Overview

1. The formalism
   - Categories
   - Slash-typing
   - Application & composition
   - Combinatory principles

2. OpenCCG demo

3. Bounded constructions

4. Other miscellaneous phenomena
   - Intonation
   - Scrambling

5. Implementation & applications
Combinatory means combinational, related to combination, being able to combine or be combined, as in ‘combinatory logic’.
Phrase-structure grammar  

Rules

1. \( S \rightarrow NP \ VP \)
2. \( NP \rightarrow A \ N \mid N \)
3. \( VP \rightarrow V \ NP \)

Categorial Grammar

Application rules

\( > \) \( X/Y \ Y \Rightarrow X \)
\( < \) \( Y \ X\backslash Y \Rightarrow X \)

Terminals & categories

living := NP/N
people := N
food := NP
need := (S\NP)/NP

Terminals

(a) \( A \rightarrow \text{living} \)
(b) \( N \rightarrow \text{people} \mid \text{food} \)
(c) \( V \rightarrow \text{need} \)
Phrase-structure grammar derivation tree

S

NP  VP

A   N  V   NP

living  people  need  N

food
Categorial grammar derivation

\[
\begin{align*}
\frac{\text{living}}{NP/N} & \quad \frac{\text{people}}{N} \\
\frac{NP}{NP} & \quad > \\
\frac{(S\backslash NP)/NP}{NP} & \quad \frac{\text{food}}{NP} \\
\frac{S\backslash NP}{S} & \quad > \\
\end{align*}
\]
Categories & (Pure) Categorial Grammar

- Categories describe syntactical & grammatical properties of constituents
- They are referred to as ‘syntactic types’
- There are two kinds of types
  - Functional types – A, V
  - Atomic types – N, NP
- The choice is arbitrary, but “verbs are functions” is a well established concept

<table>
<thead>
<tr>
<th></th>
<th>Phrase-structure</th>
<th>Categorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>Explicit</td>
<td>Generic</td>
</tr>
<tr>
<td>Derivation</td>
<td>Terminals &amp; nonterminals</td>
<td>Categories only</td>
</tr>
<tr>
<td>Expression–type association</td>
<td>Part of grammar</td>
<td>In corpus</td>
</tr>
</tbody>
</table>

Table: Phrase-structure grammars vs Categorial grammars
Another example: Transitive and intransitive verbs

Transitive verbs

\[(S\setminus NP) / NP\]

\[
\begin{array}{c}
John \\
NP
\end{array}
\quad
\begin{array}{c}
likes \\
(S\setminus NP) / NP
\end{array}
\quad
\begin{array}{c}
potatoes \\
NP
\end{array}
\]

\[
\begin{array}{c}
S\setminus NP
\end{array}
\quad
\begin{array}{c}
\quad
\end{array}
\quad
\begin{array}{c}
S
\end{array}
\]

Intransitive verbs

\[S\setminus NP\]

\[
\begin{array}{c}
John \\
NP
\end{array}
\quad
\begin{array}{c}
sleeps \\
S\setminus NP
\end{array}
\]

\[
\begin{array}{c}
S
\end{array}
\]

“Pure CG (Ajdukiewicz 1935, Bar-Hillel 1953) limits syntactic combination to rules of functional application of functions to arguments to the right or left. [...] This restriction limits expressivity to the level of context-free grammar, and CCG generalizes the context-free core by introducing further rules for combining categories.”

— Steedman and Baldridge, Combinatory Categorial Grammar
A slash has one of four feature values (⋆, ×, ◊, ⋅)
- Slash type imposes limits on possible combination
  - Formalized by application/combination rules
- Written as a subscript (/⋆, /×, /◊, /⋅)
CCG: Slash types – ⋆

- The most restrictive
- Equivalent to the simple slash in CG
- Supertype of all other slash types

Rules

\[(>) \quad X/⋆Y \quad Y \Rightarrow X\]
\[(<) \quad Y \quad X\backslash ⋆Y \Rightarrow X\]
Interlude: Building the logical form

CCG allows you to associate functions with the rules. These functions can then be used to generate the logical representation.

**Syntax**

\[
< \text{expression} > ::= [ < \text{category} > : \lambda < \text{parameter} > [...]. ] < \text{body} >
\]

**Extended rules**

\[
(>) \ X \div_f Y : f \quad Y : a \quad \Rightarrow X : fa
\]

\[
(<) \ Y : a \quad X \div_f Y : f \quad \Rightarrow X : fa
\]
The expressions are left-associative
(prove’completeness’marcel’ = (prove’completeness’’)marcel’’)

Interlude: Building the logical form

\[
\begin{array}{c}
\frac{\text{Marcel}}{\text{NP : marcel'}}} \quad \frac{\text{proved}}{(S\setminus\text{NP})/\text{NP} : \lambda x \lambda y.\text{prove}'xy} \quad \frac{\text{completeness}}{\text{NP} : \text{completeness}'} \quad \frac{}{>}
\end{array}
\]

\[
\begin{array}{c}
\frac{\text{NP} : \text{marcel'}}{\text{S}\setminus\text{NP} : \lambda y.\text{prove}'\text{completeness}'y} \quad \frac{\text{completeness}}{S : \text{prove}'\text{completeness}'\text{marcel'}} \quad \frac{\text{prove}'}{<}
\end{array}
\]
CCG: Slash types – ×

- Allows limited permutation
- Subtype of ⋆

**Rules**

(> \( B_X \)) \( X/\_Y : f \) \( Y\_Z : g \) \( \Rightarrow X\_Z : \lambda z.f(gz) \)

(< \( B_X \)) \( Y/\_Z : g \) \( X\_Y : f \) \( \Rightarrow X/\_Z : \lambda z.f(gz) \)
CCG: Slash types – ⬤

- Allows associativity (composition)
- Subtype of ⬪
- Can be iterated for a fixed $n$

### Rules

\[
(\succ B) \quad X/\diamond Y : f \quad Y/\diamond Z : g \implies X/\diamond Z : \lambda z. f(gz)
\]
\[
(\prec B) \quad Y/\diamond Z : g \quad X/\diamond Y : f \implies X/\diamond Z : \lambda z. f(gz)
\]
- Allows any of the former applications
- Subtype of both ◊ and ×
“Combinatory grammars also include type-raising rules, which turn arguments into functions over functions-over-such-arguments.”

Rules

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

Where X is a primitive category

- Mimics case marking
Marcel
\[ \frac{NP}{S/(S\backslash NP)} \rightarrow^T \frac{NP}{(S\backslash NP)/NP} \rightarrow^B \frac{I}{S/np} \rightarrow^T \frac{NP}{(S\backslash NP)/NP} \rightarrow^B \frac{(X\backslash X)/(X\backslash X)}{S/np} \rightarrow^B \frac{(S/np)\backslash X(S/np)}{S/np} \rightarrow^B \frac{completeness}{NP} \rightarrow S \]
Question
Can we derive this sentence without type raising?
There are approaches for languages with free word order [Karttunen, 1986]
Most of them treat NPs as functors (because of flexion)
Relying on prepositions as indicators of some cases doesn’t always work
... especially when there are no prepositions
**Finnish inessive:**
-ssa indicates “in that place”

<table>
<thead>
<tr>
<th>Nom</th>
<th>Ines</th>
</tr>
</thead>
<tbody>
<tr>
<td>kaupunki</td>
<td>city</td>
</tr>
<tr>
<td>kylä</td>
<td>village</td>
</tr>
<tr>
<td>huone</td>
<td>room</td>
</tr>
<tr>
<td></td>
<td>kaupungissa</td>
</tr>
<tr>
<td></td>
<td>kylässä</td>
</tr>
<tr>
<td></td>
<td>huoneessa</td>
</tr>
<tr>
<td></td>
<td>in city</td>
</tr>
<tr>
<td></td>
<td>in village</td>
</tr>
<tr>
<td></td>
<td>in room</td>
</tr>
</tbody>
</table>

Table: -ssa

All constituents of NP clusters take the same suffix: *suuri valkoinen talo* → *suuressa valkoisessa talossamme* (in our big white house)

Other properties and relationships can be expressed in similar way: *suuressa valkoisessa talossamme* (in our big white house)

More in [Karttunen, 1986]
Adjacency, Consistency

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).

\[
\begin{align*}
X/Y & \quad Y \Rightarrow Z \\
X/\circ Y & \quad Y/\circ Z \quad \Rightarrow \quad X/\times Z
\end{align*}
\]
Question
Is CCG context-free?
See [Vijay-Shanker and Weir, 1994]
OpenCCG demonstration
Bounded constructions

- Reflexivization
- Dative-shift
- Raising
- Object and Subject Control
The fixed-point theorem proved itself.
Bounded constructions: Reflexivization

\[
\text{proved} := (S\backslash NP_{3sn}) \backslash \text{LEX}((S\backslash NP_{3sn})/NP) : \lambda p \lambda y.p(\text{ana}^\prime y)y
\]

The fixed – point theorem

\[
\frac{\text{proved}}{S/(S\backslash NP_{3sn}) : \text{fptheorem}^\prime} \quad \frac{\text{itself}}{(S\backslash NP_{3sn})/(S\backslash NP_{3sn})/NP) : \lambda p \lambda y.p(\text{ana}^\prime y)y}
\]

\[
\frac{(S\backslash NP_{arg})/NP : \lambda x \lambda y.prove^\prime xy}{S/(S\backslash NP_{3sn}) : \text{fptheorem}^\prime} \quad \frac{S/(S\backslash NP_{3sn}) : \lambda y.prove^\prime(\text{ana}^\prime y)y}{S : \text{prove}^\prime(\text{ana}' \text{fptheorem}^\prime) \text{fptheorem}^\prime}
\]

- It’s a clitic!
- *“\text{Itself proved the fixed-point theorem}” is disallowed by the Principle of Inheritance
- Limitations of syntactical/lexical approach: \text{I got the book!} – \text{Can I see it?}
- Very similar approach to dative shifts
Bounded constructions: Raising

Modal verbs and verbs that behave like modals act on almost-complete sentences.

(38) seems := $(S\backslash NP)/(S_{TO}\backslash NP) : \lambda p \lambda y. \text{seem}'(py)$

The primitive $\text{seem}'$ is a modal or intensional operator which the interpretation composes with the complement predicate, thus:

(39) \[
\begin{array}{cccc}
\text{Marcel} & \quad \text{seems} & \quad \text{to} & \quad \text{drink} \\
S/(S\backslash NP) & (S\backslash NP)/(S_{TO}\backslash NP) & (S_{TO}\backslash NP)/(S_{INF}\backslash NP) & S_{INF}\backslash NP \\
: \lambda p.p \text{ marcel}' & : \lambda p.\lambda y.\text{seem}'(py) & : \lambda p.p & : \text{drink}' \\
& & & S_{TO}\backslash NP : \text{drink}' \\
& & & > \\
& & & S_{INF}\backslash NP \quad > \\
& & \quad > \\
& \quad > \\
& S : \text{seem}'(\text{drink}'\text{marcel}')
\end{array}
\]
Some verbs control the infinitival complement’s subject through the object.

- I persuaded Marcel to take a bath.
- I persuaded Marcel to bathe himself.

\[
\text{persuaded} := ((S\backslash NP)/(S_{TO} \backslash NP))/NP : \lambda x \lambda p \lambda y. \text{persuade'}(p(\text{ana}'x))xy \\
\text{persuade'}(\text{bathe'}(\text{ana}'(\text{ana}'\text{marcel'}))(\text{ana}'\text{marcel'}))\text{marcel'}\text{me'}
\]
Some verbs control the subject reference.

- John promised me to go away.
- John ordered me to go away.
(69) give  Walt the salt and Malcolm the talcum

\[
\begin{align*}
&\text{give} \quad \text{Walt} \quad \text{the salt} \quad \text{and} \quad \text{Malcolm} \quad \text{the talcum} \\
&\frac{\text{DTV} \quad \text{TV} \cdot \text{DTV} \quad \text{VP} \cdot \text{TV}}{\text{VP} \cdot \text{DTV}} < B \\
&\quad \frac{(X \cdot X)/X}{\text{TV} \cdot \text{DTV} \quad \text{VP} \cdot \text{TV}} < T \\
&\quad \frac{\text{VP} \cdot \text{DTV}}{} < B \\
&\quad \frac{(\text{VP} \cdot \text{DTV}) \cdot (\text{VP} \cdot \text{DTV})}{\text{VP} \cdot \text{DTV}} < \\
&\quad \frac{\text{VP} \cdot \text{DTV}}{} < \\
&\quad \frac{\text{VP}}{} <
\end{align*}
\]
(93) a. Kyooju-ga komonjo-o gakusee-ni kasita.
Professor-NOM manuscript-ACC student-DAT lent-PAST.CONCL
‘The professor lent the manuscript to the student.’

b. Kyooju-ga komonjo-o gakusee-ni kasita.
\[
\begin{align*}
  S/VP & \rightarrow_T S/TV \\
  VP/TV & \rightarrow_T TV/DTV \rightarrow_T DTV \\
  S/TV & \rightarrow_B S/DTV \\
  S/DTV & \rightarrow_B S
\end{align*}
\]

In this case there is another derivation for the argument cluster:

(94) Kyooju-ga komonjo-o gakusee-ni kasita.
\[
\begin{align*}
  S/VP & \rightarrow_T S/DTV \\
  VP/TV & \rightarrow_T TV/DTV \rightarrow_T DTV \\
  VP/DTV & \rightarrow_B S/DTV \\
  S/DTV & \rightarrow_B S
\end{align*}
\]
(107) Q: I know who proved soundness. But who proved COMPLETENESS?
    A: (MARCEL) (proved COMPLETENESS).
           H* L     L+H*     LH%

(108) Q: I know which result Marcel PREDICTED. But which result did Mar-
    cel PROVE?
    A: (Marcel PROVED)( COMPLETENESS).
           L+H*LH%    H*    LL%

(111) proved := (S_θ \backslash NP_θ)/NP_θ : \lambda x\lambda y. *prove'xy
Intonation

Source: [Ladd, 2008]
CCG can be parsed in low polynomial time (quadratic)
  - However, most sentences are regular
  - This is an upper bound
  - Humans can do it in linear time (or better)
  - Statistical optimization

OpenCCG
  - Parser & realizer
  - Java (and lots of XML, too)
  - Standalone or library
  - LGPL
Applications

- English
  - Dialogs
  - Intonation in generation
  - Generation in for in-car systems
- German
  - Parsing
- Italian, Greek, sign languages
- Most projects seem to be abandoned

More at
Steedman and Baldridge (2011)
Combinatory Categorial Grammar

A. Brett
Lecture notes for Linguistics 484 (University of Victoria)
http://web.uvic.ca/~ling48x/ling484/notes/index.html

Vijay-Shanker, K. and Weir, David J. (1994)

D. R. Ladd (2008)
Intonational Phonology
Cambridge University Press

L. Karttunen (2008)
Radical Lexicalism
https://www.academia.edu/1863598/Radical_Lexicalism

B. Hoffman (1992)
A CCG Approach to Free Word Order Languages