Combinatory Categorial Grammar

Constraining surface realisation in OpenCCG

Recommended Reading


OpenCCG surface realisation

Sentence plans are hybrid logic dependency structures

Sentence plans are hybrid logic dependency structures

What do the grammar and lexicon look like?

Categorial Grammar

Categorial grammars are **lexicalised** grammars

- a grammar is just a “dictionary”
- there are no language-specific grammar rules
- a grammar is a mapping from words to structures

Mapping not one-to-one!
Lexicalised grammars

Many kinds of lexicalised grammar
- Categorial grammars (including CCGs)
- Lexicalised Tree Adjoining Grammars (LTAGs)
- CFGs in Greibach Normal Form

Lexicalised grammars are more efficient than arbitrary CFGs for NLG
- search space is simpler (Koller & Striegnitz, 2002)

Categorial grammars (CGs)

A CG is a mapping from words to categories
- i.e. a set of word-category pairs

What do categories look like?

Atomic categories

Each CG is built around a finite set of atomic categories
- simple, non-composite, atomic symbols
- similar to the symbols of a CFG

Examples:
- S – sentence/clause
- NP – noun phrase
- N - noun
- PP – preposition phrase

Categories

Two kinds of category
- “atomic” categories
- “complex” categories
Atomic categories in XML

Use `atomcat` elements with a `type` attribute

```
<atomcat type="S"/>

<atomcat type="NP"/>
```

Complex categories

- Complex categories are built up from atomic category symbols
- From any *finite* set of atomic categories, can construct an *infinite* set of complex categories using two operators
  - directional slash operators: `/` and `\`

Traditional arithmetic notation is a useful analogy

Arithmetic notation

Arithmetic notation gives us a finite set of digits
- 0, 1, 2, . . . , 9

And a small set of operators for describing an infinite set of numbers: e.g.,
- concatenation: 23, 456, 92789
- addition: 2+7, 7+23, 456+65
- subtraction: 45 - 6, (2+6) - (67 - 34)

Recursive definition

Categories are defined recursively

Atomic categories constitute the “base”
- every atomic category is also a category

The recursion involves the slash operators
- if X and Y are both categories, then so is (X/Y)
- if X and Y are both categories, then so is (X\Y)
Simple examples

<table>
<thead>
<tr>
<th>category</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S\NP)</td>
<td>verb phrase, intransitive verb</td>
</tr>
<tr>
<td>(NP/N)</td>
<td>determiner</td>
</tr>
<tr>
<td>(N\N)</td>
<td>noun post-modifier, relative clause</td>
</tr>
<tr>
<td>(PP/NP)</td>
<td>preposition</td>
</tr>
<tr>
<td>(PP\NP)</td>
<td>postposition</td>
</tr>
</tbody>
</table>

Embedded examples

<table>
<thead>
<tr>
<th>category</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>((S\NP)/NP)</td>
<td>transitive verb</td>
</tr>
<tr>
<td>((S\NP)/NP)/NP</td>
<td>ditransitive verb</td>
</tr>
<tr>
<td>((NN)/NP)</td>
<td>post-nominal preposition</td>
</tr>
<tr>
<td>((S\NP)/(S\NP))</td>
<td>adverb</td>
</tr>
<tr>
<td>((S\NP)/(S\NP)/NP)</td>
<td>reflexive pronoun</td>
</tr>
<tr>
<td>((N\N)/(S\NP))</td>
<td>relative pronoun</td>
</tr>
</tbody>
</table>

Notational conveniences

Drop outermost parentheses
- (S\NP) ⇒ S\NP
- ((N\N)/(S\NP)) ⇒ (N\N)/(S\NP)

Assume left associativity of / and \n
- ((S\NP)/NP)/NP ⇒ S\NP/NP/NP
- (N\N)/(S\NP) ⇒ N\N/(S\NP)

Complex categories in XML

How to represent S\NP:

```xml
<complexcat>
  <atomcat type="S"/>
  <slash dir="\"/>
  <atomcat type="NP"/>
</complexcat>
```
Categories - summary

atomic categories \rightarrow slash operators \rightarrow complex categories

What does X/Y mean?

The kind of word or phrase that combines with a following Y to form an X.

\[ X/Y \rightarrow Y \rightarrow X \]

This rule is called forward application.
Determiners

**Determiner:** word that combines with a following N to give an NP, i.e., an NP/N.

Prepositions

**Preposition:** word that combines with a following NP to give a PP, i.e., a PP/NP.

Derivations

Attributive adjectives

**Attributive adjective:** word that combines with a following N to give another N, i.e., an N/N.
Adjective stacking

N/N great

N/N Italian

N N restaurant

What does X\Y mean?

The kind of word or phrase that combines with a preceding Y to form an X.

Y X\Y X

This rule is called **backward application**.

Intransitive verbs

NP Giovanni's

S\NP rocks

S

**Intransitive verb:** word that combines with a preceding NP to give an S, i.e., an S\NP.

Postpositions

PP\NP above

NP one floor

PP

**Postposition:** word that combines with a preceding NP to give a PP, i.e., a PP\NP.
Transitive verbs

Transitive verb: word that combines with a following NP to give an intransitive verb, S\NP.

Relative pronouns

Relative pronoun: word that combines with a following intransitive verb S\NP to give a noun postmodifier N\N.

Adverbs

Adverb: word that combines with a following intransitive verb S\NP to give another intransitive verb S\NP.

The story so far

• A categorial grammar is a mapping from words to categories
• Categories can be atomic or complex
• Words are combined into phrases by forward and backward application
Our lexicon

Giovanni's :- NP
pasta :- NP
serves :- S\NP/NP
rocks :- S\NP
restaurant :- N
great :- N/N
a :- NP/N
that :- N\N/(S\NP)

What does our grammar do?

• It tells us which strings of words are grammatical and which are not.
• It assigns derivational structure to the grammatical strings.
• But what about semantics?

Remember HLDS?

• The input to the OpenCCG realiser is a hybrid logic dependency structure
• So our categorial lexicon needs to include HLDS in some way
• We need to be able to relate the grammatical sentences with their HLDS (interpretation)
• And also to relate HLDSs to the grammatical sentences that can realise them (generation)

Adding HLDS to our lexicon

Two steps:

1. Add a nominal to each atomic category symbol
2. Add a set of elementary predications of hybrid logic to each lexical category

Then relax and let forward and backward application (i.e. unification) take care of the rest!
Our lexicon again

Giovanni's :- NP
pasta :- NP
serves :- S\NP/NP
rocks :- S\NP
restaurant :- N
great :- N/N
a : NP/N
that :- N\N/(S\NP)

Adding nominals to categories

Giovanni's :- NP
pasta :- NP
serves :- S\NP\NP
rocks :- S\NP
restaurant :- N
great :- N/N
a : NP/N
that :- N\N/(S\NP)

1. Adding nominals to categories

- Subscripts to atomic category symbols
- Referential indices: unique labels for object or event evoked by the word
- By convention, use x, y, z for objects, and e, f, g for events
- Coindexed nominals indicate the referent of the argument is the same as referent of result, e.g., "great"

Adding nominals in XML

<atomcat type="NP"/>
<atomcat type="NP">
  <fs>
    <feat attr="index">
      <lf>
        <nomvar name="X"/>
      </lf>
    </feat>
  </fs>
</atomcat>

Nominal coindexation in XML

<complexcat>
<atomcat type="N">
  <fs>
    <feat attr="index">
      <lf>
        <nomvar name="X"/>
      </lf>
    </feat>
  </fs>
<slash dir="\">
<atomcat type="N">
  <fs>
    <feat attr="index">
      <lf>
        <nomvar name="X"/>
      </lf>
    </feat>
  </fs>
</atomcat>
</complexcat>
2. Adding EPs to categories

Intransitive verbs

<table>
<thead>
<tr>
<th>Giovanni's</th>
<th>rocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP_x ( @x ) Giovanni's</td>
<td>S_e ( S_e \backslash NP_x )</td>
</tr>
<tr>
<td>pasta ( @x ) pasta</td>
<td>( @e ) great, ( @e ) &lt;THEME&gt; x</td>
</tr>
<tr>
<td>serves ( S_e \backslash NP_x / NP_y ) ( @e ) serve, ( @e ) &lt;AGENT&gt; x, ( @e ) &lt;THEME&gt; y</td>
<td>( S_e \backslash NP_y ) ( @y ) Giovanni's</td>
</tr>
<tr>
<td>rocks ( @e ) great, ( @e ) &lt;THEME&gt; x</td>
<td>( S_e \backslash NP_y ) ( @e ) great, ( @e ) &lt;THEME&gt; x</td>
</tr>
<tr>
<td>restaurant ( @e ) restaurant, ( @e ) &lt;THEME&gt; x</td>
<td>( S_e ): @e great, @e &lt;THEME&gt; x, @x Giovanni's</td>
</tr>
<tr>
<td>great ( @e ) great, ( @e ) &lt;THEME&gt; x</td>
<td>restaurant ( @e ) restaurant, @e &lt;THEME&gt; x</td>
</tr>
<tr>
<td>Italian ( N_y / N_y ) ( @f ) italian, ( @f ) &lt;THEME&gt; y</td>
<td>( @e ) restaurant, ( @e ) &lt;THEME&gt; x, ( @f ) italian, ( @f ) &lt;THEME&gt; x</td>
</tr>
<tr>
<td>restaurant ( @e ) restaurant, @e &lt;THEME&gt; x</td>
<td>( N_x ): @e restaurant, @e &lt;THEME&gt; x, @g great, @g &lt;THEME&gt; x</td>
</tr>
</tbody>
</table>

Transitive verbs

<table>
<thead>
<tr>
<th>Giovanni's</th>
<th>serves</th>
<th>pasta</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP_w ( @w ) Giovanni's</td>
<td>S_e ( S_e \backslash NP_x / NP_y ) ( @e ) serve, ( @e ) &lt;AGENT&gt; x, ( @e ) &lt;THEME&gt; y</td>
<td>NP_y ( @e ) serve, ( @e ) &lt;THEME&gt; y, ( @y ) pasta</td>
</tr>
<tr>
<td>serves ( S_e \backslash NP_x / NP_y ) ( @e ) serve, ( @e ) &lt;AGENT&gt; x, ( @e ) &lt;THEME&gt; y</td>
<td>( S_e ): @e serve, @e &lt;AGENT&gt; x, @e &lt;THEME&gt; y, @y pasta</td>
<td></td>
</tr>
<tr>
<td>pasta ( NP_y ) ( @e ) serve, ( @e ) &lt;THEME&gt; y, ( @y ) pasta</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So where are we?

- We’ve seen how to define a lexicon in CG
- We’ve learned about two important operators in CG, i.e., forward and backward application
- We’ve seen how to combine words both
  - Syntactically (derivations, unification), and
  - Semantically (set union of elementary predications)
- But, Combinatory Categorial Grammar gives us much more

From CG to CCG

CCG is an “extension” of CG.

CCG has more rules:
- forward and backward type raising
- forward and backward composition

Everything else remains the same -
- in particular the HLDS representations.

Type Raising

- CCG includes type-raising rules, which turn arguments into functions over functions over such arguments
- Forward type raising

\[
\begin{array}{c}
X \\
Y/(Y\setminus X) \\
\text{John} \\
NP \\
S/(S\setminus NP)
\end{array}
\]

- Example:

\[
\begin{array}{c}
X \\
Y/(Y\setminus X) \\
\text{John} \\
NP \\
S/(S\setminus NP)
\end{array}
\]

- The rules are order preserving. Here we turn an NP into a rightward looking function over leftward functions, preserving the linear order of constituents
Multiple derivations

Q1: I know what restaurant serves French food, but what restaurant serves Italian food?

A1: Babbo serves Italian food.

Q2: I know what kind of food Pierre’s serves, but what kind of food does Babbo serve?

A2: Babbo serves Italian food.

Forward composition

\[
\begin{align*}
X/Y & \xrightarrow{Y/Z} B \\
X/Z & \\
\end{align*}
\]

CCG is more flexible

CCG generates more sentences:

- object relative clauses – “a restaurant that [John likes]_{S/NP}”

- right node raising – “[John likes]_{S/NP} but [Charles hates]_{S/NP} Giovanni’s”

CCG is more flexible

CCG allows one sentence to be derived in many ways -

- reflecting different intonation patterns

- allowing incremental (i.e. left-branching) derivations from a right-branching lexicon
Further Reading

