12/10/87

Categorial Grammars

1. Chart Parsing and Categorial Grammars

In discussing chart parsing, we used the idea of the 'multiplication rule', which is a rule specifying how two constituents could be put together to form a larger constituent. In the CKY algorithm, this rule corresponds to the rules of grammar in a straightforward way - two constituents α and β can be combined to form γ if there is a rule of the form $\gamma \rightarrow \alpha\ \beta$ in the grammar. In the Earley algorithm, this close connection with the grammar rules is broken, so that the form of the constituent built can be determined without reference to the grammar, solely in terms of the structure of the constituents that build it, i.e. dotted rules.

However, the way that dotted rules are originally entered into the chart is by reference to the grammar, during the prediction step of parsing. We will now examine an alternative organisation of grammatical knowledge that breaks this connection entirely, so that when the lexical entries corresponding to words have been determined, parsing can be carried out without reference to a grammatical knowledge source.

Suppose that we handle subcategorisation in the manner outlined in handout 4, where a feature on a lexical item determines which phrase structure rules it appears in, e.g.

vp → v[3] np pp[to]

Without entering into the complexities of parsing context-free grammars that have features on categories, we can see that at the point in a chart parse without lookahead at which we predict a 'vp', we will be entering a large number of initial dotted rules, corresponding to possible vp expansions for any verb. When the verb is looked up in the lexicon and the multiplication rule applied to it and all active edges ending where it starts, most of these initial dotted vp rules will be inapplicable. The result will be a few edges of the form:

vp → v[3] . np pp[to]
vp → v[5] . np np

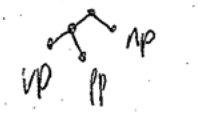
Lookahead would reduce this problem. An alternative solution is to associate these active edges directly with the verb in the lexicon. If we could find a motivated way of doing this for all lexical items, we could dispense with the prediction step altogether.

A lexicon for a context-free grammar that has been expressed in this form, so that grammar rules which mention specific categories are rendered superfluous, is called a **Categorial Grammar**. The rest of this handout explores the implications of this approach.

2. Notational Issues

The standard way to notate lexical entries in a CG is using the slash /. To illustrate this, consider the two active edges above. These would be represented as follows:

vp → v[3] . np pp[to] (vp/pp)/np
vp → v[5] . np np (vp/np)/np



= vp/np/np

These complex categories would replace the lexical assignments v[3] and v[5] to verbs such as *send* and *give*. The first of these can be glossed, identically to the active edge, as

left associative

READ

(VP/PP)/NP ≠ VP/PP/NP

12/11/87

something that combines with a np to give something that combines with a pp to give a vp. The reason for referring to the combination rule in a chart parser as 'the multiplication rule' can now be understood. Using the CG notation, we have:

Rule 1: X/Y * Y = X

for any X and Y that are categories. If X and Y were integers, say, this would be a true statement if / denoted integer division and * multiplication. Notice from the lexical assignments given above in this notation that categories are recursive. X or Y in the schema can themselves be complex. To simplify, we assume left associativity of /, so that (vp/pp)/np can be written as vp/pp/np.

So far, this probably appears like notational sleight of hand. The category v has disappeared, but we still have vp. However, vp too can be dispensed with by defining it as something that combines with a noun phrase, the subject, to give a sentence. Unlike verbs, which find their objects etc. to the right, vps find their subjects to the left. Therefore we introduce the backslash \ and the second multiplication rule:

Rule 2: Y * X\Y = X

We can continue to reduce the set of basic categories by considering the question of optional modifiers. These are typically introduced by rules such as:

vp → vp pp
np → np pp

The net effect of such rules is to allow any number of modifiers, in this case, pp's, to occur with a vp and np respectively, since these latter categories appear both in the lhs and rhs of the rules. The categorial solution is to view these modifiers as having the category X\X, which achieves the same effect. In the case of a vp modifier, X in X\X would be s\np, giving the whole a category of (s\np)\(s\np). An np post-modifier will have the category np\np. In a similar way, the effect of the phrase structure rule:

nom → adj nom

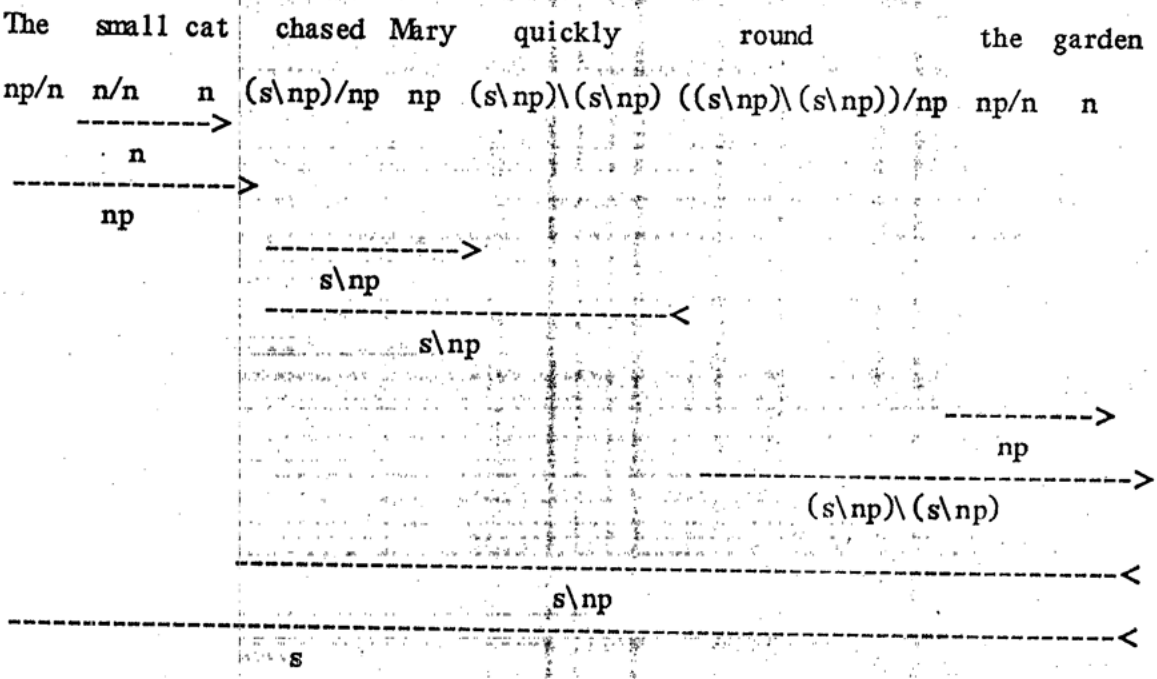
can be replicated by assigning adjectives to the category n/n. Unlike in a phrase structure grammar, the distinction between words and phrases is neutralised in a categorial grammar. An intransitive verb, for instance, has the same distributional properties as an entire verb phrase, and this is reflected in the identity of their categories, s/np. The same holds of proper nouns and noun phrases .

A typical Categorial Grammar will make use of only s, np and n as basic categories, perhaps also retaining pp for subcategorised pp's. Thus it is a very parsimonious notation for grammatical knowledge.

2 stringy

To illustrate the course of parsing a sentence in a CG, we draw a line under each pair of constituents that are combined by a combination rule, label the line with an arrow indicating the direction of combination, and write the result under it. Overleaf is an example:

12/4/87



3. Advantages of Categorial Grammar

The principle advantage of Categorial Grammar is the very flexible notion of 'syntactic category' that it permits. Consider sentences of the form.

Fred cooked, and Mary ate, the delicious spaghetti that they bought in Naples
Bill drinks coffee at breakfast, and Nancy tea
I gave Jeanie a teashirt on her birthday, and my mum a CD player at Xmas

Phrase structure grammars typically contain rule schemas of the form:

X → X and X

to handle coordination, where X ranges over all categories. But the sort of sentences given above have long been problematic for phrase structure accounts of syntax, since strings of words like *Fred cooked, Nancy tea,* and *my mum a CD player at Xmas* do not form constituents as generally understood. Even in Categorial Grammar they are not all simply characterisable, but can be handled if we allow certain mathematically well-founded operations over categories.

The two 'multiplication' rules given above are generally referred to as rules of **functional application**, since X/Y can be considered as a function from things of type Y into things of type X, and the X/Y * Y -> X as application of that function to its argument. The two rules given are called **forward function application** and **backward function application** respectively Another rule of categorial combination that has been suggested corresponds to the idea of **function composition** in recursive function theory, where

$$f \bullet g(a) = f(g(a)).$$

That is, the function obtained by composing the functions f and g, when applied to the

12/11/87

argument a, is equal to applying the function g to the result of applying the function f to a. Thus we have the further rules:

X/Y * Y/Z -> X/Z
Y\Z * X\Y -> X\Z

like Transitivity.

There is also a rule which applies to a single category, known as **type-raising**. The effect of this is to turn an argument into a functor that can combine with what it combined with before type-raising to give the same result. So, for instance, a category like np may be type-raised to a category like (s/(s\ np)). Then compare the following:

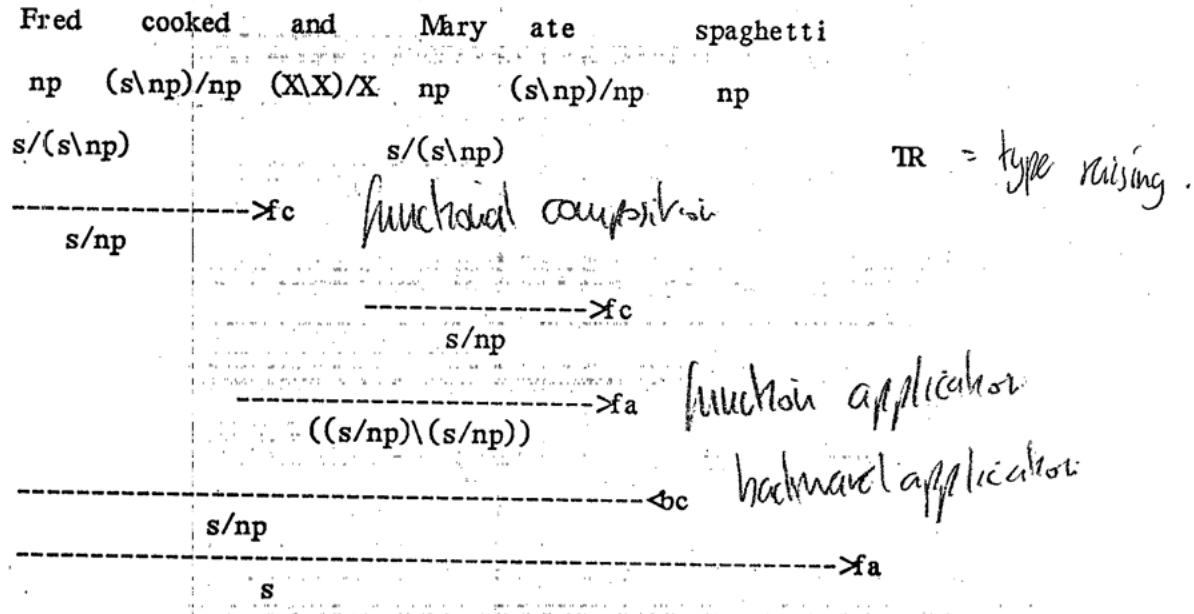
np s\np -> s
s/(s\np) s\np -> s

The general rule for type-raising is thus:

X -> Y|(Y|X)

where | indicates \ or /.

Using these algebraic manipulations on categories, we can now derive the first of the 'problem' sentences above quite easily (note lines now labelled with rule name, since there are two types of forward and two type of backward rule:



TR = type raising.

functional composition

function application

backward application

Finally, several other advantages accrue from the categorial view. In terms of Universal Grammar, it seems natural that the rules for combining constituents together are not language specific, but are taken from some small set of combinators such as given above.

The freedom with which constituents can be combined in this framework promises the psychologically attractive approach to parsing that it proceeds incrementally, building representations even for fragments.

It also appears that the syntactic combinators can be paired with semantic ones, that give a very simple account of how the meaning of a constituent is derived from the meaning of its parts. The question of **compositionality** is one that we will consider further under the heading semantics.