like him
4. *I like he

In the lab we will see an easy way of handling these problems with CFG in Prolog. On Thursday, we will see how to enrich CFG so to deal with such differences, namely how to deal with **agreements**.

20. 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

20.1. 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. After the winter break we will look into more modern approaches.

21. Trasformational Grammar

The traditional explanation basically goes like this. We have the following sentence:

Harry likes the witch

We can think of the np with the object relative clause as follows.

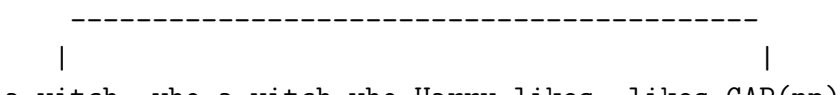
```
      ----------------------
    |                        |
the witch who Harry likes GAP(np)
```

That is, we have

1. extracted the np "Harry" from the subject position, leaving behind an np-**gap**,
2. **moved** it to the front, and
3. placed the relative pronoun "who" between it and the gap-containing sentence.

21.1. Trasformational Grammars: Relative Clauses

Recall: Relative clauses are an example of unbounded dependencies. The word 'dependency' indicates that the moved np is linked, or depends on, its original position. The word 'unbounded' is there because this "extracting and moving" operation can take place across arbitrary amounts of material. For example, from



22. 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.