# Chapter 7

# Morphology? What is it? Is there a cure?

**Morphology** is the linguistics field dedicated to the study of the internal structure of words (morph = shape, logos = word).

Being able to determine the morphology of words can be extremely important to get, for instance, their accurate lemmas; it is also a fundamental step should a syntactic analysis be required. Many applications are based on some morphological processing. For instance, if you want to know whether two words in different languages have the same origin (*organization* vs. *organização*), that is, if they are **cognates** (another fancy word to impress your friends), morphology can help.


In this chapter we will study words' constituents (morphologic parsing) and we will see different techniques to mark words in a text with morpho-syntactic tags.

This chapter is particularly inspired by Jurasfsky's book, previous Nuno Mamede's slides and also the slides of Sudeshna Sarkar. All the silly jokes and mistakes are on me.

## 7.1 Dissecting words into morphemes

Words are constituted by (meaningful) units called **morphemes**. There are two types of morphemes:

- **stems**, which carry the primary of meaning of words;

- **affixes**, which change stems meaning and/or have grammatical functions. They come in different types:
  - **prefixes**: added at the beginning of the word. For instance, *anti-* from *anti-motim*, or *des* from *desinteressante*;
  - **suffixes**: appear at the end of the word. An example is *os* from *gatos*, *alunos*, etc.
  - **infixes**: are inserted inside the stem
  - **circumfixes**: precede and follow the stem and are inserted at the same time. For instance, *amanhecer* has this type of affix.
  - **clitics**: these are morpheme that function like a word, but that do not appear alone. Example: *os* in *vi–os*.

There are languages (called **agglutinative Languages** – see Figure 7.1[1]) where words can contain an impressive number of morphemes. Turkish, for instance, has many words with 9 or 10 morphemes.

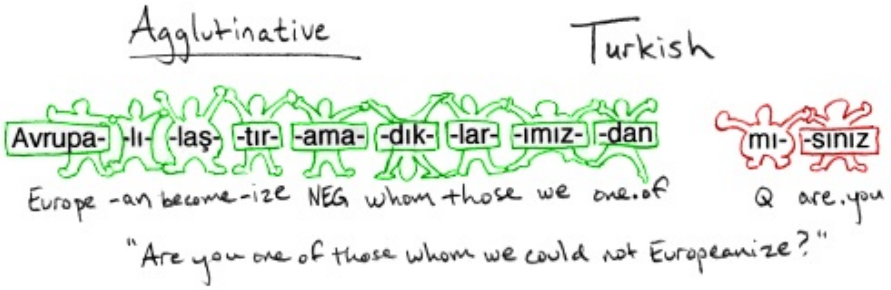


Figure 7.1: Agglutinative language

**Exercise 11: Dissecting Portuguese**

Consider the word *inacreditavelmente*. Can you find a portuguese word with more morphemes? If you do, let me know.

## 7.2 Building words

There are many different ways of building words from a word stem. Here are some of them:

- **Inflection**: Doesn't change the word class or the meaning of the word, considering the original stem. Examples are *eats* from *eat* (both verbs) and *gatas* from *gato* (both nouns).

[1] Taken from http://specgram.com/CLII.3/09.phlogiston.cartoon.3.html

- **Derivation**: Results in a word from a different word class or with a different meaning. Examples are *do* from *undo* (opposite meanings) and *amigável* from *amigo* (adjective and noun).
- **Compounding**: Combination of multiple word stems. *doghouse* and *guarda-chuva* are examples of this.
- **Cliticization**: Words with clitics, such as *apagou-o*.
- ...

Just to show you how strange this can be, have the following example (for Hebrew), representing a case of **Templatic** morphology with triconsonantal stems:

Considering that *lmd* means *to learn* or *study*:

- CaCaC triggers lamad (*he studied*, ...)
- CiCeC triggers limed (*he taught*, ...)
- CuCaC triggers lumad (*he was taught*, ...)

Original, isn't it?

## 7.3 Part-of-speech tagging

**Part-of-speech tagging** or **POS tagging** is the process of automatically assigning a morpho-syntactic tag to each word in a text. The main problem here is our darling friend ambiguity, but not because there are many ambiguous words. The problem is that these ambiguous words are extremely frequent in texts.

We will study three main approaches: a) Rule-based; b) Hybrid; c) Stochastic. The more recent approaches to POS tagging – the unsupervised approach, and the ones that use deep nets – are, for the moment, out of the scope of this course.

### 7.3.1 Rule-based approach

The rule-based approach uses hand-crafted rules to tag a text. Usually, there are two main steps:

1. With the help of a dictionary, you tag each word with all its possible labels;
2. With the help of a set of (disambiguation) rules, you disambiguate these labels.

As an example, consider the sentence *He had a book*. After the first step, you might have:

- he he/pronoun
- had have/verbpast have/auxliarypast
- a a/article

- book book/noun book/verb

If there is a rule that states that *if the previous word is an article, remove all the labels related with verbs*, after the application of this rule, we obtain:

- he he/pronoun
- had have/verbpast have/auxliarypast
- a a/article
- book book/noun

An example of a rule-based tagger is the **EngCG tagger**[2]. If you have time, give it a once-over.

### 7.3.2 Transformation based POS tagging

The **transformation based POS tagging** is also known as **Brill Tagging**. It is also rule-based, but its rules are learned from a labeled corpus (that is why is said to be an hybrid approach). It takes the following steps:

- Each word is tagged with the most frequent label;
- Transformational rules are learned from a labeled corpus (rules that replace labels);
- Transformational rules are applied until some stop condition is reached.

For instance, assume that you know that:

P(NN/race) = 0.98

P(VB/race) = 0.02

Then, the sentence *He is expected to race tomorrow* will be labeled as:

- he/PRN
- is/VBZ
- expected/VBN
- to/TO
- race/NN (not good!)
- tomorrow/NN

If you have a rule that says that you should *Replace NN by VB when the previous label is TO*, the correct labelling will be attained.

[2] http://www2.lingsoft.fi/doc/engcg/intro/

### 7.3.3 Stochastic approach

The target here is to choose the best sequence of tags,

T = $t_1$ $t_2$ ... $t_n$

for a certain sequence of words

W = $w_1$ $w_2$ ... $w_n$.

That is, we want to calculate t $\in$ $\tau$ that maximizes P(T | W), being $\tau$ the set of all possible tags' sequences:

$$argmax_{t \in \tau} P(T|W) \tag{7.1}$$

As it is difficult to calculate P(T | W), we can use **Bayes Rule**:

$$P(x|y) = P(y|x)\frac{P(x)}{P(y)} \tag{7.2}$$

and as P(W) does not depend on T, our problem is now to calculate[3]:

$$argmax_{t \in \tau} P(W|T) * \frac{P(T)}{P(W)} = argmax_{t \in \tau} P(W|T) * P(T) \tag{7.3}$$

Note that:

- P(T) is the prior probability of the sequence of tags;
- P(W | T) is the likelihood of W being given T.

#### 7.3.3.1 The Hidden Markov Models

Unfortunately, the previous formula is still too difficult to calculate and the famous **Hidden Markov Models** (HMM) are called to save the day. These models assume that the system in hands can be modeled as a Markov Process (remember?) with non observable (hidden) states. Basically, they assume that:

- The probability of occurrence of a word only depends on its label (that is, it doesn't depend on other words);
- The probability of a tag only depends on the previous tag (this is called the **bigram assumption**).

Due to this, the following approximations can be done:

$$P(W|T) \approx \prod_{i=1}^{n} P(w_i|t_i) \tag{7.4}$$

[3] Think why it is more difficult to calculate P(T | W) than P(W | T).

$$P(T) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{7.5}$$

Therefore, the formula that we need to calculate is now given by:

$$argmax_{t\in\tau} P(W|T) * P(T) \approx \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \tag{7.6}$$

The good news is that we now know how to calculate this:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}t_i)}{C(t_{i-1})}. \tag{7.7}$$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}. \tag{7.8}$$

#### 7.3.3.2 Viterbi algorithm

For models such as HMM that contain hidden variables, the task of determining which of the variables is the source of an observable sequence is called **decoding**. The **Viterbi algorithm** is often used with HMMs and it seeks the best path; it is based on probabilities and uses the technique of **dynamic programming** (solve small problems and memorize the best solution).

Considering that:

- N = # {$L_1$, ..., $L_N$};
- n = number of words in the given sequence $w_1$ ... $w_n$;
- SS is an array N*n that registers the best punctuation of the best sequence found until a certain position with label $L_i$;
- BP is an array N*n that registers the best probability of a transition from the previous to the current state;
- C is an array 1*n that registers the best sequence of labels.

Then, the Viterbi algorithm works as follows:

**Exercise 12: Viterbi**

Consider the labels N and V, and the sequence of words A B A. Knowing the following probabilities, use Viterbi to calculate the best sequence of labels for that sequence of words:

- P(N | $< s >$) = 0.25, P(V | $< s >$) = 0.3
- P(A | N) = 0.1, P(A | V) = 0.3, P(B | N) = 0.2, P(B | V) = 0.4
- P(N | V) = 0.15, P(N | N) = 0.2, P(V | V) = 0.75, P(V | N) = 0.25

**Algorithm 2** Viterbi

$i \leftarrow 1$
**while** $i < N$ **do**
  SS(i, 1) = P($w_1 \mid L_i$)*P($L_i \mid < s >$)
  BP(i, 1) = 0
  $i++$
**end while**
$t \leftarrow 2$
**while** $t < n$ **do**
  $i \leftarrow 1$
  **while** $i < N$ **do**
    SS(i, t) = $max_{j=1,...,N}$ SS(j, t-1)*P($L_i \mid L_j$)*P($w_t \mid L_i$)
    BP(i, t) = j that resulted in the maximum score
    $i++$
  **end while**
  $t++$
**end while**
C(n) = i that maximizes SS(i, n)
$i \leftarrow n - 1$
**while** $i > 1$ **do**
  $i--$
  C(i) = BP(C(i+1), i+1)
**end while**

## 7.4 Very brief review of Portuguese morphology – back to school

Each word can have more than one morphological tag, according with its function in a sentence. For instance, you can say *Estava uma noite escura* | *branca* | *clara* | *amarela* | *verde*, but you can't say *Estava uma noite de* or *Estava uma noite dormir*. That is, if you have an adjective, you can replace it by another adjective, but not with a preposition or a verb (an exception are nouns and pronouns). You don't remember what a pronoun or a preposition is? Well, let us review some morphological classes (remember this, because you will need it for the Syntax chapter); capitalized words illustrate each class:

- Nouns: *JOAQUIM estava a ler na SALA quando ouviu um BARULHO no JARDIM.*
- Determiners: *Era UM ruído estranho, O mais estranho de sempre, UM leve e suave arrastar.*
- Pronouns: *Sem que ELE o conseguisse evitar, sentiu o SEU coração a começar a bater com força.*
- Verbs: *DIRIGIU-SE à porta, mas PAROU, com a maçaneta na mão.*
- Adjectives: *Sentia um pavor INDESCRITÍVEL a entranhar-se no corpo.*
- Adverbs: *LENTA e FIRMEMENTE um frio glaciar subia por ele acima.*

- Prepositions: *O que estaria DO outro lado DA porta? O que estaria NO jardim?*
- Conjunction: *Tremendo, Joaquim abriu a porta E espreitou.*
- Interjections: *Nem teve tempo de gritar "AQUI D'EL REI", "DIACHO" ou "IRRA".*

# Chapter 8

# Syntax important Natural very Language is

**Syntax** respects the way words can be organized in sentences. At this point, I hope that it is clear in your mind that the syntax of any programming language is much easier to capture than the one of any NL.

In NLP many people work in this research field. Some try to build grammars or tools to parse sentences; others work on algorithms. In the following we will remember some basic notions of syntax, we will study some formalisms to code grammars, and we will see different algorithms that perform syntactic analysis.

## 8.1 Very brief review of the Portuguese syntax – back to school again

Different sentence types:

- Declarative: *O CORPO DE JOAQUIM FOI DESCOBERTO NA MANHÃ SEGUINTE.*
- Interrogative: *QUEM PODERIA TER FEITO TAMANHA MALDADE? – interrogava-se uma vizinha.*
- Imperative: *MINHA SENHORA, SAIA-ME DAQUI! – a voz do inspector Morcela não deixava espaço para desobediências.*
- Exclamatory: *COITADINHO! – ainda comentou a senhora antes de desaparecer. Ia ser um dia difícil: tinha tanta coisas para contar às amigas.*

Main constituents of a sentence:

- Subject: the entity (who or what) the sentence is about. Ex: *O ALUNO fez o teste*
- Predicate: what is said about the subject. Ex: *O aluno FEZ O TESTE*
- Direct Object: the receiver of the sentence action. Direct objects can be identified by asking: subject + verb + who? or what? = DIRECT OBJECT. Ex: *O professor deu-lhe O TESTE*

- Indirect Object: identifies to (or for whom or what) the sentence action is performed (transitive verbs). It typically precedes the direct object. An indirect object can be identified by asking who or what received the direct object. Ex: *O professor deu-LHE o test*
- Predicative (of the subject): gives sense to a verb, by adding features to the subject. Ex: *O professor anda contente da vida*

Main phrases:

- Noun phrase: noun (or pronoun) + complements (optional). Ex: *O JOÃO fugiu*, *AQUILO foi uma vergonha*, *A MAÇÃ VERDE E VENENOSA foi devorada pela Branca de Neve*
- Verbal phrase: verb + complements (optional). Ex: *Não COSTUMAMOS SAIR*, *COMECEI A LER o livro*, *TENHO DE SAIR hoje*, *Eu TENHO UMA BARBIE*
- Adjectival phrase: adjective (or pronoun) + complements (optional). Ex: *A rapariga MORENA fugiu*, *A aula é MARAVILHOSA*
- Propositional phrase: preposition + noun + complements (optional). Ex: *Fui a casa DO MEU AMIGO*
- Adverbial phrase: Adverb + complements (optional). Ex: *Foi um erro MUITO grande*, *FRANCAMENTE, não sei*.

Agreement:

- Subject + Verb: Ex: *O JOÃO FUGIU*
- Verb + predicative of the subject: Ex: *SÃO 10 HORAS*
- Subject + predicative of the subject: Ex: *O JOÃO estava CANSADO*

## 8.2 Formal Grammars

In this section you will see several formalisms for writing grammars. Some (at least one) you already know about. However, it should be said that there are many, many, many more formalisms.

### 8.2.1 Context Free Grammars

You have probably already met a **CFG** before. A CFG is a tuple ($N$, $T$, $S_0$, R), in which:

- $N$ is a set of non-terminal symbols;
- $T$ is a set of terminal symbols (either words or POS tags[1]);
- $S_0 \in T$ is the grammar's initial symbol;

[1] You will see why POS tags can be seen as non-terminal symbols.

- R is a set of rules in the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$.

A fundamental concept behind CFGs is the one of **derivation**. A derivation is a sequence of applications of grammar rules. In a derivation, a non terminal symbol $A$ can be replaced with a symbol $\alpha$ if there is a rule $A \rightarrow \alpha$. Also, if after the application of several rules we manage to reach $\alpha_n$ from $\alpha_1$, we say that $\alpha_1$ derives $\alpha_n$ (which is denoted by $\alpha_1 \rightarrow_* \alpha_n$). A easy way to represent a derivation is through an **analysis or parse tree** (see Figure 8.1).

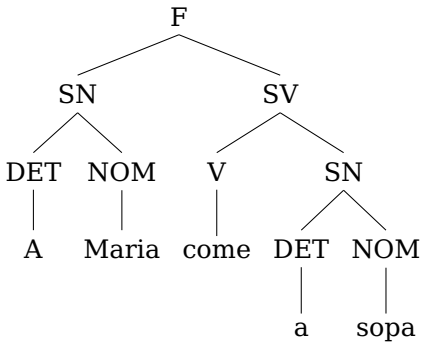


Figure 8.1: Parse tree of *A Maria come sopa*.

Another important concept associated with CFG (and grammars in general) is the one of **language generated by a grammar**, which can be defined as follows (where L(G) represents the language generated by the grammar G = (N, T, $S_0$, R)):

L(G) = $\{W : S_0 \rightarrow_* W \text{ and } W \in T^*\}$

As an example, consider the CFG $G_1$ = ($N$, $T$, $S_0$, R), in which:

- $N$ = {F, SN, SV, NOM, V, DET}
- $T$ = {a, Maria, sopa, come}
- $S_0$ = F
- R = {F $\rightarrow$ SN SV, SN $\rightarrow$ DET NOM, SV $\rightarrow$ V SN, DET $\rightarrow$ a, NOM $\rightarrow$ Maria | sopa, V $\rightarrow$ come}

The sentence *A Maria come a sopa* belongs to L($G_1$) and Figure 8.1 represents its derivation.

**Exercise 13: CFG**

Create a CFG for the following fragment of Portuguese and show that each sentence belongs to the grammar you have created by building the different derivation trees:

*Portugal é um país da Europa.*

*O Miguel come a sopa.*

*Todas as crianças gostam de batatas-fritas.*

*Todos os alunos fugiram pela janela.*

### 8.2.2 Problems with CFG

There are several problems with CFG, regarding limitations of the formalism. In this section we will talk about three of them. The first relates to the fact that CFG are **unable to properly capture long dependencies**. This can be a problem for languages such as portuguese, but there are other languages in which word order in sentences is much more flexible, and CFG are unable to properly capture this phenomena. You will see in Section 8.2.5 a formalism designed to deal with this problem (the so called dependency grammars).

**Exercise 14: Long dependencies**
Write a CFG that relates *A Maria* with *tricô* in the sentence *A Maria, que tinha uma avó que tinha sido actriz e que gostava muito, por isso, de declamar poesias e de interpretar personagens que ela própria inventava e que eram sempre muito sórdidas, deprimentes e inverosímeis, gostava muito de fazer tricô*. Hard, isn't it?

The second problem is **agreement**. CFG are not able to force that two phrases (for instance) agree in number or gender, unless non terminal systems carry that information. That is, instead of

SN $\rightarrow$ art nc

one would need to write (f stands for feminine, m for masculine, s for singular and p for plural):

SN $\rightarrow$ artfs ncfs
SN $\rightarrow$ artfp ncfp
SN $\rightarrow$ artms ncms
SN $\rightarrow$ artmp ncmp

This is obviously an impractical solution. Unification-based grammars emerged to solve this problem: attribute/value pairs represent the needed information and unification is used in the parsing process.

A third problem has to do with **ambiguity**: if many syntactic trees are generated, how can we chose the best one? There is nothing in the CFG formalism that allow us to deal with this. For instance, consider the sentences *Indique-me um hotel com piscina com água quente* and *Indique-me um hotel com piscina com suite nupcial*. Different analysis will capture different attachments. How can we find the correct one? Some possible solutions are:

- Postpone the problem and let further (semantic) processing be responsible for solving it;
- Create and use manual disambiguation rules;
- Build and use probabilistic models to give different probabilities to each tree.

Considering the last hypothesis, how can we compute these probabilistic models? In the following section you will learn how to do so.

### 8.2.3 Probabilistic Grammars

A **Probabilistic Context-Free Grammar (PCFG)** is also a tuple G = (N, T, $S_0$, R) where:

- N, T, $S_0$ are defined in the same way as in a CFG;
- R is a set of rules in the form A $\rightarrow \alpha$ [p], where:
  - A and $\alpha$ are defined as in a CFG, and
  - p is a number, between 0 and 1, that represents P($\alpha$ | A), that is, the probability of a given non terminal A to derive $\alpha$[2].

So, we can use a PCFG to calculate the probability associated with each tree, and thus solve ambiguity; however, first we need to calculate the probabilities associated with each rule. How can we do that?

Considering rules in the form A $\rightarrow \alpha_j$, we can calculate P(A $\rightarrow \alpha_i$ | A) within a **treebank** (a corpus in which each sentence is syntactically annotated[3]), by using the following formula:

$$P(A \rightarrow \alpha_i | A) = \frac{count(A \rightarrow \alpha_i)}{\sum_{j=1}^{n} count(A \rightarrow \alpha_j)} \tag{8.1}$$

Then, we can build a PCFG using rules such as:

SV $\rightarrow$ Verbo [.50]

SV $\rightarrow$ Verbo SN [.45]

SV $\rightarrow$ Verbo SN SN [.05]

Finally, we just have to associate the probabilities to each tree, by considering the probability of each constituent (that is, the probability of each tree is the product of the probabilities of each one of its subtrees):

P(A) = P(A $\rightarrow \alpha_1$, ... , $\alpha_n$ | A) * P($\alpha_1$) * ... * P($\alpha_n$).

In the leaves we use the probabilities of the POS tags:

P($\alpha_i$ | $w_i$)

[2] We will use P($\alpha$ | A), P(A $\rightarrow \alpha$) and P(A $\rightarrow \alpha$ | A) to represent this concept.

[3] Typically, the syntactic annotation is automatic (based on a grammar) and then manually corrected. There are many treebanks available (Penn Treebank, Prague dependency Treebank, and, for portuguese, the Floresta Sintáctica, which can be found in the Linguateca site.)

For illustrative purposes, consider a grammar with the following rules:

F → SN SV [0.6]
F → SV [0.4]
SN → Nome [1.0]
SV → Verbo [0.3]
SV → Verbo SN [0.7]
Nome → nota [0.1] | ... | [0.9]
Verbo → nota [0.2] | ... | [0.8]

In the following we calculate the probabilities associated with the analysis trees of the sentence *nota nota*.

The first tree is given by Figure 8.2.

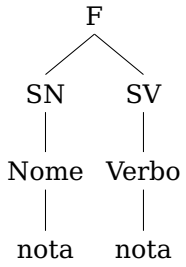


Figure 8.2: Tree 1 – “Nota nota”.

We have that:

P([Nome nota]) = P(Nome → nota) = 0.1
P([Verbo nota]) = P(Verbo → nota) = 0.2
P([SN [Nome nota]]) = P(SN → Nome) * P([Nome nota]) =
= 1.0 * 0.1 = 0.1
P([SV [Verbo nota]]) = P(SV → Verbo) * P([Verbo nota]) =
= 0.3 * 0.2 = 0.06
P [F [SN [Nome nota]] [SV [Verbo nota]]]) =
= P(F → SN VP)*0.1*0.06 =
= 0.6 * 0.1 * 0.06 = 0.0036

And for the second one (Figure 8.3),
we have:

P([SV [Verbo nota] [SN [Nome nota]]]) =
= P(SV → Verbo SN) * 0.2 * 0.1 = 0.7 * 0.2 * 0.1 = 0.014
P([F [SV [Verbo nota] [SN [Nome nota]]]]) =

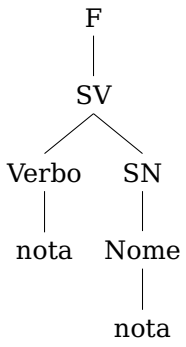


Figure 8.3: Tree 2 – “Nota nota”.

= P(F → VP) *0.014 = 0.4*.014 = 0.0056

Meaning that the second tree is more probable.

**Exercise 15: CFG**

Consider a grammar with the rules:
F → SN COORD SV [0.6] | SN COORD SN [0.4]
SN → Nome [1.0]
SV → Verbo [0.3] | Verbo COORD SN [0.7]
Nome → bucha [0.1] | estica [0.2] | ... | [0.7]
Verbo → estica [0.2] | ... | [0.8]
COORD → e [1.0]
Find the probability associated with the trees that derives *bucha e estica*

### 8.2.4 Categorial Grammars

A **Categorial Grammar** is a formalism that allows coding a grammar where grammatical categories are seen as functions and arguments, and where the category of phrases results from the application of functions to arguments. For instance, a determiner can be seen as a function (represented by `NP/N`), that takes a noun (`N`) on its right and returns a Nominal Phrase (`NP`).

A Categorial Grammar is composed of:

- A lexicon, which associates each word to a syntactic or semantic category. Each category can be:
  - an argument (such as nouns)
  - a function (such as verbs and determiners)
- A set of combination rules, which allow combining functions with arguments, namely:
  - X/Y: a function from Y to X, that is, it takes as arguments something that will result in a X when combined with Y on its right;

- X\Y: a function from Y to X, that is, it takes as arguments something that will result in a X when combined with Y on its left.

As an example, the following lexicon (plus the previous rules) represent the categorial grammar $G_2$:

- *o*, *a*: `SN/N`
- *Miguel*, *sopa*: `N`
- *come*: `(F\SN)/SN`

The sentence *O Miguel come a sopa* belongs to the language generated by $G_2$, as:

- to generate *o Miguel*, `SN/N + N` results in `SN`;
- by the same token, to generate *a sopa*, `SN/N + N` results in `SN`;
- *come a sopa* is obtained by `(F\SN)/SN + SN`, which returns `(F\SN)`;
- finally, *o Miguel come a sopa*, results from `(SN + (F\SN))`, which returns F (the initial symbol).

### 8.2.5 Dependency Grammars

The concept of constituency does not exist within dependency grammars. Here, relations (labeled or not) between words are established, that is, the grammar is described in terms of words and binary relations (either syntactic or semantic) between words (for example, subject, agent, etc.).

As the order between words is not considered, these grammars are appropriate to capture long distance dependencies and, specifically, to deal with languages in which word order is more flexible. Figure 8.4 shows an example of a dependency tree[4].

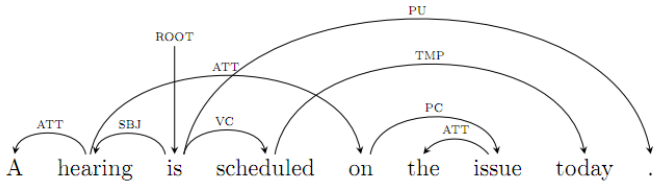


Figure 8.4: Dependency tree.

## 8.3 Parsing

The syntactic analysis of a sentence (also known as (syntactic) parsing) is the process of assigning a syntactic structure (a parse tree) to that sentence. In the following we will study several methods to perform a syntactic analysis.

[4]Taken from `http://en.wikibooks.org/wiki/LaTeX/Linguistics`.

### 8.3.1 Bottom-up and Top-down parsing

During parsing, a search strategy determines the order in which alternatives are considered. The search is guided by the input and grammar, and can involve (between others):

- a *top-down* search: the parse tree starts in the initial symbol of the grammar;
- a *bottom-up* search: the parse tree starts with the input.

The top-down strategy only searches among grammatical answers, but has a major limitation: it may suggest hypotheses that may not be consistent with the data; the bottom-up strategy only forms hypotheses consistent with the data, but may suggest hypotheses that make no sense globally (never reach the initial symbol).

Also, considering, for instance, a top-down search, if it is not possible to examine all alternatives in parallel, it becomes necessary to make further decisions, namely decide:

- which node in the current search space should be expanded first (breadth-first or depth-first)
- which of the applicable grammar rules should be expanded first (remember Prolog!)
- which leaf node in a parse tree should be expanded next (e.g., leftmost)

Before moving on to a specific top-down, left-right, depth-first (TD-LR-DF) algorithm, let us introduce the following notation. Considering:

1 Os 2 alunos 3 estudam 4

the state ((N SV) 2) means that, after position 2, we need to find an N followed by a SV. Consider also that **EC** stands for current state and **EA** for alternative states.

In Table 8.1, we illustrate the aforementioned algorithm, considering the sentence *os alunos estudam* and the grammar $G_{91}$ = ($N$, $T$, $S_0$, R), in which:

- $N$ = {F, SN, SV, N, ADJ, V, DET}
- $T$ = {os, alunos, dedicados, estudam}
- $S_0$ = F
- R = {F $\rightarrow$ SN SV, SN $\rightarrow$ DET N | DET N ADJ, SV $\rightarrow$ V | V SN, DET $\rightarrow$ os, N $\rightarrow$ alunos, V $\rightarrow$ estudam, ADJ $\rightarrow$ dedicados}

Since n = sentence length + 1, as (()n) $\in$ EC, we can say that the sentence belongs to the language generated by the grammar.

| EC | EA |
|---|---|
| ((F)1) | [ ] |
| ((SN SV)1) | [ ] |
| ((det n SV)1) | [((det n adj SV)1)] |
| ((n SV)2) | [((det n adj SV)1)] |
| ((SV)3) | [((det n adj SV)1)] |
| ((v)3) | [((v SN)3), ((det n adj SV)1)] |
| (()4) | [((v SN)3), ((det n adj SV)1)] |

Table 8.1: Algorithm TD-LR-DF.

### 8.3.2 CKY parser

The **CKY (Cocke-Kasami-Younger)** algorithm performs the analysis following the bottom-up strategy. It uses dynamic programming. Its only problem is that grammars must be in the Chomsky Normal Form (CNF), that is, rules must have the following forms (note that NT, NT1 and NT2 are non terminal symbols and t is a terminal one):

- NT $\rightarrow$ NT1 NT2
- NT $\rightarrow$ t

Being G = (N, T, $S_0$, R) a grammar in the CNF, n the number of words in the sentence W being analysed, and $w_j$ the word in position j, the CKY algorithm is defined as follows:

**Algorithm 3** CKY

$j \leftarrow 1$
**while** $j < n$ **do**
  [1, j] = {A : A $\rightarrow$ $w_j \in$ R}
  $j++$
**end while**
$i \leftarrow 2$
**while** $i < n$ **do**
  $j \leftarrow 1$
  **while** $j < n - i + 1$ **do**
    [i, j] = $\cup_{m=1}^{i-1}$ {A : A $\rightarrow$ B C $\in$ R, B $\in$ [m, j], C $\in$ [i - m, j + m]}
    $j++$
  **end while**
  $i++$
**end while**
**if** $S_0 \in$ [n, 1] **then**
  W $\in$ L(G)
**end if**

**Exercise 16: CKY**

Consider the grammar $G_{52}$ = (N, T , $S_0$, R) where:

N = {S, SN, SV, SP, det, nc, np, prep, v}

T = {o, a, Maria, saltou, fugiu, do, avião, hipopótamo, Pedro}

$S_0$ = S

R = {S → SN SV, SN → det nc | det np, SV → v SP, SP → prep nc | prep np, det → a | o,

np → Maria | Pedro, v → fugiu, prep → do, nc → hipopótamo}

Using the CKY, verify if the following sentences belong to the language generated by the previous grammar:

1. *O hipopótamo fugiu da Maria*
2. *O hipopótamo da Maria fugiu do Pedro.*

**Exercise 17: CKY and PCFG**

Consider the following grammar $G_{341}$ = (N, T, S0, P), in which:

N = {F, SN, SV, SP, nc, np, vi, vt, det, adj, prep}

T = {o, Pedro, Lisboa, aluno, etc.}

S0 = F

P = {F → SN SP SV | SN SV, SV → vi | vt SN, SN → det nc | np | nc | det np | det adj nc | SN SP,

SP → prep SN}

1. Convert the grammar to CNF;
2. Considering the CKY algorithm, show that the sentence $O_{det}$ $aluno_{nc}$ $deu_{vt}$ $um_{det}$ $estalo_{nc}$ $ao_{prep}$ $colega_{nc}$, belongs to L($G_{341}$) ;
3. Considering the following tree-bank, transform the previous grammar into a PCFG:

   [F [SN [np Lisboa]] [SV [vi despertou]]]

   [F [SN [det o np Pedro]] [SV [vi fugiu]]]

   [F [SN [det o np Pedro]] [SP [prep em SN[np Lisboa]]] [SV [vt comeu] SN [det a nc sopa]]]

   [F [SN [det o np Pedro]] [SP [prep em SN[np Lisboa)]] [SV [vi fugiu]]]

   [F [SN [det o np Pedro][SP [prep na SN[nc escola]]] [SV [vi fugiu]]]

### 8.3.3 Earley chart-parser

Consider a grammar G = (N, T, $S_0$, R), and that $w_j$ represents the word in position j of a sentence W. The Earley algorithm (also an example of dynamic programming),

**Algorithm 4** Predic

**if** A $\rightarrow$ w . B y $\in$ [i, j] **then**
  **for all** B $\rightarrow$ x $\in$ R **do**
    B $\rightarrow$ . x $\in$ [j, j]
  **end for**
**end if**

**Algorithm 5** Scann

**if** A $\rightarrow$ x . B y $\in$ [i, j] and B $\rightarrow w_j \in$ R **then**
  A $\rightarrow$ x B . y $\in$ [i, j+1]
**end if**

allows to check if W belongs to L(G), by keeping trace of the parts of the sentence analysed so far. It is seeded with a top-level rule, S' $\rightarrow$ $S_0$, and, then, repeatedly executes the following operations.

**Exercise 18: EARLEY**

Consider the grammar $G_{52}$ = (N, T , $S_0$, R) where:

N = {F, SN, SV, SP, det, nc, np, prep, vi, vt, adv}

T = {a, Maria, ressona, casa, sopa, bem, em, come}

$S_0$ = S

R = {F $\rightarrow$ SN SV | SP SN SV, SN $\rightarrow$ det nc | det np | np | nc,

SV $\rightarrow$ vt SN | vi adv | vi, SP $\rightarrow$ prep SN, det $\rightarrow$ a, np $\rightarrow$ Maria,

vi $\rightarrow$ ressona, vt $\rightarrow$ come, prep $\rightarrow$ em, nc $\rightarrow$ casa | sopa, adv $\rightarrow$ bem}

Using the Earley parser, verify if the following sentences belong to the language generated by the previous grammar:

1. *A Maria come sopa*
2. *Em casa a Maria ressona*.

### 8.3.4 Discussion

Bottom-up and top-down strategies have several problems. First, if the grammar has a left-recursion rule (for instance, A $\rightarrow$ A B) that can be a problem, as we have seen. Second, these strategies are not efficient as the same subtree can be created several times. Both CKY and Earley parser are based on dynamic programming. Thus, an analysis already performed is never repeated. In addition, they don't have the left recursion problem.

**Algorithm 6** Complete

**if** A $\rightarrow$ w . B y $\in$ [i, k] and B $\rightarrow$ x . $\in$ [k, j] **then**
  A $\rightarrow$ w B . y $\in$ [i, j]
**end if**

## 8.4 Structural Ambiguity

There are two main types of syntactic ambiguity:

- Attachment ambiguity: a constituent can be attached to the parse tree at more than one place. Ex: *Eu vi a torre de Belém a voar para Lisboa*
- Coordination ambiguity: different parts of a phrase can be conjoined by a conjunction. Ex: *Old man and woman like to..., Caros senhores e senhoras*

**Exercise 19: Syntactic ambiguity**

Consider the grammar $G_{32}$ = (N, T , $S_0$, R) where:

N = {F, SN, SV, SP, det, nc, prep, v}

T = {vi, homem, em, o, monte, com, telescópio}

$S_0$ = S

R = {F $\rightarrow$ SV, SN $\rightarrow$ det nc | SN SP,

SV $\rightarrow$ v SN | SV SP, SP $\rightarrow$ prep SN, det $\rightarrow$ o,

v $\rightarrow$ vi, prep $\rightarrow$ em | com, nc $\rightarrow$ homem | monte | telescópio}

Find the five trees to the sentence *Vi o homem no (em + o) monte com o telescópio*. Can you associate a different interpretation to each one of the syntactic trees?

# Chapter 9

# Symbolic Representation of Language

As you know, the field where we study the meaning of words, sentences and documents is generally called **Semantics**. The enterprise of designing meaning representations and associated semantic parsers is referred to as **Computational Semantics**. In the field of computational semantics, researchers also try to **understand natural language**, a long-standing goal of Artificial Intelligence.

In this chapter, we will focus on the frameworks that share the notion that a meaning representation consists of structures composed from **a set of symbols**, **human-interpretable**, corresponding to objects, properties of objects or relations among objects in some world. We will start by talking about how to represent the meaning of words and sentences, but also how we (systematically) get to the meaning of sentences (based on the meaning of the words and phrases).

## 9.1 Symbolic semantic representation (in brief)

### 9.1.1 What makes a good semantic representation?

According to Jurafsky [5], this is what makes a good semantic representation:

- **Expressiveness**: the formalism should be expressive enough to handle the phenomena we are dealing with. For instance, if you want to represent the meaning of a whole book it should be clear that Propositional Logic is not expressive enough. In fact, to be completely honest, if you want to represent the meaning of a book, there is no formalism expressive enough to do it.
- **Canonical Form**: here, the idea is that inputs with the same meaning (Remember? Paraphrases!) should have a single semantic representation. For instance, consider that you want to create a natural language interface with a database about cinema. If you want to capture the semantics of *Where was Keanu Reeves born?* and *What was Keanu Reeves birth place?*, you should get a formula such as born(Keanu_Reeves) to represent both sentences, and not an extra one such as birth_place(Keanu Reeves). This makes the mapping step more complicated (semantic analysis), but reasoning becomes easier.

- **Verifiability**: it must be possible to use the chosen representation to determine the relationship between the meaning of a sentence and the world (our application domain), that is to say, it should be possible to compare or match the representation of the meaning of an input against the store of information about the world. For instance, if I'm able to map *Where was Keanu Reeves born?* into the appropriate representation born(Keanu_Reeves), but I'm unable to interpret this one against the knowledge base, then I'm not able to verify it.
- **Unambiguous Representations**: an input can have different meaning representations assigned to it. Regardless of any ambiguity in the input, it is critical that a meaning representation language provides representations with a single unambiguous interpretation (otherwise semantic representations should be designed in order to capture ambiguities). For instance, consider that we have the predicate howManyOscars, with one argument. If that argument can take values in the set of actors and also in the set of movies, we will not know which table we need to consult.
- **Inference and Variables**: here, the idea is that the system should be able to draw valid conclusions based on the meaning representation of inputs.

Sometimes some of these characteristics are skipped depending on the task we are dealing with. In fact, they are just a wish list. Anyway, nowadays, nobody really cares too much in developing a new (symbolic) meaning representation.

### 9.1.2 First Order Logic and Reification

First Order Logic (FOL) is based on Constants, Variables, Functions, Predicates and Quantifiers. For instance, being:

- DUSTIN and DART constants;
- friend(x, y) a predicate that is true if x is a friend of y;
- likes(x, y) a predicate that is true if x likes y.

then, the sentences:

- *If Dustin likes Dart then Dart is his friend*; and
- *Not all Dustin's friends like Dart*

can be represented in FOL, respectively, as:

- likes(Dustin, Dart) $\rightarrow$ friend(Dart, Dustin)
- $\exists$x[friend(x, Dustin) $\wedge$ $\neg$likes(x, Dart)]

Notice that in FOL, predicates should not be arguments of predicates, so

likes(x, friend(x, y))

is not a well formed FOL formula[1].

But let us move on to a more complicated situation. Consider the following sentences and their (possible) representations:

- *I ate* – eat(I)
- *I ate soup* – eat(I, soup)
- *I ate soup yesterday* – eat(I, soup, yesterday)
- *I ate soup yesterday at "La Ratatouille"* – eat(I, soup, yesterday, LaRatatouille)

Now consider that you want to represent *I ate at "La Ratatouille"*. How would you represent this? eat(I, LaRatatouille) makes no sense as it collides with eat(I, soup). Is eat(I, _, _, LaRatatouille) any better?

In fact, choosing the correct number of arguments for the predicate representing the meaning of eat is a tricky problem. You can implement multiple predicates with different number of arguments. You can also implement one single predicate with all the expected arguments (and the ones not occurring are empty). However, none of these are satisfactory ways of dealing with this problem. Many years ago, Hobbs [4] presented a solution for this: the concept of **reification**. Basically, you introduce a variable to represent an event (or object) and then you use it every time you want to say something about the event (or object). Logical approaches based on reification are known in the literature as 'neo-Davidsonian' approaches, as the concept was originally introduced by the philosopher Donald Davidson. Reification allows us to move from standard notations in FOL to another notation in FOL.

Considering reification, the previous cases would be represented as:

- *I ate* – $\exists$e ISA(e, Eat) $\wedge$ Eater(e, I) // e is a (ISA) event
- *I ate soup* – $\exists$e ISA(e, Eat) $\wedge$ Eater(e, I) $\wedge$ object_ate(e, soup)
- *I ate soup yesterday* – $\exists$e ISA(e, Eat) $\wedge$ Eater(e, I) $\wedge$ object_ate(e, soup) $\wedge$ Time(e, yesterday)
- *I ate soup yesterday at "La Ratatouille"* – $\exists$e ISA(e, Eat) $\wedge$ Eater(e, I) $\wedge$ object_ate(e, soup) $\wedge$ Time(e, yesterday) $\wedge$ Location(e, "LaRatatouille")

Then, in order to represent *I ate at "La Ratatouille"*, we have:

$\exists$e ISA(e, Eat) $\wedge$ Eater(e, I) $\wedge$ Location(e, "LaRatatouille")

Cool, right?
Hum... really? :-(

Here are more examples, in which we also introduce a variable that is associated with an object (such as a book), so that we can say things about that object:

[1] However, it would be ok if friend was a function (and not a predicate), that, for instance, returns the friends of x and y.

- *Pedro reads a book* – ∃x ∃e ISA(e, read) ∧ reader(e, Pedro) ∧ ISA(x, book) ∧ object_read(e, x)
- *Pedro reads a blue book* – ∃x ∃e ISA(e, read) ∧ reader(e, Pedro) ∧ ISA(x, book) ∧ object_read(e, x) ∧ blue(x)
- *A boy reads a book* – ∃x ∃y ∃e ISA(e, read) ∧ ISA(y, boy) ∧ reader(e, y) ∧ ISA(x, book) ∧ object_read(e, x)
- *A nice boy reads a book* – ∃x ∃y ∃e ISA(e, read) ∧ ISA(y, boy) ∧ reader(e, y) ∧ ISA(x, book) ∧ object_read(e, x) ∧ nice(y)

### 9.1.3 Other ways of representing meaning

Not only FOL is used to represent meaning. Much more formalisms exist. For instance, **frames** (roughly, sets of attribute/value pairs) are widely used in the NLP research area of (task-oriented) **Dialogue Systems**. The original concept was coined by Marvin Minsky[2] and was much more complicated. A simplified definition of frame [8] includes:

- a type: types of actions that can be taken by the system;
- a set of slots (attribute-values pairs): pieces of information relevant to the action.

You can find an example of a semantic representation based on frames in Figure 9.1.

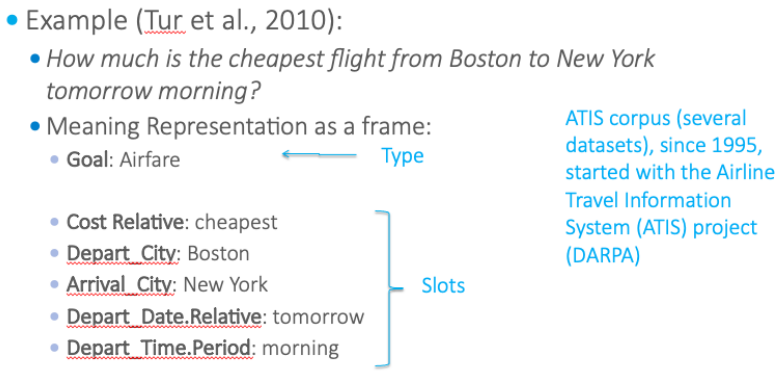


Figure 9.1: Example of a representation based in frames.

As another example, consider an application in the home automation (or domotics) field, where the user can ask a virtual butler to control several devices in his/her house. That is, the butler can be asked to switch on/off the lights of a specific room, change the colour of a table, or switch on/off the television, just to name a few possibilities. In this scenario, frames can be used to represent the user's requests. Possible attributes would be: device, room and action. Examples of values associated with these frames would be *tv* and *table* for the device attribute, *kitchen*, *bedroom* and

[2] If you do not know who Minsky was, please, check it!

*hall* for the room attribute, and *switch on/off*, *turn_blue* or *turn_yellow* for the action attribute.

The next examples shows a frame associated with a request to Ambrósio, a butler developed at INESC-ID many years ago (before Edgar).

```
User: Ambrósio, please switch-off the tv of the living room
<SpeechAct>
<Action><Value>off</Value></Action>
<Room><Value>living-room</Value></Room>
<Device>
<Item><Value>tv</Value></Item>
</Device>
</SpeechAct>
```

Finally, I should mention another formalism: the Abstract Meaning Representation (AMR) [1], based on rooted, directed, edge-labeled, leaf-labeled graphs. Figure 9.2 illustrates this concept.

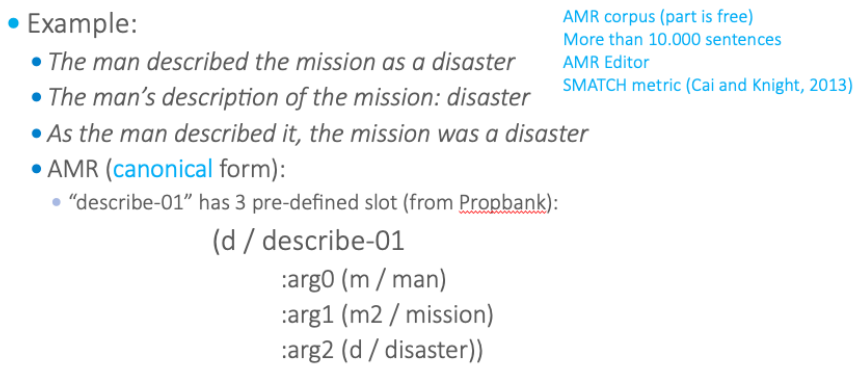


Figure 9.2: Example of a representation in AMR.

To be totally honest, this was the last semantic representation I heard about. There are, however, many recent works that try to map natural language into AMR. The mapping process, although just the classic version, is the target of the next section.

## 9.2 Semantic analysis

So, now the question is: how to automatically map a sentence in natural language into its semantic representation? Let's see how to do this. You already had some insights from professor Nuno classes.

### 9.2.1 Semantic analysis – the compositional process

In this chapter, we are considering a symbolic representation of language, and we will go a little further in the semantic analysis based on a compositional process, showing

how to take advantage of a grammar to attain a semantic representation of a sentence in FOL. Remember your LP classes? (some of you will remember) You were asked to represent sentences in FOL. However, no algorithm was taught to you, so that you could do it systematically. In one of the next classes you will learn another notation associated with this process. Now, we will learn how to do it in a very simple way.

Richard Montague (Figure 9.3) was the first person to propose, in the beginning of the 70s, a framework to map syntactic structures into semantic representations. That is to say: he was responsible for the first advances in finding a systematic way of mapping NL into a semantic representation. His framework used syntax as the support to the semantic analysis, and one of his famous sentences is:

*I fail to see any great interest in syntax except as a preliminary to semantics* (Richard Montague)[3]

Figure 9.3: Richard Montague.

His ideas were at the basis of what is called **compositional semantic analysis**:

- a **semantic representation** is given to words;
- each syntactic rule has a **semantic rule** associated (**syntax/semantic parallelism**);
- the semantic analysis is made in a bottom-up process, by using semantic rules in order to compose the semantics of the constituents.

Consider a CFG. Each syntactic rule of such grammars (A $\rightarrow \alpha_1 ... \ \alpha_n$) can be enriched with a semantic rule, that shows how to combine the associated elements, as follows:

A $\rightarrow \alpha_1 ... \ \alpha_n$ $\{sem_i(sem_j(\alpha_1),... \ , \ sem_k(\alpha_n))\}$

This means that the meaning representation assigned to the constituent A can be computed by functions $sem_i$, which is fed by some subset of the semantics of A's constituents.

Making things very simple, we will use the following operators to combine the semantics of the constituents:

[3] He also said *I reject the contention that an important theoretical difference exists between formal and natural languages.*. I think he was totally wrong about this one.

- X.sem, which denotes the semantic of X;
- replace(A N B), which denotes the formula obtained by inserting A in the Nth position of B.

As an example, consider...

a) the following grammar:
S → NP VP {replace (NP.sem 1 VP.sem)} // S for sentence, NP for Noun Phrase and VP for Verb Phrase
VP → VI {VI.sem} | VT NP {replace (NP.sem 2 VT.sem)}
NP → pn {pn.sem} // pn for proper noun
pn → Manuel | Maria
VI → snores // VI for intransitive verb
VT → adores // VT for transitive verb
b) the meaning representations (right) associated with the lexicon (left):
snores: snore(X)
adores: adore(X, Y)
Manuel: MANUEL01
Maria: MARIA1

Then, let us:

1. Assign a semantic representation for *Manuel snores*;
2. Assign a semantic representation for *Manuel adores Maria*.

First you build the syntactic tree. Then, you apply the rules, in a bottom-up process. When you reach the S (sentence) you get the meaning representation of the sentence. Results can be seen in Figure 9.4. The wildcards indicate the semantics obtained in each part of the syntactic tree.

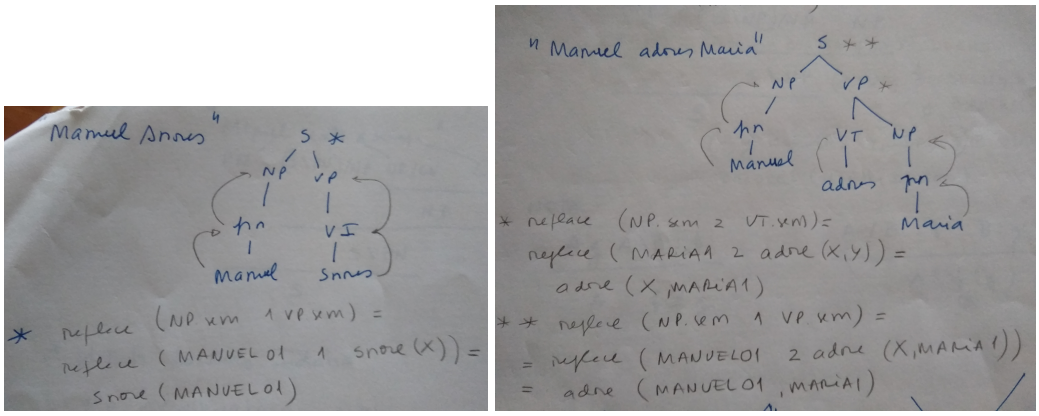


Figure 9.4: Manuel Snores and Manuel adores Maria.

A problem that may arise from this approach: ill-formed formulas may result from the composition process. As previously said, a predicate cannot have another predicate inside. As an example, consider:

- the rule replace(NP.sem 1 PP.sem) // PP for Propositional Phrase
- the NP with semantic representation *hotel(X)*, and
- the PP with semantic representation *from(Y, Lisbon)*,

The direct application of the rule will results in the formula *from(hotel(X), Lisbon)*, which is not a good one, as the predicate hotel/1 is inside the predicate from/2. In order to avoid this problem, you should abstract the variable associated with the predicate that is going to appear inside the other predicate and only use this variable in the substitution/replacement, keeping the predicate in the main formula. Considering the previous example, you should obtain:

*hotel(X)* $\wedge$ *from(X, Lisbon)*.

Another problem is that these approaches consider that the **meaning/semantic of a sentence** is associated with its **literal (context independent) meaning**. Therefore, the meaning of a sentence is captured based on the semantics of its words (lexical semantics) and syntactic structure. As you should know, the meaning of each word is important to reach a correct representation of a sentence, but the way words are organised in a sentence (syntax) or in a text (discourse) is also very important. It should be clear that context brings a major contribution. In order to have a really good semantic representation all these factors (and others) should be considered.

### 9.2.2 Semantic analysis – other approaches

Early systems were rule-based. Then, statistical (compositional) semantic parsers emerged. When enough data is present you can probably try a deep learning approach.

We will also see how to obtain the semantic of a sentence when we are dealing with vectors. We will see that it is much simpler than this hand-crafted stuff. However, it is impossible for us, humans, to understand the meaning of a sentence in their vectorial form; the logic version is, undoubtedly, more comprehensive.

# Bibliography


[1] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider, *Abstract Meaning Representation for sembanking*, Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse (Sofia, Bulgaria), Association for Computational Linguistics, August 2013, pp. 178–186.

[2] Jacob Cohen, *A coefficient of agreement for nominal scales*, Educational and Psychological Measurement **20** (1960), no. 1, 37.

[3] Joseph L Fleiss, *Measuring nominal scale agreement among many raters.*, Psychological bulletin **76** (1971), no. 5, 378.

[4] Jerry R. Hobbs, *Ontological promiscuity*, Proc. ACL'85, University of Chicago, Association for Computational Linguistics, 1985, pp. 61–69.

[5] Daniel Jurafsky and James H. Martin, *Speech and language processing (2nd edition) (prentice hall series in artificial intelligence)*, 2 ed., Prentice Hall, May 2008.

[6] V. I. Levenshtein, *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Physics Doklady **10** (1966), 707–710.

[7] João Silva, *Qa+ml@wikipedia&google*, Master's thesis, Instituto Superior Técnico, Technical University of Lisboa, 2009.

[8] Wayne Ward and Sunil Issar, *Recent improvements in the cmu spoken language understanding system*, Proceedings of the Workshop on Human Language Technology (USA), HLT '94, Association for Computational Linguistics, 1994, p. 213–216.

[9] Joseph Weizenbaum, *Eliza - a computer program for the study of natural language communication between man and machine*, Commun. ACM **9** (1966), 36–45.