Computational Linguistics: History & Comparison of Formal Grammars

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

Via della Mostra, Room: 1.06, e-mail: bernardi@inf.unibz.it

Contents

1. Formal Grammars

- We have seen that Formal Grammars play a crucial role in the research on Computational Linguistics.
- We have looked at Context Free Grammars/Phrase Structure Grammars, Categorial Grammar and Lambek calculus

But through the years, computational linguists have developed other formal grammars too.

Today, we will look at the most renown ones, at their generative capacity and their complexity. Next time we will mention some applications.

2. 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.

2.1. 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?

2.2. 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.

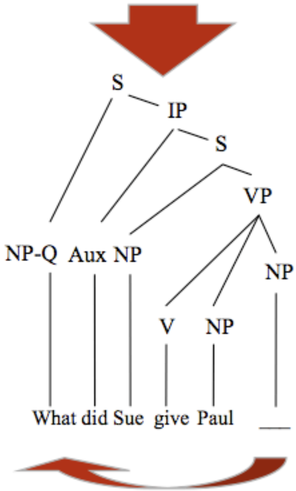
```
      -----------------------
     |                       |
the witch who Harry likes GAP(np)
```

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



3. History of Formal Grammars

Important steps in the historical developments of Formal grammar started in the 1950's and can be divided into five phases:

1. Formalization: Away from descriptive linguistics and behavioralism (performance vs. competence) [1950's 1960's]
2. Inclusion of meaning: Compositionality [1970's]
3. Problems with word order: Need of stronger formalisms [1970's 1980's]
4. Grammar meets logic & computation [1990's]
5. Grammar meets statistic [1990's 2000's]

In these phases, theoretical linguists addressed similar issues, but worked them out differently depending on the perspective they took:

- constituency-based or
- dependency-based.

3.1. Constituency-based vs. Dependency-based

Constituency (cf. structural linguists like Bloomfield, Harris, Wells) is a **horizontal** organization principle: it groups together constituents into phrases (larger structures), until the entire sentence is accounted for.

- Terminal and non-terminal (phrasal) nodes.
- Immediate constituency: constituents need to be adjacent (CFPSG).
- But we have seen that meaningful units may not be adjacent –Discontinuous constituency or long-distance dependencies.
- This problem has been tackled by allowing flexible constituency: "phrasal rebracketing"

Dependency is an asymmetrical relation between a head and a dependent, i.e. a **vertical** organization principle.

3.2. Constituency vs. Dependencies

Dependency and constituency describe different dimensions.

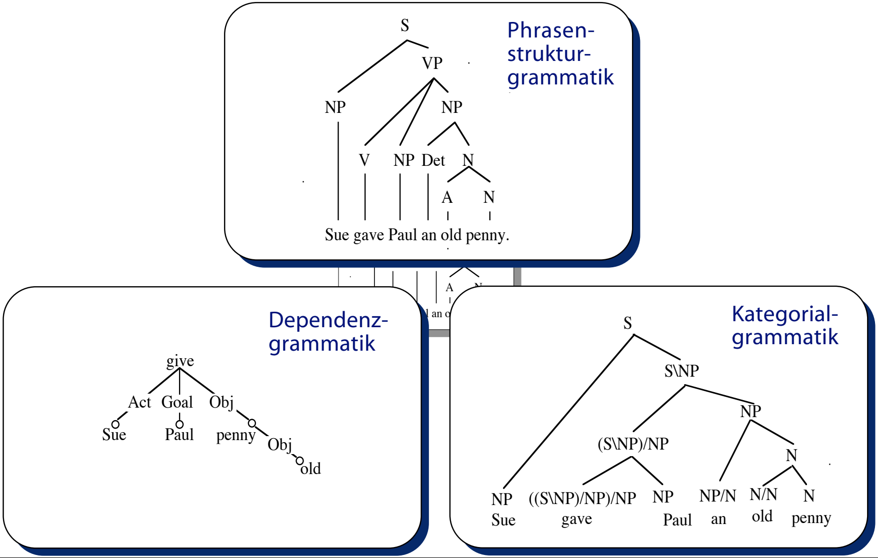
1. A phrase-structure tree is closely related to a derivation, whereas a dependency tree rather describes the product of a process of derivation.
2. Usually, given a phrase-structrue tree, we can get very close to a dependency tree by constructing the transitive collapse of headed structures over nonterminals.

Constituency and dependency are not adversaries, they are complementary notions. Using them together we can overcome the problems that each notion has individually.

Possible topic for a project.

4. DG & CFPSG & CG

THREE TRADITIONS



4.1. Combining Constituency and Dependencies

In 1975, Joshi et ali introduced a grammatical formalism called Tree-Adjoining Grammars (TAGs), which are tree-generating systems. The application of TAGs to natural language is known as LTAGs.

- New way of thinking of domain of dependencies
- Localization of dependencies : elementary structures of a formalisms over which dependencies such as agreement, subcategorization and filler-gap relation can be specified.

5. TAG & CFG

CFG:

```
S --> NP VP       NP --> Harry        ADV --> passionately
VP --> V NP       NP --> peanuts
VP --> VP ADV     V --> likes
```

TAG:

```
a1   S                 a2     NP            a3 NP
   /   \                      |                |
 NP|     VP                   peanuts          Harry
       /    \
      V     NP |
      |
    likes
```

5.1. TAG rules

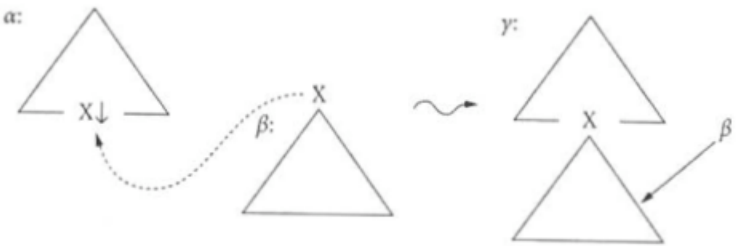


Fig. 26.2 Substitution

5.2. Example

Try to apply the substitution rules to the entries given above:

```
a1    S                 a2     NP             a3 NP
   /    \                      |                 |
 NP|      VP                   peanuts           Harry
       /    \
      V    NP |
      |
   likes
```

What does this rule correspond to in CG?

Do you think this rule is going to be enough?

5.3. Example

"Harry thinks Bill likes John"

what's the entry for "thinks"?

```
       S
     /   \
  NP|     VP
         /  \
        V    S|
        |
      think
```

And what about the sentence "Who does Harry think Bill likes?"

5.4. Example

To account for gaps, new elementary trees are assigned to e.g. TV:

```
       S
    /      \
NP(wh)|     S
          /   \
       NP|     VP
              /  \
             V     NP|
             |     |
           likes empty
```

5.5. Adjunction

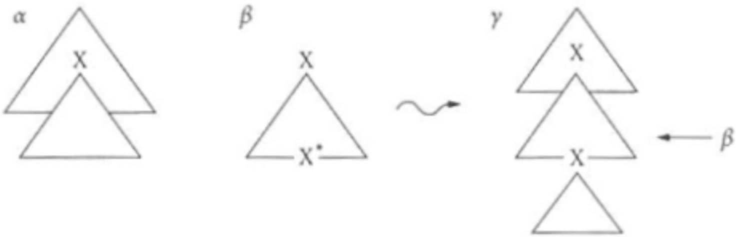
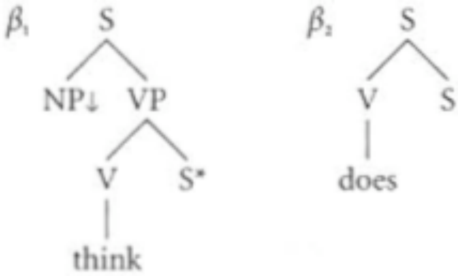


Fig. 26.5 Adjoining

The lexical entries “does” and “think” carry the special marker:



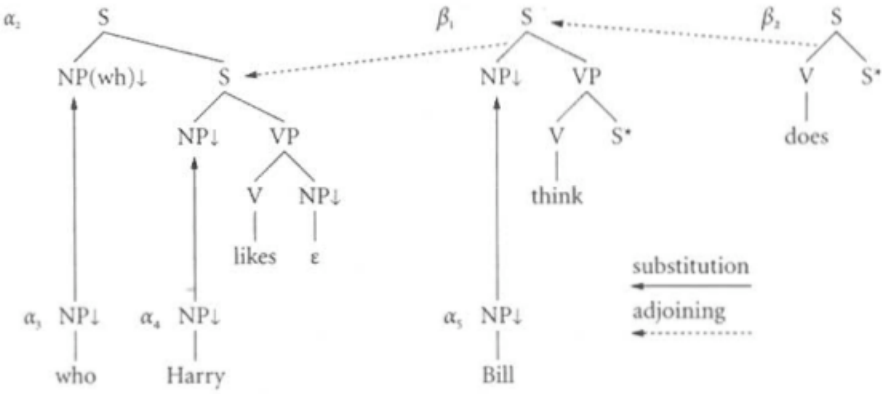


Fig. 26.9 LTAG derivation for *who does Bill think Harry likes*

Again, do you see any corresponds between TAG and CTL/CG?

Possible topic for review project.

6. Recall: Generative Power and Complexity of FGs

Recall, every (formal) grammar generates a unique language. However, one language can be generated by several different (formal) grammars.

Formal grammars differ with respect to their **generative power**:

One grammar is of a greater generative power than another if it can recognize a language that the other cannot recognize.

Two grammars are said to be

- **weakly** equivalent if they generate the same string language.
- **strongly** equivalent if they generate both the same string language and the same tree language.

6.1. DG, CG, CTL, CCG, and TAG

- DG: Gross (1964)(p.49) claimed that the dependency languages are **exactly** the context-free languages. This claim turned out to be a mistake, and now there is new interested in DG. (Used in QA)
- CG: Chomsky (1963) conjectured that **Lambek calculi** were also **context-free**. This conjectured was proved by Pentus and Buszkowski in 1997.
- TAG and CCG: have been proved to be Mildly Context Free.
- CTL has been proved to be Mildly Sensitive (Moot), or Context Sensitive (Moot) or Turing Complete (Carpenter), accordingly to the structural rules allowed.
- LG has been proved to be Mildly Context Free. (Moot 2008)

7. Meaning entered the scene

Chomsky was, in general, **sceptical of efforts to formalize semantics**. Interpretative semantics or the autonomy of syntax: Syntax can be studied without reference to semantics (cf. also Jackendoff).

Criticism on both transformational and non-transformational approaches:

- Transformations do not correspond to syntactic relations, relying too much on linear order.
- Similarly, Curry (1961; 1963) criticized Lambek for the focus on order (directionality).

7.1. Different ongoing efforts

- Developing a notion of (meaningful) logical form, to which a syntactic structure could be mapped using transformations. Efforts either stayed close to a constituency-based notion of structure, like in generative semantics (Fodor, Katz), or were dependency-based (Sgall et al, particularly Panevová (1974; 1975); Fillmore (1968)). Cf. also work by Starosta, Bach, Karttunen.
- Montague's formalization of semantics – though Montague and the semanticists in linguistics were unaware of one another, cf. (Partee, 1997)

7.2. Montague and the development of formal semantics

The foundational work by Frege, Carnap, and Tarski had led to a rise in work on modal logic, tense logic, and the analysis of **philosophically interesting issues in natural language**. Philosophers like Kripke and Hintikka added model theory.

These developments went hand-in-hand with the **logical syntax** tradition (Peirce, Morris, Carnap), distinguishing syntax (well-formedness), from semantics (interpretation), and pragmatics (use).

Though the division was inspired by language, **few linguists attempted to apply the logician's tools in linguistics as such**.

This changed with **Montague**.

> "I reject the contention that an important theoretical difference exists between formal and natural languages." (Montague, 1974)(p.188)

A compositional approach, using a "rule-by-rule" translation (Bach) of a syntactic structure into a first-order, intensional logic. This differed substantially from transformational approaches (generative or interpretative semantics).

8. Grammars meet Logic & ...

Logics to specify a grammar framework as a mathematical system:

- Feature logics: HPSG, cf. (King, 1989; Pollard and Sag, 1993; Richter et al., 1999)
- Categorial Type Logics (Kurtonina, 1995; Moortgat, 1997)

Logics to interpret linguistically realized meaning:

- Montague semantics: used in early LFG, GPSG, Montague Grammar, Categorial Type Logic, TAG (Synchronous LTAG)
- Modal logic: used in dependency grammar frameworks, e.g. (Broeker, 1997; Kruijff , 2001).
- Linear logic: used in contemporary LFG, (Crouch and van Genabith, 1998).

9. .. Computation

Computation of linguistic structures

- Unification (constraint-based reasoning): LFG, HPSG, categorial grammar (UCG), dependency grammar (UDG, DUG, TDG)
- "Parsing as deduction": CTL
- Optimality theory: robust constraint-solving, e.g. LFG

9.1. Unification

The development of Unification Grammars has strongly been influenced by the:

- use of tools developed in Logics and in AI;
- the progress made in the area of Natural Language Processing;
- Development of Logic Programming: Prolog.
    1. Declarative character: grammar is not a set of rules, but a set of constraints that a sequence needs to satisfy in order for it to be a grammatical phrase.
    2. Constraints do not need to be ordered.

Transformational grammars are inadequate if faced with implementation problems. Derivations proceed from deep structures while automatic sentence analysis requires the inverse process.

Unification grammars or constraint based grammars represent the new syntactic models of the 80's.

10. Recall: 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

10.1. 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.

10.2. 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
```

11. 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

$$s \longrightarrow np\ vp$$

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).

12. 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.

13. 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.

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

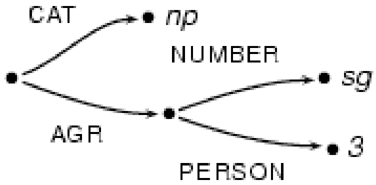
This notion of paths brings us to an alternative graphical way of illustrating FS, namely directed graphs.

14.1. 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

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

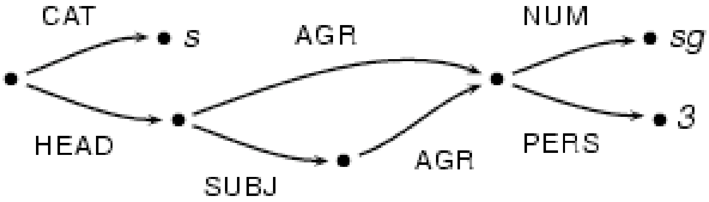
can also be represented by the following directed graph.



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`.

14.2. 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.

Structure sharing (cont.)

Example: Kim and Sandy are twins.

(4)

$$
\begin{bmatrix}
\text{TWIN1} & \begin{bmatrix} \text{NAME} & \textit{Kim} \\ \text{AGE} & \boxed{1}\ 29 \\ \text{FAVORITES} & \begin{bmatrix} \text{COLOR} & \textit{blue} \end{bmatrix} \end{bmatrix} \\
\text{TWIN2} & \begin{bmatrix} \text{NAME} & \textit{Sandy} \\ \text{AGE} & \boxed{1}\ 29 \\ \text{FAVORITES} & \begin{bmatrix} \text{COLOR} & \textit{blue} \end{bmatrix} \end{bmatrix}
\end{bmatrix}
$$

Structure sharing (cont.)

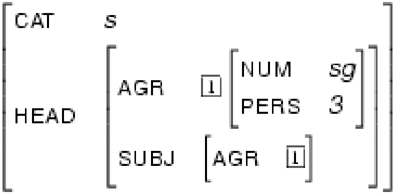
It is a fact about the world that twins have the same age, so the values for AGE are token identical, or "structure-shared." This is notated with matching boxed indices. On the other hand, the fact that Kim and Sandy have the same favorite color is accidental; these values are not structure-shared.

In DAG terms—

- Type identity: The two attribute edges have distinct end nodes.
- Token identity: The two attribute edges point to the same end node (structure-sharing).

14.3. Reentrancy as Coindexing

In other words, 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**.



Structure sharing (cont.)

Structure sharing is not allowed to result in circular or cyclical paths:

(5)

$$\boxed{1}\begin{bmatrix} \text{NAME} & \textit{Sandy} \\ \text{SIBLING} & \begin{bmatrix} \text{NAME} & \textit{Kim} \\ \text{SIBLING} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

This is not a well-formed feature structure description because is does not describe a DAG (directed acyclic graph).

14.4. 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`.

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

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.

Subsumption (cont.)

Formal definition of subsumption:

(6) a. For atomic values a and b (remember these are considered to be AVMs), $a \preceq b \Leftrightarrow b \preceq a \Leftrightarrow a = b$.[1]

b. For non-atomic AVMs A and B, A $\preceq$ B iff

i. for every attribute path in A, the same path exists in B and its value in A subsumes its value in B

ii. for every pair of paths that is structure-sharing in A, the same pair of paths is structure-sharing in B.

---

[1] **NB**: This will change once we introduce *sorts*.

14.5. 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.

14.6. Exercise

And how about the following two feature structures?

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

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.

14.7. 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.

15. 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.

15.1. 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

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

and let `F2` be the feature structure

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

Then, what is `F1` ⊔ `F2`, the unification of these two feature structures?

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

15.1.1. 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).)

15.1.2. 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) $F \sqcup G$ is the feature structure with the following three properties:

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

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**.

16. 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.

17. 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:

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

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.

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

See course web site for a project on this. Parsing with Feature Structures (PATR).

17.1. 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]$_{\texttt{Sif}}$

17.2. Schema

Schematically the subcategorization is represented as below.

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

17.3. Example

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

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

18. Conclusion

Next time we will look at the application of FG.

Topics for projects should be fixed by next time.