Combinatory Categorial Grammar

L614
Spring 2010

Based on Steedman and Baldridge (2007)
Combinatory Categorial Grammar (CCG)

- Motivation for Categorial Grammar
- Categorial Grammar (CG)
  - Categories
  - Functional Application
  - Modalities
  - Bounded constructions: Raising and Control
- Combinatory Categorial Grammar (CCG)
  - Functional Composition
  - Type-raising
  - Coordination
  - Unbounded constructions
Motivation for Categorial Grammar

Why is categorial grammar a formalism worth investigating?

- Only a “minimal” extension to CFGs
  - Operations are in terms of the simple combination of adjacent constituents
- Close relation to (compositional) semantics
- Cross-linguistic generalizations can be made easily since the same set of rules always apply
- Flexible constituency allows for coordination of unlikes and treatment of unbounded constructions
- Arguably psychologically plausible (since processing can proceed in a left-to-right fashion)
• The rules of grammar are entirely conditioned on the lexical categories i.e., There are lots of categories and only a small set of applicable rules

• Categories—sometimes referred to as types—come in two varieties:
  - **Primitive categories**: N, NP, PP, S, etc.
    1. Marcel := NP
    2. man := N
    → can further be distinguished by features
  - **Functions**: a combination of primitive categories, more specifically a function from one category (primitive or function) to another: e.g., S/NP, (S/NP)/(S/NP), etc.
Functional application (FA)

• Syntactically potent elements such as verbs are associated with a syntactic category that identifies them as functions
  – Functions specify the type and directionality of their arguments and the type of their result
  – $S \backslash NP$ is an intransitive verb because it is looking for an NP (to the left) in order to form an S

• A “result leftmost” notation is used here:
  – $\alpha/\beta$ is a rightward-combining functor over a domain $\beta$ into a range $\alpha$
  – $\alpha \backslash \beta$ is the corresponding leftward-combining functor.
  – $\alpha$ and $\beta$ may themselves be functional categories.
Example lexical entries

- Intransitive and transitive verbs

  (3) a. ran := \( S \backslash NP \)
  
  b. proved := \( S \backslash NP \)/\( NP \)
  
  c. gave := \( (S \backslash NP)/NP \)/\( NP \)

- Sentence and verb modifying adverbs

  (4) a. yesterday := \( S \backslash S \)
  
  b. always := \( S \backslash NP \)/(\( S \backslash NP \))
Rules and derivations

• Functor categories can combine with their arguments by the following rules:

(5) Forward application ($\triangleright$)

\[ X/Y \ Y \Rightarrow X \]

(6) Backward application ($\triangleleft$)

\[ Y \ X\Y \Rightarrow X \]

• Derivations are written as shown below. Note the direct correspondence to an upside-down constituency tree.

\[
\begin{array}{c}
\text{Marcel} \\
\text{NP}
\end{array}
\quad
\begin{array}{c}
\text{proved} \\
(S\backslash \text{NP})/\text{NP}
\end{array}
\quad
\begin{array}{c}
\text{completeness} \\
\text{NP}
\end{array}
\]

\[
\begin{array}{c}
(S\backslash \text{NP})/\text{NP} \\
\text{NP}
\end{array}
\Rightarrow
\begin{array}{c}
\text{S}\backslash \text{NP} \\
\text{S}
\end{array}
\]

\[
\begin{array}{c}
\text{S}\backslash \text{NP} <
\end{array}
\]
Other examples

```
Marcel          gave          me          fits
NP              (NP)/(NP)   NP         NP
               (NP)         >         >
               S\NP            >
               S
```

```
Marcel          always        gave        me          fits
NP              (NP)/(NP)   (NP)/(NP)   NP         NP
              (NP)         >         >
              S\NP            >
              S\NP            >
              S
```
Adjuncts and Complements

The procedure for adding adjuncts is the same as for complements: use functional application

- Note how in the previous proof tree, always was added with the same forward application as we used to add fits

- What is the tree for Marcel proved completeness yesterday?

⇒ Adjuncts are generally defined as being of type $X/X$ (or $X \setminus X$)
The lexical categories are augmented with an explicit identification of their semantic interpretation and the rules of functional application are accordingly expanded with an explicit semantics.

(7) \( \text{proved} := (S\backslash NP)/NP : prove' \)

(8) a. Forward application (\( > \))
\[
X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa
\]
b. Backward application (\( < \))
\[
Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : fa
\]
Example derivation with semantics

\[
\begin{align*}
\text{Marcel} & \quad \text{proved} & \quad \text{completeness} \\
\text{NP: marcel'} & \quad (S\setminus\text{NP})/\text{NP: prove'} & \quad \text{NP: completeness'} \\
& \quad \quad \quad \quad \quad \quad S\setminus\text{NP: prove'} \text{ completeness'} & \quad < \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad S: \text{prove'} \text{ completeness'} \text{ marcel'}
\end{align*}
\]

→ What would be the syntactic and semantic category for *always*?
We will mostly leave the semantics off of the derivations, but note the following way we handle the dative shift:

- \( \text{give}_1: \ (\text{\textipa{S}}\backslash \text{NP})/\text{NP}: \lambda x \lambda y \lambda z. \text{give}' yxz \)

- \( \text{give}_2: \ (\text{\textipa{S}}\backslash \text{NP})/\text{PP[to]}): \text{NP}: \lambda y \lambda x \lambda z. \text{give}' yxz \)

The semantics are almost identical, except the order in which they combine

- A lexical rule can relate such lexical entries

After we discuss lambda abstraction, work out the derivations for *Marcel gave me money* and *Marcel gave money to me*, to see how to obtain the same semantics
Principle of Type Transparency

- The semantic interpretation of all combinatory rules is fully determined by the *Principle of Type Transparency*:
  
  - Categories: All syntactic categories reflect the semantic type of the associated logical form
  
  - Rules: All syntactic combinatory rules are type-transparent versions of one of a small number of semantic operations over functions including application, composition, and type-raising.
Modal operators can further restrict the rules and function categories

For example, Steedman and Baldridge use the $\Diamond$ modality to prevent heavy NP shifts in the following:

(9) *I gave a book my very heavy friend from Hoboken.

- $\text{give}_1$: $((S\setminus NP)/NP)/\Diamond NP: \lambda z \lambda y \lambda x [\text{give}'(x, y, z)]$
Modality operators can also be organized into a hierarchy.

- That way, some operations can be restricted to a specific class
  - e.g., function composition cannot apply to coordination

- And other modalities are less restrictive, allowing more operations to apply
  (e.g., limited permutation)

The modalities are mostly used to handle word order and allow for constraining things like cross-serial dependencies in Dutch.

- It also means that lexical items can determine whether they allow an operation to be performed
Bounded constructions

For bounded constructions, CCG expresses underlying dependencies and structures at the level of logical form, not at the level of derivation.

- reflexivization
- dative-shift
- raising
- object & subject-control
- passivization

Treatment on a par with HPSG, LFG, & TAG
Raising and Control

• Raising verb

(10) a. I seemed to be angry.
   b. seem := \((S\backslash NP)/(S_{TO}\backslash NP) : \lambda p\lambda y.\text{seem}'(py)\)

• (Subject-)Control verb

(11) a. I promised to take a bath.
   b. promise := \((S\backslash NP)/(S_{TO}\backslash NP) : \lambda p\lambda y.\text{promise}'(p(ana'y)y)\)

Syntactic functions are identical, but the semantics of control include an anaphoric reference to the subject

• i.e., \textit{ana'y} serves as the subject of \textit{to take a bath} \((p)\).
Raising example

\[
\frac{\text{Marcel}}{S/(S/NP)} \quad \frac{\text{seems}}{(S/NP)/(S_{TO}/NP)} \quad \frac{\text{to}}{(S_{TO}/NP)/(S_{INF}/NP)} \quad \frac{\text{drink}}{S_{INF}/NP}
\]

\[
\lambda p. \text{pmarcel'} \\
\lambda p \lambda y. \text{seem'(py)} \\
\lambda p. p \\
\text{S}_{TO}/NP: \text{drink'} \\
\text{S}_{TO}/NP: \lambda y. \text{seem'(drink'y)} \\
\text{S: seem'(drink'marcel')} \\
\]
To obtain passives, we need to assume lexical function composition

- Morpheme -en applies to the first rightward argument of the base category

- /... schematizes over categories with 0+ further rightward arguments

\[(12) \text{-en := } ((S\backslash \text{NP/...}))_{LEX}(\text{
}((S\backslash \text{NP/...})/\text{NP}) : \lambda p \lambda \ldots \lambda x.p \ldots x \text{ one}'\]

Application of this rule yields, e.g.,

\[(13) \text{proven := } S_{EN} \text{\backslash NP : } \lambda x.\text{prove}'x \text{ one}'\]
CCG includes linguistically motivated rule schemata such as the one for coordination of constituents of like type shown below:

(14) Coordination (< & >)
\[ X \text{ conj } X \Rightarrow X \]

(15) and := \((X\backslash X)/X\)

Any identical categories can coordinate in this manner.
Constituent Coordination

Marcel conjectured
NP  (S\NP)/NP

and

(X\X)/X

proved

(S\NP)/NP

(S\NP)/NP

((S\NP)/NP)\((S\NP)/NP)

(S\NP)/NP

completeness

NP

((S\NP)/NP)\((S\NP)/NP)

(S\NP)/NP

S\NP

S

What about so-called Right Node Raising, e.g., *John conjectured and Harry proved completeness*?
What we have seen so far is what makes up *categorial grammar*

**Combinatory Categorial Grammar (CCG)** increases the power of CG by adding two kinds of rule:

- Functional composition
- Type-raising
In order to account for coordination of contiguous strings that do not constitute traditional constituents, CCG allows certain operations on functions called “combinators”, including the rule of functional composition in (16).

(16) a. Forward composition ($\triangleright \mathbf{B}$)

\[ X/\triangleright Y : f \quad Y/\triangleright Z : g \quad \Rightarrow \quad X/\triangleright Z : \lambda x. f(gx) \]

b. Backward composition ($\triangleleft \mathbf{B}$)

\[ Y/\triangleleft Z : g \quad X/\triangleleft Y : f \quad \Rightarrow \quad X/\triangleleft Z : \lambda x. f(gx) \]

What this means is that functors can now select for items which are “missing” elements, but we won’t have to change any categories!

NB: These are the “harmonic rules” because all the slashes are in the same direction

- These slashes are typed, so only certain modalities can apply
Marcel conjectured and might prove completeness.

\[
\begin{array}{cccc}
\text{Marcel} & \text{NP} & \text{proven} & \text{NP} \\
(S \backslash \text{NP})/\text{NP} & (X \backslash X)/X & (S \backslash \text{NP})/((S \backslash \text{NP})/\text{NP}) & (S \backslash \text{NP})/\text{NP} \\
& & >_B & \text{NP} \\
& & & (S \backslash \text{NP})/\text{NP} \\
& & & ((S \backslash \text{NP})/\text{NP})/((S \backslash \text{NP})/\text{NP}) \\
& & & (S \backslash \text{NP})/\text{NP} \\
& & & S \backslash \text{NP} \\
& & & S
\end{array}
\]

NB: to save space in the future, we can abbreviate \( S \backslash \text{NP} \) as \( VP \), thus making \textit{might}'s lexical category more intuitive: \( VP/VP \)
If we had not used functional composition, we would have had to posit *might*'s lexical entry as:

\[(17) \text{might} := ((S\backslash NP)/NP)/((S\backslash NP)/NP)\]

- Select a transitive verb, \((S\backslash NP)/NP\)
- Then select the object of that transitive verb, \(NP\)

But we then need separate rules for every different kind of complement verb (intransitive, etc.)
Using function composition

- What are the two different analyses for *Marcel might prove completeness*?
- Do they correspond to different semantic representations?
Generalizing function composition

What do we need to do to allow for Marcel (has and) might give money to me?
Is function composition too powerful?

Dowty (1988) shows that, in terms of word order, no new orders are allowed by function composition:

**Application:**

\[
\frac{A/B}{B/C} \frac{C}{B} > \frac{A}{C}
\]

**Composition:**

\[
\frac{A/B}{B/C} >_B C
\]

Other word orderings simply won’t work for either case.

Plus, as we mentioned before, modalities can limit the power of function composition.
Combinatory Projection Principle

- **The Principle of Adjacency**
  Combinatory rules may only apply to finitely many phonologically realized and string-adjacent entities.

- **The Principle of Consistency**
  All syntactic combinatory rules must be consistent with the directionality of the principal function.

- **The Principle of Inheritance**
  If the category that results from the application of a combinatory rule is a function category, then the slash type of a given argument in that category will be the same as the one(s) of the corresponding argument(s) in the input function(s).
Crossing functional composition

While those principles rule out many rules (e.g., $X/Y Y/Z \implies X\backslash Z$), it does allow crossing composition:

(18) a. Forward composition ($\rhd B_\times$)

\[
X/\times Y : f \quad Y\backslash_\times Z : g \implies X\backslash_\times Z : \lambda x.f(gx)
\]

b. Backward composition ($\lhd B_\times$)

\[
Y/\times Z : g \quad X\backslash_\times Y : f \implies X/\times Z : \lambda x.f(gx)
\]

The composition rules allow us to permute
Heavy NP Shift

Backward crossed composition allows adjuncts & second arguments to invert with the first verb argument

\[
\begin{array}{c}
I \quad \text{introduced} \\
S/(S/\text{NP}) \quad (((S/\text{NP})/\text{PP}_{TO})/\text{NP}) \\
(S/\text{PP}_{TO})/\text{NP} \\
S/\text{NP} \\
S
\end{array}
\begin{array}{c}
to\text{Marcel} \\
S/(S/\text{PP}_{TO}) \\
S/(S/\text{NP}) \\
S/\text{NP} \\
S
\end{array}
\]

Note, though, that the PP *to Marcel* has a funky category. To understand how this is derived from PP, we need to turn to type-raising
CCG includes type-raising rules, which turn arguments into functions over functions-over-such-arguments.

(19) Forward type-raising ($\geq \mathbf{T}$)

\[ X:a \quad \Rightarrow \quad T/(T\setminus X) : \lambda f.f a \]

(20) Backward type-raising ($\leq \mathbf{T}$)

\[ X:a \quad \Rightarrow \quad T\setminus(T/X) : \lambda f.f a \]

- $X$ ranges over argument categories (e.g., NP and PP).

- The rules are order-preserving, e.g., (19) can turn an NP into a rightward-looking function over leftward functions, preserving the linear order of subjects and predicates.
With type-raising, we have more than one option for derivation, where the semantics work out to be the same:

\[
\frac{\text{Marcel} \quad \text{ran}}{\text{NP: marcel'} \quad \text{S/NP: run'}} < \frac{\text{Marcel}}{\text{NP: marcel'}} \quad \frac{\text{run}}{\text{S/NP: run'}} \quad \frac{\text{S/(S/NP): } \lambda f. f \text{ marcel'}}{\text{S: run' marcel'}} >^T \frac{\text{S: run' marcel'}}{\text{S/NP: run'}}
\]

The variable $X$ on the previous slide is restricted to primitive argument categories:

- this makes it a lexical or morphological-level process (cf. giving an NP case)
Interlude: Object-Control

Object-control verbs differ in their acceptance of heavy NP shift

(21) a. I persuaded Marcel to take a bath.
    b. I persuaded to take a bath my very heavy friend from Hoboken.
    c. I expect Marcel to take a bath.
    d. *I expected to take a bath my very heavy friend from Hoboken.

Lexical entry:

(22) persuade := ((S\NP)/(S_{TO}\NP))/NP : \lambda x \lambda p \lambda y. persuade'(p(ana'x))xy
(23) expect := ((S\NP)/(S_{TO}\NP))/\diamond NP : \lambda x \lambda p \lambda y. persuade'(p(ana'x))xy
Complement-taking verbs like *think*, VP/S, can compose with fragments like *Marcel proved*, S/NP, which accounts for right-node raising (24), and also provides the basis for an analysis of unbounded dependencies (25).

(24) \([I \text{ disproved}]_{S/NP}\) and \([\text{you think that Marcel proved}]_{S/NP}\) completeness.

(25) the result that \([\text{you think that Marcel proved}]_{S/NP}\)

Strings such as *you think that Marcel proved* are taken to be surface constituents of type S/NP.
By using type-raising and functional composition, we can account for such phenomena as right node raising:

\[
\begin{array}{c}
\frac{\text{Marcel}}{\text{NP}} \quad \frac{\text{proved}}{\text{(S\backslash NP)/NP}} \\
\frac{\frac{\frac{S/(S\backslash NP)}{S/\text{NP}}} {\text{T}}}{\text{B}} \quad \frac{\frac{\frac{(X\backslash X)/X}{X}}{X}}{X} \quad \frac{\frac{\frac{\frac{I}{NP}}{NP}}{NP}}{NP} \\
\frac{\frac{\frac{\frac{S/(S\backslash NP)}{S/\text{NP}}} {\text{T}}}{\text{B}}}{\text{B}} \quad \frac{\frac{\frac{\frac{\frac{\frac{\frac{\frac{\text{disproved}}{(S\backslash NP)/NP}}{NP}}{NP}}{NP}}{NP}}{(S/\text{NP})\backslash (S/\text{NP})}}{\text{B}}}{\text{B}} \quad \frac{\frac{\text{completeness}}{\text{NP}}}{\text{NP}}
\end{array}
\]
Likewise, argument cluster coordination is licensed:

\[
\begin{array}{c}
give \\
D TV
\end{array}
\quad
\begin{array}{c}
\frac{\text{Walt}}{TV\setminus D TV} < T \\
\frac{\text{the salt}}{VP\setminus TV} < T \\
\frac{}{VP\setminus D TV} < B
\end{array}
\quad
\begin{array}{c}
\frac{}{VP\setminus D TV}
\end{array}
\quad
\begin{array}{c}
\frac{\text{Malcolm}}{TV\setminus D TV} < T \\
\frac{\text{the tcalcum}}{VP\setminus TV} < T \\
\frac{}{VP\setminus D TV} < B
\end{array}
\quad
\begin{array}{c}
\frac{}{VP\setminus D TV}
\end{array}
\]

\[
\frac{\frac{}{(VP\setminus D TV)/(VP\setminus D TV)}}{VP\setminus D TV} <
\]

\[
\frac{}{VP}
\]

where:

- \( VP = S\setminus NP \)
- \( TV \) (transitive verb) = \( VP/NP \)
- \( DTV \) (ditransitive verb) = \( (VP/NP)/NP \)
Non-standard surface structures are licensed throughout

- The non-traditional constituents motivated for right-node raising and similar coordinations are also possible constituents of non-coordinate sentences like *Marcel proved completeness*.

\[
\begin{align*}
\frac{\text{Marcel}}{\text{NP : marcel}'} & \quad \frac{\text{proved}}{(S\backslash NP) / NP : \text{prove}'} & \quad \frac{\text{completeness}}{\text{NP : completeness}'} \\
\frac{S / (S\backslash NP) : \lambda f . \text{marcel}'}{\text{T} >} & \quad \frac{(S\backslash NP) / NP : \text{prove}'}{\text{B} >} & \frac{\text{S\backslash (S\backslash NP) : } \lambda p . p \text{ completeness}'}{\text{T} <} \\
\frac{S / NP : \lambda x . \text{prove} ' \times \text{marcel}'}{\text{B} <} & \quad \frac{\text{S : prove} ' \text{ completeness} ' \text{ marcel}'}{\text{S : prove} ' \text{ completeness} ' \text{ marcel}'}
\end{align*}
\]

\[
\begin{align*}
\frac{\text{Marcel}}{\text{NP : marcel}'} & \quad \frac{\text{proved}}{(S\backslash NP) / NP : \text{prove}'} & \quad \frac{\text{completeness}}{\text{NP : completeness}'} \\
\frac{S / (S\backslash NP) : \lambda f . \text{marcel}'}{\text{T} >} & \quad \frac{\text{S\backslash (S\backslash NP) : } \lambda p . p \text{ completeness}'}{\text{T} <} & \frac{\text{S\backslash (S\backslash NP) : } \lambda p . p \text{ completeness}'}{\text{T} <} \\
\frac{S / NP : \lambda x . \text{prove} ' \times \text{marcel}'}{\text{B} <} & \quad \frac{\text{S : prove} ' \text{ completeness} ' \text{ marcel}'}{\text{S : prove} ' \text{ completeness} ' \text{ marcel}'}
\end{align*}
\]

- The Principle of Type Transparency guarantees that all such non-standard derivations yield identical interpretations.
Traditionally, there has been the notion of the **Constituent Condition on Rules**

The inputs and outputs of all rules of syntax should be constituents

Thus, CCG has to claim one of the following:

- This claim is not true.

- Items like *Marcel proved* can be constituents.
Try to work out derivations for the following object extractions:

(26) what John knows (NP)

Remember that categories can select for complex categories, and a pronoun like what can select for a gapped construction

(27) What does John know?
One potential advantage of CCG is that you can derive sentences in a completely incremental, left-to-right fashion (some type-raising omitted here) (Dowty 1988):
Summary

- CG has complex lexical entries with syntactic and semantic information and a few rules to apply to them.

- CCG adds combinators and type-raising, which allows for flexible constituency and all sorts of coordination phenomena.

- The “non-constituent” clustering of information is claimed to be consistent with intonational/information structure.
References

