Parsing Combinatory Categorial Grammar with Answer Set Programming: Preliminary Report

Yuliya Lierler  Peter Schüller

Computer Science Department, University of Kentucky
KBS Group – Institut für Informationssysteme, Technische Universität Wien

WLP – September 30, 2011

supported by: CRA/NSF 2010 Computing Innovation Fellowship, Vienna Science and Technology Fund (WWTF) project ICT08-020
Natural Language Parsing

- Required for transforming natural language into KR language(s)
- First step: obtaining sentence structure
- Example:
  
  John saw the astronomer with the telescope.
  
  ⇒ two distinct structures = “structural ambiguity”
  
  John [saw the astronomer] [with the telescope].
  John saw [the astronomer [with the telescope]].

- “Wide-coverage parsing”
  
  ⇒ parsing unrestricted natural language (e.g., newspaper)
This Work

- **Goals of this work:**
  - Wide-coverage parsing
  - obtaining all distinct structures

- **Approach:**
  - Parsing represented as planning
  - Answer Set Programming for realizing the planning
  - Use of ASP with Function symbols
  - Optimization for best-effort parsing
  - Framework using python, gringo, clasp
  - Visualization
Planning:

- actions, executability, effects
- initial and goal state
  \[ \Rightarrow \text{find sequence of actions from initial to goal state} \]

Answer Set Programming:

- declarative programming paradigm
- logic programming rules and function symbols
- stable model semantics
- guess & check — resp. GENERATE - DEFINE - TEST paradigm
Using ASP for Planning

- GENERATE all possible action sequences
- DEFINE action effects starting from initial state
- TEST executability
- TEST goal conditions
Categories for words and constituents:

- **Atomic categories**, e.g.: noun $N$, noun phrase $NP$, sentence $S$
- **Complex categories**: specify argument and result, e.g.:
  - $S\backslash NP \Rightarrow$ expect $NP$ to the left, result is $S$
  - $(S\backslash NP)/NP \Rightarrow$ expect $NP$ to the right, result is $S\backslash NP$

Given CCG lexicon $\Rightarrow$ **represent words by corresponding categories**:

<table>
<thead>
<tr>
<th>The</th>
<th>dog</th>
<th>bit</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP/N$</td>
<td>$N$</td>
<td>$(S\backslash NP)/NP$</td>
<td>$NP$</td>
</tr>
</tbody>
</table>

- **Words may have multiple categories** $\Rightarrow$ handle all combinations
Combinators are grammar rules that combine categories:

- **Application**
  \[
  \frac{A/B \quad B}{A} > A
  \]

- **Composition**
  \[
  \frac{A/B \quad B/C}{A/C} > B
  \]

- **Type Raising**
  \[
  \frac{A}{B/(B\setminus A)} > T
  \]
Combinatory Categorial Grammar (2)

- **Combinators** are grammar rules that combine categories:

  - application: \( \frac{A/B \ B}{A} > \)
  - composition: \( \frac{A/B \ B/C}{A/C} > B \)
  - type raising: \( \frac{A}{B/(B\setminus A)} > T \)

- **Instantiation** of combinators used for parsing, e.g.:

  \[
  \frac{NP/N \ N}{NP} >
  \]

- **Example derivation, resp. parse tree:**

  \[
  \frac{The}{NP/N} \frac{dog}{N} > \frac{bit}{(S\setminus NP)/NP} \frac{John}{NP} > \frac{(S\setminus NP)/NP}{S\setminus NP} < \frac{NP}{S}
  \]
State = Abstract Sequence Representation (ASR):
ASR contains categories, numbered from left to right.
Example:

\[
\begin{array}{cccc}
\text{The} & \text{dog} & \text{bit} & \text{John} \\
NP/N & N & (S\backslash NP)/NP & NP
\end{array}
\]

is represented by the Initial State ASR:

\[ [NP/N^1, N^2, (S\backslash NP)/NP^3, NP^4] \]
State = Abstract Sequence Representation (ASR): ASR contains categories, numbered from left to right.

Example:

\[
\begin{align*}
\text{The} & \quad \text{dog} & \quad \text{bit} & \quad \text{John} \\
\frac{NP}{N} & \quad \frac{N}{NP} & \quad \frac{(S\backslash NP)/NP}{NP} & \quad \frac{NP}{NP}
\end{align*}
\]

is represented by the Initial State ASR:

\[
[NP/N^1, \ N^2, \ (S\backslash NP)/NP^3, \ NP^4]
\]

Actions = Combinators that operate on precondition ASR. Combinators yield a single result category. Result category is numbered like the leftmost precondition category.

Example:

\[
\begin{align*}
\frac{NP}{N^1} & \quad \frac{N^2}{NP^1} \\
\frac{NP^1}{NP^1}
\end{align*}
\]
Action Effect = replace precondition sequence by result category.

Example:

- Time step 1: \( \text{ASR} = [NP^1, (S\setminus NP)/NP^3, NP^4] \)
  \[ \Rightarrow \text{action} \quad \frac{(S\setminus NP)/NP^3}{S\setminus NP^3} \quad NP^4 > \]

- Time step 2: \( \text{ASR} = [NP^1, S\setminus NP^3] \)
  \[ \Rightarrow \text{action} \quad \frac{NP^1}{S\setminus NP^3} \quad S^1 > \]

- Time step 3: \( \text{ASR} = [S^1] \)

Goal State = ASR \([S^1]\)

Concurrent execution of actions possible.
Spurious CCG Parsers

- Redundant parse trees yield same semantic result.

Example:

\[
\begin{array}{ccc}
\text{The} & \text{dog} & \text{bit} \\
\text{NP}/N & \lambda \alpha. \alpha & (S\setminus NP)/NP \\
& N d & \lambda \alpha \beta. b(\beta, \alpha) \\
& NP d & S\setminus NP \\
& & \lambda \beta. b(\beta, j) \\
& & S \ b(d, j) \\
\end{array}
\]
Spurious CCG Parses

- Redundant parse trees yield same semantic result.

Example:

\[
\begin{align*}
\text{The} & \quad \text{dog} \\
\frac{\text{NP}/\text{N}}{\lambda \alpha. \alpha} & \quad \frac{\text{N}}{d} \\
\frac{\text{NP}}{d} & > \\
\frac{\text{(S/NP)}/\text{NP}}{\lambda \alpha \beta. b(\beta, \alpha)} & > \\
\frac{\text{S/NP}}{\lambda \beta. b(\beta, j)} & < \\
\text{S} & b(d, j)
\end{align*}
\]

versus

\[
\begin{align*}
\text{The} & \quad \text{dog} \\
\frac{\text{NP}/\text{N}}{\lambda \alpha. \alpha} & \quad \frac{\text{N}}{d} \\
\frac{\text{NP}}{d} & > \text{T} \\
\frac{\text{S/(S/NP)}}{\lambda \gamma \delta. \gamma(d, \delta)} & > \text{B} \\
\frac{\text{S/NP}}{\lambda \delta. [\lambda \alpha \beta. b(\beta, \alpha)](d, \delta)} & = \lambda \delta. b(d, \delta) \\
\frac{\text{S}}{b(d, j)} & >
\end{align*}
\]

- Such parse trees are called spurious and should be suppressed.
AspCCGTk implements known methods for eliminating spurious parses:

- Allow only one branching direction for functional compositions:

  $W/X \ X/Y \ Y/Z \xrightarrow{B} W/Y \xrightarrow{B} W/Z$  

  $W/X \ X/Y \ Y/Z \xrightarrow{B} X/Z \xrightarrow{B} W/Z$  

- Disallow certain combinations of rule applications:
ASPCCGTk implements known methods for eliminating spurious parses:

- **Allow only one branching direction** for functional compositions:

  \[
  \frac{W/X \ X/Y \ Y/Z}{W/Y} \xrightarrow{B} \frac{W/Z}{W/Y} \xrightarrow{B} \frac{W/Z}{B}
  \]

  \[
  \xrightarrow{\text{normalize}} \frac{W/X \ X/Y \ Y/Z}{X/Z} \xrightarrow{B} \frac{W/Z}{B}
  \]

- **Disallow certain combinations** of rule applications:

  \[
  \frac{X/Y \ Y/Z \ Z}{X/Z} \xrightarrow{B} \frac{X/Z}{X} \xrightarrow{B} \frac{X/Z}{X}
  \]

  \[
  \xrightarrow{\text{normalize}} \frac{X/Y \ Y/Z \ Z}{Y} \xrightarrow{X}
  \]

- **Implemented as executability conditions** of actions.
ASP Encoding (State Representation)

- posCat\((p, c, t)\) $\Rightarrow$ category \(c\) is annotated with (position) \(p\) at time \(t\)
- posAdjacent\((p_L, p_R, t)\) $\Rightarrow$ position \(p_L\) is adjacent to position \(p_R\) at time \(t\)
- categories represented as function symbols \(rfunc\), \(lfunc\), and strings

Example: “The dog bit John.” is represented as the EDB

\[
\begin{align*}
&\text{posCat}(1, rfunc(“NP”, “N”), 0). \quad \text{posCat}(2, “N”, 0). \\
&\text{posCat}(3, rfunc(lfunc(“S”, “NP”), “NP”), 0). \quad \text{posCat}(4, “NP”, 0). \\
&\text{posAdjacent}(1, 2, 0). \quad \text{posAdjacent}(2, 3, 0). \quad \text{posAdjacent}(3, 4, 0).
\end{align*}
\]
GENERATE part of encoding for $\frac{A/B}{B} > A$

\[
\{\text{occurs}(\text{ruleFwdAppl}, L, R, T)\} \leftarrow \\
\text{posCat}(L, \text{rfunc}(A, B), T), \text{posCat}(R, B, T), \text{posAdjacent}(L, R, T), \text{not ban}(\text{ruleFwdAppl}, L, T), \text{time}(T), T < \text{maxsteps}.
\]

DEFINE part for $\text{ban}/2$ realizes normalizations
DEFINE part of encoding for explicit effects of $\frac{A/B}{B} > A$:

$$posCat(L, A, T+1) \leftarrow \begin{array}{l}
\text{occurs}(\text{ruleFwdAppl}, L, R, T), \\
\text{posCat}(L, rfunc(A, B), T), \\
\text{time}(T), \; T < \text{maxsteps}.
\end{array}$$

DEFINE part of encoding for implicit effect called “affectedness”:

$$posAffected(L, T+1) \leftarrow \begin{array}{l}
\text{occurs}(\text{Act}, L, R, T), \; \text{binary}(\text{Act}), \\
\text{time}(T), \; T < \text{maxsteps}.
\end{array}$$
DEFINE part of encoding for ASR inertia:

\[ \text{posCat}(P, C, T+1) \leftarrow \text{posCat}(P, C, T), \]
\[ \text{not posAffected}(P, T+1), \]
\[ \text{time}(T), \quad T < \text{maxsteps}. \]

TEST forbids invalid concurrency

TEST enforces reaching the goal state
• implemented in ASP controlled by python
• using/exteding BioASP library in potassco

http://www.kr.tuwien.ac.at/staff/ps/aspccggtk/
Visualisation

\[
\begin{align*}
\frac{\text{The}}{\text{NP/N}} & \quad \frac{\text{dog}}{\text{N}} \\
\frac{\text{NP}}{\text{N}} & \quad > \\
\frac{\text{bit}}{(S\backslash\text{NP})/\text{NP}} & \quad \frac{\text{John}}{\text{NP}} \\
\frac{\text{S\backslashNP}}{\text{S}} & \quad < \\
\end{align*}
\]

- uses IDPDraw
- in python: convert \( rfunc(NP, N) \) into “NP/N”
Best-effort parsing

- Assume, in our lexicon, “bit” always requires someone being bitten (i.e., assume there is no intransitive category for “bit”).
- “The dog bit” then is not recognized as a sentence.
Assume, in our lexicon, “bit” always requires someone being bitten (i.e., assume there is no intransitive category for “bit”).

“The dog bit” then is not recognized as a sentence.

ASPCCGTK will not find a parse and provide a best-effort parse:

```
The          dog          bit
NP/N          N            (S\NP)/NP
\--------------\         \----------\          \        \        \   
 NP            NP          T            B
\----------\        \----------\          \        \        \   
 S/(S\NP)      S/NP
\-------------------\                      \        \        \   
 S/NP
```
Recent, Ongoing and Future Work

Recent and Ongoing:

- using incremental solver ICLINGO
- performance evaluation on large corpus CCGBank
- different encodings (configuration, CYK)
  (⇒ there we have the main effort in grounding)

Future:

- add features to make ASPCCGTk comparable to C&C
  (probably the most widely used wide coverage CCG parser)
- make compatible with Boxer
- correctness evaluation on large corpus


