Computational Linguistics: Syntax II

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani, Room: 2.28, e-mail: bernardi@inf.unibz.it

Contents

1. Recall

We have spoken of:

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

- Linguistic Theories
  1. Morphology: Stems vs. Affixes; Inflectional and derivational forms.
  2. Syntax: Constituents; Head; (local, and long-distance) Dependencies.
- Natural Language as Formal Language
  1. Morphology can be formalized by means of Regular Languages and as such modeled by FSA.
  2. Syntax cannot be formalized by RL. We have seen how to use CFG to recognize NL syntax. However, the basic CFG we have looked at both overgenerates (agreement) and undergenerates (long-distance dependencies).
- Implementation
  1. We have used PROLOG as the programming language to computationally deal with the Linguistic Theory modeled by the Formal Languages.

2. Features

In the lab session you have started familiarizing with the concept of **features**. In particular, you have looked at a way of representing case (subject, object).

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

3. Features (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.
- 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.

4. Today's topics

Today, we will see how to

- use features to account for agreement,
- represent complex feature structures, and
- augment CFG with these feature structures.

5. Agreement

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.

5.1. 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) or plural intransitive verbs (IVpl).

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

5.2. 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.

5.3. Loss of generalization

Another reason why this strategy of adding new non-terminals is inappropriate: it **misses some important generalizations**.

In the extended grammar above, for example, the non-terminals NPsg and NPpl are not more similar to each other than to say VPsg. The generalization that they are both NPs is lost. Similarly, the information that sentence are always built out of a noun phrase and a verb phrase (although their number can vary) is lost.

So, what we would like to say is something like this:

> "Sentences consist of an NP and a VP. They must both be of the same number; either plural or singular."

5.4. 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
```

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. 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).

8. 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.

9. 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.

10. 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.

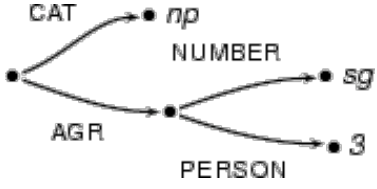
This notion of paths brings us to the an alternative graphical way of illustrating FS, namely directed graphs.

10.1. 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

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

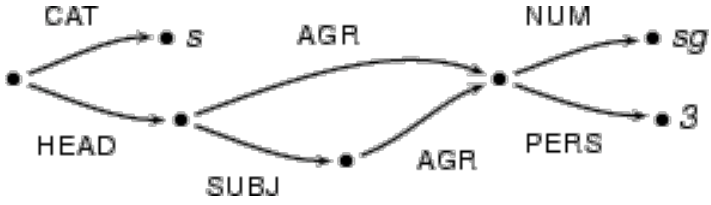
can also be represented by the following directed graph.



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`.

10.2. Reentrancy

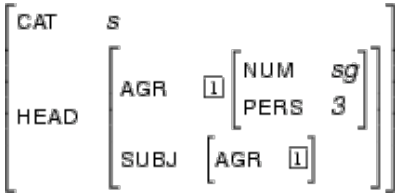
The graph that we have just looked at had a tree structure, i.e., there was no node that had more than one incoming edge. This need not always be the case. Look at the following example:



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.

10.3. Reentrancy as Coindexing

In attribute value matrices, reentrancy is 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 one value**.



10.4. 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`.

10.5. 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.

10.6. 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.

10.7. 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.

11. 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.

11.1. 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?

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

11.1.1. 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.

11.1.2. 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) $\mathtt{F} \sqcup \mathtt{G}$ is the feature structure with the following three properties:

1. $\mathtt{F} \sqsubseteq \mathtt{F} \sqcup \mathtt{G}$ ($\mathtt{F} \sqcup \mathtt{G}$ is subsumed by F)
2. $\mathtt{G} \sqsubseteq \mathtt{F} \sqcup \mathtt{G}$ ($\mathtt{F} \sqcup \mathtt{G}$ is subsumed by G)
3. If H is a feature structure such that $\mathtt{F} \sqsubseteq \mathtt{H}$ and $\mathtt{G} \sqsubseteq \mathtt{H}$, then $\mathtt{F} \sqcup \mathtt{G} \sqsubseteq \mathtt{H}$ ($\mathtt{F} \sqcup \mathtt{G}$ 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 $\mathtt{F} \sqcup \mathtt{G}$.)

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**.

11.1.3. Similarity with set operations The **subsumption** relation on feature structures can be thought of as analogous to the **subset relation** on sets.

Similarly, **unification** is rather like an analog of **set-theoretic union** (recall that the union of two sets is the smallest set that contains all the elements in both sets).

But there is a difference: union of sets is an operation (that is, the union of two sets is always defined) whereas (as we have discussed) unification is only a **partial operation** on feature structures.

11.1.4. Exercises Now that the formal definition is in place, let's look at a few more examples of feature structure unification. First, let F5 be

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

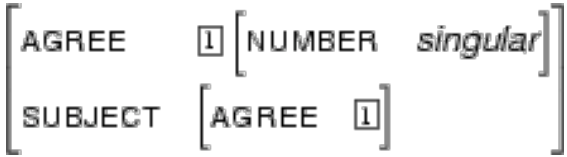
and let F6 be the feature structure

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

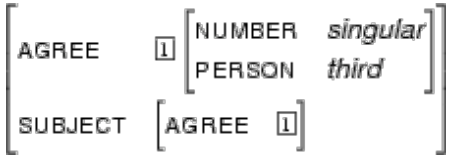
What is the result of F5 $\sqcup$ F6?

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

11.1.5. Exercises: Reentrancy Next, an example involving reentrancies. Let `F7` be



What is then `F7` ⊔ `F6`?



11.1.6. Exercise: Empty feature structure What happens if one of the feature structures we are trying to unify is the empty feature structure `[]` — that is, the feature structure containing no information at all?

Pretty clearly, for any feature structure `F`, we have that:

$\texttt{F} \sqcup [] = [] \sqcup \texttt{F} = \texttt{F}$

That is, the empty feature structure is the identity element for the (partial) operation on feature structure unification. That is, the empty feature structure behaves like the empty set in set theory (recall that the union of the empty set with any set S is just S).

11.2. Generalization

Generalization can be thought of as an analog of the **set-theoretic operation of intersection**. Recall that the intersection of two sets is the set that contains only the elements common to both sets. Similarly, the generalization of two feature structures contains only the information that is common to both feature structures.

For example, let `F9` be

$$\begin{bmatrix} \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & \textit{pl} \\ \texttt{PERS} & 3 \end{bmatrix} \\ \texttt{ARITY} & 3 \end{bmatrix}$$

and let `F10` be

$$\begin{bmatrix} \texttt{AGR} & \begin{bmatrix} \texttt{NUM} & \textit{pl} \\ \texttt{PERS} & 1 \end{bmatrix} \\ \texttt{ARITY} & 1 \end{bmatrix}$$

What is then `F9` $\sqcap$ `F10`, the generalization of `F` and `G`?

$$\left[\ \texttt{AGR}\ \ \left[\ \texttt{NUM}\ \ \mathit{pl}\ \right]\ \right]$$

Clearly $\texttt{F9} \sqcap \texttt{F10}$ contains only information which can be found in both `F` and `G`.

11.3. Generalization: Formal Definition

Here's a precise definition of generalization: The generalization of two feature structures F and G is the largest feature structure that subsumes both F and G. That is, $\mathtt{F} \sqcap \mathtt{G}$ is the feature structure with the following three properties:

1. $\mathtt{F} \sqcap \mathtt{G} \sqsubseteq \mathtt{F}$
2. $\mathtt{F} \sqcap \mathtt{G} \sqsubseteq \mathtt{G}$
3. If H is a feature structure such that$\mathtt{H} \sqsubseteq \mathtt{F}$ and $\mathtt{H} \sqsubseteq \mathtt{G}$, then $\mathtt{H} \sqsubseteq \mathtt{F} \sqcup \mathtt{G}$.

11.4. Unification vs. Generalization

There is an important difference between unification and **generalization**: unlike unification, generalization is **an operation** on feature structures, not just a partial operation. That is, **the generalization of two features is always defined**.

Think about it. Consider the worst case — two feature structures `F` and `G` that contain no common information at all. Is there a largest feature structure that subsumes both? Of course there is — the empty feature structure `[]`!

Thus, from a mathematical perspective, generalization is somewhat easier to work with than unification. Nonetheless, it is not used nearly so often in computational linguistics as unification is, so we won't discuss how to implement it.

12. 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.

13. 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.

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

13.1. Exercise

Try to build a feature based grammar for the (tiny) fragment of English you have worked in the lab and that is repeated below.

```
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].
```

13.2. Head Features and Subcategorization

Last week 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]$_{\texttt{Sif}}$

13.3. Schema

Schematically the subcategorization is represented as below.

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

13.4. Example

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

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

14. Conclusion

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

We will return to this after the winter break.

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 for a fragment of a language of your choice.

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