Towards a proof-theoretic natural language semantics for wide-coverage grammars

Koji Mineshima
Ochanomizu University

New Landscapes in Theoretical Computational Linguistics
Ohio State University
October 15, 2016
Collaborators

- Daisuke Bekki
- Pascual Martínez-Gómez
- Ribeka Tanaka
- Yusuke Miyao

With funding from the JST CREST program, “Establishment of Knowledge-Intensive Structural Natural Language Processing and Construction of Knowledge Infrastructure”
Two faces of this research

1. Applications to mainstream computational linguistics and NLP
2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): "Proof-theoretic turn"
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP

2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP
2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP
2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP

2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP
2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Two faces of this research

1. Applications to mainstream computational linguistics and NLP

2. New methodology for theoretical linguistics
   - Implementation of syntax-semantics interface
   - with statistical, corpus-based wide-coverage parsers
   - from truth-condition (Tarski) to proof-condition (Gentzen): “Proof-theoretic turn”
   - inferences-as-tests: empirical evaluation of semantic theories in terms of textual entailment (Bekki and Mineshima, 2016)
   - Importance of (automated) theorem proving: implementation of syntax-semantics-prover interface
   - Establishing a “benchmark” for formal semantics
Why “computational” theoretical linguistics?

- Computational modelling is needed to specify detailed syntactic and semantic architectures. 

... although it can be very frustrating I know from experience that in order to write a complete fragment, you normally have to include decisions about things for which you have no principled basis to choose, or for which you know that none of the available alternatives are really good. (Partee 2005)
Why “computational” theoretical linguistics?

- Computational modelling is needed to specify detailed syntactic and semantic architectures.
- Making it possible to compute the predictions of each syntactic/semantic framework quickly and precisely.
Why “computational” theoretical linguistics?

- Computational modelling is needed to specify detailed syntactic and semantic architectures.
- Making it possible to compute the predictions of each syntactic/semantic framework quickly and precisely.
- A necessary step towards establishing a meaningful and systematic way to compare/evaluate each framework.
Why “computational” theoretical linguistics?

- Computational modelling is needed to specify detailed syntactic and semantic architectures.
- making it possible to compute the predictions of each syntactic/semantic framework quickly and precisely.
- A necessary step towards establishing a meaningful and systematic way to compare/evaluate each framework.
- ... although it can be very frustrating

> I know from experience that in order to write a complete fragment, you normally have to include decisions about things for which you have no principled basis to choose, or for which you know that none of the available alternatives are really good. (Partee 2005)
Implemented syntax-semantics-prover interface

Wide-coverage “formal semantics” parsers

- Bos et al. (2004): Boxer/Nutcraker
  CCG + DRT + FOL prover/model builder (English)
- Moot (2010):
  TLG + DRT (French)
- Butler and Yoshimoto (2012):
  SCT + Treebank Semantics (English and Japanese)
- Abzianidze (2015):
  CCG + Natural Logic Tableau prover (English)
- Mineshima et al. (2015)
  CCG + HOL prover (English and Japanese)
To test the capacity of formal semantics systems, we would also need a shared dataset ("benchmark").
Benchmarking

- To test the capacity of formal semantics systems, we would also need a shared dataset ("benchmark").
- Cf. the TPTP library in Automated Theorem Proving:
  - a comprehensive and up-to-date list of problems
  - a guideline for adding new problems and for correcting errors in existing problems
  - competition for evaluating system performance
Benchmarking

- To test the capacity of formal semantics systems, we would also need a shared dataset (“benchmark”).
- Cf. the TPTP library in Automated Theorem Proving:
  - a comprehensive and up-to-date list of problems
  - a guideline for adding new problems and for correcting errors in existing problems
  - competition for evaluating system performance
- Unshared Task at LENLS 13: Theory and System analysis with FraCaS, MultiFraCaS and JSeM Test Suites (November 13, NINJAL, Japan)
http://www.compling.jp/fracas_task/index.html
The framework

Our favorite theories:

- **Syntax**: CCG (wide-coverage parsers are available for English and Japanese)
- **Semantics**: DTS (dynamic/underspecification semantics smoothly combined with modern categorial grammar and prover)
- **Prover**: HOL (but it’s almost first-order; it’s better than first-order reification)
The framework

Our favorite theories:

- Syntax: CCG (wide-coverage parsers are available for English and Japanese)
- Semantics: DTS (dynamic/underspecification semantics smoothly combined with modern categorial grammar and prover)
- Prover: HOL (but it’s almost first-order; it’s better than first-order reification)

Three steps:

1. Build a flexible platform to implement the syntax-semantics-prover interface: \texttt{ccg2lambda (CCG + HOL)}
2. Adding a robust external knowledge base
3. Extend it with underspecification semantics (@-terms) to handle dynamics (anaphora/presupposition): \texttt{ccg2lambda + DTS}
Plan

1. Introduction
2. ccg2lambda
3. Experiments on RTE datasets
4. Dependent Type Semantics
ccg2lambda

Compositional semantics and higher-order inference system for wide-coverage CCG parsers
Natural Language Inferences (Textual Entailment)

• Does Premise $P$ entail Hypothesis $H$?

$P$ Smoking in restaurants is prohibited by law in most cities in Japan.
$H$ Smoking in public spaces is not allowed in some cities.

Yes (Entailment)
Natural Language Inferences (Textual Entailment)

- Does Premise P entail Hypothesis H?
  - P: Smoking in restaurants is prohibited by law in most cities in Japan.
  - H: Smoking in public spaces is not allowed in some cities.

  *Yes* (Entailment)

- *The best way of testing an NLP system’s semantic capacity* (Cooper et al. 1996)
- Many applications (Question Answering, Machine Translation, etc.)
Natural Language Inferences (Textual Entailment)

- Does Premise $P$ entail Hypothesis $H$?
  
  $P$  Smoking in restaurants is prohibited by law in most cities in Japan.
  
  $H$  Smoking in public spaces is not allowed in some cities.
  
  Yes (Entailment)

- The best way of testing an NLP system’s semantic capacity (Cooper et al. 1996)

- Many applications (Question Answering, Machine Translation, etc.)

- Relevant factors:

  1. Syntax
  2. Logical words: most, not, some, every
  3. Content words: 

     restaurant $\rightarrow$ public space
     prohibited $\rightarrow$ $\neg$ allowed


Formal Semantics
Logic-based approaches to entailment

- **Natural Logic**
  - formalizes inferences with surface form
  - only allows single premise inferences (mononicity inference)
  - more efficient
  - less expressive
  - MacCartney (2009)

- **First-order logic (FOL)**
  - efficient provers exist
  - dominate computational linguistics
  - limited expressive power

- **Higher-order logic (HOL)**
  - high expressive power
  - dominate formal semantics
  - no general-purpose efficient prover exists
  - less efficient
  - more expressive

Our approach

Boxer (Bos 2008)
Logic-based approaches to entailment

**Natural Logic**
- formalizes inferences with surface form
  - only allows single premise inferences (mononicity inference)
- more efficient
- less expressive

**First-order logic (FOL)**
- efficient provers exist
- dominate computational linguistics
- limited expressive power

**Higher-order logic (HOL)**
- high expressive power
- dominate formal semantics
  - no general-purpose efficient prover exists
- less efficient
- more expressive

MacCartney (2009)  
Boxer (Bos 2008)  
Our approach
Logic-based approaches to entailment

### Natural Logic
- formalizes inferences with surface form
- only allows single premise inferences (mononicity inference)

### First-order logic (FOL)
- efficient provers exist
- dominate computational linguistics
- limited expressive power

MacCartney (2009)

### Higher-order logic (HOL)
- high expressive power
- dominate formal semantics
- no general-purpose efficient prover exists

Boxer (Bos 2008)

Our approach
Logic-based approaches to entailment

Natural Logic
- formalizes inferences with surface form
- ▲ only allows single premise inferences (mononicity inference)

First-order logic (FOL)
- efficient provers exist
- ▲ dominate computational linguistics
- ▲ limited expressive power

Higher-order logic (HOL)
- ▲ high expressive power
- ▲ dominate formal semantics
- ▲ no general-purpose efficient prover exists

Goal: To develop a higher-order inference system specialized for natural language inferences, and combine it with a wide-coverage parser

MacCartney (2009)  Boxer (Bos 2008)  Our approach
Higher-order inference system: ccg2lambda

https://github.com/mynlp/ccg2lambda

Wide-coverage CCG parsers

- **C&C parser** for English
  (Clark and Curran, 2007)

- **Jigg** for Japanese
  (Noji and Miyao, 2016)
CCG-based Compositional Semantics

Basic category: \( S, NP, N \)
Function category: \( X/Y, X\setminus Y \)
Combinatory rules in CCG:

\[
\frac{X/Y : f \quad Y : a}{X : fa} > \frac{Y : a \quad X\setminus Y : f}{X : fa}
\]

\[
\text{John} \quad \frac{\text{likes}}{(S\setminus NP)/NP :} \quad \frac{\text{Mary}}{NP :} \quad > \quad \frac{NP :}{S\setminus NP :} \quad \frac{S :}{S :} <
\]
CCG-based Compositional Semantics

Basic category: $S, NP, N$
Function category: $X/Y, X\backslash Y$
Combinatory rules in CCG:

$$\frac{X/Y : f \quad Y : a}{X : fa} \quad > \quad \frac{Y : a \quad X\backslash Y : f}{X : fa}$$

John \hspace{1cm} \text{likes} \hspace{1cm} \text{Mary}
\hline
(S\backslash NP)/NP : \lambda y.\lambda x.\text{like}(x,y) \quad \text{NP : mary}
\hline
NP : john \quad \text{S}\backslash NP : \quad \text{S} : \\
\hline
CCG-based Compositional Semantics

Basic category: $S, NP, N$

Function category: $X/Y, X\backslash Y$

Combinatory rules in CCG:

$$
\frac{X/Y : f \quad Y : a}{X : fa} \quad \frac{Y : a \quad X\backslash Y : f}{X : fa}
$$

John

NP : john

$(S\backslash NP)/NP : \lambda y.\lambda x.\text{like}(x, y)$

Mary

NP : mary

$S\backslash NP : \lambda x.\text{like}(x, \text{mary})$

$S :$

like$(john; mary)$
CCG-based Compositional Semantics

Basic category: $S, NP, N$

Function category: $X/Y, X\backslash Y$

Combinatory rules in CCG:

$$
\frac{X/Y : f \quad Y : a}{X : fa} \quad > \quad \frac{Y : a \quad X\backslash Y : f}{X : fa}
$$

$$
\frac{John}{NP : john} \quad \frac{likes}{(S\backslash NP)\backslash NP : \lambda y.\lambda x.\text{like}(x, y)} \quad \frac{Mary}{NP : mary}
\quad \frac{S\backslash NP : \lambda x.\text{like}(x, mary))}{S : \text{like}(john, mary)} <
$$
CCG-based Compositional Semantics

Basic category: \( S, NP, N \)
Function category: \( X/Y, X\backslash Y \)
Combinatory rules in CCG:

\[
\begin{align*}
X/Y : f & \quad Y : a \\
\overrightarrow{X : fa} & >
\end{align*}
\]

\[
\begin{align*}
Y : a & \quad X\backslash Y : f \\
\overleftarrow{X : fa} & <
\end{align*}
\]

\[
\begin{align*}
\text{John} & \quad \frac{\text{likes}}{(S\backslash NP)/NP : \lambda y.\lambda x.\text{like}(x, y)} \\
\overrightarrow{NP : \text{john}} & \quad \frac{\text{Mary}}{NP : \text{mary}} \\
\overleftarrow{S\backslash NP : \lambda x.\text{like}(x, \text{mary})} & <
\end{align*}
\]

\[
\begin{align*}
S : \text{like}(\text{john, mary}) & <
\end{align*}
\]
Quantifier: Standard analysis (Steedman, 2000)

\[ \text{every} \quad NP/N \quad : \quad \lambda F \lambda G. \forall x (F(x) \rightarrow G(x)) \]
\[ \text{student} \quad N \quad : \quad \lambda x. \text{student} (x) \]
\[ \text{runs} \quad S/\text{NP} \quad : \quad \lambda x. \text{run} (x) \]

\[ NP \quad : \quad \lambda G. \forall x (\text{student} (x) \rightarrow G(x)) \]
\[ S \quad : \quad \forall x (\text{student} (x) \rightarrow \text{run} (x)) \]
Quantifier: Standard analysis (Steedman, 2000)

\[
\begin{align*}
\text{every} & \quad \text{student} \\
NP/N & \quad N \\
\lambda F \lambda G. \forall x (F(x) \rightarrow G(x)) & \quad \lambda x. \text{student}(x) \\
S/(S\setminus NP) & \quad \lambda G. \forall x (\text{student}(x) \rightarrow G(x)) \\
S \quad \lambda x. \text{run}(x) < \quad \forall x (\text{student}(x) \rightarrow \text{run}(x))
\end{align*}
\]
Quantifier: Standard analysis (Steedman, 2000)

\[
\text{every } (S/(S\backslash NP))/N : \lambda F \lambda G. \forall x (F(x) \rightarrow G(x))
\]

\[
S/(S\backslash NP) : \lambda G. \forall x (\text{student}(x) \rightarrow G(x))
\]

\[
S : \forall x (\text{student}(x) \rightarrow \text{run}(x))
\]

\[
\text{student } N : \lambda x. \text{student}(x)
\]

\[
\text{runs } S\backslash NP : \lambda x. \text{run}(x)
\]
Quantifier: Standard analysis (Steedman, 2000)

\[
\begin{align*}
\text{every} & \quad (S/(S\setminus NP))/N \\
& \quad : \lambda F \lambda G. \forall x (F(x) \rightarrow G(x)) \\
\text{student} & \quad N \\
& \quad : \lambda x. \text{student}(x) \\
\end{align*}
\]

\[
\begin{align*}
S/(S\setminus NP) & \quad : \lambda G. \forall x (\text{student}(x) \rightarrow G(x)) \\
S & \quad : \forall x (\text{student}(x) \rightarrow \text{run}(x)) \\
\end{align*}
\]

Output of the C&C parser

\[
\begin{align*}
\text{every} & \quad NP/N \\
& \quad : \lambda F. \lambda G. \forall x (F(x) \rightarrow G(x)) \\
\text{student} & \quad N \\
& \quad : \lambda x. \text{student}(x) \\
\end{align*}
\]

\[
\begin{align*}
NP & \quad : \lambda G. \forall x (\text{student}(x) \rightarrow G(x)) \\
S & \quad : \forall x (\text{student}(x) \rightarrow \text{run}(x)) \\
\end{align*}
\]
Quantifier: Standard analysis (Steedman, 2000)

\[
\text{every} \\
\frac{(S/(S\setminus NP))/N}{\lambda F \lambda G. \forall x (F(x) \rightarrow G(x))}
\]

\[
\frac{S/(S\setminus NP)}{\lambda G. \forall x (\text{student}(x) \rightarrow G(x))}
\]

Output of the C&C parser

\[
\text{every} \\
\frac{NP/N}{\lambda F. \lambda G. \forall x (F(x) \rightarrow G(x))}
\]

\[
\frac{NP}{\lambda G. \forall x (\text{student}(x) \rightarrow G(x))}
\]

\[
\frac{S}{\lambda Q. Q(\lambda x. \text{run}(x))}
\]

\[
\frac{S\setminus NP}{\lambda x. \text{run}(x)}
\]

\[
\frac{S}{\forall x (\text{student}(x) \rightarrow \text{run}(x))}
\]
Continuation-based semantics of post NP-modifiers

- The standard analysis of NP-modifiers

\[
\text{Every } + \quad [ \text{student } + \text{ who works } ]
\]

\[
\lambda F \lambda G. \forall x (Fx \rightarrow Gx) \quad \lambda x. (\text{student}(x) \land \text{work}(x))
\]
Continuation-based semantics of post NP-modifiers

• The standard analysis of NP-modifiers

\[
\text{Every } + \ [ \text{student } + \text{ who works }] \\
\lambda F \lambda G. \forall x (Fx \rightarrow Gx) \quad \lambda x. (\text{student}(x) \land \text{work}(x))
\]

• The C&C parser (NP-S analysis)

\[
[ \text{Every } + \text{ student } ] + \text{ who works} \\
\lambda F \lambda G. \forall x (Fx \rightarrow Gx) \quad \lambda x. (\text{student}(x) \land \text{work}(x))
\]
Continuation-based semantics of post NP-modifiers

- The standard analysis of NP-modifiers
  
  \[ \lambda F \lambda G. \forall x (Fx \rightarrow Gx) \quad \lambda x. (\text{student}(x) \land \text{work}(x)) \]

- The C&C parser (NP-S analysis)

  \[ [ \text{Every} \ + \ \text{student} ] \ + \ \text{who works} \]

  \[ NP / N \quad N \quad N \backslash N \]

- Boxer’s output for “Every student who works comes.”

  \[ \forall x (\text{student}(x) \rightarrow \text{work}(x) \land \text{come}(x)) \]
Continuation-based semantics of post NP-modifiers

- Solution: NPs have an extra argument position

\[
[\text{Every} + \text{student}] + \text{who works}
\]

\[
\lambda F \lambda G. \forall x (\text{student}(x) \land Fx \rightarrow Gx)
\]

\[
\lambda Q \lambda F \lambda G. Q(\lambda x. (\text{work}(x) \land Fx))(G)
\]

Cf. Bach and Cooper’s (1979) NP-S analysis; Champollion’s (2014) event semantics
Continuation-based semantics of post NP-modifiers

- Solution: NPs have an extra argument position
  \[ [\text{Every } + \text{ student}] + \text{ who works} \]
  \[
  \lambda F \lambda G. \forall x (\text{student}(x) \land Fx \rightarrow Gx)
  \quad \lambda Q. \lambda F \lambda G. Q(\lambda x. (\text{work}(x) \land Fx))(G)
  \]
  Cf. Bach and Cooper’s (1979) NP-S analysis; Champollion’s (2014) event semantics

- VPs reset the extra argument
  Every student who works + comes
  \[
  \lambda F \lambda G. \forall x (\text{student}(x) \land \text{work}(x) \land Fx \rightarrow Gx)
  \quad \lambda Q. Q(\lambda x. \text{True})(\lambda x. \text{come}(x))
  \]

- Output: \( \forall x (\text{student}(x) \land \text{work}(x) \land \text{True} \rightarrow \text{come}(x)) \)
Continuation-based semantics of post NP-modifiers

- Solution: NPs have an extra argument position

  \[
  \text{[Every + student] + who works}
  \]

  \[
  \begin{align*}
  NP/N & \quad N & \quad NP\backslash NP \\
  \lambda F \lambda G. \forall x (\text{student}(x) \land Fx \rightarrow Gx) & \quad \lambda Q \lambda F \lambda G. Q(\lambda x. (\text{work}(x) \land Fx))(G)
  \end{align*}
  \]

  Cf. Bach and Cooper’s (1979) NP-S analysis; Champollion’s (2014) event semantics

- VPs reset the extra argument

  Every student who works + comes

  \[
  \begin{align*}
  NP & \quad NP\backslash S \\
  \lambda F \lambda G. \forall x (\text{student}(x) \land \text{work}(x) \land Fx \rightarrow Gx) & \quad \lambda Q. Q(\lambda x. \text{True})(\lambda x. \text{come}(x))
  \end{align*}
  \]

- Output: \( \forall x (\text{student}(x) \land \text{work}(x) \land \text{True} \rightarrow \text{come}(x)) \)

- Advantages:
  1. Bare quantifiers: everyone in the room
  2. Non-restrictive relative clauses: John, who is the president, ...
  3. NP-modifiers in Japanese:
     hataraku subete-no gakusei (every student who works)
     works    every    student
HOL as representation language

1. Generalized quantifiers
   \[ \text{Most students work} \leadsto \text{most}(\lambda.\text{student}(x), \lambda x.\text{work}(x)) \]

2. Modals
   \[ \text{John might come} \leadsto \text{might}(\text{come}(j)) \]

3. Veridical and anti-veridical predicates
   \[ \text{Someone managed to come} \leadsto \exists x(\text{manage}(x, \text{come}(x))) \]
   \[ \text{Someone failed to come} \leadsto \exists x(\text{fail}(x, \text{come}(x))) \]

4. Intensional (privative) adjectives
   \[ \text{John is a former student} \leadsto \text{former}(j, \lambda x.\text{student}(x)) \]

5. Attitude verbs
   \[ \text{John knows that some student came.} \leadsto \text{know}(j, \exists x(\text{student}(x) \land \text{come}(x))) \]

- Alternative: FOL decomposition/reification (Hobbs, 1985)
  \[ \forall w_1 (R_{\text{john}} w_0 w_1 \rightarrow \exists x(\text{student}(w_1, x) \land \text{come}(w_1, x))) \]
HOL as representation language

**E**: type of entities

**Prop**: type of propositions

<table>
<thead>
<tr>
<th>Examples</th>
<th>Semantic Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>most</td>
<td>$(E \rightarrow \text{Prop}) \rightarrow (E \rightarrow \text{Prop}) \rightarrow \text{Prop}$</td>
</tr>
<tr>
<td>might</td>
<td>$\text{Prop} \rightarrow \text{Prop}$</td>
</tr>
<tr>
<td>manage</td>
<td>$\text{Prop} \rightarrow E \rightarrow \text{Prop}$</td>
</tr>
<tr>
<td>former</td>
<td>$(E \rightarrow \text{Prop}) \rightarrow (E \rightarrow \text{Prop})$</td>
</tr>
<tr>
<td>know</td>
<td>$\text{Prop} \rightarrow E \rightarrow \text{Prop}$</td>
</tr>
</tbody>
</table>
Using Coq as HOL theorem prover

Proof Assistant Coq

- Inferences in the style of natural deduction
- Rich language (Dependent Type Theory)
- Powerful automation (Tactics)
  - Built-in tactics for first-order inferences and equational reasoning
  - User-defined tactics for higher-order inferences (Ltac)
Axioms for non-first-order constructions

<table>
<thead>
<tr>
<th>Inference pattern</th>
<th>Axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existential import</td>
<td>$\forall F \forall G (\text{most}(F, G) \rightarrow \exists x (Fx \land Gx))$</td>
</tr>
<tr>
<td>Conservativity</td>
<td>$\forall F \forall G (\text{most}(F, G) \rightarrow \text{most}(F, \lambda x. (Fx \land Gx)))$</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>$\forall F \forall G \forall H (\text{most}(F, G) \rightarrow (\forall x (Gx \rightarrow Hx) \rightarrow \text{most}(F, H)))$</td>
</tr>
<tr>
<td>Veridicality</td>
<td>$\forall x \forall P (\text{manage}(x, P) \rightarrow P)$ \hspace{0.5cm} $\forall x \forall P (\text{know}(x, P) \rightarrow P)$</td>
</tr>
<tr>
<td>Anti-veridicality</td>
<td>$\forall x \forall P (\text{fail}(x, P) \rightarrow \neg P)$</td>
</tr>
</tbody>
</table>
Lexical entries

1. For **closed words**: lexical entries directly assigned to surface form: 129 entries

   **Example**
   - category: *NP/N*
   - semantics: $\lambda F \lambda G \lambda H. \forall x (Fx \land Gx \rightarrow H)$
   - surf: every

2. For **open words**: schematic lexical entry (semantic templates) assigned to syntactic categories: 100 entries

   **Example**
   - category: *N*
   - semantics: $\lambda x. E(x)$
2. Experiments on FraCaS and JSeM
Experiment 1: FraCaS

- The FraCaS test suite (Cooper et al., 1994): the textual inference problems to test theories of formal and computational semantics
- Most problems do not require lexical/world knowledge, but contain linguistically challenging problems.
- Three types of answer: yes, no, unknown
- Single-premise problems (55%) and multiple-premise problems (45%)

Example

fracas-026  
answer: yes (the premises entail the hypothesis)
P1  Most Europeans are resident in Europe.
P2  All Europeans are people.
P3  All people who are resident in Europe can travel freely within Europe.
H  Most Europeans can travel freely within Europe.

fracas-038  
answer: no (the premise contradicts the hypothesis)
P1  No delegate finished the report.
H  Some delegate finished the report on time.
Results (Mineshima et al. 2015)

- Nutcracker = C&C parser + Boxer* + FOL prover \textit{(bliksem)}
  + FOL model-builder \textit{(mice)} + WordNet

### Accuracy

<table>
<thead>
<tr>
<th>Section</th>
<th>#</th>
<th>Ours</th>
<th>Nut</th>
<th>L &amp; S 13</th>
<th>Tian 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantifiers</td>
<td>74</td>
<td>.77</td>
<td>.53</td>
<td>.62</td>
<td>.80</td>
</tr>
<tr>
<td>Plurals</td>
<td>33</td>
<td>.67</td>
<td>.52</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Adjectives</td>
<td>22</td>
<td>.68</td>
<td>.32</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Comparatives</td>
<td>31</td>
<td>.48</td>
<td>.45</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Verbs</td>
<td>8</td>
<td>.62</td>
<td>.62</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Attitudes</td>
<td>13</td>
<td>.77</td>
<td>.46</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Total</td>
<td>181</td>
<td>.69*</td>
<td>.50</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

* Total accuracy drops to 59% when ablating the higher-order rules

* 30 seconds time-out (after that, the system outputs “unknown”)

- L &S 13 = Lewis and Steedman (2013): CCG + FOL prover
- Tian 14 = Tian et al. (2014): DCS-based inference system

### Speed

<table>
<thead>
<tr>
<th>Parsing and inference</th>
<th>sec./problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCG Parsing (C &amp;C parser)</td>
<td>3.76</td>
</tr>
<tr>
<td>Our system with higher-order inference</td>
<td>3.72</td>
</tr>
<tr>
<td>Our system w/o higher-order inference</td>
<td>3.46</td>
</tr>
<tr>
<td>Nutcracker with first-order inference (first-order prover + model builder)</td>
<td>11.23</td>
</tr>
</tbody>
</table>
Why not reach 100% accuracy?

A reviewer: why not reach an accuracy close to 100% if tuned on a test set of only about 100 examples?

- syntactic parse error (no output CCG tree)
- disambiguation
  - syntactic (e.g. PP-attachment)
  - semantic (e.g. negation scope, collective vs. distributive)
- lack of lexical semantics (e.g. verb aspect classification)
- the existing analyses are not good:
  - comparative/superlative
  - numerals
- context-dependency (anaphora and ellipsis)
- gold labels are wrong/problematic
Why not reach 100% accuracy?
A reviewer: why not reach an accuracy close to 100% if tuned on a test set of only about 100 examples?

- syntactic parse error (no output CCG tree)
- disambiguation
  - syntactic (eg. PP-attachment)
  - semantic (eg. negation scope, collective vs. distributive)
- lack of lexical semantics (eg. verb aspect classification)
- the existing analyses are not good:
  - comparative/superlative
  - numerals
- context-dependency (anaphora and ellipsis)
- gold labels are wrong/problematic

Fixing parse/disambiguation errors improved the accuracy:
- combining two CCG parsers (C&C + EasyCCG):
- Accuracy of Quantifier Sec. : 77% to 95%
Japanese CCGbank (Uematsu et al., 2015)
  - Base category: only S and NP
  - A few non-lexical type-shifting rules:
    eg. the relativizer from $S \backslash NP$ to NP/NP

Japanese CCG parser Jigg (Noji and Miyao, 2016)

Size of semantic lexicon:

<table>
<thead>
<tr>
<th></th>
<th>Japanese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics templates for syntactic categories</td>
<td>37</td>
<td>129</td>
</tr>
<tr>
<td>Lexical entries for particular lexical items</td>
<td>113</td>
<td>100</td>
</tr>
</tbody>
</table>
Adding Neo-Davidsonian Event Semantics

<table>
<thead>
<tr>
<th>Syntactic category</th>
<th>Semantic representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>( \lambda NF. \exists x(N((\text{Base}, x) \land F(x))) )</td>
</tr>
<tr>
<td>S(\setminus)NP(_{ga})</td>
<td>( \lambda QK.Q(\lambda I.I, \lambda x.\exists v(K((\text{Base}, v)\land(\text{Nom}(v) = x))) )</td>
</tr>
<tr>
<td>S(\setminus)NP(_{ga})(\setminus)NP(_o)</td>
<td>( \lambda Q_2 Q_1 K.Q_1(\lambda I.I, \lambda x_1.\lambda x_2.\exists v(K((\text{Base}, v)\land(\text{Nom}(v) = x_1)\land(\text{Acc}(v) = x_2))) )</td>
</tr>
<tr>
<td>S/S</td>
<td>( \lambda SK.S(\lambda Jv.K(\lambda v'.(J(v') \land \text{Base}(v'))), v) )</td>
</tr>
<tr>
<td>NP/NP</td>
<td>( \lambda QNF.Q(\lambda Gx.N(\lambda y.(\text{Base}(y) \land G(y))) , x) , F) )</td>
</tr>
</tbody>
</table>

Semantic types

\[ T ::= E \mid Ev \mid Prop \mid T_1 \Rightarrow T_2 \]

Mapping from syntactic categories to semantic types

\[
NP^* = ((E \Rightarrow Prop) \Rightarrow E \Rightarrow Prop) \Rightarrow (E \Rightarrow Prop) \Rightarrow Prop \\
S^* = ((Ev \Rightarrow Prop) \Rightarrow Ev \Rightarrow Prop) \Rightarrow Prop \\
(C1/C2)^* = (C1\setminus C2)^* = C2^* \Rightarrow C1^*
\]
Experiment 2 : JSeM

- Dataset: Japanese Semantics Test Suite (JSeM) (Kawazoe et al., 2015)
  http://researchmap.jp/community-inf/JSeM/

- Two separate parts
  - translation of English FraCaS Problems
  - a set of problems specific to Japanese syntax/semantics

- each problem is tagged with:
  - phenomena type (quantifier, adjective, negation, etc.)
  - inference type (logical entailment, presupposition)

- answer labels: yes, no, unknown

- single-premised problems (66%) and multi-premised problems (34%)
Results: Mineshima et al. (2016)

- five phenomena: quantifier, plural, adjective, verb, and attitude
- inference type: logical entailment
- Distribution of gold answer labels: yes 297, no 50, unknown 176
- Morphological analyzer: Kuromoji

<table>
<thead>
<tr>
<th>Section</th>
<th>#Problem</th>
<th>Gold</th>
<th>System</th>
<th>SLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantifier</td>
<td>337</td>
<td>92.3</td>
<td>78.0</td>
<td>88.4</td>
</tr>
<tr>
<td>Plural</td>
<td>41</td>
<td>68.3</td>
<td>56.1</td>
<td>51.2</td>
</tr>
<tr>
<td>Adjective</td>
<td>65</td>
<td>67.7</td>
<td>63.1</td>
<td>44.6</td>
</tr>
<tr>
<td>Verb</td>
<td>36</td>
<td>77.8</td>
<td>75.0</td>
<td>55.5</td>
</tr>
<tr>
<td>Attitude</td>
<td>44</td>
<td>88.6</td>
<td>86.4</td>
<td>75.0</td>
</tr>
<tr>
<td>Total</td>
<td>523</td>
<td>86.0</td>
<td>75.0</td>
<td>76.7</td>
</tr>
</tbody>
</table>

- **Plain**: system syntactic parse (1-best)
- **Gold**: gold syntactic parse (manually selected from n-best parses)
- **SLC**: supervised learning classifier (NTCIR RITE baseline tool)

http://www.cl.ecei.tohoku.ac.jp/rite2/doku.php
3. ccg2lambda + Dependent Type Semantics
Dependent Type Semantics (DTS)

- ESSLII2016 course:
  Daisuke Bekki and Koji Mineshima
  An Introduction to Dependent Type Semantics
  http://esslli2016.unibz.it/?page_id=216

Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Martin-Löf</th>
<th>Logic</th>
<th>Agda</th>
<th>DTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Pi )-type</td>
<td>((\Pi x : A) B)</td>
<td>(\forall x : A. B)</td>
<td>((x : A) \rightarrow B)</td>
<td>((x : A) \rightarrow B)</td>
</tr>
<tr>
<td>( \Sigma )-type</td>
<td>((\Sigma x : A) B)</td>
<td>(\exists x : A. B)</td>
<td>((x : A) \times B)</td>
<td>(\begin{bmatrix} x : A \ B \end{bmatrix})</td>
</tr>
<tr>
<td>Function-type Implication</td>
<td>(A \rightarrow B)</td>
<td>(A \rightarrow B)</td>
<td>(A \rightarrow B)</td>
<td>(A \rightarrow B)</td>
</tr>
<tr>
<td>Product-type</td>
<td>(A \land B)</td>
<td>(A \land B)</td>
<td>(A \times B)</td>
<td>(\begin{bmatrix} A \ B \end{bmatrix})</td>
</tr>
<tr>
<td>Conjunction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dependent Types

1. Σ and Π types: generalization of \( \land \) and \( \to \)

\[
\exists x. \text{walk } x \leadsto \left[ \begin{array}{c} x : \text{entity} \\ \text{walk}(x) \end{array} \right]
\]

\[
\forall x. \text{walk } x \leadsto (x : \text{entity}) \to \text{walk}(x)
\]

2. propositions-as-types

\[
\left[ \begin{array}{c} x : \text{entity} \\ \text{walk}(x) \end{array} \right] : \text{type}
\]

\[
(j, p) : \left[ \begin{array}{c} x : \text{entity} \\ \text{walk}(x) \end{array} \right]
\]

\((j, p)\) is a proof-term of the proposition someone walks
Dependent Types

In FOL

(1) Someone entered. He smiled.
\[ \exists x \ (\text{enter } x) \land \text{smile } x \]

In Dependent Type Theory

(2) Someone entered. He smiled.
\[ \left[ u : \begin{array}{l} x : \text{entity} \\ \text{enter}(x) \\ \text{smile}(\pi_1 u) \end{array} \right] \]
Underspecified terms

What is the semantic representation (SR) for (3)?

(3) The elevator is clean.

(4) There is an elevator and it is clean.  
   Presupposition    Assertion
   ▷ Anaphoric dependency across the two-dimensions!

SR for (3) in DTS:

(5) \textbf{clean} \left( \pi_1 \left( \@i \left[ x : \text{entity} \vphantom{\text{elevator}} \left( \vphantom{\text{elevator}} x \right) \vphantom{\text{elevator}} \right] \right) \right)

- The annotated type $A$ in $\@i A$ represents the presupposition
Underspecified terms

(5) \[ \textbf{clean} \left( \pi_1 \left( \bigoplus_{i} \left[ x : \text{entity} \varepsilon \text{levator}(x) \right] \right) \right) \]

- For (5) to be well-formed, one needs to construct a term for the type \[ \left[ x : \text{entity} \varepsilon \text{levator}(x) \right] \]
- Take the first projection \( \pi_1 \) of a pair \((x, p)\) where we have \(x : \text{entity}\) and \(p : \text{elevator}(x)\).
Nested presupposition

John’s sister is happy.

\[
\text{happy} \left( \pi_1 \left( \ominus_i \left[ x : \text{entity} \right] \text{ sister}(x, \text{john}) \right) \right)
\]

John’s sister’s husband is happy.

\[
\text{happy} \left( \pi_1 \left( \ominus_1 \left[ x : \text{entity} \right] \text{ husband}(x, \pi_1 \left( \ominus_2 \left[ y : \text{entity} \right] \text{ sister}(y, \text{john}) \right)) \right) \right)
\]
Pipeline

Text

CCG parsing + semantic composition

Underspecified SR

Type checking + proof construction

@-elimination / accommodation

Fully specified SR

\(\Sigma\)-type elimination

Theorem proving
System demonstration

Examples:

(6) a. A farmer who owns a donkey beats it.
   b. The farmer owns the donkey.
   c. Someone met his daughter yesterday.
Example

CCG Parsing and semantic composition:

*The elevator is clean*

\[ \rightsquigarrow \text{clean} \left( \pi_1 \left( \emptyset_1 \left[ x : \text{entity} \ \text{elevator}(x) \right] \right) \right) \]
Type Checking

\[
\text{clean} \left( \pi_1 \left( \mathcal{O}_1 \left[ x : \text{entity elevator}(x) \right] \right) \right) : \text{type}
\]
Type Checking

\[
\begin{align*}
\text{clean} : \text{entity} & \to \text{type} \\
\pi_1 \left( \@_1 \left[ x : \text{entity} \right. \right. \left. \left. \text{elevator}(x) \right] \right) : \text{type}
\end{align*}
\]
Type Checking

\[
\text{clean} : \text{entity} \rightarrow \text{type} \quad \Rightarrow \quad \pi_1 \left( \oplus_1 \left[ \begin{array}{c} x \colon \text{entity} \\ \text{elevator}(x) \end{array} \right] \right) : \text{entity} \\
\text{clean} \left( \pi_1 \left( \oplus_1 \left[ \begin{array}{c} x \colon \text{entity} \\ \text{elevator}(x) \end{array} \right] \right) \right) : \text{type}
\]
Type Checking

\[
\begin{align*}
\text{clean} : \text{entity} & \rightarrow \text{type} \\
\pi_1 \left( \pi_1 \left( \text{elevator} (x) \right) \right) : \text{entity} \\
\text{clean} \left( \pi_1 \left( \pi_1 \left( \text{elevator} (x) \right) \right) \right) : \text{type}
\end{align*}
\]
Type Checking

\[
\begin{align*}
\text{clean : entity \rightarrow type} & \quad \pi_1 \left( \underbrace{\text{clean}}_{\text{entity}} \left( \underbrace{\pi_1 \left( \underbrace{\text{clean}}_{\text{entity}} \left( \text{elevator}(x) \right) \right)}_{\text{entity}} \right) \right) : \text{type} \\
\end{align*}
\]
Type Checking

\[
\begin{align*}
\text{clean : entity} & \rightarrow \text{type} \\
\pi_1 \left( \Theta_1 \left[ x : \text{entity elevator}(x) \right] \right) & : \text{entity} \\
\text{clean} \left( \pi_1 \left( \Theta_1 \left[ x : \text{entity elevator}(x) \right] \right) \right) & : \text{type}
\end{align*}
\]
Type Checking

\[
\begin{align*}
\text{clean} : \text{entity} & \rightarrow \text{type} \\
\pi_1 \left( \@_1 \left[ x : \text{entity elevator}(x) \right] \right) : \text{entity} \\
\text{clean} \left( \pi_1 \left( \@_1 \left[ x : \text{entity elevator}(x) \right] \right) \right) : \text{type}
\end{align*}
\]
Option 1. Proof construction + @-elimination

Suppose we have

$$\mathcal{K} \equiv t : \text{entity}, \ u : \text{elevator}(t).$$

Then we can prove:

$$\mathcal{K} \vdash (t, u) : \left[ x : \text{entity} \right. \left. \vphantom{\text{elevator}(x)} \text{elevator}(x) \right]$$

Replace the @-term with a constructed proof term:

$$\text{clean} \left( \pi_1 \left( @_1 \left[ x : \text{entity} \vphantom{\text{elevator}(x)} \text{elevator}(x) \right] \right) \right) \rightsquigarrow \text{clean} (\pi_1 (t, u))$$

$$\rightarrow_\beta \text{clean} (t)$$
Option 2. Accommodation

- **Σ-accommodation:**
  John’s daughter is happy.
  \[ \leadsto \text{John has a daughter and she is happy.} \]

  \[
  \text{happy} \left( \pi_1 \left( \ominus_1 \left[ x : \text{entity} \right. \right. \right. \\
  \left. \left. \left. \text{daughter} \left( x, j \right) \right] \right) \right) \leadsto \left[ \begin{array}{c}
  u : [ x : \text{entity} ] \\
  \text{daughter} ( x, j ) \\
  \text{happy} ( \pi_1 u )
  \end{array} \right]
  \]

- **Π-accommodation:**
  John’s daughter is happy.
  \[ \leadsto \text{If John has a daughter, she is happy.} \]

  \[
  \text{happy} \left( \pi_1 \left( \ominus_1 \left[ x : \text{entity} \right. \right. \right. \\
  \left. \left. \left. \text{daughter} \left( x, j \right) \right] \right) \right) \leadsto \left[ \begin{array}{c}
  u : [ x : \text{entity} ] \\
  \text{daughter} ( x, j ) \\
  \text{happy} ( \pi_1 u )
  \end{array} \right]
  \]
\[\text{\textbf{\(\Sigma\)-type elimination}}\]

Obtain the \(\Sigma\)-free form using the equivalence:

\[
\begin{align*}
\left[ t : \begin{array}{c}
u : A \\
B \\
C \\
\end{array} \right] & \equiv \left[ \begin{array}{c}
u : A \\
\pi_1 t \mapsto u, \pi_2 t \mapsto v \\
C \\
\end{array} \right] \\
\left( t : \begin{array}{c}
u : A \\
B \end{array} \right) \rightarrow C & \equiv (u : A) \rightarrow (v : B) \rightarrow C (\pi_1 t \mapsto u, \pi_2 t \mapsto v)
\end{align*}
\]

**Example**

\[
\begin{align*}
\left[ t : \begin{array}{c}
x : \text{entity} \\
\text{boy}(x) \\
\text{enter}(\pi_1 u) \\
\text{whistle}(\pi_1 \pi_1 t) \end{array} \right] & \equiv \left[ \begin{array}{c}
x : \text{entity} \\
\text{boy}(x) \\
\text{enter}(\pi_1 u) \\
\text{whistle}(\pi_1 u) \end{array} \right] \\
& \equiv \left[ \begin{array}{c}
x : \text{entity} \\
\text{boy}(x) \\
\text{enter}(x) \\
\text{whistle}(x) \end{array} \right]
\end{align*}
\]
Example:

P1. Every farmer who owns a donkey beats it.
P2. John owns a donkey.
P3. John is a farmer.

C John beats a donkey.


