Computational Linguistics: Feature Agreement

Raffaella Bernardi

Contents

1. Formal Grammars

We have seen

- that NL syntax cannot be represented by a Regular Language, because it has nested dependencies $a^n b^n$, $a^n b^m c^m c^n$
- how to use CFG to recognize/generate and parse NL strings.
- that FGs can be weakly or strongly equivalent.

Today we introduce Feature Structures and augument CFG with features.

1.1. Recall: 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.

1.2. Undergeneration: Long-distance dep.

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?

1.3. Relative clauses

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.

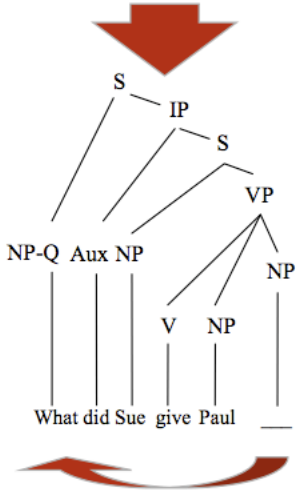
```
      ------------------------
     |                        |
the witch who Harry likes GAP(np)
```

That is, we have

1. extracted the np "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.

The Transformational Tradition (cont.)

Sue gave Paul an old penny



1.4. 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

2. Features and values

A 'linguistic feature" is a property-like element that changes the grammatical behaviour of syntactic constituents:

- **person**: I go, you go, he goes
- **number**: he dances, they dance
- **case**: he brings John, John brings him
- **tense**: go, went, gone

- **person**: 1st, 2nd, 3rd
- **number**: singular, plural,
- **case**: accusative, locative etc
- **tense**: past, present, future,

See more at: http://grammaticalfeatures.net/ and RZ's course.

2.1. 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.

2.2. 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
```

3. 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).

4. 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)** (It is common practice to refer to AVMs as "feature structures" although strictly speaking they are feature structure **descriptions**.)

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

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.

Note that, the order of rows is unimportant, and within a single AVM, an attribute can only take one value.

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.

5. 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.

6. 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.

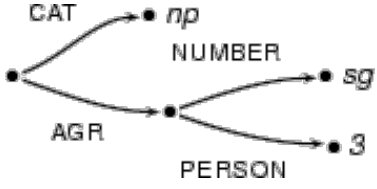
This notion of paths brings us to an alternative graphical way of illustrating FS, namely directed graphs.

6.1. 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

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

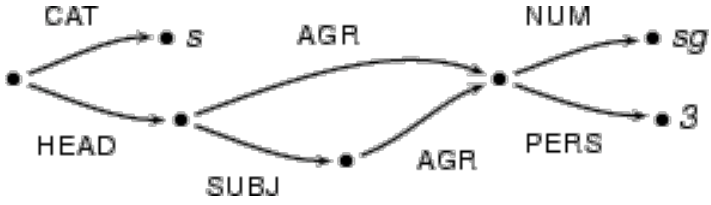
can also be represented by the following directed graph.



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`.

6.2. Reentrancy

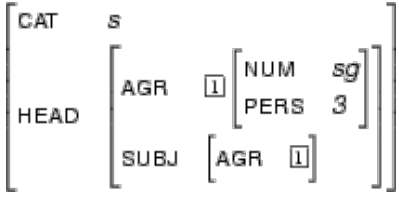
The graph that we have just looked at had a tree structure, i.e., there was no node that had more than one incoming edge. This need not always be the case. Look at the following example:



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.

6.3. Reentrancy as Coindexing

In AVM, 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**.



6.4. 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**.

We say that a less specific feature structure subsumes an equally or more specific one, namely a feature structure `F1` subsumes ($\sqsubseteq$) another feature structure `F2` iff all the information that is contained in `F1` is also contained in `F2`.

The minimum element w.r.t. the subsumption ordering is the feature structure that specifies no information at all (no attributes, no values). It is called the "top" and is written `T` or [ ]. Top subsumes every other AVM, because every other AVM contains at least as much information as top.

6.5. FS: Formal definition of Subsumption. RVD

A feature structure `F1` subsumes ($\sqsubseteq$) another feature structure `F2` iff

- For every feature x in $F1$, $F1(x) \sqsubseteq F2(x)$ (where $F1(x)$ means "the value of the feature x of feature structure $F1$"),
- For all paths p and q in $F1$ such that $F1(p) = F1(q)$, it is also the case that $F2(p) = F2(q)$.

Notice that subsumption is reflexive, transitive and anti-symmetric.

6.6. 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.

6.7. 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.

6.8. 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.

7. Operations on FS

The two principal operations we need to perform on 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.

7.1. 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 $\texttt{F1} \sqcup \texttt{F2}$, the unification of these two feature structures?

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

7.1.1. 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. (It's result is marked by $\perp$, i.e. an improper AVM that cannot describe any object (the opposite of `T`).)

7.1.2. Unification: Formal Definition Those are the basic intuitions about unification, so let's now give a precise definition. This is easy to do if we make use of the idea of subsumption, which we discussed above.

The unification of two feature structures F and G (if it exists) is the **smallest** feature structure that is subsumed by both F and G. That is, (if it exists) F ⊔ G is the feature structure with the following three properties:

1. F ⊑ F ⊔ G (F ⊔ G is subsumed by F)
2. G ⊑ F ⊔ G (F ⊔ G is subsumed by G)
3. If H is a feature structure such that F ⊑ H and G ⊑ H, then F ⊔ G ⊑ H (F ⊔ G is the smallest feature structure fulfilling the first two properties. That is, there is no other feature structure that also has properties 1 and 2 and subsumes F ⊔ G.)

If there is no smallest feature structure that is subsumed by both F and G, then we say that the unification of F and G is **undefined**.

7.2. Unification: Examples

$$\begin{bmatrix} \texttt{NUMBER} & sg \end{bmatrix} \sqcup \begin{bmatrix} \texttt{PERSON} & 3rd \end{bmatrix} = \begin{bmatrix} \texttt{NUMBER} & sg \\ \texttt{PERSON} & 3rd \end{bmatrix}$$

$$\begin{bmatrix} \texttt{NUMBER} & sg \end{bmatrix} \sqcup \begin{bmatrix} \texttt{NUMBER} & [] \end{bmatrix} = \begin{bmatrix} \texttt{NUMBER} & sg \end{bmatrix}$$

[] indicates that the value is left unspecified, hence it can be successfully matched to any value in a corresponding feature in another structure.

8. 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.

9. 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.

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

9.1. Head Features and Subcategorization

We've 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).

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.

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**.

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

See COMLEX set (Macelod et. al. '98) for a subcategorization-frame tagset.

9.2. FG with Head and Subcategorization information

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). The HEAD value of any sign is always unified with that of its phrasal projections.

Schema Schematically the subcategorization is represented as below.

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

9.3. Example

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

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

Also other words have subcategorization frames. For instance, the prepositions **while** vs. **during**

- Keep your seathbelt fasted while we are taking off.
- *Keep your seathbelt fasted while takeoff
- *Keep your seathbelt fasted during we are taking off.
- Keep your seathbelt fasted during takeoff

Exercises

9.4. Home-work

- For tomorrow, remember to bring the exercises on CFG.
- For the 03.10, who is writing a short summary of the following paper and summarize it to the others (with or without slides)?

  "Head-Driven Phrase Structure Grammar Linguistic Approach, Formal Foundations, and Computational Realization" by Robert D. Levine and W. Detmar Meurers

9.5. Not done on FS:

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

If you know Python and what to learn more on FS the http://www.nltk.org/ has a module nltk.featstruct. Try to instal NLTK by Monday.