Efficient NORMAL–FORM Parsing
for Combinatory Categorial Grammar

Jason M. Eisner
University of Pennsylvania
June 26, 1996 at ACL
CCG and the Spurious Ambiguity Problem

\[ \text{[John likes Mary]} \]
\[ \text{John [likes Mary]} \]
\[ \text{[John likes]} \text{ Mary} \]

S (sentence)
S/NP (sentence missing NP to its left – "\")
S/NP (sentence missing NP to its right – "/")

... can conjoin this with other predicates

[John likes], and [Sue hates], that woman in the hat

... can ask who satisfies it

Who does [John like]?

... can state who satisfies it

It is MARY that [John likes]. / [John likes] MARY.

CCG allows linguistically useful extra constituents ...
CCG and the Spurious Ambiguity Problem

Two parses for an unambiguous sentence:

[[John likes] Mary]  (non-standard parse)
[John [likes Mary]]  (standard parse)

the [aide in the] Senate [that D’Amato says Clinton tried to] bribe

... but CCG forces hundreds of extra parses on us.
Today’s Talk

- Sketch of CCG formalism
  + the B combinators

- A solution to spurious ambiguity

- Why the solution works (formal intuitions)

- Important extensions of the solution
  + the S combinator (straightforward)
  + the T combinator (work in progress)
  + restrictions on the rules
<table>
<thead>
<tr>
<th>forward rules</th>
<th>backward rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;B0: A/B   B  \rightarrow A</td>
<td>&lt;B0: B  A\B  \rightarrow A</td>
</tr>
<tr>
<td>&gt;B1: A/B   B/C  \rightarrow A/C</td>
<td>&lt;B1: B\C  A\B  \rightarrow A\C</td>
</tr>
<tr>
<td>A/B   B\C  \rightarrow A\C</td>
<td>B/C   A\B  \rightarrow A/C</td>
</tr>
<tr>
<td>&gt;B2: A/B   B/C/D  \rightarrow A/C/D</td>
<td>&lt;B2: B\C/D  A\B  \rightarrow A/C/D</td>
</tr>
<tr>
<td>A/B   B/C\D  \rightarrow A/C\D</td>
<td>B\C/D   A\B  \rightarrow A/C\D</td>
</tr>
<tr>
<td>A/B   B\C/D  \rightarrow A\C\D</td>
<td>B/C\D   A\B  \rightarrow A/C\D</td>
</tr>
<tr>
<td>A/B   B\C\D  \rightarrow A\C\D</td>
<td>B/C\D   A\B  \rightarrow A/C\D</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>
Sketch of CCG Formalism:

Example

$\lambda u \text{ bribed}(u) \rightarrow A/C$

$\lambda u f(g(u))$

$\Rightarrow A$

$\Rightarrow f(x)$

$A/B \xrightarrow{\ell} B$
A Solution to Spurious Ambiguity: The Goal

Exactly one parse per reading.
(Efficiently suppress all other parses.)
A Solution to Spurious Ambiguity: The Strategy

How can we rule out extra parses?

Yes, allow all of CCG’s non–standard constituents,
both when useful \[D’Amato \text{ said } Clinton \text{ tried},\]
and [maybe he said she failed], to bribe that aide.
and when useless. \[D’Amato \text{ said Clinton tried}] to bribe that aide.

BUT:

1 parse not 5
1 parse not 5
assemble 1 parse not 25
and in this case, disallow even that 1 parse!
(but do allow: [ [D’Amato] [said Clinton tried to bribe that aide] ] )
A Solution to Spurious Ambiguity: The Tactics

Standard kind of spurious ambiguity:
Forward (or backward) "chains"

<table>
<thead>
<tr>
<th>VP/NP</th>
<th>NP/N</th>
<th>N</th>
<th>A/A</th>
<th>A/B</th>
<th>B\C/D/E</th>
<th>E/F</th>
<th>F\G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 parses</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The OUTPUT of forward composition

\( (>B_1, >B_2, >B_3, ...) \)

may not be the primary (left) INPUT to any forward rule.

\( (>B_0, >B_1, >B_2, >B_3 ... \)

The OUTPUT of backward composition

\( (>B_1, >B_2, >B_3, ...) \)

may not be the primary (right) INPUT to any backward rule.

\( (>B_0, >B_1, >B_2, >B_3 ... \)
The OUTPUT of forward composition $(>B_1, >B_2, >B_3, ...)$ may not be the primary (left) INPUT to any forward rule.

$(>B_0, >B_1, >B_2, >B_3 ...)$

A Solution to Spurious Ambiguity: The Tactics in Action

VP

bribed(the(aide))

satisfies constraint

violates constraint

(a "normal-form" tree)
A Solution to Spurious Ambiguity: The Result

For CCG with the generalized composition rules (including mixed), these tactics

(1) eliminate ONLY spurious ambiguity (safety)
(2) eliminate ALL spurious ambiguity (completeness)

1–1 correspondence:

*semantic equiv. classes*  \[\overset{\leftrightarrow}{\longrightarrow}\]  *normal–form trees*
Formal Intuitions: What is Spurious Ambiguity?

A syntax tree takes interps of the words, and combines them semantically into an interp. of the phrase.

So a syntax tree on n words computes an n–ary function: \( \lambda f \lambda g \lambda h \lambda k \ ( \lambda z \lambda y \ f(g(h(\lambda w \ k(z)(w))))(y)) \)

Two trees on the same n words are semantically equivalent iff they compute the same n–ary semantic function.
Formal Intuitions: What is Spurious Ambiguity?

Two trees on the same n words are semantically equivalent iff they compute the same n–ary semantic function.

What this definition is NOT:

(1) Does this mean "iff they compute the same lambda–term"?

(2) Do we eliminate one parse from each of these pairs?

[quietly [knock twice]]

[[quietly knock] twice]

[\(\pi\) equals [2 plus 3 over 4]]

[\(\pi\) equals [2 plus [3 over 4]]]

denote same action

denote same truth value ("false")
Formal Intuitions: Existence Theorem

**Theorem.** For every tree $T$ we cut down with our constraints, we leave standing a semantically equivalent tree, $\text{NF}(T)$.

**Proof.** To construct $\text{NF}(T)$ from $T$, essentially replace $>B_m$ throughout with $>B(m+n-1)$.

Construction used is inductive.
Takes $O(1)$ time, if $\text{NF}(T')$ is known for $T'$ smaller than $T$. 
Formal Intuitions: Uniqueness Theorem

Theorem. We never leave two equivalent trees standing.

Proof. Given two distinct trees that we keep. They must differ somewhere syntactically:

so contain either

(tree 1)

\[ \ldots x \quad y \quad \ldots \]

one rule

(tree 2)

\[ \ldots x \quad y \quad \ldots \]

another rule

or

\[ \ldots x \quad y \quad \ldots \]

\[ \ldots y \quad z \quad \ldots \]

Show that they differ semantically as a result.
Formal Intuitions: The Spurious Ambiguity Lemma

2 parses on the same sequence of words are spuriously ambiguous ...

Def. ... iff spuriosity is robust under changes to words’ semantics.

Equiv def. ... iff ambiguity is robust under changes to words’ syntax.

Easy syntactic characterization of a semantic property!
Formal Intuitions:

Proof of Spurious Ambig. Lemma

\[ \lambda z \lambda y f(g(h(\lambda w k(z)(w)))y) \]

\[ \lambda x \lambda y f(g(x)y) \]

\[ \lambda x \lambda y \lambda z \lambda w k(z)(w) \]

\[ \lambda f \lambda g \lambda h \lambda k \]

\[ \lambda f \lambda g \lambda h \lambda k (\lambda z \lambda y f(g(h(\lambda w k(z)(w)))y))) \]

restricted combinator

injunctive

injunctive

most general polymorphic type

\[ (B \rightarrow A) \rightarrow (D \rightarrow C \rightarrow B) \rightarrow (X \rightarrow D) \rightarrow (G \rightarrow X) \rightarrow (G \rightarrow C \rightarrow A) \]

can write as \[ (A|C|G) | (X|G) | (D|X) | (B|C|D) | (A|B) \]
Extensions: The S and T combinators

If we add the S (substitution) combinator, we need a new restriction:

Just as

The OUTPUT of (>B1, >B2, >B3, ...)
may not be the primary (left) INPUT to (>B0, >B1, >B2, >B3, ...)

now

The OUTPUT of (>B2, >B3, ...)
may not be the primary (left) INPUT to >S

If we add the T (type-raising) combinator,
the ambiguities get much trickier!  Work in progress.
Extensions: Making TR visible to the grammar

If type-raising is only lexical, our definition can’t see this ambiguity:

\[
\begin{align*}
\text{John} & \quad \text{likes} & \quad \text{Mary} \\
\text{NP} & \quad \text{(S\(\text{NP})/\text{NP)} & \quad \text{NP} \\
\text{parses of different sentences!}
\end{align*}
\]

and the ambiguity below depends on funny "lexical" properties of type-raised arguments, so doesn’t look spurious:

In fact, he parroted yesterday [her stand on Bosnia]

\[
\begin{align*}
\text{S/}(\text{S}\lbrack \text{NP}\rbrack) & \quad \text{S/}(\text{S}\lbrack \text{NP}\rbrack) \\
\text{NP} & \quad \text{NP}
\end{align*}
\]
Extensions: Restrictions on CCG rules

In practice, a CCG grammar may state WHICH rules can apply, & WHEN.

Don’t change the theorems, change the parser!

Karttunen 1986: No constraints on parses. Whenever we find a new parse of a constituent, check that it’s not redundant.

But checking new parse against old parses takes exponential time.

New idea: See if its NF matches an old parse’s. Can do in O(1) time.
Extensions: Finding Equiv Classes instead of NFs

Have proved 1–1 correspondence:

Semantic equiv. classes \( \leftrightarrow \) Normal−form trees

So use each NF tree as a magnet for its equivalence class:

- Not found by parser (disallowed by grammar, or conflict with prior "incremental" commitments)
- Keep just one of these legal parses – e.g. the first, or the best according to prosody or discourse module
Summary of Results

+ A useful model–theoretic definition of spurious ambiguity ... and a lemma giving a syntactic test for it.

+ Easy, fast parser for CCG with the B and S rules.
  Simple constraints provably eliminate all spurious ambiguity.

+ Fast parser still possible if grammar rules have nasty restrictions:
  Rapidly group legal (sub)trees by semantic equivalence class –
  just have each NF tree point to the legal trees in its class.