Lexicon formalism: categorial grammar

Kuan Yu

Universität Tübingen

kuanyu@sfs.uni-tuebingen.de

May 3, 2016
Type-driven interpretation

Montague grammar

\[
\begin{array}{c}
\text{"likes"} \\
\lambda x \lambda y [\text{likes}(y, x)] \\
\text{"books"} \\
\end{array}
\begin{array}{c}
\text{"likes books"} \\
\lambda y [\text{likes}(y, \text{books})] \\
\end{array}
\]

\text{"Peter likes books"}

\text{"likes (peter, books)"}

Categorial grammar[1]

\[
\begin{array}{c}
\text{"likes"} \\
(s \backslash np) / np \\
\end{array}
\begin{array}{c}
\text{"books"} \\
np \\
\end{array}
\]

\text{"likes books"}

\[
\begin{array}{c}
\text{\(s \backslash np\)} / np \\
\text{np} \\
\end{array}
\begin{array}{c}
\text{\(s\)} \\
\text{np} \\
\end{array}
\]

\text{'Peter likes books"}

\text{\(s\)}
Functors and applications

Functors

Functions which map between categories.

Forward application

\[ \frac{X}{Y} \ast Y = X \]

\[
\begin{array}{c}
s/np \\
\hline
np \\
\end{array}
\]

\[
\frac{s}{s}
\]

Backward application

\[ Y \ast \frac{X}{Y} = X \]

\[
\begin{array}{c}
np \\
\hline
s/np \\
\end{array}
\]

\[
\frac{s}{s}
\]
Type raising and compositions

Type raising

\[ X \Rightarrow T / (T \backslash X) \]

\[
\begin{align*}
np \\
\frac{s / (s \backslash np)}{s / (s \backslash np)}
\end{align*}
\]

Composition

\[ X/Y * Y/Z = X/Z \]

\[
\frac{s / (s \backslash np) \ (s \backslash np) / np}{s / np}
\]

Backwards

\[ X \Rightarrow T \backslash (T / X) \]

\[
Y \backslash Z * X \backslash Y = X \backslash Z
\]
More rules[2]

- forward crossing composition
- backward crossing composition
- forward substitution
- backward substitution
- forward crossing substitution
- backward crossing substitution
Reductionism

Non-reducible systems

- Emergent properties.
- Example: the formation of words from the alphabet.
- Plato's Cratylus: naturalism vs conventionalism.

Forced composition

\[
F(i) : \{ \text{lex}(i), \text{lex}(\text{iambus}), \text{lex}(\text{iambic}), \ldots, \text{lex}(\text{Icarian}), \text{lex}(\text{Icarus}), \text{lex}(\text{ice}), \ldots, \text{lex}(\text{izzat}) \}
\]

\[
F(i)(c) : \{ \text{lex}(\text{Icarian}), \text{lex}(\text{Icarus}), \text{lex}(\text{ice}), \text{lex}(\text{iceburg}), \ldots, \text{lex}(\text{ictus}), \text{lex}(\text{icy}) \}
\]

\[
F(i)(c)(e) : \{ \text{lex}(\text{ice}), \text{lex}(\text{iceburg}), \ldots, \text{lex}(\text{icewine}) \}
\]

\[
F(i)(c)(e)(\$) : \text{lex}(\text{ice})
\]
Semantic holism

A certain part of language can only be understood through its relations to a larger segment of language. A transitive verb is understood as: \((s \backslash np) / np\).

Holism & compositionality

Contradictory only on the surface:
Bottom-up & top-down perspective of the same thing.

A holistic compositional grammar formalism

“Possible? Yes. But is it desirable?”
— Leto II Atreides, Frank Herbert’s Children of Dune
Why the trouble?

Adaptability for non-Indo-European languages.
- Languages with free word order.
- Languages with crazy morphology.
- Languages with no morphology.
- ...

Integrated syntactic & semantic parsing.
- Traditional pipeline approach does not really work.
- Semantic feedback for syntactic analysis.
History

Esther König

- 1995[3]: lexicalized HPSG
- 1999[4]: categorial grammar

Implementations

- CUF\(^1\): comprehensive unification formalism
- DII[5]: DYANA integrated implementation (English)

Sadly, out of maintenance.

\(^1\)ftp://ftp.ims.uni-stuttgart.de/pub/Users/jochen/Esslili95/cuf/cuf_flyer_short.html
Comparison with HPSG

Schemata vs lexicon

- No phrase structure schemata
- Syntactic info in the lexicon

Advantage: efficient parsing

- Restricted search space
  Each step in producing analyses is licensed by some input word.
- Compositional
  No global rules means freedom in top-down & bottom-up strategies.
- Lean formalism
  Phrase structure schema & lexical schema $\Rightarrow$ lexical schema.
Subcategorization

Category sort
category(Nonterminal, Arguments)

Argument sort
goal(Direction, Nonterminal, Arguments, Slash)

Examples:

```
lex(books) := category(np, []).
lex(walks) := category(s, [goal(left, np, [], [])]).
lex(likes) := category(s, [goal(right, np, [], []),
                          goal(left, np, [], [])]).
```
CUF vs CG

nonterminal = np | s.

%% np
category(np, []).

%% s np
category(s, [goal(left, np, [], [])]).

%% (s np)/np
category(s, [goal(right, np, [], []),
            goal(left, np, [], [])]).
Recursive arguments

Control verbs

%% ((s_fin\np)/(s_inf\np))/np

\text{lex}(\text{promised}) :=
\begin{align*}
\text{category}&(s_{\text{fin}}, [\text{goal(right,np,[]][[])}], \\
&\text{goal(right,s_{\text{inf}}, [\text{goal(left,np,[]][[])}], \\
&[[]], \\
&\text{goal(left,np,[]][[])}].
\end{align*}

%% s_{\text{inf}}/np

\text{lex}(\text{to_go}) :=
\begin{align*}
\text{category}&(s_{\text{inf}}, [\text{goal(left,np,[]][[])}].
\end{align*}
Adjunctions

```prolog
%% x/x
category(N,[goal(right,N,Args,[]) | Args]).
```

Illegal in CUF

Categories must be propositional:
Variables must neither range over categories nor over argument lists.

Solution
Enumerate all possible nodes where adjunction can take place.
Unbounded dependencies

books_i \quad (Peter \quad likes \quad \epsilon_i)

Trace postulation

%% s_fin/(s_fin|np)
lex(books) := category(s_fin,
    [goal(right,s_fin,[]),
     [category(np,[])])]).

Trace percolation

The inherited slash **MotherSlash** of the mother node is the disjoint multi-set union of the functor’s slash **FunSlash** and the argument’s slash **ArgSlash**.

Trace realization

Category for an empty string unified with the category of its inherited slash, which is not a lexical entry.
Lexically licensed trace

```
“books”

s / (s | np) {} \\
np {} \\
\( (s \backslash np) / np \{\} \) \( np \{np\} \)
```

```
“Peter”

\( np \{\} \) \\
\( s \backslash np \{np\} \)
```

```
“likes”

\( (s \backslash np) / np \{\} \) \( np \{np\} \)
```

```
“Peter likes”

\( s \{np\} \)
```

```
“books Peter likes”

\( s \{\} \)
```

Alternatively, combinatory categorial grammar:

```
“books”

s / (s | np) \\
\( s \backslash (s \backslash np) \) \( \text{Type} \uparrow \) \\
\( s \backslash np \) \\
\( (s \backslash np) / np \) \\
\( Comp \rightarrow \)
```

```
“Peter”

\( np \) \\
\( s \backslash (s \backslash np) \)
```

```
“likes”

\( (s \backslash np) / np \)
```

```
Books Peter likes

s \{\} \\
s \{}
```

```
Alternatively, combinatory categorial grammar:

“books”

s / (s | np) \\
\( s \backslash (s \backslash np) \) \( \text{Type} \uparrow \) \\
\( s / np \) \\
\( s \)
```

```
“Peter”

\( np \) \\
\( s \backslash (s \backslash np) \)
```

```
“likes”

\( (s \backslash np) / np \)
```

```
Books Peter likes

s \{\} \\
s \{}
```
Montague again

Quantifier raising and quantifying-in

\[
\begin{align*}
\lambda P \left[ \exists x \left[ \text{unicorn}(x) \land P(x) \right] \right] & \\
\exists x \left[ \text{unicorn}(x) \land (\lambda x_0 [\text{seek}(\text{john}, x_0)]) (x) \right] & \\
\exists x \left[ \text{unicorn}(x) \land \text{seek}(\text{john}, x) \right]
\end{align*}
\]
Lexical generalizations

Originally

\[
\text{lex(likes)} := \text{category}(s, [\text{goal(right,np,[]},[[]]), \text{goal(left,np,[]},[[]])]).
\]

Generalized

\[
\text{lex(likes)} := \text{verb(v2)}.
\]
\[
\text{lex(hates)} := \text{verb(v2)}.
\]
\[
\text{verb(v2)} := \text{category}(s,[\text{goal(right,np,[]},[[]]), \text{goal(left,np,[]},[[]])]).
\]

Hierarchical description

CUF sorts
Base generation vs movement

Base-generated

\[
\text{base\_generated} := \text{category}(_\text{Root}, \text{slashes\_empty}).
\]
\[
\text{slashes\_empty} := \[].
\]
\[
\text{slashes\_empty} := [\text{goal}(\_,\_,\_,\[]) \mid \text{slashes\_empty}].
\]

Filler category

\[
\text{movement\_schema}(\text{Trace}) := \text{category}(_\text{Root}, [\text{goal}(\_,\_,\_,\[\text{Trace}\]) \mid \_]).
\]
Saturated category & head category

Maximal projection

\[
\text{maxproj}(\text{Nonterminal}) := \text{category}(\text{Nonterminal}, []).\
\]

Head category

\[
\text{head\_category}(\text{Nonterminal}, \text{Args}) := \\
\text{category}(\text{Nonterminal}, \\
(\text{Args} \& \text{nelist})) \& \\
\text{base\_generated}.\
\]
Non-head categories

Adjunct

\[
\text{adjunct}(\text{category}(\text{Nonterminal}, \text{Args}), \text{Direction}) := \\
\text{category}(\text{Nonterminal}, \\
[(\text{goal}[\text{Direction}, \text{Nonterminal}, \text{Args}, _]) \\
| \text{Args}]).
\]

Specifier

\[
\text{specifier}(\text{Nonterminal}, \\
\text{category}(\text{ArgNT}, \text{Args}), \text{Direction}) := \\
\text{category}((\text{Nonterminal} \& (~ \text{ArgNT})), \\
[\text{goal}(\text{Direction}, \text{NT}, \text{Args}) \\
| \text{Args}]).
\]
Language universal

```
basic_adjective := adjunct(maxproj(n),right).

basic_adverb := adjunct(category(s,subcat(_,_),left).

subcat(0) := [].
subcat(v1) := [maxproj(np)].
subcat(v2) := [maxproj(np),maxproj(np)].
%%% and so on ...

basic_noun_phrase := maxproj(np).

basic_nominal := maxproj(n).

basic_determiner := specifier(np,category(n,[]),right).
```
Constituents in non-basic word order
The notion of a topicalized constituent can be defined as the intersection of an s-specifier and a moved phrase.

topicalized(Category) := specifier(s, category(s, []), right) & movement_schema(Category).

Syntactic alternations

noun_phrase := basic_noun_phrase.
noun_phrase := topicalized(basic_noun_phrase).
Words classes

Lexical variation

\[
\text{noun}(\text{NType, Number}) := \text{basic_nominal}. \\
\text{noun}(\text{NType, Number}) := \\
(\text{true}(\text{NType & mass_noun}); \\
(\text{true}(\text{NType & count_noun}) \& \\
\text{true}(\text{Number & pl}))) \& \\
\text{noun_phrase}. \\
\text{true}(_) := _.
\]
Words classes

Description of a syntactic category as a combination of two simpler categories.

\[
\text{join_trees} (\text{category}(NT1,\text{Args}1), \\
\text{category}(_\text{NT}2,\text{Args}2)) := \\
\text{category}(NT1,\text{append}(\text{Args}2,\text{Args}1)).
\]

\[
\text{determiner} := \text{join_trees}(\text{noun_phrase}, \\
\text{basic_determiner}).
\]
Surface lexicon

Theory independent mappings from words onto word classes and linguistic features.

\[
\text{lex(books)} := \text{noun(count, pl)}.
\]

\[
\text{lex(peter)} := \text{noun_phrase}.
\]

\[
\text{lex(the)} := \text{determiner}.
\]
Stuff Kuan would like

Variable category
- adjunction
- conjunction & disjunction

Treatments for passive constructions

Lexicon building
- dynamic training
- probabilistic parsing


