23/11/87 -

Unification Grammars and their Lexicons

1. Templates

In the previous sections on representing grammars in a Unification framework such as PATR-II, you may have been struck by how much information was associated with individual lexical entries. For instance, handling agreement by putting the requisite specification for the path <subj agr> in the lexical entry for a tensed verb means that each such verb must contain that information. Similarly, the treatment of subcategorisation proposed above gives rise to extremely complicated lexical entries. Unlike grammar rules, which come in ones and twos, particularly in **lexicalist** grammars, lexical entries come in tens of thousands. Therefore in developing a grammar with a lexicalist orientation, it is very important to consider the question of how dictionary entries can be specified compactly.

One device that PATR-II provides to make dictionary writing easier is called the **template**. A template gives a name to a set of path equations that define a DAG that needs to be used in several places. The template name can then appear rather than the set of equations. Some simple examples are:

```
Let V be
<cat> = v

Let 3sing be
<head subject head agr num> = sing
<head subject head agr pers> = 3
<head form> = finite
```

A more complex example is:

```
Let Transitive be
<subcat first cat> = np
<subcat rest first cat> = np
<subcat rest first> = <head subject>
<subcat rest rest> = end
```

Then the entry for a particular transitive verb in its third singular form will just be:

```
storms:
V 3sing Transitive
```

This shows the use of templates at the top level. Templates can also be used in path equations, e.g.

```
Let Dummy_NP be
<cat> = np
<nform> = it
<lex> = it

seems:
<head subj> = Dummy_NP
```

Recursive Procedure.

23/11/87

Also, since a template definition may include a call to another template, the evaluation procedure must be recursive.

2. Lexical Rules

A more elaborate system for specification of lexical entries has been developed by Ritchie et al. at Edinburgh and Cambridge. Like PATR-II this system employs template-like devices, which are called **aliases.** This sort of device acts merely as an abbreviatory convention and is thus similar to a macro in a programming language. It has little linguistic import. Ritchie's system has three other types of **lexical rule.** I will illustrate these types of rule, though I will adopt a somewhat different notation.

First are the rules which refer only to a single feature structure. The idea behind these is that there are certain combinations of feature specifications which as a matter of linguistic hypothesis always occur together. Therefore mention of one of them is sufficient to guarantee the presence of others. Conversely, the presence of one without the others is an indication of an ill-formed feature structure. Ritchie's system chooses to separate out these two uses of cooccurence information as distinct rule types, that is, having a distinct procedural interpretation. The two types are called **completion rules** and **consistency checks**, but they can be more generally subsumed under the heading **redundancy rules**

Examples of the sort of facts that one might wish to describe using redundancy rules follow.

Grammar rules which refer to verbs will often specify that the verb is finite, without requiring it to be specifically either present or past. On the other hand, there will be distinct morphemes for present and past. The lexical entries for these morphemes need not give a specification for finiteness, only for tense, once the following rule has been stated:

[tense = _-> finite=yes]

A similar sort of rule might be used in a language which made grammatical distinctions between humans, animals and inanimate entities:

[human = yes -> animate = yes]
[animate = no -> human = no]

Notice that if we had the full power of logic we could deduce the second of these rules from the first. In general, however, if we wish to write a procedurally tractable linguistic programming language, we must restrict ourselves to some subset of full logic. This is analogous to Prolog's inability to prove certain conclusions that follow from the Prolog clauses treated as full FOPC.

More interesting rules of this type are those in which the antecedent is not a single feature specification, e.g.

[subcat = intrans & vform = pastp -> perf = yes]

Bearing in mind that in English, past and perfective participles are never distinguished, but that intransitive verbs cannot passivise, we get the above rule (assuming that the lexicon does not have separate entries for the two forms).

A similar type of rule is the **default** specification. Like other redundancy rules, these will add feature specifications to a single entry. Unlike the others, no error condition will arise

23/11/85

if the antecedent holds but the consequent fails, since this merely indicates that the default has been overridden. In Ritchie's system, defaults are not a separate rule type, but merely a way of using completion rules. Typical defaults would be:

[cat = verb -> voice = active]
[cat = noun -> nform = norm]

The second type of rule in Ritchie's system is called a **multiplication rule.** Such rules do not refer to a single feature structure; rather they define how to build a new entry on the basis of an existing one, and hence multiply the number of entries. PATR-II also has this sort of rule, and we will illustrate using that notation:

```
Define DiTrans as
<out cat> = <in cat>
<out head> = <in head>
<out subcat first sem> = <in subcat rest first sem>
<out subcat first cat> = np
<out subcat rest first sem> = <in subcat first sem>
<out subcat rest first cat> = np
<out subcat rest rest> = <in subcat rest rest>
```

Here the two special path names 'in' and 'out' refer to the DAGs that constitute the input and output to the lexical rule. In this example 'in' is an entry for a ditransitive verb, which would take a direct object and an indirect object marked with the preposition 'to', and 'out' is a form which takes two np objects.

Notice that unlike the basic PATR-II system, including templates, a system which contains the sort of lexical rules that have been introduced above is no longer purely declarative. Defaults, in particular, are difficult to give a declarative semantics. The effect of applying a default rule referring to a particular feature will obviously vary according to whether it is applied before or after a multiplication rule which assigns a particular value to the same feature.

3. Feature passing principles (FPPs)

Just as there has been a trend to move subcategorisation information out of grammatical rules, there has also been a trend to eliminate some of the complexity of feature specifications from grammatical rules. Ritchie's system incorporates three such principles (which they call 'conventions') in their **word grammar**, the set of rules that specify which morphemes may join together in which orders to form words. The idea of FPPs is that the DAGs associated with the mother node and daughter nodes in a rule do not in general vary arbitrarily. We know, for instance, that the head features will be the same on mother and head daughter, which we can state as the **Head Feature Principle**. Now, instead of adding equations of the form:

$<X_0$ head$> = <X_i$ head$>$

to each rule, we merely need to indicate which of the X_i is in fact the head, e.g. by using the identifier H. Thus the PATR-II rule:

23/11/87

$X_0 \to X_1\ X_2$
$\langle X_0\ \text{head}\rangle = \langle X_2\ \text{head}\rangle$

may be replaced by:

$X_0 \to X_1\ H$

Other principles that have been suggested include: the **Control Agreement Principle**, whcih specifies identity of agreement features between various daughter DAGs in a rule according to certain criteria which need not concern us here; and the **Foot Feature Principle**, which concerns the distribution of foot features. By analogy with head features, foot features are those shared between the mother and one or more of the non-head daughters. An example of foot features are those concerning the placement of gaps, as illustrated in the context of DCGs.

4. Conclusion

A Unification based approach to linguistic description now seems to be a consensus amongst computational linguists and theoretical linguists interested in computational implementation of their ideas. We can draw a distinction between those Unification based formalisms that are intended as tools, and those intended as linguistic theories. In the first category fall PATR-II and Martin Kay's FUG, which is described in Readings in NLP, as well as DCGs. In the latter category are: Generalised Phrase Structure Grammar (GPSG), whose treatment of subcategorisation was exemplified in HO4 and whose lexical rules and feature passing conventions are approximately described in this handout (Ritchie's system being part of a larger project to implement GPSG); Head-Driven Phrase Structure Grammar (HPSG), which is a development of GPSG with a treatment of subcategorisation as described in HO9; Categorial Unification Grammar (CUG/UCG), which is Categorial Grammar as described in HO7 plus unification; and Lexical Functional Grammar, a fairly direct descendant of Chomskyan TG, without the T.

In Britain alone, the Institutions and Projects using Unification Grammars include: The Alvey-funded Edinburgh, Cambridge and Lancaster GPSG system; The Alvey-funded UMIST English-Japanese Translation System; The IBM Winchester HPSG Project; The Esprit-funded Acord Project at Edinburgh Cognitive Science, a CUG grammar for Natural Language Interrogation of Data Bases; and several others.