FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# 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, Answer Set Programming

Planning:

- actions, executability, effects
- initial and goal state
- $\Rightarrow$ 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

# Combinatory Categorial Grammar (1)

- 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:

$$\frac{The}{NP/N} \qquad \frac{dog}{N} \qquad \frac{bit}{(S\backslash NP)/NP} \qquad \frac{John}{NP}$$

- Words may have multiple categories $\Rightarrow$ handle all combinations

# Combinatory Categorial Grammar (2)

- Combinators are grammar rules that combine categories:

| application | composition | type raising |
| --- | --- | --- |
| $\dfrac{A/B \quad B}{A} >$ | $\dfrac{A/B \quad B/C}{A/C} >\mathbf{B}$ | $\dfrac{A}{B/(B\backslash A)} >\mathbf{T}$ |

# Combinatory Categorial Grammar (2)

- Combinators are grammar rules that combine categories:

| application | composition | type raising |
| --- | --- | --- |
| $\dfrac{A/B \quad B}{A} >$ | $\dfrac{A/B \quad B/C}{A/C} >\mathbf{B}$ | $\dfrac{A}{B/(B\backslash A)} >\mathbf{T}$ |

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

$$\frac{NP/N \quad N}{NP} >$$

- Example derivation, resp. parse tree:

$$\cfrac{\cfrac{\cfrac{The}{NP/N} \quad \cfrac{dog}{N}}{NP} > \qquad \cfrac{\cfrac{bit}{(S\backslash NP)/NP} \quad \cfrac{John}{NP}}{S\backslash NP} >}{S} <$$

# Using Planning to Realize CCG (1)

- State = Abstract Sequence Representation (ASR):
  ASR contains categories, numbered from left to right.

  Example:

$$\frac{\textit{The}}{NP/N} \qquad \frac{\textit{dog}}{N} \qquad \frac{\textit{bit}}{(S\backslash NP)/NP} \qquad \frac{\textit{John}}{NP}$$

  is represented by the Initial State ASR:

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

# Using Planning to Realize CCG (1)

- State = Abstract Sequence Representation (ASR):
  ASR contains categories, numbered from left to right.

  Example:

  $$\frac{\textit{The}}{NP/N} \qquad \frac{\textit{dog}}{N} \qquad \frac{\textit{bit}}{(S\backslash NP)/NP} \qquad \frac{\textit{John}}{NP}$$

  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:

  $$\frac{NP/N^1 \quad N^2}{NP^1} >$$

# Using Planning to Realize CCG (2)

- Action Effect = replace precondition sequence by result category.

  Example:

  time step 1: ASR $= [NP^1, (S\backslash NP)/NP^3, NP^4]$

  $\Rightarrow$ action $$\frac{(S\backslash NP)/NP^3 \quad NP^4}{S\backslash NP^3} >$$

  time step 2: ASR $= [NP^1, S\backslash NP^3]$

  $\Rightarrow$ action $$\frac{NP^1 \quad S\backslash NP^3}{S^1} >$$

  time step 3: ASR $= [S^1]$

- Goal State = ASR $[S^1]$
- Concurrent execution of actions possible.

# Spurious CCG Parses

- Redundant parse trees yield same semantic result.

Example:

$$\dfrac{\dfrac{\dfrac{The}{NP/N\ \lambda\alpha.\alpha}\quad\dfrac{dog}{N\ d}}{NP\ d}> \qquad \dfrac{\dfrac{bit}{(S\backslash NP)/NP\ \lambda\alpha\beta.b(\beta,\alpha)}\quad\dfrac{John}{NP\ j}}{S\backslash NP\ \lambda\beta.b(\beta,j)}>}{S\ b(d,j)}<$$

# Spurious CCG Parses

- Redundant parse trees yield same semantic result.

Example:

$$\dfrac{\dfrac{\dfrac{\mathit{The}}{NP/N\ \lambda\alpha.\alpha}\quad\dfrac{\mathit{dog}}{N\ d}}{NP\ d}> \qquad \dfrac{\dfrac{\mathit{bit}}{(S\backslash NP)/NP\ \lambda\alpha\beta.b(\beta,\alpha)}\quad\dfrac{\mathit{John}}{NP\ j}}{S\backslash NP\ \lambda\beta.b(\beta,j)}>}{S\ b(d,j)}<$$

versus

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{The}}{NP/N\ \lambda\alpha.\alpha}\quad\dfrac{\mathit{dog}}{N\ d}}{NP\ d}>}{S/(S\backslash NP)\ \lambda\gamma\delta.\gamma(d,\delta)}>\mathbf{T} \qquad \dfrac{\mathit{bit}}{(S\backslash NP)/NP\ \lambda\alpha\beta.b(\beta,\alpha)}}{S/NP\ \lambda\delta.[\lambda\alpha\beta.b(\beta,\alpha)](d,\delta) = \lambda\delta.b(d,\delta)}>\mathbf{B} \qquad \dfrac{\mathit{John}}{NP\ j}}{S\ b(d,j)}>$$

- Such parse trees are called spurious and should be suppressed.

# Spurious Parse Normalization

ASPCCGTK implements known methods for eliminating spurious parses:

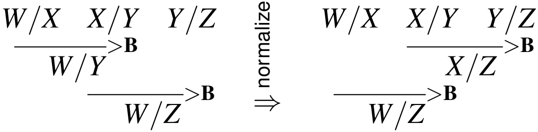
- Allow only one branching direction for functional compositions:

$$\frac{\frac{W/X \quad X/Y}{W/Y}{>}\mathbf{B} \quad Y/Z}{W/Z}{>}\mathbf{B} \quad \xRightarrow{\text{normalize}} \quad \frac{W/X \quad \frac{X/Y \quad Y/Z}{X/Z}{>}\mathbf{B}}{W/Z}{>}\mathbf{B}$$

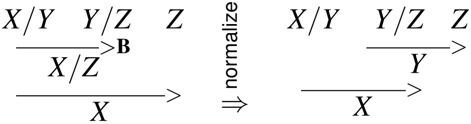
# Spurious Parse Normalization

ASPCCGTK implements known methods for eliminating spurious parses:

- Allow only one branching direction for functional compositions:




- Disallow certain combinations of rule applications:




- 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

$$posCat(1, rfunc(\text{“}NP\text{”}, \text{“}N\text{”}), 0).\ \ posCat(2, \text{“}N\text{”}, 0).$$
$$posCat(3, rfunc(lfunc(\text{“}S\text{”}, \text{“}NP\text{”}), \text{“}NP\text{”}), 0).\ \ posCat(4, \text{“}NP\text{”}, 0).$$
$$posAdjacent(1, 2, 0).\ \ posAdjacent(2, 3, 0).\ \ posAdjacent(3, 4, 0).$$

# ASP Encoding (Action Generation)

- GENERATE part of encoding for $\dfrac{A/B \quad B}{A} >$

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

- DEFINE part for *ban*/2 realizes normalizations

# ASP Encoding (Effects)

- DEFINE part of encoding for explicit effects of $\frac{A/B \quad B}{A} >$

$$posCat(L, A, T{+}1) \leftarrow \quad occurs(ruleFwdAppl, L, R, T), \\ posCat(L, rfunc(A, B), T), \\ time(T),\ T < maxsteps.$$

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

$$posAffected(L, T{+}1) \leftarrow \quad occurs(Act, L, R, T),\ binary(Act), \\ time(T),\ T < maxsteps.$$

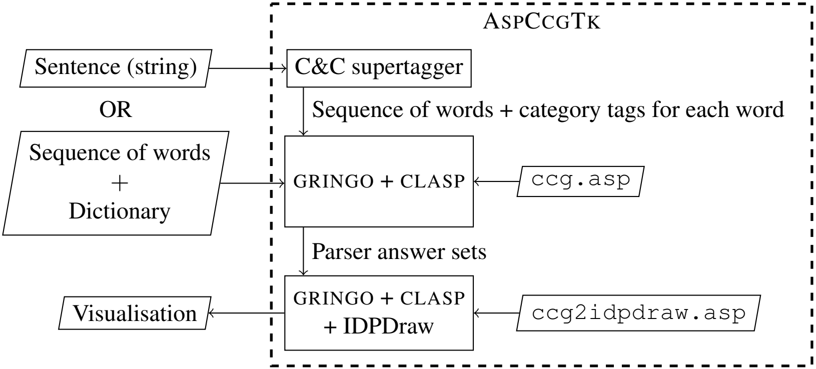
# ASP Encoding (Inertia and Goal)

- DEFINE part of encoding for ASR inertia:

$$
\begin{aligned}
posCat(P, C, T{+}1) \leftarrow \quad & posCat(P, C, T), \\
& not\ posAffected(P, T{+}1), \\
& time(T),\ T < maxsteps.
\end{aligned}
$$

- TEST forbids invalid concurrency
- TEST enforces reaching the goal state

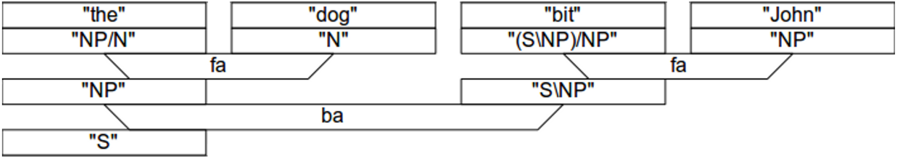
# ASPCCG Toolkit




- implemented in ASP controlled by python
- using/exteding BioASP library in potassco
- `http://www.kr.tuwien.ac.at/staff/ps/aspccgtk/`

# Visualisation

$$\dfrac{\dfrac{\dfrac{\mathit{The}}{NP/N}\quad \dfrac{\mathit{dog}}{N}}{NP}> \qquad \dfrac{\dfrac{\mathit{bit}}{(S\backslash NP)/NP}\quad \dfrac{\mathit{John}}{NP}}{S\backslash NP}>}{S}<$$




- 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.

# 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.
- AspCcgTk will not find a parse and provide a best-effort parse:

$$
\begin{array}{ccc}
\textit{The} & \textit{dog} & \textit{bit} \\
\hline NP/N & \hline N & \hline (S\backslash NP)/NP \\
\multicolumn{2}{c}{\overline{\qquad NP \qquad}{>}} & \\
\overline{S/(S\backslash NP)}{>}\mathbf{T} & & \\
\multicolumn{3}{c}{\overline{\qquad\qquad S/NP \qquad\qquad}{>}\mathbf{B}}
\end{array}
$$

# Recent, Ongoing and Future Work

Recent and Ongoing:

- using incremental solver ICLINGO
- performance evaluation on large corpus CCGBank
- different encodings (configuration, CYK)
  ($\Rightarrow$ 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

# References I


- Alessandro Cimatti, Marco Pistore, and Paolo Traverso. Automated planning. In *Handbook of Knowledge Representation*. Elsevier, 2008.
- Jason Eisner. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics (ACL'96)*, pages 79–86, 1996.
- Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Comput. Logic*, 5:206–263, April 2004.
- Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *IJCAI'07*, pages 386–392, 2007.
- Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- Julia Hockenmaier and Mark Steedman. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Comput. Linguist.*, 33:355–396, 2007.

# Another sample visualisation