Proceedings of the 34th Annual Meeting of the ACL, Santa Cruz, June 1996, pp. 79-86.

Efficient Normal-Form Parsing for Combinatory Categorial Grammar*

Jason Eisner
Dept. of Computer and Information Science
University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104-6389, USA
jeisner@linc.cis.upenn.edu

Abstract

Under categorial grammars that have powerful rules like composition, a simple n-word sentence can have exponentially many parses. Generating all parses is inefficient and obscures whatever true semantic ambiguities are in the input. This paper addresses the problem for a fairly general form of Combinatory Categorial Grammar, by means of an efficient, correct, and easy to implement normal-form parsing technique. The parser is proved to find *exactly one* parse in each semantic equivalence class of allowable parses; that is, spurious ambiguity (as carefully defined) is shown to be both safely and completely eliminated.

1 Introduction

Combinatory Categorial Grammar (Steedman, 1990), like other "flexible" categorial grammars, suffers from *spurious ambiguity* (Wittenburg, 1986). The non-standard constituents that are so crucial to CCG's analyses in (1), and in its account of intonational focus (Prevost & Steedman, 1994), remain available even in simpler sentences. This renders (2) syntactically ambiguous.

(1) a. **Coordination:** [[John likes]$_{S/NP}$, and [Mary pretends to like]$_{S/NP}$], the big galoot in the corner.
b. **Extraction:** Everybody at this party [whom [John likes]$_{S/NP}$] is a big galoot.

(2) a. John [likes Mary]$_{S\backslash NP}$.
b. [John likes]$_{S/NP}$ Mary.

The practical problem of "extra" parses in (2) becomes exponentially worse for longer strings, which can have up to a Catalan number of parses. An exhaustive parser serves up 252 CCG parses of (3), which must be sifted through, at considerable cost, in order to identify the two distinct meanings for further processing.[1]

(3) the galoot in the corner
NP/N N (N\N)/NP NP/N N
that I said Mary
(N\N)/(S/NP) S/(S\NP) (S\NP)/S S/(S\NP)
pretends to
(S\NP)/(S_{inf}\NP) (S_{inf}\NP)/(S_{stem}\NP)
like
(S_{stem}\NP)/NP

This paper presents a simple and flexible CCG parsing technique that prevents any such explosion of redundant CCG derivations. In particular, it is proved in §4.2 that the method constructs *exactly one* syntactic structure per semantic reading—e.g., just two parses for (3). All other parses are suppressed by simple normal-form constraints that are enforced throughout the parsing process. This approach works because CCG's spurious ambiguities arise (as is shown) in only a small set of circumstances. Although similar work has been attempted in the past, with varying degrees of success (Karttunen, 1986; Wittenburg, 1986; Pareschi & Steedman, 1987; Bouma, 1989; Hepple & Morrill, 1989; König, 1989; Vijay-Shanker & Weir, 1990; Hepple, 1990; Moortgat, 1990; Hendriks, 1993; Niv, 1994), this appears to be the first full normal-form result for a categorial formalism having more than context-free power.

2 Definitions and Related Work

CCG may be regarded as a generalization of context-free grammar (CFG)—one where a grammar has infinitely many nonterminals and phrase-structure rules. In addition to the familiar *atomic* nonterminal categories (typically `S` for sentences, `N` for

*This material is based upon work supported under a National Science Foundation Graduate Fellowship. I have been grateful for the advice of Aravind Joshi, Nobo Komagata, Seth Kulick, Michael Niv, Mark Steedman, and three anonymous reviewers.

[1] Namely, Mary pretends to like the galoot in 168 parses and the corner in 84. One might try a statistical approach to ambiguity resolution, discarding the low-probability parses, but it is unclear how to model and train any probabilities when no single parse can be taken as the standard of correctness.

nouns, NP for noun phrases, etc.), CCG allows infinitely many *slashed* categories. If x and y are categories, then x/y (respectively $x\backslash y$) is the category of an incomplete x that is missing a y at its right (respectively left). Thus verb phrases are analyzed as subjectless sentences S\NP, while "John likes" is an objectless sentence or S/NP. A complex category like ((S\NP)\(S\NP))/N may be written as S\NP\(S\NP)/N, under a convention that slashes are left-associative.

The results herein apply to the TAG-equivalent CCG formalization given in (Joshi et al., 1991).[2] In this variety of CCG, every (non-lexical) phrase-structure rule is an instance of one of the following binary-rule templates (where $n \geq 0$):

(4) Forward generalized composition >Bn:
$x/y \quad y\,|_n z_n \cdots |_2 z_2\,|_1 z_1 \;\rightarrow\; x\,|_n z_n \cdots |_2 z_2\,|_1 z_1$
Backward generalized composition <Bn:
$y\,|_n z_n \cdots |_2 z_2\,|_1 z_1 \quad x\backslash y \;\rightarrow\; x\,|_n z_n \cdots |_2 z_2\,|_1 z_1$

Instances with $n = 0$ are called *application* rules, and instances with $n \geq 1$ are called *composition* rules. In a given rule, $x, y, z_1 \ldots z_n$ would be instantiated as categories like NP, S/NP, or S\NP\(S\NP)/N. Each of $|_1$ through $|_n$ would be instantiated as either / or \.

A fixed CCG grammar need not include every phrase-structure rule matching these templates. Indeed, (Joshi et al., 1991) place certain restrictions on the rule set of a CCG grammar, including a requirement that the rule degree n is bounded over the set. The results of the present paper apply to such restricted grammars and also more generally, to any CCG-style grammar with a *decidable* rule set.

Even as restricted by (Joshi et al., 1991), CCGs have the "mildly context-sensitive" expressive power of Tree Adjoining Grammars (TAGs). Most work on spurious ambiguity has focused on categorial formalisms with substantially less power. (Hepple, 1990) and (Hendriks, 1993), the most rigorous pieces of work, each establish a normal form for the syntactic calculus of (Lambek, 1958), which is weakly context-free. (König, 1989; Moortgat, 1990) have also studied the Lambek calculus case. (Hepple & Morrill, 1989), who introduced the idea of normal-form parsing, consider only a small CCG fragment that lacks backward or order-changing composition; (Niv, 1994) extends this result but does not show completeness. (Wittenburg, 1987) assumes a CCG fragment lacking order-changing or higher-order composition; furthermore, his revision of the combinators creates new, conjoinable constituents that conventional CCG rejects. (Bouma, 1989) proposes to replace composition with a new combinator, but the resulting product-grammar scheme assigns different types to "John likes" and "Mary pretends to like," thus losing the ability to conjoin such constituents or subcategorize for them as a class. (Pareschi & Steedman, 1987) do tackle the CCG case, but (Hepple, 1987) shows their algorithm to be incomplete.

[2]This formalization sweeps any type-raising into the lexicon, as has been proposed on linguistic grounds (Dowty, 1988; Steedman, 1991, and others). It also treats conjunction lexically, by giving "and" the generalized category $x\backslash x/x$ and barring it from composition.

3 Overview of the Parsing Strategy

As is well known, general CFG parsing methods can be applied directly to CCG. Any sort of chart parser or non-deterministic shift-reduce parser will do. Such a parser repeatedly decides whether two adjacent constituents, such as S/NP and NP/N, should be combined into a larger constituent such as S/N. The role of the grammar is to state which combinations are allowed. The key to efficiency, we will see, is for the parser to be less permissive than the grammar—for it to say "no, redundant" in some cases where the grammar says "yes, grammatical."

(5) shows the constituents that untrammeled CCG will find in the course of parsing "John likes Mary." The spurious ambiguity problem is not that the grammar allows (5c), but that the grammar allows both (5f) and (5g)—distinct parses of the *same* string, with the *same* meaning.

(5) a. $[\text{John}]_{\text{S/(S\NP)}}$
b. $[\text{likes}]_{\text{(S\NP)/NP}}$
c. $[\text{John likes}]_{\text{S/NP}}$
d. $[\text{Mary}]_{\text{NP}}$
e. $[\text{likes Mary}]_{\text{S\NP}}$
f. $[[\text{John likes}]\ \text{Mary}]_{\text{S}}$ ⟵ *to be disallowed*
g. $[\text{John}\ [\text{likes Mary}]]_{\text{S}}$

The proposal is to construct all constituents shown in (5) except for (5f). If we slightly constrain the use of the grammar rules, the parser will still produce (5c) and (5d)—constituents that are indispensable in contexts like (1)—while refusing to *combine* those constituents into (5f). The relevant rule S/NP NP → S will actually be blocked when it attempts to construct (5f). Although rule-blocking may eliminate an analysis of the sentence, as it does here, a semantically equivalent analysis such as (5g) will always be derivable along some other route.

In general, our goal is to discover exactly one analysis for each <substring, meaning> pair. By practicing "birth control" for each bottom-up generation of constituents in this way, we avoid a population explosion of parsing options. "John likes Mary" has only one reading semantically, so just one of its analyses (5f)–(5g) is discovered while parsing (6). Only that analysis, and not the other, is allowed to continue on and be built into the final parse of (6).

(6) that galoot in the corner that thinks [John likes Mary]$_{\text{S}}$

For a chart parser, where each chart cell stores the analyses of some substring, this strategy says that

all analyses in a cell are to be semantically distinct. (Karttunen, 1986) suggests enforcing that property directly—by comparing each new analysis semantically with existing analyses in the cell, and refusing to add it if redundant—but (Hepple & Morrill, 1989) observe briefly that this is inefficient for large charts.[3] The following sections show how to obtain effectively the same result without doing any semantic interpretation or comparison at all.

4 A Normal Form for "Pure" CCG

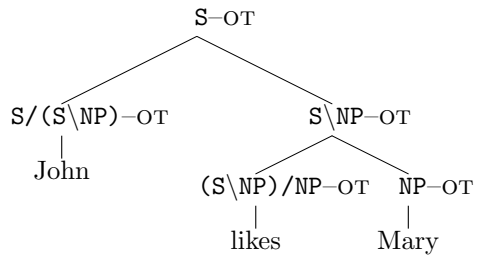
It is convenient to begin with a special case. Suppose the CCG grammar includes not some but *all* instances of the binary rule templates in (4). (As always, a separate lexicon specifies the possible categories of each word.) If we group a sentence's parses into semantic equivalence classes, it always turns out that exactly one parse in each class satisfies the following simple declarative constraints:

(7) a. No constituent produced by `>B`n, any $n \geq 1$, ever serves as the primary (left) argument to `>B`n', any $n' \geq 0$.
b. No constituent produced by `<B`n, any $n \geq 1$, ever serves as the primary (right) argument to `<B`n', any $n' \geq 0$.

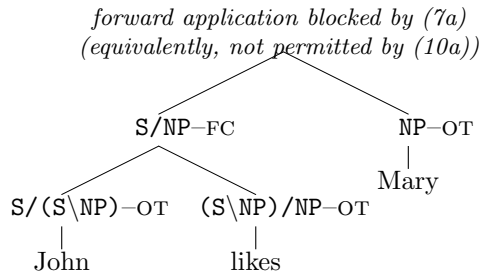
The notation here is from (4). More colloquially, (7) says that the output of rightward (leftward) composition may not compose or apply over anything to *its* right (left). A parse tree or subtree that satisfies (7) is said to be in *normal form* (NF).

As an example, consider the effect of these restrictions on the simple sentence "John likes Mary." Ignoring the tags –OT, –FC, and –BC for the moment, (8a) is a normal-form parse. Its competitor (8b) is not, nor is any larger tree containing (8b). But non-standard constituents are allowed when necessary: (8c) *is* in normal form (cf. (1)).

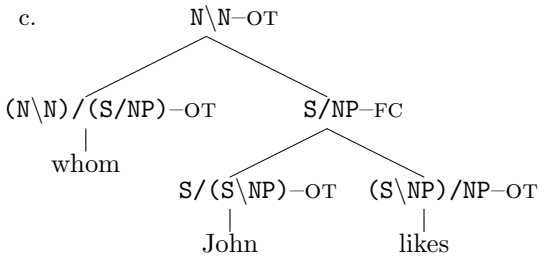
(8) a.



b.



c.



It is not hard to see that (7a) eliminates all but right-branching parses of "forward chains" like `A/B B/C C` or `A/B/C C/D D/E/F/G G/H`, and that (7b) eliminates all but left-branching parses of "backward chains." (Thus every functor will get its arguments, if possible, before it becomes an argument itself.) But it is hardly obvious that (7) eliminates *all* of CCG's spurious ambiguity. One might worry about unexpected interactions involving crossing composition rules like `A/B B\C` → `A\C`. Significantly, it turns out that (7) really does suffice; the proof is in §4.2.

It is trivial to modify any sort of CCG parser to find only the normal-form parses. No semantics is necessary; simply block any rule use that would violate (7). In general, detecting violations will not hurt performance by more than a constant factor. Indeed, one might implement (7) by modifying CCG's phrase-structure grammar. Each ordinary CCG category is split into three categories that bear the respective tags from (9). The 24 templates schematized in (10) replace the two templates of (4). Any CFG-style method can still parse the resulting spuriosity-free grammar, with tagged parses as in (8). In particular, the polynomial-time, polynomial-space CCG chart parser of (Vijay-Shanker & Weir, 1993) can be trivially adapted to respect the constraints by tagging chart entries.

[3] How inefficient? (i) has exponentially many semantically *distinct* parses: $n = 10$ yields 82,756,612 parses in $\binom{20}{10} = 48{,}620$ equivalence classes. Karttunen's method must therefore add 48,620 representative parses to the appropriate chart cell, first comparing each one against all the previously added parses—of which there are 48,620/2 on average—to ensure it is not semantically redundant. (Additional comparisons are needed to reject parses other than the lucky 48,620.) Adding a parse can therefore take exponential time.

(i) $\overbrace{\ldots\ \texttt{S/S}\ \ \texttt{S/S}\ \ \texttt{S/S}}^{n}\ \ \texttt{S}\ \ \overbrace{\texttt{S\S}\ \ \texttt{S\S}\ \ \texttt{S\S}\ \ldots}^{n}$

Structure sharing does not appear to help: parses that are grouped in a parse forest have only their syntactic category in common, not their meaning. Karttunen's approach must tease such parses apart and compare their various meanings individually against each new candidate. By contrast, the method proposed below is purely syntactic—just like any "ordinary" parser—so it never needs to unpack a subforest, and can run in polynomial time.

(9) –FC output of >Bn, some $n \geq 1$ (a forward composition rule)
–BC output of <Bn, some $n \geq 1$ (a backward composition rule)
–OT output of >B0 or <B0 (an application rule), or lexical item

(10) a. Forward application >B0: $\left\{ \begin{array}{l} x/y\text{–BC} \\ x/y\text{–OT} \end{array} \right\} \left\{ \begin{array}{l} y\text{–FC} \\ y\text{–BC} \\ y\text{–OT} \end{array} \right\} \rightarrow x\text{–OT}$

b. Backward application <B0: $\left\{ \begin{array}{l} y\text{–FC} \\ y\text{–BC} \\ y\text{–OT} \end{array} \right\} \left\{ \begin{array}{l} x\backslash y\text{–FC} \\ x\backslash y\text{–OT} \end{array} \right\} \rightarrow x\text{–OT}$

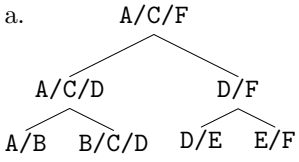
c. Fwd. composition >Bn ($n \geq 1$): $\left\{ \begin{array}{l} x/y\text{–BC} \\ x/y\text{–OT} \end{array} \right\} \left\{ \begin{array}{l} y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–FC} \\ y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–BC} \\ y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–OT} \end{array} \right\} \rightarrow \; x\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–FC}$

d. Bwd. composition <Bn ($n \geq 1$): $\left\{ \begin{array}{l} y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–FC} \\ y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–BC} \\ y\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–OT} \end{array} \right\} \left\{ \begin{array}{l} x\backslash y\text{–FC} \\ x\backslash y\text{–OT} \end{array} \right\} \rightarrow \; x\,|_n z_n \cdots |_2 z_2\,|_1 z_1\text{–BC}$

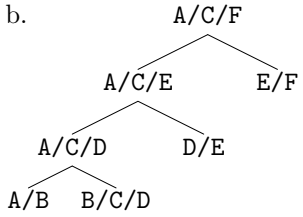
(11) a. Syn/sem for >Bn ($n \geq 0$): $\begin{array}{ccccc} x/y & & y\,|_n z_n \cdots |_2 z_2\,|_1 z_1 & \rightarrow & x\,|_n z_n \cdots |_2 z_2\,|_1 z_1 \\ \Updownarrow & & \Updownarrow & & \Updownarrow \\ f & & g & & \lambda c_1 \lambda c_2 \ldots \lambda c_n . f(g(c_1)(c_2)\cdots(c_n)) \end{array}$

b. Syn/sem for <Bn ($n \geq 0$): $\begin{array}{ccccc} y\,|_n z_n \cdots |_2 z_2\,|_1 z_1 & & x\backslash y & \rightarrow & x\,|_n z_n \cdots |_2 z_2\,|_1 z_1 \\ \Updownarrow & & \Updownarrow & & \Updownarrow \\ g & & f & & \lambda c_1 \lambda c_2 \ldots \lambda c_n . f(g(c_1)(c_2)\cdots(c_n)) \end{array}$

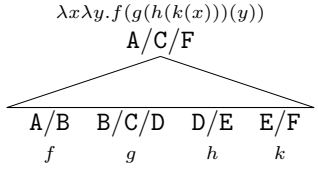
(12) a.



b.



c.



It is interesting to note a rough resemblance between the tagged version of CCG in (10) and the tagged Lambek calculus **L***, which (Hendriks, 1993) developed to eliminate spurious ambiguity from the Lambek calculus **L**. Although differences between CCG and **L** mean that the details are quite different, each system works by marking the output of certain rules, to prevent such output from serving as input to certain other rules.

4.1 Semantic equivalence

We wish to establish that each semantic equivalence class contains exactly one NF parse. But what does "semantically equivalent" mean? Let us adopt a standard model-theoretic view.

For each leaf (i.e., lexeme) of a given syntax tree, the lexicon specifies a *lexical interpretation* from the model. CCG then provides a *derived interpretation* in the model for the complete tree. The standard CCG theory builds the semantics compositionally, guided by the syntax, according to (11). We may therefore regard a syntax tree as a static "recipe" for combining word meanings into a phrase meaning.

One might choose to say that two parses are semantically equivalent iff they derive the same phrase meaning. However, such a definition would make spurious ambiguity sensitive to the fine-grained semantics of the lexicon. Are the two analyses of VP/VP VP VP\VP semantically equivalent? If the lexemes involved are "softly knock twice," then yes, as softly(twice(knock)) and twice(softly(knock)) arguably denote a common function in the semantic model. Yet for "intentionally knock twice" this is not the case: these adverbs do not commute, and the semantics are distinct.

It would be difficult to make such subtle distinctions rapidly. Let us instead use a narrower, "intensional" definition of spurious ambiguity. The trees in (12a–b) will be considered equivalent because they specify the same "recipe," shown in (12c). No matter what lexical interpretations f, g, h, k are fed into the leaves A/B, B/C/D, D/E, E/F, both the trees end up with the same derived interpretation, namely a model element that can be determined from f, g, h, k by calculating $\lambda x \lambda y . f(g(h(k(x)))(y))$.

By contrast, the two readings of "softly knock

twice" are considered to be distinct, since the parses specify different recipes. That is, given a suitably free choice of meanings for the words, the two parses can be made to pick out two different VP-type functions in the model. The parser is therefore conservative and keeps both parses.[4]

4.2 Normal-form parsing is safe & complete

The motivation for producing only NF parses (as defined by (7)) lies in the following existence and uniqueness theorems for CCG.

Theorem 1 *Assuming "pure CCG," where all possible rules are in the grammar, any parse tree α is semantically equivalent to some NF parse tree $NF(\alpha)$.*

(This says the NF parser is *safe* for pure CCG: we will not lose any readings by generating just normal forms.)

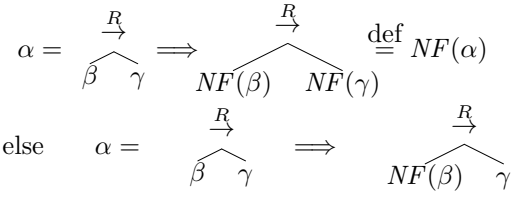
Theorem 2 *Given distinct NF trees $\alpha \neq \alpha'$ (on the same sequence of leaves). Then α and α' are not semantically equivalent.*

(This says that the NF parser is *complete*: generating only normal forms eliminates *all* spurious ambiguity.)

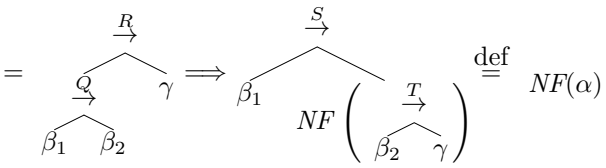
Detailed proofs of these theorems are available on the cmp-lg archive, but can only be sketched here. Theorem 1 is proved by a constructive induction on the order of α, given below and illustrated in (13):

- For α a leaf, put $NF(\alpha) = \alpha$.
- ($<R, \beta, \gamma>$ denotes the parse tree formed by combining subtrees β, γ via rule R.)
  If $\alpha = <R, \beta, \gamma>$, then take $NF(\alpha) = <R, NF(\beta), NF(\gamma)>$, which exists by inductive hypothesis, unless this is not an NF tree. In the latter case, WLOG, R is a forward rule and $NF(\beta) = <Q, \beta_1, \beta_2>$ for some forward composition rule Q. Pure CCG turns out to provide forward rules S and T such that $\alpha' = <S, \beta_1, NF(<T, \beta_2, \gamma>)>$ is a constituent and is semantically equivalent to α. Moreover, since β_1 serves as the primary subtree of the NF tree $NF(\beta)$, β_1 cannot be the output of forward composition, and is NF besides. Therefore α' is NF: take $NF(\alpha) = \alpha'$.

(13) If $NF(\beta)$ not output of fwd. composition,



[4](Hepple & Morrill, 1989; Hepple, 1990; Hendriks, 1993) appear to share this view of semantic equivalence. Unlike (Karttunen, 1986), they try to eliminate only parses whose denotations (or at least λ-terms) are systematically equivalent, not parses that happen to have the same denotation through an accident of the lexicon.



This construction resembles a well-known normal-form reduction procedure that (Hepple & Morrill, 1989) propose (without proving completeness) for a small fragment of CCG.

The proof of theorem 2 (completeness) is longer and more subtle. First it shows, by a simple induction, that since α and α' disagree they must disagree in at least one of these ways:

(a) There are trees β, γ and rules $R \neq R'$ such that $<R, \beta, \gamma>$ is a subtree of α and $<R', \beta, \gamma>$ is a subtree of α'. (For example, `S/S S\S` may form a constituent by either `<B1x` or `>B1x`.)

(b) There is a tree γ that appears as a subtree of both α and α', but combines to the left in one case and to the right in the other.

Either condition, the proof shows, leads to different "immediate scope" relations in the full trees α and α' (in the sense in which f takes immediate scope over g in $f(g(x))$ but not in $f(h(g(x)))$ or $g(f(x))$). Condition (a) is straightforward. Condition (b) splits into a case where γ serves as a secondary argument inside both α and α', and a case where it is a primary argument in α or α'. The latter case requires consideration of γ's ancestors; the NF properties crucially rule out counterexamples here.

The notion of scope is relevant because semantic interpretations for CCG constituents can be written as restricted lambda terms, in such a way that constituents having distinct terms must have different interpretations in the model (for suitable interpretations of the words, as in §4.1). Theorem 2 is proved by showing that the terms for α and α' differ somewhere, so correspond to different semantic recipes.

Similar theorems for the Lambek calculus were previously shown by (Hepple, 1990; Hendriks, 1993). The present proofs for CCG establish a result that has long been suspected: the spurious ambiguity problem is not actually very widespread in CCG. Theorem 2 says *all* cases of spurious ambiguity can be eliminated through the construction given in theorem 1. But that construction merely ensures a right-branching structure for "forward constituent chains" (such as `A/B B/C C` or `A/B/C C/D D/E/F/G G/H`), and a left-branching structure for backward constituent chains. So these familiar chains are the *only* source of spurious ambiguity in CCG.

5 Extending the Approach to "Restricted" CCG

The "pure" CCG of §4 is a fiction. Real CCG grammars can and do choose a subset of the possible rules.

For instance, to rule out (14), the (crossing) backward rule `N/N N\N` $\rightarrow$ `N/N` must be omitted from English grammar.

(14) $[\text{the}_{\text{NP/N}}\ [[\text{big}_{\text{N/N}}\ [\text{that likes John}]_{\text{N\N}}]_{\text{N/N}}\ \text{galoot}_{\text{N}}]_{\text{N}}]_{\text{NP}}$

If some rules are removed from a "pure" CCG grammar, some parses will become unavailable. Theorem 2 remains true ($\leq$ 1 NF per reading). Whether theorem 1 ($\geq$ 1 NF per reading) remains true depends on what set of rules is removed. For most linguistically reasonable choices, the proof of theorem 1 *will* go through,[5] so that the normal-form parser of §4 remains safe. But imagine removing only the rule `B/C C` $\rightarrow$ `B`: this leaves the string `A/B B/C C` with a left-branching parse that has no (legal) NF equivalent.

In the sort of restricted grammar where theorem 1 does *not* obtain, can we still find one (possibly non-NF) parse per equivalence class? Yes: a different kind of efficient parser can be built for this case.

Since the new parser must be able to generate a non-NF parse when no equivalent NF parse is available, its method of controlling spurious ambiguity cannot be to enforce the constraints (7). The old parser refused to build non-NF constituents; the new parser will refuse to build constituents that are semantically equivalent to already-built constituents.

This idea originates with (Karttunen, 1986). However, we can take advantage of the core result of this paper, theorems 1 and 2, to do Karttunen's redundancy check in $O(1)$ time—no worse than the normal-form parser's check for –FC and –BC tags. (Karttunen's version takes worst-case exponential time for each redundancy check: see footnote §3.)

The insight is that theorems 1 and 2 establish a one-to-one map between semantic equivalence classes and normal forms of the pure (unrestricted) CCG:

(15) Two parses α, α' of the pure CCG are semantically equivalent iff they have the *same* normal form: $NF(\alpha) = NF(\alpha')$.

The *NF* function is defined recursively by §4.2's proof of theorem 1; semantic equivalence is also defined independently of the grammar. So (15) is meaningful and true even if α, α' are produced by a restricted CCG. The tree $NF(\alpha)$ may not be a legal *parse* under the restricted grammar. However, it is still a perfectly good data structure that can be maintained outside the parse chart, to serve as a magnet for α's semantic class. The proof of theorem 1 (see (13)) actually shows how to construct $NF(\alpha)$ in $O(1)$ time from the values of *NF* on smaller constituents. Hence, an appropriate parser can compute and cache the NF of each parse in $O(1)$ time as it is added to the chart. It can detect redundant parses by noting (via an $O(1)$ array lookup) that their NFs have been previously computed.

Figure (1) gives an efficient CKY-style algorithm based on this insight. (Parsing strategies besides CKY would also work, in particular (Vijay-Shanker & Weir, 1993).) The management of cached NFs in steps 9, 12, and especially 16 ensures that duplicate NFs never enter the *oldNFs* array: thus any alternative copy of $\alpha.nf$ has the same array coordinates used for $\alpha.nf$ itself, because it was built from identical subtrees.

The function `PreferableTo`(σ, τ) (step 15) provides flexibility about *which* parse represents its class. `PreferableTo` may be defined at whim to choose the parse discovered first, the more left-branching parse, or the parse with fewer non-standard constituents. Alternatively, `PreferableTo` may call an intonation or discourse module to pick the parse that better reflects the topic-focus division of the sentence. (A variant algorithm ignores `PreferableTo` and constructs one parse forest per reading. Each forest can later be unpacked into individual equivalent parse trees, if desired.)

(Vijay-Shanker & Weir, 1990) also give a method for removing "one well-known source" of spurious ambiguity from restricted CCGs; §4.2 above shows that this is in fact the only source. However, their method relies on the grammaticality of certain intermediate forms, and so can fail if the CCG rules can be *arbitrarily* restricted. In addition, their method is less efficient than the present one: it considers parses in pairs, not singly, and does not remove any parse until the entire parse forest has been built.

6 Extensions to the CCG Formalism

In addition to the **B**n ("generalized composition") rules given in §2, which give CCG power equivalent to TAG, rules based on the **S** ("substitution") and **T** ("type-raising") combinators can be linguistically useful. **S** provides another rule template, used in the analysis of parasitic gaps (Steedman, 1987; Szabolcsi, 1989):

(16) a. `>S`: $x/y\,|_1 z \quad y\,|_1 z \rightarrow x\,|_1 z$
$\Updownarrow \quad \Updownarrow \quad \Updownarrow$
$f \quad g \quad \lambda z.f(z)(g(z))$

b. `<S`: $y\,|_1 z \quad x\backslash y\,|_1 z \rightarrow x\,|_1 z$

Although **S** interacts with **B**n to produce another source of spurious ambiguity, illustrated in (17), the additional ambiguity is not hard to remove. It can be shown that when the restriction (18) is used together with (7), the system again finds exactly one

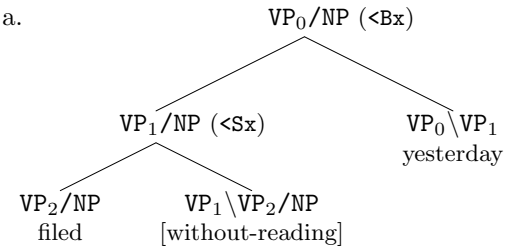
[5] For the proof to work, the rules S and T must be available in the restricted grammar, given that R and Q are. This is usually true: since (7) favors standard constituents and prefers application to composition, most grammars will not block the NF derivation while allowing a non-NF one. (On the other hand, the NF parse of `A/B B/C C/D/E` uses `>B2` twice, while the non-NF parse gets by with `>B2` and `>B1`.)

```
1. for i := 1 to n
2.    C[i − 1, i] := LexCats(word[i])                (* word i stretches from point i − 1 to point i *)
3. for width := 2 to n
4.    for start := 0 to n − width
5.       end := start + width
6.       for mid := start + 1 to end − 1
7.          for each parse tree α = <R, β, γ> that could be formed by combining some
            β ∈ C[start, mid] with some γ ∈ C[mid, end] by a rule R of the (restricted) grammar
8.             α.nf := NF(α)          (* can be computed in constant time using the .nf fields of β, γ, and
                                          other constituents already in C. Subtrees are also NF trees. *)
9.             existingNF := oldNFs[α.nf.rule, α.nf.leftchild.seqno, α.nf.rightchild.seqno]
10.            if undefined(existingNF)                          (* the first parse with this NF *)
11.               α.nf.seqno := (counter := counter + 1)      (* number the new NF & add it to oldNFs *)
12.               oldNFs[α.nf.rule, α.nf.leftchild.seqno, α.nf.rightchild.seqno] := α.nf
13.               add α to C[start, end]
14.               α.nf.currparse := α
15.            elsif PreferableTo(α, existingNF.currparse)          (* replace reigning parse? *)
16.               α.nf := existingNF                    (* use cached copy of NF, not new one *)
17.               remove α.nf.currparse from C[start, end]
18.               add α to C[start, end]
19.               α.nf.currparse := α
20. return(all parses from C[0, n] having root category S)
```

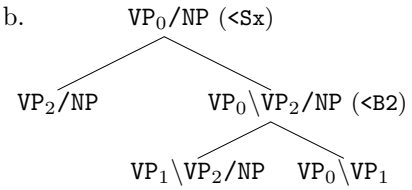
Figure 1: Canonicalizing CCG parser that handles arbitrary restrictions on the rule set. (In practice, a simpler normal-form parser will suffice for most grammars.)

parse from every equivalence class.

(17) a.



b.



(18) a. No constituent produced by >Bn, any $n \geq 2$, ever serves as the primary (left) argument to >S.

b. No constituent produced by <Bn, any $n \geq 2$, ever serves as the primary (right) argument to <S.

Type-raising presents a greater problem. Various new spurious ambiguities arise if it is permitted freely in the grammar. In principle one could proceed without grammatical type-raising: (Dowty, 1988; Steedman, 1991) have argued on linguistic grounds that type-raising should be treated as a mere lexical redundancy property. That is, whenever the lexicon contains an entry of a certain category X, with semantics x, it also contains one with (say) category T/(T\X) and interpretation $\lambda p.p(x)$. As one might expect, this move only sweeps the problem under the rug. If type-raising is lexical, then the definitions of this paper do not recognize (19) as a spurious ambiguity, because the two parses are now, technically speaking, analyses of different sentences. Nor do they recognize the redundancy in (20), because—just as for the example "softly knock twice" in §4.1—it is contingent on a kind of lexical coincidence, namely that a type-raised subject commutes with a (generically) type-raised object. Such ambiguities are left to future work.

(19) $[\text{John}_{\text{NP}}\ \text{left}_{\text{S}\backslash\text{NP}}]_{\text{S}}$ vs. $[\text{John}_{\text{S}/(\text{S}\backslash\text{NP})}\ \text{left}_{\text{S}\backslash\text{NP}}]_{\text{S}}$

(20) $[\text{S}/(\text{S}\backslash\text{NP}_{\text{S}})\ [\text{S}\backslash\text{NP}_{\text{S}}/\text{NP}_{\text{O}}/\text{NP}_{\text{I}}\ \mathbf{T}\backslash(\mathbf{T}/\text{NP}_{\text{O}})]]_{\text{S}/\text{S}_{\text{I}}}$
vs. $[\text{S}/(\text{S}\backslash\text{NP}_{\text{S}})\ \text{S}\backslash\text{NP}_{\text{S}}/\text{NP}_{\text{O}}/\text{NP}_{\text{I}}]\ \mathbf{T}\backslash(\mathbf{T}/\text{NP}_{\text{O}})]_{\text{S}/\text{S}_{\text{I}}}$

7 Conclusions

The main contribution of this work has been formal: to establish a normal form for parses of "pure" Combinatory Categorial Grammar. Given a sentence, every reading that is available to the grammar has exactly one normal-form parse, no matter how many parses it has *in toto*.

A result worth remembering is that, although TAG-equivalent CCG allows free interaction among forward, backward, and crossed composition rules of any degree, two simple constraints serve to eliminate all spurious ambiguity. It turns out that all spurious ambiguity arises from associative "chains" such as A/B B/C C or A/B/C C/D D/E\F/G G/H. (Wit-

tenburg, 1987; Hepple & Morrill, 1989) anticipate this result, at least for some fragments of CCG, but leave the proof to future work.

These normal-form results for pure CCG lead directly to useful parsers for real, restricted CCG grammars. Two parsing algorithms have been presented for practical use. One algorithm finds only normal forms; this simply and safely eliminates spurious ambiguity under *most* real CCG grammars. The other, more complex algorithm solves the spurious ambiguity problem for *any* CCG grammar, by using normal forms as an efficient tool for grouping semantically equivalent parses. Both algorithms are safe, complete, and efficient.

In closing, it should be repeated that the results provided are for the TAG-equivalent $\mathbf{B}n$ (generalized composition) formalism of (Joshi et al., 1991), optionally extended with the **S** (substitution) rules of (Szabolcsi, 1989). The technique eliminates all spurious ambiguities resulting from the interaction of these rules. Future work should continue by eliminating the spurious ambiguities that arise from grammatical or lexical type-raising.

References

Gosse Bouma. 1989. Efficient processing of flexible categorial grammar. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 19–26, University of Manchester, April.

David Dowty. 1988. Type raising, functional composition, and non-constituent conjunction. In R. Oehrle, E. Bach and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures.* Reidel.

Mark Hepple. 1987. Methods for parsing combinatory categorial grammar and the spurious ambiguity problem. Unpublished M.Sc. thesis, Centre for Cognitive Science, University of Edinburgh.

Mark Hepple. 1990. *The Grammar and Processing of Order and Dependency: A Categorial Approach.* Ph.D. thesis, University of Edinburgh.

Mark Hepple and Glyn Morrill. 1989. Parsing and derivational equivalence. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 10–18, University of Manchester, April.

Herman Hendriks. 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics.* Ph.D. thesis, Institute for Logic, Language, and Computation, University of Amsterdam.

Aravind Joshi, K. Vijay-Shanker, and David Weir. 1991. The convergence of mildly context-sensitive grammar formalisms. In *Foundational Issues in Natural Language Processing*, MIT Press.

Lauri Karttunen. 1986. Radical lexicalism. Report No. CSLI-86-68, CSLI, Stanford University.

E. König. 1989. Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver.

J. Lambek. 1958. The mathematics of sentence structure. *American Mathematical Monthly* 65:154–169.

Michael Moortgat. 1990. Unambiguous proof representations for the Lambek Calculus. In *Proceedings of the Seventh Amsterdam Colloquium.*

Michael Niv. 1994. A psycholinguistically motivated parser for CCG. In *Proceedings of the 32nd Annual Meeting of the ACL*, Las Cruces, NM, June. cmp-lg/9406031.

Remo Pareschi and Mark Steedman. 1987. A lazy way to chart parse with combinatory grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, July.

Scott Prevost and Mark Steedman. 1994. Specifying intonation from context for speech synthesis. *Speech Communication*, 15:139-153. cmp-lg/9407015.

Mark Steedman. 1990. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207–264.

Mark Steedman. 1991. Structure and intonation. *Language*, 67:260–296.

Mark Steedman. 1987. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5:403–439.

Anna Szabolcsi. 1989. Bound variables in syntax: Are there any? In R. Bartsch, J. van Benthem, and P. van Emde Boas (eds.), *Semantics and Contextual Expression*, 295–318. Foris, Dordrecht.

K. Vijay-Shanker and David Weir. 1990. Polynomial time parsing of combinatory categorial grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics.*

K. Vijay-Shanker and David Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

K. Vijay-Shanker and David Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.

Kent Wittenburg. 1986. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism.* Ph.D. thesis, University of Texas.

Kent Wittenburg. 1987. Predictive combinators: A method for efficient parsing of Combinatory Categorial Grammars. In *Proceedings of the 25th Annual Meeting of the ACL*, Stanford Univ., July.