— Lecture Notes —

Subcategorization Features

1 We talked in class about how to implement subcategorization features for verbs. The starting point is the assumption that verb phrases are to be expanded by a DCG rule like the following.

```
vp( ... ) -->
        verb( ..., SubCat, ... ),
        complement( SubCat, ... ).

complement( np, ... ) -->
        np( ... ).
complement( sBar, ... ) -->
        s_bar( ... ).
```

That is, as a consequence of the information in our lexicon, every verb is mapped to one (or more) subcategorization feature(s), an atom which is used as an index into a table of `complement` rules.

2 To analyze the given sentences, you will need to have verbs in your lexicon that can (a) subcategorize for more than one complement and (b) subcategorize for particular features of that complement. For example, the verb *bauen* can be analyzed as taking two arguments: the thing constructed and the material from which the thing is constructed. (The latter argument is optional. There is a fine line between "optional arguments" and adjunct modifiers. We will not worry about this distinction here. Just treat the prepositional phrase *aus W"ortern* as an argument.) As an example of (b), according to this micro-corpus of data, the verb *dirigieren* takes a complement NP in the accusative case, while the verb *folgen* takes a complement NP in the dative case.

3 The simplest way to solve this problem, which uses no fundamentally new ideas, is have subcategorization features which are specific to the cases we actually observe. So, for example, we could have a subcategorization feature `np_pp` and a clause in the definition of `complement` that expands as follows.

```
complement( np_pp, ... ) -->
        np( ... ),
        pp( ... ).
```

I recommend this approach for this morning's "Ubung. It can be extended easily to government of particular cases. Assuming that noun phrases have feature terms like the following: `$($(Person,Number),Case)`, then you might have a clause in the definition of `complement` like this:

```
complement( np_dat, ... ) -->
        np( $(_Agr,dat), ... ).
```

4 If the corpus of examples you needed to recognize were any bigger you would soon find that the number of clauses in the definition of `complement` was getting out of hand! You could recover some generality as follows. First, let's tackle the problem of multiple complements. We can take a subcategorization frame like `np_pp` and turn it into a list of simpler subcategorization features `[np,pp]` Assuming we already had `complement` rules to handle `np` and `pp`, then having multiple complements does not increase the number of clauses in the definition of `complement` itself. We do however need some extra code to handle *lists* of subcategorization features. So assume now for the sake of generality that *all* verbs have lists of subcategorization features, even if they take no complements (`SubCat=[]`) or only one (`SubCat=[np]`) Then we just need a predicate like

```
complement_list( [], ..., L, L ).
complement_list( [SC|SCs], ..., L0, L ):-
        complement( SC, ..., L0, L1 ),
        complement_list( SCs, ..., L1, L ).
```

Note that I have written this as a Prolog predicate, not a DCG rule, so it needs the string arguments to be explicitly given! You can always mix Edinburgh syntax and DCG syntax in a single program, but you must be careful about those two extra arguments!

Now you just have to modify the VP rule to call `complement_list` rather than `complement`, directly.

5 To deal with subcategorization for particular features, you just have to do to subcategorization features what we have already done to the nonterminal categories of context free grammars: add arguments. So we can now distinguish subcategorization for an NP of any sort (as in English) (`SubCat=np(_)`) from subcategorization for an accusative NP (`SubCat=np(acc)`) from subcategorization for a dative NP(`SubCat=np(dat)`), and so on.

6 It's possible to go even farther than this, so that each verb is associated directly with a callable Prolog goal as its subcategorization frame, but that extra step only works well if you are using "X-Bar Theory" (which has the effect of turning even major categories like `np` and `s` into features of a single all-purpose syntactic category term). We have not covered that yet, so I will postpone further discussion of subcategorization until then.