Computational Linguistics: Syntax II

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani, Room: 2.21, e-mail: bernardi@inf.unibz.it

Contents

1. Summary

The main issues of last lecture were:

- Syntax: Constituents; Head; Dependencies.
- NL Syntax cannot be formalized by RL.
- CFG can be used to recognize NL syntax, but
- In NL, there are evidences of cross-dependencies

2. 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]

Today we will speak about the problems "the grammar" has to face, next week we will move to speak about the "parsing techniques".

3. Exercise

Try exercises 1. a)-b)

4. Undergeneration and Overgeneration

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

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

5. 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

5.1. 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?

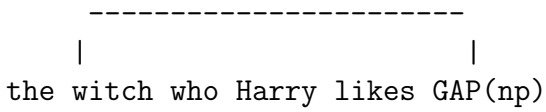
Today we will briefly see a fairly traditional explanation in terms of movement, gaps, extraction, and so on. In the second part of the course, we will look into more modern approaches.

6. 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.

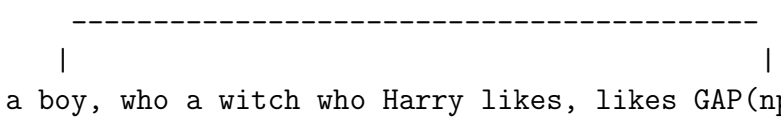


That is, we have

1. extracted the np "the witch" from the object 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.

6.1. 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



7. Overgeneration: Agreement

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

1. He hates a red shirt
2. *He like a red shirt
3. He hates him
4. *He hates he

7.1. Agreement between SUB and Verb

When working with agreement a first important fact to be taken into account is that we use

- a plural VP if and only if we have a plural NP, and
- a singular VP if and only if we have a singular NP.

For instance,

1. "the gangster dies"
2. *"the gangster die"

That is, we have to distinguish between singular and plural VPs and NPs.

7.2. First try

One way of doing this would be to **invent new non-terminal symbols** for plural and singular NPs and VPs. Our grammar would then look as follows.

We would have two rules for building sentences: one for building a sentence out of a singular NP (NPsg) and a singular VP (VPsg), and the other one for using a plural NP (NPpl) with a plural VP (VPpl).

Singular NPs are built out of a determiner and a singular noun (Nsg) and plural NPs are built out of a determiner and a plural noun (Npl). Note that we don't have to distinguish between singular and plural determiners as we are only using "the" at the moment, which works for both.

Similarly, singular VPs are built out of singular intransitive verbs (IVsg) and plural VPs out of plural intransitive verbs (IVpl).

Finally, we have singular and plural nouns and verbs in our lexicon.

7.3. Loss of efficiency

Now, the grammar does what it should:

1. "the gangster dies"
2. "the gangsters die"
3. *"the gangster die"
4. *"the gangsters dies".

However, compared to the grammar we started with, it has become **huge** – we have twice as many phrase structure rules, now. And we only added the information whether a noun phrase or a verb phrase is plural or singular.

Imagine we next wanted to add transitive verbs and pronouns. To be able to correctly accept "he shot him" and reject "him shot he", we would need case information in our grammar. And if we also wanted to add the pronouns "I" and "you", we would further have to distinguish between first, second and third person.

If we wanted to code all this information in the non-terminal symbols of the grammar, we would need non-terminal symbols for all combinations of these features. Hence, the **size of the grammar would explode** and the rules would probably become very difficult to read.

7.4. Second Try

We use features to represent case (subject, object), gender (female, masculine), number (singular, plural).

```
s --> np(subj), vp.
vp --> vt, np(obj).
vp --> vi.
np(CASE) --> pro(CASE).
np(_) --> det, n.
np(_) --> pn.
Lexicon
det --> the
n --> whiskey
pn --> bill
pro(subj) --> he
pro(obj) --> him
vi --> fights
vt --> kills
```

Try the second exercise.

7.5. Second try (cont'd)

While doing the exercise, you might have noticed that the extra argument — the feature — is simply **passed up the tree** by ordinary **unification**. And, depending on whether it can correctly unify or not, this feature controls the facts of English case avoiding duplications of categories.

Summing up,

- features let us get rid of lots of unnecessary rules in a natural way.
- In the lab we will see that PROLOG enables us to implement rules with feature information in a natural way.

This way of handling features however has some limits, in particular it does not provide an adequate syntax descriptions of the agreement phenomena in general terms.

7.6. Feature Pergolation

Last time we have spoken of the **head** of the phrase as the word characterizing the phrase itself. E.g. the head of a noun phrase is the noun, the head of a verb phrase is the verb, the head of a prepositional phrase is the preposition, etc.

Notice that its the head of a phrase that **provides the features of the phrase**. E.g. in the noun phrase "this cat", it's the noun ("cat") that characterizes the np as singular.

Note, this also means that the noun requires the article to match its features.

7.7. Set of properties

This can be captured in an elegant way, if we say that our **non-terminals are** no longer atomic category symbols, but a **set of properties**, such as type of category, number, person, case ....

Certain rules can then impose **constraints** on the individual properties that a category involved in that rule may have.

These constraints can force a certain property to have **some specific value**, but can also just say that two properties must have the **same value**, no matter what that value is. Using this idea, we could specify our grammar like this:

```
s ---> np  vp : number of np= number of vp
np ---> Det  n : number of np= number of n
vp ---> iv
Det ---> the
n ---> gangster : number of n= singular
n ---> gangsters : number of n= plural
iv ---> dies: number of iv = singular
iv ---> die : number of iv = plural
```

8. Constraint Based Grammars

In computational linguistics such sets of properties are commonly represented as feature structures.

The grammars that use them are known as **constraint-based** grammars, i.e. grammars that can express **constrains on the properties** of the categories to be combined by means of its rules. Roughly, a rule would have to say

$$s \rightarrow np\ vp$$

only if the number of the *np* is equal to the number of the *vp*.

The most well known Constraint Based Grammars are Lexical Functional Grammar (LFG, Bresnan '82), Generalized Phrase Structure Grammar (GPSG, Gazdar et al. '85), Head-driven Phrase Structure Grammar (HPSG, Pollard and Sag, '87), Tree Adjoining Grammar (TAG, Joshi et al. '91).

9. Feature Structures

Constraints-Based Grammars usually encode properties by means of **Feature Structures** (FS). They are simply sets of feature-value pairs, where features are unalayzable atomic symbols drown from some finite set, and values are either atomic symbols or feature structures.

They are traditionally illustrated with the following kind of matrix-like diagram, called **attribute-value matrix (AVM)**:

$$\begin{bmatrix} \texttt{Feature}_1 & \texttt{Value}_1 \\ \texttt{Feature}_2 & \texttt{Value}_2 \\ \ldots & \ldots \\ \texttt{Feature}_n & \texttt{Value}_n \end{bmatrix}$$

For instance, the number features `sg` (singular) and `pl` plural, are represented as below.

$$\begin{bmatrix} \texttt{NUM} & sg \end{bmatrix} \quad \begin{bmatrix} \texttt{NUM} & pl \end{bmatrix}$$

Similarly, the slightly more complex feature 3rd singular person is represented as

$$\begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix}$$

Next, if we include also the category we obtain, e.g.

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix}$$

which would be the proper representation for "Raffaella" and would differ from the FS assigned to "they" only with respect to (w.r.t.) the number.

Therefore, FS give a way to encode the information we need to take into consideration in order to deal with **agreement**. In particular, we obtain a way to encode the constraints we have seen before.

10. Agreement Feature

In the above example all feature values are atomic, but they can also be feature structures again. This makes it possible to group features of a common type together.

For instance, the two important values to be considered for agreement are `NUM` and `PERS`, hence we can group them together in one `AGR` feature obtaining a more compact and efficient representation of the same information we expressed above.

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix} \end{bmatrix}$$

Given this kind of arrangement, we can test for the equality of the values for both `NUM` and `PERS` features of two constituents by testing for the equality of their `AGR` features.

11. Feature Path

A **Feature Path** is a list of features through a FS leading to a particular value. For instance, in the FS below

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix} \end{bmatrix}$$

the $\langle$AGR NUM$\rangle$ path leads to the value sg, while the $\langle$AGR PERS$\rangle$ path leads to the value 3.

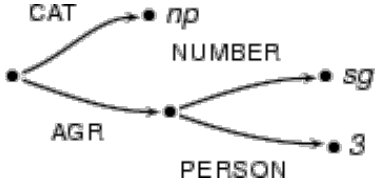
This notion of paths brings us to an alternative graphical way of illustrating FS, namely directed graphs.

11.1. Directed Graphs

Another common way of representing feature structures is to use directed graphs. In this case, values (no matter whether atomic or not) are represented as nodes in the graph, and features as edge labels. Here is an example. The attribute value matrix

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix} \end{bmatrix}$$

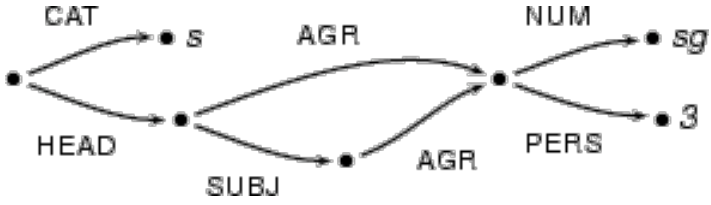
can also be represented by the following directed graph.



Paths in this graph correspond to sequences of features that lead through the feature structure to some value. The path carrying the labels `AGR` and `NUM` corresponds to the sequence of features ⟨`AGR, NUM`⟩ and leads to the value `sg`.

11.2. Reentrancy

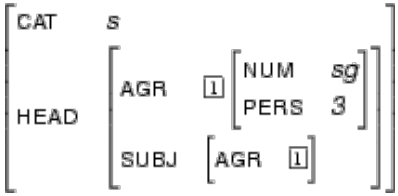
The graph that we have just looked at had a tree structure, i.e., there was no node that had more than one incoming edge. This need not always be the case. Look at the following example:



Here, the paths ⟨`Head, AGR` ⟩ and ⟨`Head, SUBJ, AGR` ⟩ both lead to the same node, i.e., they lead to the same value and **share that value**. This property of feature structures that several features can share one value is called **reentrancy**. It is one of the reasons why feature structures are so useful for computational linguistics.

11.3. Reentrancy as Coindexing

In attribute value matrices, reentrancy is commonly expressed by coindexing the values which are shared. Written in the matrix notation the graph from above looks as follows. The **boxed 1** indicates that the two features sequences leading to it **share the value**.



11.4. FS: Subsumption

We have said that feature structures are essentially sets of properties. Given two different sets of properties an obvious thing to do is to **compare the information they contain**.

A particularly important concept for comparing two feature structures is **subsumption**.

A feature structure `F1` subsumes ($\sqsubseteq$) another feature structure `F2` iff all the information that is contained in `F1` is also contained in `F2`.

11.5. Examples

The following two feature structures for instance subsume each other.

$$\begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix} \quad \begin{bmatrix} \texttt{PERS} & 3 \\ \texttt{NUM} & sg \end{bmatrix}$$

They both contain exactly the same information, since the order in which the features are listed in the matrix is not important.

11.6. Exercise

And how about the following two feature structures?

$$\begin{bmatrix} \texttt{NUM} & sg \end{bmatrix} \qquad \begin{bmatrix} \texttt{PERS} & 3 \\ \texttt{NUM} & sg \end{bmatrix}$$

Well, the first one subsumes the second, but not vice versa. Every piece of information that is contained in the first feature structure is also contained in the second, but the second feature structure contains additional information.

11.7. Exercise: (Cont'd)

Do the following feature structures subsume each other?

$$\begin{bmatrix} \texttt{NUM} & sg \\ \texttt{GENDER} & masc \end{bmatrix} \quad \begin{bmatrix} \texttt{PERS} & 3 \\ \texttt{NUM} & sg \end{bmatrix}$$

The first one doesn't subsume the second, because it contains information that the second doesn't contain, namely `GENDER masc`.

But, the second one doesn't subsume the first one either, as it contains `PERS 3` which is not part of the first feature structure.

12. Operations on FS

The two principal operations we need to perform of FS are **merging** the information content of two structures and **rejecting** the merger of structures that are incompatible.

A single computational technique, namely **unification**, suffices for both of the purposes.

Unification is implemented as a binary operator that accepts two FS as arguments and returs a FS when it succeeds.

12.1. Unification of FS

Unification is a (partial) operation on feature structures. Intuitively, it is the operation of combining two feature structures such that the new feature structure contains **all the information of the original two, and nothing more**. For example, let F1 be the feature structure

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & sg \end{bmatrix} \end{bmatrix}$$

and let F2 be the feature structure

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{PERS} & 3 \end{bmatrix} \end{bmatrix}$$

Then, what is F1 $\sqcup$ F2, the unification of these two feature structures?

$$\begin{bmatrix} \texttt{CAT} & np \\ \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & sg \\ \texttt{PERS} & 3 \end{bmatrix} \end{bmatrix}$$

12.1.1. Partial Operation Why did we call unification a **partial operation**? Why didn't we just say that it was an operation on feature structures?

The point is that unification **is not guaranteed to return a result**. For example, let `F3` be the feature structure

$$\left[\ \texttt{CAT}\quad np\ \right]$$

and let `F4` be the feature structure

$$\left[\ \texttt{CAT}\quad vp\ \right]$$

Then `F3` $\sqcup$ `F4` does not exist. There is no feature structure that contains all the information in `F3` and `F4`, because the information in these two feature structures is contradictory. So, the value of this unification is undefined.

13. Augmenting CFG with FS

We have seen that agreement is necessary, for instance, between the np and vp: they have to agree in number in order to form a sentence.

The basic idea is that non-terminal symbols no longer are atomic, but are feature structures, which specify what properties the constituent in question has to have.

So, instead of writing the (atomic) non-terminal symbols `s`, `vp`, `np` , we use feature structures `CAT` where the value of the attribute is `s`, `vp` , `np` . The rule becomes

$$[\texttt{CAT}\ s] \rightarrow [\texttt{CAT}\ np]\ \ [\texttt{CAT}\ vp]$$

That doesn't look so exciting, yet.

14. Augmenting CFG wiht FS (cont'd)

But what we can do now is to add further information to the feature structures representing the non-terminal symbols. We can, e.g., add the information that the `np` must have nominative case:

$$[\texttt{CAT}\ s] \rightarrow \begin{bmatrix} \texttt{CAT} & np \\ \texttt{CASE} & nom \end{bmatrix} [\texttt{CAT}\ vp]$$

Further, we can add an attribute called `NUM` to the `np` and the `vp` and require that the values be shared. Note how we express this requirement by co-indexing the values.

$$[\texttt{CAT}\ s] \rightarrow \begin{bmatrix} \texttt{CAT} & np \\ \texttt{CASE} & nom \\ \texttt{NUM} & \boxed{1} \end{bmatrix} \begin{bmatrix} \texttt{CAT} & vp \\ \texttt{NUM} & \boxed{1} \end{bmatrix}$$

See course web site for a project on this. Parsing with Feature Structures (PATR). (two persons: morphological analyzer plus syntactic analyser).

14.1. Exercise

Try to build a feature based grammar for the (tiny) fragment of English below.

Lexicon

```
det --> a
det --> the
n --> bride
n --> whiskey
pn --> bill
pn --> gogo
pro(subj) --> he
pro(subj) --> she
pro(obj) --> him
pro(obj) --> her
vi --> whistles
vi --> fights
vt --> drinks
vt --> kills
```

14.2. Head Features and Subcategorization

We have seen that to "put together" words to form constituents two important notions are the "head" of the constituent and its dependents (also called the arguments the head subcategorize for).

In some constraints based grammars, e.g. HPSG, besides indicating the category of a phrase, FS are used also to sign the head of a phrase and its arguments.

In these grammars, the `CAT` (category) value is an object of sort category (cat) and it contains the two attributes `HEAD` (head) and `SUBCAT` (subcategory).

Head Recall, the features are percolated from one of the children to the parent. The child that provides the features is called the **head** of the phrase, and the features copied are referred to as head features. Therefore, the `HEAD` value of any sign is always unified with that of its phrasal projections.

Subcategorization The notion of subcategorization, or valence, was originally designed for verbs but many other kinds of words exhibit form of valence-like behavior. This notion expresses the fact that such words determine which patterns of argument they must/can occur with. They are used to express **dependencies**.

For instance,

1. an intransitive verb subcategorizes (requires) a subject.
2. a transitive verb requires two arguments, an object and a subject.
3. ...

Other verbs

- want [to see a girl called Evelyn]$_{\texttt{Sto}}$
- asked [him]$_{\texttt{NP}}$ [whether he could make it]$_{\texttt{Sif}}$

14.3. Schema

Schematically the subcategorization is represented as below.

$$\begin{bmatrix} \texttt{ORTH} & \textit{word} \\ \texttt{CAT} & \textit{category} \\ \texttt{HEAD} & \left[\texttt{SUBCAT} \quad \langle 1st\ required\ argument, 2nd\ required\ argument, \ldots \rangle \right] \end{bmatrix}$$

14.4. Example

For instance, the verb "want" would be represented as following

$$\begin{bmatrix} \texttt{ORTH} & \textit{want} \\ \texttt{CAT} & \textit{verb} \\ \texttt{HEAD} & \left[\texttt{SUBCAT} \quad \langle [\texttt{CAT}\ np], \left[\begin{matrix} \texttt{CAT} & vp \\ \texttt{HEAD} & [\texttt{VFORM}\ INFINITIVE] \end{matrix} \right] \rangle \right] \end{bmatrix}$$

Exercise nr. 3.

15. Conclusion

The model of subcategorization we have described so far helps us solving the over-generation problem described last time. However, we still have to see how to deal with long-distance dependencies.

We will return to this in May.

Not done on FS:

1. Implementing Unification
2. Parsing with Unification Constraints
3. Types and Inheritance

Projects 1, and 2 are possible topics for projects. Another possible project on today topic is to build a Constraint Based Grammar for a fragment of a language of your choice.

Next week, we will look at parsing techniques. Then we move to semantics and discourse.

16. Practical Info

Topics for projects so far:

- PoS Tagging.
- Formal Grammars: present some FGs not discussed in class, or in depth presentation about HPSG.
- Formal Grammars: CFG+FS implementation: (programming languages?).
- Parsing techniques.
- Semantics: extending CFG with lambda calculus (Prolog).
- Semantics: underspecification.
- Projects within our project on IQA (BoB): eg. QA recognizers, learning models. etc..

Suggestion: Make up your mind by the 24th of April –when we have covered all these topics already – and you have one month to work on it (while doing all the other studies)