1. Overly complicated, and wrong as well?

[Before we go on to agreement, a brief diversion]

You might feel that all these (mostly binary) rules are missing the point

- Particularly, because they allow all kinds of wrong orders

\[
\text{Verbal} \\
\text{Aux} \quad \text{Modal} \quad \text{Verbal} \\
\text{have} \quad \text{may} \quad \text{V} \\
\quad \text{left}
\]

Why don't we just make the order explicit?

\[
\text{CNP} \rightarrow \text{Det? Card? Ord? Quant? AP* Nominal}
\]

\[
\text{Verbal} \rightarrow \text{Modal? Aux? AdvP? V}
\]

where by e.g. "Det?" is meant the Det is optional
and the "*" is a Kleene star, i.e. 0 or more APs are allowed

That is, why not allow regular expressions over \( T \cup NT \) on the right hand side?

- We could, and people have
- Either as an extension to CFGs
- Or as an extension to FSAs, called \textbf{Pushdown Automata}
  - Or sometimes \textbf{Recursive Transition Networks}
2. Extending CFGs

You can understand such an extension to CFGs in one of two ways:

- As a change to the formalism itself, i.e.
  - rhs a regular expression whose alphabet is \( T \cup NT \)
  - corresponding (non-trivial) changes to the rewriting and node-admissibility definitions

- As an extension to the notation *only*, not to the formalism as such
  - i.e., we treat rules notated like so:
    - \( X \rightarrow \ldots_1 Y \ldots_2 \)
    - As just shorthand notation for the more verbose pair of notations
      \[
      \begin{align*}
      X & \rightarrow \ldots_1 Y \ldots_2 \\
      X & \rightarrow \ldots_1 Y \ldots_2
      \end{align*}
      \]

On this account, our VP 'rule' on the previous slide is a shorthand notation for *eight* actual rules.

What about the NP rule, with its Kleene star?

3. Infinite CFGs

Including Kleene star in our notation for the right-hand side of rules turns out to have a surprising consequence.

If we take the same approach as we did for question-mark

- i.e., we treat rules notated like so:
  - \( X \rightarrow \ldots_1 Y^* \ldots_2 \)
  - As just shorthand notation for the more verbose pair of notations
    \[
    \begin{align*}
    X & \rightarrow \ldots_1 Y \ldots_2 \\
    X & \rightarrow \ldots_1 Y Y^* \ldots_2
    \end{align*}
    \]

we have what amounts to (a notation for) a CFG with an *infinite* number of rules!

- That actually has the potential to change the status of the formalism
  - Its **weak generative capacity**
  - AKA its position on the **Chomsky hierarchy**

[End of diversion]

4. Back to agreement

The NP and VP rules we’ve seen so far *overgenerate*
5. Possible CFG Solution for Agreement

We could try to address our agreement problems by expanding the non-terminal categories to encode agreement:

Where we’ve used ‘sg’ and ‘pl’ for singular and plural
And the above isn’t enough: more doubling of rules would be needed
  ◦ E.g. for Det

We could use the same approach for all the verb/VP classes

But this clearly has become quite obscure
And the (multiplicative) interaction between number agreement and subcategorisation will make things much worse

6. CFG Solution for Agreement

Good thing
  It works and stays within the power of CFGs

Less good things
  • It’s inelegant
  • It doesn’t scale
    ◦ The interaction among various families of constraints explodes the number of categories and rules in the grammar
  • It still overgenerates!
    ◦ It can’t deal with unbounded dependency
7. CFG conclusions

CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.

But there are problems:

- Agreement
- Unbounded dependencies

There are more elegant solutions:

- But they go beyond the formal power of CFGs
  - Sign-based theories (GPSG, HPSG)
  - Tree-adjoining grammars

We'll look at LTAG, one variety of tree-adjoining grammar, next week.

But first, we'll expand our approach to categories:

- By adding features

8. Features

The name feature has been around in Linguistics for a long time.

The basic idea is to capture generalisations by decomposing monolithic categories into collections of simpler features.

Originally developed for phonology, where we might have e.g.

- /i/ +high, +front
- /e/ -high, +front
- /o/ -high, -front
- /u/ +high, -front

Where we can now 'explain' why /i/ and /u/ behave similarly in certain cases, while /i/ and /e/ go together in other cases.

Those are all binary features:

- sometimes also used at the level of syntax:

  +/- singular; +/- finite

9. Features, cont'd

But more often we find features whose values are some enumerated type:

- person: {1st, 2nd, 3rd}; number: {sg, pl}; ntype: {count, mass}

We'll follow J&M and write collections of features like this:
It will be convenient to generalise and allow features to take feature bundles as values:

\[
\begin{array}{c|c}
\text{ntype} & \text{count} \\
\hline
\text{agreement} & \begin{array}{c}
\text{person} \\
\text{number} \\
\text{3rd} \\
\text{pl}
\end{array}
\end{array}
\]

10. Features in use

We can now add feature bundles to categories in our grammars. In practice we allow some further notational conveniences:

- Not all features need be specified (e.g. the \textbf{number} feature for 'sheep')
- In rules, we allow the values of features to be variables
- And we can add constraints in terms of those variables to rules

For example

\[
\begin{array}{c|c|c}
\text{ntype} & \text{count} \\
\hline
\text{agreement} & \begin{array}{c}
\text{person} \\
\text{number} \\
\text{3rd} \\
\text{pl}
\end{array}
\end{array}
\rightarrow \text{men} \mid \text{dogs} \mid \text{cats}\ldots
\]

\[
\text{NP[ agreement } x \text{ ] } \rightarrow \text{D[ agreement } y \text{ ] } \text{N[ agreement } z \text{ ] } x = y = z
\]

11. Features: more than notational convenience?

At one level, features are just a convenience.

The allow us to write lexicon entries and rules more transparently.

But they also "capture generalisations."

If we write a pair of rules using some (in principle opaque) complex category labels, they are not obviously related in any way:
It appears as if we have to justify each of these independently

- or that we *might* have had one without the other

- or had \( S \rightarrow \text{NP}_{sg} \text{VP}_{pl} \) just as well

Whereas when we write

\[
S \rightarrow \text{NP} \text{VP} \left[ <\text{NP agreement}> = <\text{VP agreement}> \right]
\]

we are making a stronger claim, even though 'behind the scenes' this single line corresponds to a collection of simple atomic-category rules

### 12. Infinity again: categories

Once you move to feature bundles as the values of features

- You can in principle have an infinite number of categories
- And, as with infinite numbers of rules, that actually changes your position on the Chomsky hierarchy

One strand of modern grammatical theory

- From GPSG to HPSG to so-called sign-based grammatical theories

Puts essentially *all* the expressive power of the grammar into feature structures

### 13. Unification

When we write '=' between two feature paths or feature variables, we mean more than an equality test

Consider the noun phrase "a sheep", and the following rules
The resulting parse tree reveals that we have not only tested for compatibility between the various feature structures, we’ve actually merged them:

where by the \( \varnothing \) we mean that all three agreement values are the *the same* feature structure

**14. Unification, cont'd**

The implications of unification run deep

The three occurrences of \( \varnothing \) don’t just appear the same

- They *are* the same
- That is, a single structure, shared 3 times
- So any change to one in the future will be a change to all
- As would be the case with e.g. "the sheep runs" or "the sheep run"

J&M give a detailed introduction to unification, which is what this is called, in section 15.2, and a formal definition in section 15.4.

The directed acyclic graph (DAG) way of drawing feature structures used in J&M 15.4 makes clearer when necessary structure identity is the case, as opposed to contingent value equality

**15. Parsers**

A **parser** is an algorithm that computes a structure for an input string given a grammar.

All parsers have two fundamental properties:

**Directionality**
- The sequence in which the structures are constructed
  - Almost always **top-down** or **bottom-up**

**Search strategy**
- The order in which the search space of possible analyses is explored
  - Usually **depth-first**, **breadth-first** or **best-first**
16. Recursive Descent Parsing

A recursive descent parser treats a grammar as a specification of how to break down a top-level goal into subgoals.

This means that it works very similarly to a particular blind approach to constructing a rewriting interpretation derivation:

**Directionality**
*Top-down:*
- starts from the start symbol of the grammar
- works down to the terminals

**Search strategy**
*Depth-first:*
- expands the left-most unsatisfied non-terminal
- until it gets to a terminal
  - which either matches the next item in the input
  - or it doesn't

17. Depicting a WFST as a graph

A sample graph, for the same situation mid-parse

- Here, **nodes** (or **vertices**) represent positions in the text string, starting **before** the first word, ending **after** the final word.
- **arcs** (or **edges**) connect vertices at the start and the end of a span to represent a particular substring
  - Edges can be labelled with the same information as in a cell in the matrix representation

![Graph Diagram]