8. Mildly Context-Sensitive Grammar Formalisms

Gerald Penn
Department of Computer Science, University of Toronto

Based on slides by David Smith, Dan Klein, Stephen Clark and Eva Banik
Combinatory Categorial Grammar
Combinatory Categorial Grammar (CCG)

- Categorial grammar (CG) is one of the oldest grammar formalisms
- *Combinatory* Categorial Grammar now well established and computationally well founded (Steedman, 1996, 2000)
- Account of syntax; semantics; prosody and information structure; automatic parsers; generation
Combinatory Categorial Grammar (CCG)

- CCG is a lexicalized grammar
- An elementary syntactic structure – for CCG a lexical category – is assigned to each word in a sentence:
  \[ \text{walked: } S\NP \text{“give me an NP to my left and I return a sentence”} \]
- A small number of rules define how categories can combine:
  - Rules based on the combinators from Combinatory Logic
CCG Lexical Categories

- Atomic categories: S, N, NP, PP, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: S/NP walked
  - transitive verb: (S/NP)/NP respected
  - ditransitive verb: ((S/NP)/NP)/NP gave
- Complex categories can encode modification
  - PP nominal: (NP/NP)/NP
  - PP verbal: ((S/NP)/(S/NP))/NP
Simple CCG Derivation

interleukin − 10 inhibits production

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

> forward application

< backward application
Function Application Schemata

- Forward (>) and backward (<) application:

\[
\begin{align*}
X / Y & \quad Y \quad \Rightarrow \quad X \quad (>)
\quad Y & \quad X \backslash Y \quad \Rightarrow \quad X \quad (<)
\end{align*}
\]
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

```
S
   /\        /
  |  \       |
 NP   S\NP   NP
       |      |
      /\     |
     interleukin-10 inhibits production
```

interleukin-10 inhibits production
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

```
interleukin-10 inhibits production
```
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N N (NP/NP)/(S/NP) NP (S\NP)/NP
Extraction out of a Relative Clause

The company which Microsoft bought

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

S/(S\NP)  \to^T

> T  type-raising
Extraction out of a Relative Clause

The company which Microsoft bought

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

Microsoft bought

NP (S\NP)/NP

S/(S\NP) >T

S/NP >B

> T type-raising

> B forward composition
Extraction out of a Relative Clause

The company which Microsoft bought

\[NP/N \quad N \quad (NP/\ NP)/(S/\ NP)\quad NP \quad (S/\ NP)/NP\]

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

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
Extraction out of a Relative Clause

The company which Microsoft bought

\[
\begin{align*}
\text{The} & \quad \text{company} & \quad \text{which} & \quad \text{Microsoft} & \quad \text{bought} \\
NP/N & \quad N & \quad (NP/NP)/(S/NP) & \quad NP & \quad (S/NP)/NP \\
\quad & \quad T & \quad S/(S/NP) & \quad B & \\
\quad & \quad \text{\textless} & \quad S/NP & \quad \text{\textrangle} \\
\quad & \quad \text{\textless} & \quad NP\backslash NP & \quad \text{\textrangle} \\
\quad & \quad NP & \\
\end{align*}
\]
Forward Composition and Type-Raising

- Forward composition ($\succ_B$):

\[ X / Y \quad Y / Z \Rightarrow X / Z \ (\succ_B) \]

- Type-raising ($T$):

\[ X \Rightarrow T / (T \setminus X) \ (\succ_T) \]
\[ X \Rightarrow T \setminus (T / X) \ (\prec_T) \]

- Extra combinatory rules increase the weak generative power to mild context-sensitivity
“Non-constituents” in CCG – Right Node Raising

Google $\rightarrow$ NP

$sells$ $\rightarrow$ $(S\backslash NP)/NP$

but $\rightarrow$ conj

Microsoft $\rightarrow$ NP

$\rightarrow$ $(S\backslash NP)/NP$

shares $\rightarrow$ NP

$\rightarrow$ T

> T type-raisning
“Non-constituents” in CCG – Right Node Raising

Google \( \text{sells} \) but Microsoft \( \text{buys} \) shares

\[
\begin{align*}
\text{NP} \quad \frac{(S \backslash NP) / NP}{\text{conj}} \quad \text{NP} \quad \frac{(S \backslash NP) / NP}{\text{NP}} \\
S / (S \backslash NP) \quad S / (S \backslash NP) \\
\end{align*}
\]

\( > T \) type-raising

\( > B \) forward composition

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\begin{align*}
S/(S\backslash NP) \rightarrow^T & \quad S/NP \\
NP \quad (S\backslash NP)/NP \rightarrow^B & \quad (S\backslash NP)/NP \\
S \rightarrow^T & \quad S/(S\backslash NP) \\
NP \quad (S\backslash NP)/NP \rightarrow^B & \quad (S\backslash NP)/NP \\
S/NP & \quad <\Phi>
\end{align*}
\]
"Non-constituents" in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\begin{align*}
  &NP & (S\backslash NP)/NP & \text{conj} & NP & (S\backslash NP)/NP & NP \\
  &S/(S\backslash NP) & >^T & S\backslash NP & >^T & S\backslash NP & <\Phi>
\end{align*}
\]
Combinatory Categorial Grammar

- CCG is *mildly* context sensitive
- Natural language is provably non-context free
- Constructions in Dutch and Swiss German (Shieber, 1985) require more than context free power for their analysis
  - these have *crossing* dependencies (which CCG can handle)

![Language Hierarchy Diagram]

**Language Hierarchy Diagram**

- Type 0 languages
- Context sensitive languages
- Context free languages
- Regular languages
- Mildly context sensitive languages = natural languages (?)

*Stephen Clark  
Practical Linguistically Motivated Parsing  
JHU, June 2009*
CCG Semantics

- Categories encode argument sequences
- Parallel syntactic combinator operations and lambda calculus semantic operations

\[
\begin{align*}
John & \vdash NP : john' \\
shares & \vdash NP : shares' \\
buys & \vdash (S\setminus NP)/NP : \lambda x. \lambda y. buys'xy \\
sleeps & \vdash S\setminus NP : \lambda x. sleeps'x \\
well & \vdash (S\setminus NP)\setminus(S\setminus NP) : \lambda f. \lambda x. well'(fx)
\end{align*}
\]
## CCG Semantics

<table>
<thead>
<tr>
<th>Left arg.</th>
<th>Right arg.</th>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X/Y : f$</td>
<td>$Y : a$</td>
<td>Forward application</td>
<td>$X : f(a)$</td>
</tr>
<tr>
<td>$Y : a$</td>
<td>$X\backslash Y : f$</td>
<td>Backward application</td>
<td>$X : f(a)$</td>
</tr>
<tr>
<td>$X/Y : f$</td>
<td>$Y/Z : g$</td>
<td>Forward composition</td>
<td>$X/Z : \lambda x.f(g(x))$</td>
</tr>
<tr>
<td>$X : a$</td>
<td></td>
<td>Type raising</td>
<td>$T/(T\backslash X) : \lambda f.f(a)$</td>
</tr>
</tbody>
</table>

etc.
Tree Adjoining Grammar
TAG Building Blocks

• Elementary trees (of many depths)
• Substitution at ↓
• Tree Substitution Grammar equivalent to CFG
TAG Building Blocks

- Auxiliary trees for *adjunction*
- Adds extra power beyond CFG

\[ \begin{align*}
\alpha_1 & \quad \text{NP} \\
\text{Harry} & \\
\alpha_2 & \quad \text{S} \\
\text{NP} & \quad \text{VP} \\
\text{likes} & \quad \text{NP} \\
\alpha_3 & \quad \text{NP} \\
\text{peanuts} & \\
\beta & \quad \text{VP} \\
\text{Adv} & \quad \text{passionately}
\end{align*} \]
Semantics

\[ Harry(x) \land \text{likes}(e, x, y) \land \text{peanuts}(y) \land \text{passionately}(e) \]