Sentence Realisation with OpenCCG

Lecture 4
February 1, 2013

Recap

- What have you learned so far?
  - hybrid logic dependency structures (HLDSs)
    - What goes into the OpenCCG surface realiser?
  - categorial grammars (CGs)
  - integrating HLDSs into CGs

- Today:
  - Combinatory Categorial Grammar
    - an extension of Categorial Grammar
    - How chart realisation works

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 more expressive power

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

\[
\begin{array}{c}
X \\
\rightarrow^T \\
Y/(Y\backslash X)
\end{array}
\]

- Example:

\[
\begin{array}{c}
\text{John} \\
\rightarrow^T \\
\text{NP} \\
\rightarrow^T \\
\text{S}/(\text{S}\backslash \text{NP})
\end{array}
\]

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

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”
Right Node Raising

Example:

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

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.

CCC allows Incremental Processing

Can interpret from left to right
Modularity - the NLG pipeline

Communicative Goals

Document Planning

Document Plans

Microplanning

Sentence Plans

Linguistic Realisation

Surface Text

Linguistic realisation with OpenCCG

Sentence Plan

OpenCCG realiser

Grammar

Surface Text

Sentence plans are labelled directed graphs

very-good

serve

Italian

Giovanni's

food

cheap

i.e. nodes, edges, labels

Labelled directed graphs can be represented as hybrid logic formulas

@e (very-good ^ <THEME> (x ^ Giovanni's))
^ @f (serve ^ <AGENT> x ^ <THEME> (y ^ food))
^ @g (Italian ^ <THEME> y)
^ @h (cheap ^ <THEME> y)
Sentence plans are sets of elementary predications in hybrid logic

Why hybrid logic?

Ideal for representing labelled directed graphs
• extension of modal logic

Hybrid logic is well understood
• decidable fragment of first order logic

Hybrid logic allows a graph to be represented in two different, but equivalent, ways -
• hierarchically - good for viewing by humans
• flat - good for processing by computers

Flat representations are particularly useful for doing surface realisation.

Linguistic realisation with OpenCCG

Representing Syntax in XML

Atomic categories - S, NP, N, ...

<atomcat type="S"/>
<atomcat type="NP"/>
<atomcat type="N"/>
Complex categories - (N\N)/(S\NP)

```
<complexcat>
  <atomcat type="N"/>
  <slash dir="\"/>
  <atomcat type="N"/>
  <slash dir="/"/>
  <complexcat>
    <atomcat type="S"/>
    <slash dir="\"/>
    <atomcat type="NP"/>
  </complexcat>
</complexcat>
```

Adding Semantics in XML

```
<satop nomvar="E">
  <prop name="very-good"/>
</satop>

<satop nomvar="E">
  <diamond mode="theme">
    <nomvar name="x"/>
  </diamond>
</satop>
```

A grammar is a lexicon

- Giovanni's :- NP
- food :- N
- Italian :- N/N
- cheap :- N/N
- rocks :- S\NP
- serves :- S\NP/NP
- some :- NP/N

Forward and backward application
Integrating HLDS - (1) Add nominals

Giovanni's :- NPₓ
food :- Nₓ
Italian :- Nₓ/Nₓ
cheap :- Nₓ/Nₓ
rocks :- Sₑ\NPₓ
serves :- Sₑ\NPₓ/NPᵧ
some :- NPₓ/Nₓ

Integrating HLDS - (2) Add EPs

Giovanni's :- NPₓ : @x Giovanni's
food :- Nₓ : @x food
Italian :- Nₓ/Nₓ : @e Italian, @e <THEME> x
cheap :- Nₓ/Nₓ : @e cheap, @e <THEME> x
rocks :- Sₑ\NPₓ : @e very-good, @e <THEME> x
serves :- Sₑ\NPₓ/NPᵧ : @e serve, @e <AGENT> x, @e <THEME> y
some :- NPₓ/Nₓ :

Adding nominals in XML

<atomcat type="S">
  <fs>
    <feat attr="index">
      <lf>
        <nomvar name="x"/>
      </lf>
    </feat>
    <atomcat/>
  </fs>
</atomcat>

Semantic construction
### Type raising and composition

<table>
<thead>
<tr>
<th>Giovanni's</th>
<th>serves</th>
<th>some</th>
<th>cheap</th>
<th>Italian</th>
<th>food</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPx</td>
<td>@x Giovanni's</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sh/(Sh/NPx)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@x Giovanni's</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@h serve, @h &lt;AGENT&gt; x, @h &lt;THEME&gt; z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sh/NPz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@h serve, @h &lt;AGENT&gt; x, @x Giovanni's, @h &lt;THEME&gt; z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sh/Nz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@h serve, @h &lt;AGENT&gt; x, @x Giovanni's, @h &lt;THEME&gt; z, @f cheap, @f &lt;THEME&gt; z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@h serve, @h &lt;AGENT&gt; x, @x Giovanni's, @h &lt;THEME&gt; z, @f cheap, @f &lt;THEME&gt; z, @g Italian, @g &lt;THEME&gt; z, @z food</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Why CCG?

**CCGs are lexicalised**
- allows for efficient NLG

**CCGs are powerful**
- it is easy to generate sentences with unbounded dependencies (object relative clauses, right node raising)

**CCGs are flexible**
- we can simulate incremental processing
- easy to integrate with models of intonation for spoken language generation

**CCGs have transparent semantics**
- easy to integrate with HLDSs

### Chart realisation

An algorithm for converting flat semantic representations into text, using a lexicalised grammar.

First proposed in Martin Kay (1996): "Chart generation" (ACL).

Adapted for OpenCCG by Michael White.

More efficient than other realisation algorithms
- semantic head-driven generation

Inspired by chart parsing.

### Chart parsing?

More efficient than normal top-down or bottom-up parsing
- keeps a record of what it has learned
- so doesn't have to keep repeating the same computations
Example - input sentence plan

1. @a serve
2. @b Italian
3. @c Giovanni's
4. @d food
5. @a <AGENT> c
6. @a <THEME> d
7. @b <THEME> d

Every sentence plan EP gets a unique index.

Example - lexicon

Giovanni's :- NP_x : @x Giovanni's
food :- N_x : @x food
Italian :- N_x/N_x : @y Italian, @y <THEME> x
cheap :- N_x/N_x : @y cheap, @y <THEME> x
rocks :- S_x/NP_y : @x very-good, @x <THEME> y
serves :- S_x/NP_y/NP_z : @x serve, @x <AGENT> y, @x <THEME> z
some :- NP_x/N_x :

Every entry has exactly one "indexing EP", represented by the underline.

Example - sentence plan + lexicon

1. @a serve 5. @a <AGENT> c
2. @b Italian 6. @a <THEME> d
3. @c Giovanni's 7. @b <THEME> d
4. @d food

Every entry has exactly one "indexing EP", represented by the underline.

Step 1: add lexical edges

repeat for every EP $\phi$ in the sentence plan:
repeat for every entry $E$ in the lexicon:
if $E$'s indexing EP matches $\phi$
then add the relevant lexical edge to the chart.
1. @a serve             5. @a <AGENT> c
2. @b Italian            6. @a <THEME> d
3. @c Giovanni's     7. @b <THEME> d
4. @d food

Giovanni's :- NP_x : @x Giovanni's
food :- N_x : @x food
Italian :- N_y/N_x : @y Italian, @y <THEME> x
cheap :- N_y/N_x : @y cheap, @y <THEME> x
rocks :- S_y/N_y : @x very-good, @x <THEME> y
serves :- S_y/N_y/N_z : @x serve, @x <AGENT> y, @x <THEME> z
some :- NP_x/N_x :

The chart (1)

1. serves SaNPcNPd
   1.5,6
   2-4,7

The chart (2)

1. serves SaNPcNPd
   1.5,6
   2-4,7

Italian NcNd
   2.7
   1.3-6
1. @a serve* 5. @a <AGENT> c
2. @b Italian* 6. @a <THEME> d
3. @c Giovanni's 7. @b <THEME> d
4. @d food

Giovanni's :- NP_x : @x Giovanni's
food :- N_x : @x food
Italian :- N_y/N_x : @y Italian, @y <THEME> x
cheap :- N_y/N_x : @y cheap, @y <THEME> x
rocks :- S_y/NP_y : @x very-good, @x <THEME> y
serves :- S_y/NP_y/NP_z : @x serve, @x <AGENT> y, @x <THEME> z
some :- NP_x/N_x :

1. @a serve* 5. @a <AGENT> c*
2. @b Italian* 6. @a <THEME> d*
3. @c Giovanni's* 7. @b <THEME> d*
4. @d food*

Giovanni's :- NP_x : @x Giovanni's
food :- N_x : @x food
Italian :- N_y/N_x : @y Italian, @y <THEME> x
cheap :- N_y/N_x : @y cheap, @y <THEME> x
rocks :- S_y/NP_y : @x very-good, @x <THEME> y
serves :- S_y/NP_y/NP_z : @x serve, @x <AGENT> y, @x <THEME> z
some :- NP_x/N_x :

The chart (3)
Step 1: add lexical edges

repeat for every EP $\phi$ in the sentence plan:
repeat for every entry $E$ in the lexicon:
  if $E$’s indexing EP matches $\phi$
  then add the relevant lexical edge
to the chart.

1. $@a$ serve*
2. $@b$ Italian*
3. $@c$ Giovanni’s*
4. $@d$ food*
5. $@a$ <AGENT> c*
6. $@a$ <THEME> d*
7. $@b$ <THEME> d*

The chart (4)
Step 2: apply CCG rules

repeat for every edge E on the chart:
repeat for every edge F on the chart:
if there is a CCG rule
that can combine E and F
and the EP in-sets of E and F are disjoint
then add the relevant edge to the chart.
### The chart (8)

<table>
<thead>
<tr>
<th>serves</th>
<th>Italian</th>
<th>Giovannii's</th>
<th>food</th>
<th>some</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaNPcNPd</td>
<td>1.5,6</td>
<td>2.7</td>
<td>1,2,4,7</td>
<td>1-3,5,7</td>
</tr>
<tr>
<td>Italian food</td>
<td>Nd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
<tr>
<td>some food</td>
<td>NPd</td>
<td>4</td>
<td>1-3,5-7</td>
<td></td>
</tr>
</tbody>
</table>

### The chart (9)

<table>
<thead>
<tr>
<th>serves</th>
<th>Italian</th>
<th>Giovannii's</th>
<th>food</th>
<th>some</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaNPcNPd</td>
<td>1.5,6</td>
<td>2.7</td>
<td>1,2,4,7</td>
<td>1-3,5,7</td>
</tr>
<tr>
<td>Italian food</td>
<td>Nd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
<tr>
<td>some food</td>
<td>NPd</td>
<td>4</td>
<td>1-3,5-7</td>
<td></td>
</tr>
</tbody>
</table>

### The chart (10)

<table>
<thead>
<tr>
<th>serves</th>
<th>Italian</th>
<th>Giovannii's</th>
<th>food</th>
<th>some</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaNPcNPd</td>
<td>1.5,6</td>
<td>2.7</td>
<td>1,2,4,7</td>
<td>1-3,5,7</td>
</tr>
<tr>
<td>Italian food</td>
<td>Nd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
<tr>
<td>some food</td>
<td>NPd</td>
<td>4</td>
<td>1-3,5-7</td>
<td></td>
</tr>
<tr>
<td>some Italian food</td>
<td>NPd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
</tbody>
</table>

### The chart (11)

<table>
<thead>
<tr>
<th>serves</th>
<th>Italian</th>
<th>Giovannii's</th>
<th>food</th>
<th>some</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaNPcNPd</td>
<td>1.5,6</td>
<td>2.7</td>
<td>1,2,4,7</td>
<td>1-3,5,7</td>
</tr>
<tr>
<td>Italian food</td>
<td>Nd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
<tr>
<td>some food</td>
<td>NPd</td>
<td>4</td>
<td>1-3,5-7</td>
<td></td>
</tr>
<tr>
<td>some Italian food</td>
<td>NPd</td>
<td>2,4,7</td>
<td>1,3,5,6</td>
<td></td>
</tr>
</tbody>
</table>
The chart (16)

<table>
<thead>
<tr>
<th>serves</th>
<th>Italian</th>
<th>Giovanni's</th>
<th>food</th>
<th>some</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sa/NPc/NPd 1.5</td>
<td>Nd/No 1.2</td>
<td>3</td>
<td>1-7</td>
<td>-</td>
</tr>
<tr>
<td>2-4.7</td>
<td>4-7</td>
<td>1.3-6-7</td>
<td>1-7</td>
<td>-</td>
</tr>
</tbody>
</table>

Italian food
Nd 24.7
1.3-5.6
some food
NPd 4
1-3.5-7
Serves some Italian food
Sa/NPc 1.2-7
3
Giovanni's serves some food
Sa 1.3-6
2.7

Giovanni's serves some Italian food
Sa 1-7
-

Chart realisation algorithm

1. repeat for every EP $\phi$ in the sentence plan:
   repeat for every entry $E$ in the lexicon:
   if $E$'s indexing EP matches $\phi$
   then add the relevant lexical edge to the chart.

2. repeat for every entry $E$ in the lexicon:
   if $E$ has no EPs
   then add the relevant lexical edge to the chart.

3. repeat for every edge $E$ on the chart:
   repeat for every edge $F$ on the chart:
   if there is a CCG rule that can combine $E$ and $F$
   and the EP in-sets of $E$ and $F$ are disjoint
   then add the relevant edge to the chart.

Result!

"Giovanni's serves some Italian food"

Another example

1. @a cheap
2. @b Italian
3. @c food
4. @a <THEME> c
5. @b <THEME> c

Giovanni's :: NP_x : @x Giovanni's
food :: N_x : @x food
Italian :: N_y/N_z : @y Italian, @y <THEME> x
cheap :: N_y/N_z : @y cheap, @y <THEME> x
rocks :: S_z/NP_x : @x very-good, @x <THEME> y
serves :: S_x/NP_y/NP_z : @x serve, @x <AGENT> y, @x <THEME> z
some :: NP_x/N_x :
The chart - lexical edges added

The chart - CCG rules applied

Result

One for the road

"Italian cheap food"
"some Italian cheap food"
"cheap Italian food"
"some cheap Italian food"

Giovanni's :- NPₓ : @x Giovanni's
great :- Aₓ : @x very-good
rocks :- S₁, NPᵧ : @x very-good, @x <THEME> y
is :- Sₓ, NPᵧ/Aₓ : @x <THEME> y
What you need to know

Convert a labelled directed graph into a set of hybrid logic elementary predications, and vice versa.

Given a CCG lexicon, show how sentence S can be derived
• including semantic representations

Given a CCG lexicon, show how labelled directed graph G can be realised, using the chart realisation algorithm.