ASP for Integral Syntactic Parsing and Semantic Disambiguation

Yuliya Lierler

University of Nebraska at Omaha

Joint work with Peter Schüller (Sabanci University)
Overview

1. Introduction to Answer Set Programming
2. Natural Language Parsing
1. Introduction to Answer Set Programming

2. Natural Language Parsing
Answer Set Programming (ASP)

- ASP is a declarative programming paradigm intended to solve difficult (NP-hard) combinatorial search problems (Marek and Truszczyński, 2000; Niemelä, 2000)
  - Declarative programming: Constraint Programming, Integer Linear Programming, SQL
- Combinatorial search problems consist of finding the combinations of a discrete set of items that satisfy specified requirements (constraints).
- Such problems are often NP-hard and occur in various areas in engineering and science applications.
- ASP origins go back to semantic foundations of Prolog: answer sets (Gelfond and Lifschitz, 1988)
ASP Systems

- **Grounding** — instantiates variables
  - GRINGO (Potsdam), DLV (Calabria), LPARSE (Helsinki)
- **Answer set solving** — finds answer sets:
  - SMODELS (Helsinki),
  - DLV (Vienna, Calabria),
  - CMODELS (UT),
  - CLASP (Potsdam) . . .

ASP Solving is a relative of Propositional Satisfiability (SAT)
Grounding — task of instantiating variables

\[ \Pi \]

| \{a(1), a(2), a(3)\} | \{a(1), a(2), a(3)\} | \{a(1), a(2), a(3)\} |
| \{b(1)\} | \{b(1)\} | \{b(1)\} |
| c(X) \leftarrow a(X), b(X) | c(1) \leftarrow a(1), b(1) | c(1) \leftarrow a(1), b(1) |
| c(2) \leftarrow a(2), b(2) | c(3) \leftarrow a(3), b(3) |  |
ASP Solving: backtrack search through exponential size search space

Propositional Satisfiability (SAT) — task of finding satisfying truth assignments for propositional formulas

Classic backtrack search SAT algorithm: Davis-Putnam-Logemann-Loveland (DPLL)

SAT solvers: MINISAT, SATZILLA, PLINGELING . . .

- performance boost $\Rightarrow$ success story in automated reasoning

ASP solvers *dwell* on this success story
# ASP vs SAT

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>set of rules <strong>with variables</strong> Grounders Recursive definitions NonMonotonic Reasoning</td>
<td>set of <strong>ground</strong> clauses</td>
</tr>
<tr>
<td><strong>Solutions</strong></td>
<td>answer sets</td>
<td>truth assignments (models)</td>
</tr>
<tr>
<td><strong>Solving</strong></td>
<td>DPLL-like</td>
<td>DPLL+ backjumping, learning, ...</td>
</tr>
</tbody>
</table>
Using ASP to Find a Clique of $\geq 4$

ASP programming methodology: *Generate - Define - Test*

A *clique* of 4 in a graph is a set of 4 pairwise adjacent vertexes.

```
node(1). node(2). node(3). node(4). node(5).
edge(1,2). edge(1,3). edge(1,4). edge(1,5).
edge(2,3). edge(2,4). edge(2,5).
edge(3,4). edge(3,5).
```
Using ASP to Find a Clique of \( \geq 4 \)

% GENERATE
% at least 4 inCliq nodes form a potential solution
  \( 4\{\text{inCliq}(X) : \text{node}(X)\} \).

% DEFINE
adjacent(X,Y) :- edge(X,Y).
adjacent(X,Y) :- edge(Y,X).

% TEST
% Impossible: different X and Y are inCliq, but not adjacent
:- X!=Y, inCliq(X), inCliq(Y), not adjacent(X,Y).

Answer set 1: inCliq(1), inCliq(2), inCliq(3), inCliq(4)
Answer set 2: inCliq(1), inCliq(2), inCliq(3), inCliq(5)
1. Introduction to Answer Set Programming
2. Natural Language Parsing
Natural Language Parsing:

- Obtaining sentence structure

  I eat spaghetti with chopsticks.

  ⇒ two distinct structures (ambiguity): multiple parses

  I [eat spaghetti] [with chopsticks].
  I eat [spaghetti [with chopsticks]].

- Required for transforming natural language into formal (KR) language(s) for semantic analysis

- **Wide-coverage parsing**
  ⇒ parsing unrestricted natural language (e.g., newspaper)
Syntactic Structures and Information they Carry

✓ I [eat spaghetti] [with chopsticks].

✓: with chopsticks modifies eat
   chopsticks is a tool of eat

✗ I [eat spaghetti] [with meatballs].

✗: with meatballs modifies eat
   meatballs is a tool of eat

✓ I [eat [spaghetti with meatballs]].

✓: with meatballs modifies spaghetti
   meatballs is food as well as spaghetti is food

✗ I [eat [spaghetti with chopsticks]].

✗: with chopsticks modifies spaghetti
   chopsticks is food as well as spaghetti is food
Modern Parsers

Given

I eat spaghetti with chopsticks.

I eat spaghetti with meatballs.

Advanced parsers (nine of them, incl. Stanford and Berkeley) favor the same structure for sentences:

- Either
  - ✓ I [eat spaghetti] [with chopsticks].
  - ⊘ I [eat spaghetti] [with meatballs].

- or
  - ⊘ I [eat [spaghetti with chopsticks]].
  - ✓ I [eat [spaghetti with meatballs]].

Pipeline architecture prevailing in natural language processing: postpones semantic analysis until the time when parsing is completed.
Goal

Given

I eat spaghetti with chopsticks.

I eat spaghetti with meatballs.

Goal is to produce correct structures for sentences:

✓ I [eat spaghetti] [with chopsticks].
✓ I [eat [spaghetti with meatballs]].

We achieve this by incorporating semantic information in the phase of syntactic parsing: pipeline architecture.
ASPCCCG Toolkit

ASPCCCGTk (Lierler and Schüller, 2012; 2013)

- ASP parsing component:
  - Combinatory Categorial Grammar
  - data-structures of Cocke-Younger-Kasami (CYK) parsing algorithm
  - CYK adapted to the task of generating parse trees
- Uses part-of-speech tagger of C&C parser to achieve *wide-coverage*
- Finds *multiple* parse trees
- Uses IDPDraw (Leuven, Belgium) for visualization
- PYTHON is glue language

http://www.kr.tuwien.ac.at/staff/ps/aspccggtk/
ASPCCGtk on “chopsticks”

**ASPCCGtk:**

- I [eat spaghetti] [with chopsticks].

- I [eat [spaghetti with chopsticks]].
Combinatory Categorial Grammar (1)

- **Categories** for words:
  - **Atomic** categories, e.g.: noun $N$, noun phrase $NP$, sentence $S$
  - **Complex** categories: specify argument and result, e.g.:
    
    $$\text{determiner } NP / N \Rightarrow \text{expect } N \text{ to the right, result is } NP$$
    $$\text{the } \overline{NP / N}$$
    $$\text{intransitive verb } S \setminus NP \Rightarrow \text{expect } NP \text{ to the left, result is } S$$
    $$\text{walks } \overline{S \setminus NP}$$

- **Combinators** are grammar rules that combine categories, e.g.:
  
  forward application
  $$A / B \quad B \quad \Rightarrow \quad A$$
  $$\text{fa}$$
  
  backward application
  $$B \quad A \setminus B \quad \Rightarrow \quad A$$
  $$\text{ba}$$
Words in sentences are identified with corresponding categories:

\[
\begin{array}{cccc}
I & NP & (NP/NP)/PP & NP \\
NP & (NP/NP)/PP & NP & (NP/NP)/PP \\
I & NP & (NP/NP)/PP & NP \\
I & NP & (NP/NP)/PP & S \\
\end{array}
\]

Combinators are applied to produce a parse tree:

\[
\begin{array}{cccc}
I & NP & (NP/NP)/PP & NP \\
NP & (NP/NP)/PP & NP & (NP/NP)/PP \\
I & NP & (NP/NP)/PP & S \\
\end{array}
\]

The task of CCG parsing — find CCG parse trees for a sentence

Steedman (2000): *nine* combinators are required to parse English
The task of CCG parsing — find CCG parse trees for a sentence

Both parses are as good:

✓ I [eat spaghetti] [with chopsticks].

✗ I [eat [spaghetti with chopsticks].

Information about semantics has to be incorporated

- *with chopsticks* is *no food* to modify *spaghetti*
FRAMENET (Fillmore and Baker, 2001) – lexical dataset of semantic relations

- provides information for semantic disambiguation:
  - the frame `food` corresponds to the word “spaghetti”

<table>
<thead>
<tr>
<th>Frame</th>
<th>Frame Element</th>
<th>Semantic Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>food</td>
<td>CONSTITUENT</td>
<td>food_constituent</td>
</tr>
</tbody>
</table>

- it suggests: `food` only takes other `food` as constituents ⇒ it is impossible to modify “spaghetti” with “chopsticks”.

Yuliya Lierler  
ASP for Integral Syntactic Parsing and Semantic Disambiguation
Our approach is to define semantically coherent parse trees:

- Each node in a tree is tagged with semantic information
- FRAMENET is used to populate the semantic information of leaves
- Linguistic constraints guard tags of intermediate nodes
Tags I

A tag is either ⊥ or a pair $T||F$:

For leaves of a tree (= words of a sentence):

- $T$: semantic types associated with leaf-word
  $T_{\text{spaghetti}}$ is $\{\text{food, food\_constituent, ingestible}\}$

- $F$: semantic types associated with the frame elements in a frame evoked by a leaf-word

  “eat” evokes the frame ingestion:

<table>
<thead>
<tr>
<th>Frame</th>
<th>Frame Element</th>
<th>Semantic Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingestion</td>
<td>INGESTOR</td>
<td>sentient</td>
</tr>
<tr>
<td></td>
<td>INGESTIBLE</td>
<td>ingestible</td>
</tr>
<tr>
<td></td>
<td>INSTRUMENT</td>
<td>tool</td>
</tr>
</tbody>
</table>

$F_{\text{eat}}$ is $\{\text{sentient, ingestible, tool}\}$. 
For a non-leaf node \( p \):

\[
T_p \parallel F_p = \begin{cases} 
\perp & \text{if a tag of either } f \text{ or } a \text{ is } \perp \\
\perp & \text{if } F_f \cap T_a = \emptyset \\
T_f \parallel (F_f \setminus \{s\}) & \text{if there is a semantic type } s \in F_f \cap T_a
\end{cases}
\]

where \( f \) and \( a \) are a functor and an argument children of \( p \).

\( \{\text{sentient, ingestible, tool}\} \cap \{\text{food, food\_constituent, ingestible}\} = \{\text{ingestible}\} \neq \emptyset \)

\[
\begin{array}{l}
\text{eat} \\
(VP/PP)/NP : \emptyset \parallel \{\text{sentient, ingestible, tool}\} \\
NP : \{\text{food, food\_constituent, ingestible}\} \parallel \{\text{food\_constituent}\}
\end{array}
\begin{array}{l}
\text{spaghetti} \\
VP/PP : \emptyset \parallel \{\text{sentient, tool}\} \\
\{\text{sentient, ingestible, tool}\} \setminus \{\text{ingestible}\}
\end{array}
\]
Parse tree is *semantically coherent* if there is no node in the tree annotated by the ⊥ tag.

✓ semantically coherent subtree:

\[
\begin{array}{c}
\text{eat} \\
(VP/PP)/NP: \emptyset || F_{\text{eat}} \\
NP: T_{\text{sp}} || \{\text{food.constituent}\} \\
VP/PP: \emptyset || \{\text{sentient, tool}\} \\
VP: \emptyset || \{\text{sentient}\}
\end{array}
\]

\[
\begin{array}{c}
\text{spaghetti} \\
NP: T_{\text{sp}} || \{\text{tool, instrument}\} || \emptyset \\
VP: \emptyset || \{\text{sentient}\}
\end{array}
\]

\[\text{with chopsticks} \]
Example II

\[ T_p \parallel F_p = \begin{cases} \bot & \text{if a tag of either } f \text{ or } a \text{ is } \bot \\ \bot & \text{if } F_f \cap T_a = \emptyset \\ T_f \parallel (F_f \setminus \{s\}) & \text{if there is a semantic type } s \in F_f \cap T_a \end{cases} \]

where \( f \) and \( a \) are a functor and an argument children of \( p \)

\[ \{\text{food constituent}\} \cap \{\text{tool, instrument}\} = \emptyset \]

\( \emptyset \) semantically incoherent subtree:

\[
\begin{array}{c}
\text{spaghetti} \\
NP : T_{sp} \parallel \{\text{food constituent}\} \\
\hline
\text{with chopsticks} \\
NP \setminus NP : \{\text{tool, instrument}\} \parallel \emptyset \\
\hline
NP : \bot
\end{array}
\]
Snapshot of CYK

Given:

\[
\begin{align*}
I & \quad eat & \quad spaghetti \\
NP & \quad (S \backslash NP) / NP & \quad NP
\end{align*}
\]

and combinators:

\[
\begin{align*}
A / B & \quad B & \quad fa \\
A & \quad fa
\end{align*}
\]

\[
\begin{align*}
B & \quad A \backslash B & \quad ba \\
A & \quad ba
\end{align*}
\]

CYK table:

\[
\begin{array}{c|c|c|c}
I & \text{eat} & \text{spaghetti} \\
NP & (S \backslash NP) / NP & S \backslash NP & S \\
NP & & & fa
\end{array}
\]
Populating diagonal by given categories for words:

grid(P,P,C) :- category_at(C,P).

Populating nondiagonal cells:

applicable(fa,Pj,Pi,PL,PD,X,rfunc(X,Y),Y) :-
  grid(PL,Pi,rfunc(X,Y)), grid(Pj,PD,Y).

grid(Pj,Pi,X) :- applicable(_,Pj,Pi,_,_,_X,_,_).

Constructing a parse tree:

{ applied(Comb,Pj,Pi,PL,PD,Result,SrcL,SrcD) } :-
  applicable(Comb,Pj,Pi,PL,PD,Result,SrcL,SrcD).
Populating diagonal by semantic types for words:

\[
\text{grid-semtype}(P,P,Sem) :- \text{word-at}(W,P), \\
\text{instance-of}(W,Sem).
\]

Populating nondiagonal cells in accordance with the definition of coherent semantic tree.

Defining a state that constitutes incoherent semantic tree.

Forbidding such bad states.
AspCcgTk + Semantics = Goal

AspCcgTk + semantics:

✓ I [eat spaghetti] [with chopsticks].
✓ I [eat [spaghetti with meatballs]].
Future Work

- Automate a process of fetching relevant information from FRAMENET
- Evaluate approach on CCGbank (collection of annotated sentences from Wall Street Journal)
- Exploring the possibility of relying on other sources of lexical/semantic information than FRAMENET: VERBNET, PROPBANK...