Adapting Chart Realization to CCG

Michael White and Jason Baldridge
School of Informatics
University of Edinburgh
{mwhite,jmb}@inf.ed.ac.uk

Abstract

We describe a bottom-up chart realization algorithm adapted for use with Combinatory Categorial Grammar (CCG), and show how it can be used to efficiently realize a wide range of coordination phenomena, including argument cluster coordination and gapping. The algorithm has been implemented as an extension to the OpenNLP open source CCG parser. As an avenue for future exploration, we also suggest how the realizer could be used to simplify the treatment of aggregation in conjunction with higher level content planning components.

1 Introduction

In this paper, we describe our initial efforts to develop a practical, open source realizer for Combinatory Categorial Grammar (CCG, Steedman (2000b)). While CCG provides theoretically attractive accounts of numerous linguistic phenomena — including unique accounts of coordination and intonation, which is of particular interest to builders of dialog systems[1] — its adoption by the NLG community has been hindered by the lack of a practical realizer. As a first step towards making such a realizer available, we have implemented a bottom-up chart realization algorithm (Carroll et al., 1999) adapted for use with CCG. The implementation builds upon the Java-based OpenNLP CCG parser[2] described in Baldridge (2002).

The paper is organized as follows. We provide the rationale for our algorithm choice in §2. In §3 and §4, we provide background for the realization algorithm and the algorithm itself. In §5, we show how the realizer handles a wide range of coordination phenomena. In §6, we provide initial evidence that the realizer can be reasonably efficient in practice. In §7, we discuss related work and conclude with a discussion of future directions.

2 Rationale

Since our chart realization algorithm may not be the most efficient algorithm one might consider implementing, we provide the following rationale for our choice:

Completeness The simple nature of our algorithm makes it relatively straightforward to achieve completeness;[3] in contrast, it would be more difficult to do so for all combinatory rules with Hoffman's (1995) adaptation of semantic head driven realization for CCG.

Parser reuse Since the algorithm is entirely bottom-up, it can directly reuse the parsing-oriented optimizations of the CCG rules described in Baldridge (2002).

[1] We are primarily targeting the realizer for use in dialog systems, and intend to use it in the IST project COMIC (COnversational Multimodal Interaction with Computers), http://www.mpi.nl/comic/.

[2] http://opennlp.sourceforge.net/

[3] That is, to ensure that all derivations licensed by the grammar can be reversed.

LF order independence The algorithm does not rely on the order of conjuncts in the input logical form, and thus handles this oft-discussed aspect of the logical form equivalence problem (Shieber, 1993).

Anytime search The use of an agenda makes it easy to control the search for possible realizations, and thus to run the algorithm in anytime mode.[4]

3 Background

3.1 Combinatory Categorial Grammar

We provide here a brief overview of CCG; see Steedman (2000b) for an extensive introduction.

A given CCG grammar is defined almost entirely in terms of the entries of the lexicon, which are (possibly complex) categories bearing standard feature information (such as tense, agreement, etc.) and subcategorization information. Some (simplified) lexical entries are given below:

(1) a. *man* $\vdash$ n
b. *that* $\vdash$ (n\n)/($s_{uform=fin}$/np)
c. *Bob* $\vdash$ np
d. *saw* $\vdash$ ($s_{tense=past,uform=fin}$\np)/np

CCG has a small set of rules which can be used to combine categories in derivations. The two most basic rules are forward (>) and backward (<) function application:

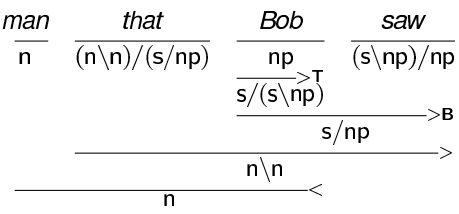
(>) X/Y Y ⇒ X
(<) Y X\Y ⇒ X

CCG also employs further rules based on the composition (**B**), type-raising (**T**), and substitution (**S**) combinators of combinatory logic. Each combinator gives rise to several directionally-distinct rules; for example, there are forward and backward rules for both composition and type-raising:

(>**B**) X/Y Y/Z ⇒ X/Z
(<**B**) Y\Z X\Y ⇒ X\Z
(>**T**) X ⇒ Y/(Y\X)
(<**T**) X ⇒ Y\(Y/X)

These rules are crucial for building the "non-standard" constituents for which CCG is well-known, and which are essential for CCG's handling of coordination, extraction, intonation, and other phenomena. For example, CCG's rules and the categories given in (1) lead to the following derivation of the relative clause *man that Bob saw*:

(2)



The OpenNLP CCG system uses a multi-modal version of CCG (Baldridge, 2002; Baldridge and Kruijff, 2003), which has a fully universal rule component that makes it possible to write more efficient unification schemes for rule application than for the original CCG framework.

3.2 Hybrid Logic Dependency Semantics

Like other compositional grammatical frameworks, CCG allows logical forms to be built in parallel with the derivational process. Traditionally, the λ-calculus has been used to express semantic interpretations, but work in other frameworks has moved to using more flexible representations in computational implementations, such as the MRS framework (Copestake et al., 2001) used for HPSG. In the context of categorial grammar, Kruijff (2001) proposes a framework that utilizes hybrid logic (Blackburn, 2000) to realize a dependency-based perspective on meaning. Baldridge and Kruijff (2002) show how this framework, Hybrid Logic Dependency Semantics (HLDS), relates closely to MRS, and show how terms of HLDS can be built compositionally with CCG via unification. In the next section, we show how HLDS's flexibility enables an approach to semantic construction that ensures semantic monotonicity, simplifies equality tests, and avoids copying in coordinate constructions.

Hybrid logic provides a language for representing relational structures that overcomes standard modal logic's inability to directly reference states in a model. It does so by using *nominals*, a new sort of basic formula with which we can explicitly name states. In addition to propositions, nominals are first-class citizens of the object language: formulas can be formed using propositions, nominals, standard boolean operators, and the *satisfaction operator* "@". A formula $@_i(p \wedge \langle \mathrm{F} \rangle (j \wedge q))$

[4] That is, to allow the client program to request the best solution found so far at any time. We are currently exploring strategies for ranking partial solutions based on n-gram measures (Varges, 2001).

indicates that the formulas p and $\langle \mathrm{F} \rangle (j \wedge q)$ hold at the state named by i and that the state j is reachable via the modal relation F.

In HLDS, hybrid logic is used as a language for describing discourse representation structures — which have their own underlying semantics — as follows. Each semantic head is associated with a nominal that identifies its *discourse referent*, and heads are connected to their dependents via dependency relations, which are modeled as modal relations. As an example, the sentence *Bob saw Gil* receives the represention in (3).

(3) $@_e(\textbf{see} \wedge \langle \textsc{tense} \rangle \textbf{past} \wedge \langle \textsc{act} \rangle (b \wedge \textbf{Bob}) \wedge \langle \textsc{pat} \rangle (g \wedge \textbf{Gil}))$

In this example, e is a nominal that labels the predications and relations for the head **see**, and b and g label those for the the **Bob** and **Gil**, respectively. The relations ACT and PAT represent the dependency roles **Actor** and **Patient**, respectively.[5]

By using the @ operator, hierarchical terms such as (3) can be flattened to an equivalent conjunction of fixed-size *elementary predications* (EPs), closely related to MRS terms:

(4) $@_e\textbf{see} \wedge @_e\langle \textsc{tense} \rangle \textbf{past} \wedge @_e\langle \textsc{act} \rangle b \wedge @_e\langle \textsc{pat} \rangle g \wedge @_b\textbf{Bob} \wedge @_g\textbf{Gil}$

As (4) shows, EPs come in three varieties: lexical predications, (e.g. $@_e\textbf{see}$); semantic features (e.g. $@_e\langle \textsc{tense} \rangle \textbf{past}$); and relations, (e.g. $@_e\langle \textsc{act} \rangle b$).

3.3 Semantic Construction

To facilitate realization from HLDS terms, we have slightly changed Baldridge and Kruijff's (2002) approach to semantic construction to one which uses maximally flat representations such as (4). In our revised approach, EPs are paired with syntactic categories in the lexicon to form signs, as shown in (5)–(7) below. Each atomic category has an index feature which makes a nominal available for capturing syntactically induced dependencies; these indices are shown as subscripts on the category labels.

(5) *saw* $\vdash (\mathsf{s}_e \backslash \mathsf{np}_x)/\mathsf{np}_y$: $@_e\textbf{see} \wedge @_e\langle \textsc{tense} \rangle \textbf{past} \wedge @_e\langle \textsc{act} \rangle x \wedge @_e\langle \textsc{pat} \rangle y$

(6) *Bob* $\vdash \mathsf{np}_b$: $@_b\textbf{Bob}$

(7) *Gil* $\vdash \mathsf{np}_g$: $@_g\textbf{Gil}$

[5]To refrain from committing to a particular set of dependency roles, relations such as ARG1, ARG2, etc. can be used.

In derivations, applications of the combinatory rules coindex the appropriate nominals via unification on the categories, and the EPs are then conjoined to form the resulting interpretation. For example, (6) can type-raise and compose with (5) to yield (8), where x has been coindexed with b, and where the EPs have been conjoined; (8) can then apply to (7) to yield (9), which has the same conjunction of predications as (4).[6]

(8) *Bob saw* $\vdash \mathsf{s}_e/\mathsf{np}_y$: $@_e\textbf{see} \wedge @_e\langle \textsc{tense} \rangle \textbf{past} \wedge @_e\langle \textsc{act} \rangle b \wedge @_e\langle \textsc{pat} \rangle y \wedge @_b\textbf{Bob}$

(9) *Bob saw Gil* $\vdash \mathsf{s}_e$: $@_e\textbf{see} \wedge @_e\langle \textsc{tense} \rangle \textbf{past} \wedge @_e\langle \textsc{act} \rangle b \wedge @_e\langle \textsc{pat} \rangle g \wedge @_b\textbf{Bob} \wedge @_g\textbf{Gil}$

Since the EPs are always conjoined by the combinatory rules, semantic construction is guaranteed to be *monotonic* — in the sense that no semantic information can be dropped during the course of a derivation — which is an essential property for ensuring that the realization algorithm is complete (Copestake et al., 2001).

Another benefit of this approach to semantic construction is that it becomes easier to perform equality tests on signs, since the flat conjunctions of EPs can be sorted into a canonical order and compared in turn. Such equality tests can be used to avoid adding duplicate entries into the chart when there are multiple equivalent derivations for a given sign, thereby alleviating the problem of so-called "spurious" ambiguity (Steedman, 2000b).

A final benefit of simply conjoining EPs in derivations is that it avoids any copying of predications in coordinate constructions. In contrast, the approach implicit in Baldridge and Kruijff (2002) yields duplicate predications in examples such as *Bob heard and Ted saw Gil*, where the proposition **Gil** appears twice (ignoring tense):

(10) $@_{e_1}(\textbf{hear} \wedge \langle \textsc{act} \rangle (b \wedge \textbf{Bob}) \wedge \langle \textsc{pat} \rangle (g \wedge \textbf{Gil}) \wedge \langle \textsc{coord} \rangle (e_2 \wedge \textbf{see} \wedge \langle \textsc{act} \rangle (t \wedge \textbf{Ted}) \wedge \langle \textsc{pat} \rangle (g \wedge \textbf{Gil})))$

As we will show in §5, by avoiding such duplicate predications, the present approach to semantic construction keeps the output of the parser in line with the expected input of the realizer.

[6]There is another more traditional (but less incremental) derivation for *Bob saw Gil*, where *saw* combines first with the object *Gil* before combining with the subject *Bob*.

4 The Algorithm

4.1 Data Structures

The input to the algorithm is a logical form encoded as an HLDS term. The input term is flattened to a list of EPs, so that the extent to which partial realizations cover the input LF can be tracked positionally. For example, to realize *man that Bob saw*, the hierarchically structured input in (11) is flattened to (12):

(11) $@_x(\mathbf{man} \wedge \langle \text{GENREL} \rangle (e \wedge \mathbf{see} \wedge \langle \text{TENSE} \rangle \mathbf{past} \wedge \langle \text{ACT} \rangle (b \wedge \mathbf{Bob}) \wedge \langle \text{PAT} \rangle x))$

(12) $0 : @_x\mathbf{man},\ 1 : @_x\langle \text{GENREL} \rangle e,\ 2 : @_e\mathbf{see}$
$3 : @_e\langle \text{TENSE} \rangle \mathbf{past},\ 4 : @_e\langle \text{ACT} \rangle b$
$5 : @_e\langle \text{PAT} \rangle x,\ 6 : @_b\mathbf{Bob}$

The algorithm makes use of three principal data structures: edges, an agenda and a chart. An *edge* is just a CCG sign plus a couple of bit vectors which record the sign's coverage of the input LF and the sign's indices (nominals) that are syntactically available. These bit vectors make it possible to instantly check whether two edges cover disjoint parts of the input LF and whether they have any indices in common. For example, the edges for the finite past and non-finite forms of *see* are given below, with the bit vectors for the EPs and indices shown in braces:

(13) $\{2,3,4,5\}\ \{e,b,x\}$
saw $\vdash$ $(\mathsf{s}_{e,fin} \backslash \mathsf{np}_b)/\mathsf{np}_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{TENSE} \rangle \mathbf{past} \wedge @_e\langle \text{ACT} \rangle b \wedge @_e\langle \text{PAT} \rangle x$

(14) $\{2,4,5\}\ \{e,b,x\}$
see $\vdash$ $(\mathsf{s}_{e,nonfin} \backslash \mathsf{np}_b)/\mathsf{np}_x$:
$@_e\mathbf{see} \wedge @_e\langle \text{ACT} \rangle b \wedge @_e\langle \text{PAT} \rangle x$

The *agenda* is a priority queue of edges which manages the edges that have yet to be added to the chart. Using the agenda makes it easy to vary the search order by changing the edge sorting strategy.

The *chart* is a collection of edges that enables a dynamic programming search for realizations. Whereas a chart for parsing uses string positions to track partial parses, one for realization uses an edge's coverage vector to track partial realizations.

4.2 Lexical Lookup

In the first phase of the algorithm, for each EP in the flattened input LF, relevant lexical entries are accessed according to the following indexing scheme. Most lexical items are indexed by the principal lexical predicate which they introduce. However, if a lexical item (e.g. a relative pronoun) only introduces a dependency relation or a semantic feature, it is indexed by the relation or feature. Semantically null lexical items, i.e. ones which introduce no EPs (e.g. infinitival *to*), are not indexed at all; instead, they receive special handling in the combinatory rule phase (see step 4 in figure 1). Case marking prepositions and particles are only considered when there is a matching feature on one of the indexed lexical items indicating that they may be needed.

Once a lexical entry indexed by the current EP has been accessed, instantiation is attempted. During instantiation, the current EP is unified first, and then unification of the remaining EPs in the lexical entry is attempted against the remaining EPs in the input LF. The lexical entry is allowed to introduce extra semantic features, enabling some limited underspecification in the input LF.

For example, the predicational EP $@_e\mathbf{see}$ triggers the lookup of the edges shown in (13) and (14). Note that the present tense form *sees* is accessed as well, but instantiation fails due to its incompatible $\langle \text{TENSE} \rangle$ value (whereas the non-finite form *see* has no $\langle \text{TENSE} \rangle$ value). The relational EP $@_x\langle \text{GENREL} \rangle e$ triggers the lookup and instantiation of the two edges for the relative pronoun shown in (15) and (16) below. Similarly, the featural EP $@_e\langle \text{TENSE} \rangle \mathbf{past}$ triggers the introduction of the auxiliary *did*.

(15) $\{1\}\ \{e,x\}$
that $\vdash$ $(\mathsf{n}_x \backslash \mathsf{n}_x)/(\mathsf{s}_{e,fin} \backslash \mathsf{np}_x) : @_x\langle \text{GENREL} \rangle e$

(16) $\{1\}\ \{e,x\}$
that $\vdash$ $(\mathsf{n}_x \backslash \mathsf{n}_x)/(\mathsf{s}_{e,fin} / \mathsf{np}_x) : @_x\langle \text{GENREL} \rangle e$

4.3 Combinatory Rules

In the second, main phase of the algorithm — at a high level — edges are successively moved from the agenda to the chart and combined with the edges already on the chart, with any resulting new edges added to the agenda, until no more combinations are possible and the agenda becomes empty. Figure 1 describes the main loop in more detail.

Continuing our example, some of the edges generated during the combinatory rule phase are shown in (17)–(21) below, without the bit vectors. The edge for *Bob* is type-raised, yielding (17), and

Until the agenda is empty:

1. Remove the first edge from the agenda and set it to be the current edge. If the chart contains an already derived equivalent edge, skip the rest of the loop.
2. Combine the current edge with the edges already on the chart. More specifically, for each chart edge:
   (a) Check the coverage bit vectors for the current edge and the chart edge for intersection. If they overlap, skip the chart edge.
   (b) Check the index bit vectors for intersection. If they do not overlap, only combine the current edge with the chart edge if the input LF contains an appropriate ⟨PAIREDWITH⟩ relation (cf. §5 for discussion).
   (c) Combine the current edge with the chart edge using all available binary combinatory rules, and add any resulting new edges to the agenda.
3. Apply all unary combinatory rules to the current edge, adding any resulting new edges to the agenda.
4. Combine the current edge with edges for all semantically null lexical items, as if these were chart edges.
5. Add the current edge to the chart.

Figure 1: Main loop

the edge for *see* (14) combines with the semantically null infinitival *to*, yielding (18); (17) then forward composes with both *saw* (13) and *to see* (18), yielding (19) and (20). Since *Bob to see* (20) is marked syntactically as infinitival rather than finite, the relative pronoun edge (16) will only combine (via forward application) with *Bob saw* (19), before combining (via backward application) with *man* to yield the complete edge in (21).

(17) *Bob* $\vdash \mathsf{s}_{\mathit{1}}/(\mathsf{s}_{\mathit{1}}\backslash\mathsf{np}_b) : @_b\mathbf{Bob}$

(18) *to see* $\vdash (\mathsf{s}_{e,\mathit{inf}}\backslash\mathsf{np}_b)/\mathsf{np}_x :$
$@_e\mathbf{see} \wedge @_e\langle\textsc{Act}\rangle b \wedge @_e\langle\textsc{Pat}\rangle x$

(19) *Bob saw* $\vdash \mathsf{s}_{e,\mathit{fin}}/\mathsf{np}_x :$
$@_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\mathbf{past}$
$\wedge @_e\langle\textsc{Act}\rangle b \wedge @_e\langle\textsc{Pat}\rangle x \wedge @_b\mathbf{Bob}$

(20) *Bob to see* $\vdash \mathsf{s}_{e,\mathit{inf}}/\mathsf{np}_x :$
$@_e\mathbf{see} \wedge @_e\langle\textsc{Act}\rangle b \wedge @_e\langle\textsc{Pat}\rangle x \wedge @_b\mathbf{Bob}$

(21) *man that Bob saw* $\vdash \mathsf{n}_x :$
$@_x\mathbf{man} \wedge @_x\langle\textsc{GenRel}\rangle e$
$\wedge @_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\mathbf{past}$
$\wedge @_e\langle\textsc{Act}\rangle b \wedge @_e\langle\textsc{Pat}\rangle x \wedge @_b\mathbf{Bob}$

5 Coordination

5.1 Sentential Coordination

CCG's flexible approach to constituency delivers derivations for a wide variety of coordinate structures, often involving the coordination of such "non-standard" constituents as s/np, as in the following right node raising example:

(22) [Bob saw]$_{\mathsf{s/np}}$ and [Ted heard]$_{\mathsf{s/np}}$ Gil.

Examples like (22) can be handled using the category for *and* given in (23), where s$ schematizes over functions into s:[7]

(23) *and* $\vdash (\mathsf{s}_e\$1\backslash\mathsf{s}_{e_1}\$1)/\mathsf{s}_{e_2}\$1 :$
$@_e\mathbf{and} \wedge @_e\langle\textsc{List}\rangle e_1 \wedge @_{e_1}\langle\textsc{Coord}\rangle e_2$

Category (23) enables *Bob saw* and *Ted heard* to coordinate as follows:

(24) *Bob saw and Ted heard* $\vdash$
$\mathsf{s}_e/\mathsf{np}_x :$
$@_e\mathbf{and} \wedge @_e\langle\textsc{List}\rangle e_1 \wedge @_{e_1}\langle\textsc{Coord}\rangle e_2$
$\wedge @_{e_1}\mathbf{see} \wedge \ldots @_{e_1}\langle\textsc{Pat}\rangle x$
$\wedge @_{e_2}\mathbf{hear} \wedge \ldots @_{e_2}\langle\textsc{Pat}\rangle x$

Category (24) can then be combined with *Gil* to yield a flat conjunction of HLDS terms equivalent to the one below (ignoring tense), which has been collapsed into hierarchical form for readability:

(25) $@_e(\mathbf{and}$
$\wedge\langle\textsc{List}\rangle(e_1 \wedge \mathbf{see} \wedge \langle\textsc{Act}\rangle(m \wedge \mathbf{Bob}) \wedge \langle\textsc{Pat}\rangle g$
$\wedge \langle\textsc{Coord}\rangle(e_2 \wedge \mathbf{hear}$
$\wedge \langle\textsc{Act}\rangle(t \wedge \mathbf{Ted}) \wedge \langle\textsc{Pat}\rangle g)))$
$\wedge @_g\mathbf{Gil}$

Since the present approach to semantic construction does not produce duplicate EPs for *Gil*, the output of the CCG parser for (22) shown in (25) can be directly reversed by the realizer. In contrast, the duplicate EPs seen in (10) (cf. §3.3) would cause problems for the realizer's tracking of input LF coverage. Indeed, the LF in (10) is perhaps more similar to the one for the clause-level coordination in (26) below than it is to (25):[8]

(26) *Bob heard Gil and Ted saw Gil* $\vdash$
$\mathsf{s} : @_e(\mathbf{and} \wedge \langle\textsc{List}\rangle(e_1 \wedge \mathbf{heard} \wedge \langle\textsc{Act}\rangle(b \wedge \mathbf{Bob})$
$\wedge \langle\textsc{Pat}\rangle(g_1 \wedge \mathbf{Gil}) \wedge \langle\textsc{Coord}\rangle(e_2 \wedge \mathbf{see}$
$\wedge \langle\textsc{Act}\rangle(t \wedge \mathbf{Ted}) \wedge \langle\textsc{Pat}\rangle(g_2 \wedge \mathbf{Gil}))))$

[7] The relations ⟨LIST⟩ and ⟨COORD⟩ encode a linked list; ⟨LIST⟩ points to the first item in the list, and ⟨COORD⟩ points from one item to the next.

[8] Note that each use of a lexical item gives rise to a distinct index nominal, similarly to DRT.

The HLDS terms in (25) and (26) show how differences in the realizer's input logical form — which are reminiscent of the differences between reduced and unreduced λ-terms — can be used to control the choice of coordination options made available by the grammar.[9]

5.2 NP Coordination

Of the multiple possible readings involving NP coordination, we will only focus on the distributive reading here. As Moore (1989) points out, NPs such as *Ted and Gil* in (27) below pose a challenge for first-order unification–based approaches to semantic construction, since the index x cannot be unified with the referents for both *Ted* and *Gil*:[10]

(27) [Bob saw]$_{\mathsf{s}_e/\mathsf{np}_x}$ Ted and Gil.

Following (Moore, 1989), we tackle this problem by introducing a λ-binder into the semantic representation for (27), while still eschewing the use of λ's in variable binding:

(28) $@_s(\mathbf{and} \wedge \langle \textsc{List} \rangle (t \wedge \mathbf{Ted} \wedge \langle \textsc{Coord} \rangle (g \wedge \mathbf{Gil})) \wedge \langle \textsc{Pred} \rangle (l \wedge \mathbf{lambda} \wedge \langle \textsc{BoundVar} \rangle x \wedge \langle \textsc{Body} \rangle (e \wedge \mathbf{see} \wedge \langle \textsc{Act} \rangle (b \wedge \mathbf{Bob}) \wedge \langle \textsc{Pat} \rangle x)))$

The HLDS term in (28) is intended to be equivalent to the conjunction of the terms formed by distributing the λ-term across each member of the list. (27) can be parsed and realized with the semantics in (28) using the category (29), which takes the two NPs and forms a type-raised NP:

(29) *and* $\vdash$ $((\mathsf{s}_s\$\backslash(\mathsf{s}_e\$/\mathsf{np}_x))\backslash\mathsf{np}_{x_1})/\mathsf{np}_{x_2}$: $@_s\mathbf{and} \wedge @_s\langle \textsc{List} \rangle x_1 \wedge @_{x_1}\langle \textsc{Coord} \rangle x_2 \wedge @_s\langle \textsc{Pred} \rangle l \wedge @_l\mathbf{lambda} \wedge @_l\langle \textsc{BoundVar} \rangle x \wedge @_l\langle \textsc{Body} \rangle e$

5.3 Argument Clusters and Gapping

The above approach to distributive NP coordination can be extended to handle argument clusters — as in (30) below — without the need to invoke otherwise unnecessary deletion operations.

(30) [Bob gave]$_{(\mathsf{s}_e/\mathsf{np}_y)/\mathsf{np}_x}$ [Ted$_t$ a dog$_d$]$_{\mathsf{s}\backslash(\mathsf{s}/\mathsf{np}_d/\mathsf{np}_t)}$ and [Gil$_g$ a cat$_c$]$_{\mathsf{s}\backslash(\mathsf{s}/\mathsf{np}_c/\mathsf{np}_g)}$

To handle (30), we introduce a $\langle$PAIREDWITH$\rangle$ relation to connect pairs of NP referents and bound variables, in the following category for *and*:

(31) *and* $\vdash$ $((\mathsf{s}_s\$\backslash((\mathsf{s}_e\$/\mathsf{np}_y)/\mathsf{np}_x))\backslash(\mathsf{s}\$\backslash((\mathsf{s}\$/\mathsf{np}_{y_1})/\mathsf{np}_{x_1})))/(\mathsf{s}\$\backslash((\mathsf{s}\$/\mathsf{np}_{y_2})/\mathsf{np}_{x_2}))$: $@_s\mathbf{and} \wedge @_s\langle \textsc{List} \rangle x_1 \wedge @_{x_1}\langle \textsc{PairedWith} \rangle y_1 \wedge @_{x_1}\langle \textsc{Coord} \rangle x_2 \wedge @_{x_2}\langle \textsc{PairedWith} \rangle y_2 \wedge @_s\langle \textsc{Pred} \rangle l \wedge @_l\mathbf{lambda} \wedge @_l\langle \textsc{Body} \rangle e \wedge @_l\langle \textsc{BoundVar} \rangle x \wedge @_x\langle \textsc{PairedWith} \rangle y$

Category (31) enables (30) to be parsed into a semantic representation analogous to (28). The derivation of (30) requires the base NPs *Ted*$_t$ and *a dog*$_d$ to type raise and compose together into the category $\mathsf{s}\backslash(\mathsf{s}/\mathsf{np}_d/\mathsf{np}_t)$, as indicated (and similarly for *Gil*$_g$ and *a cat*$_c$). Reversing this derivation during realization thus requires *Ted*$_t$ and *a dog*$_d$ to combine, even though they have no indices in common. Since removing the index intersection filter from the realization algorithm entirely would let all NPs combine via type-raising and composition in all possible orders, we instead require the indices to be in a $\langle$PAIREDWITH$\rangle$ relation in the input LF in order for the NPs to combine.

To handle gapping examples like (32), a similar category can be supplied for *and*, as shown in (33) without the semantics, which remains unchanged:

(32) Ted$_t$ received$_{(\mathsf{s}_e\backslash\mathsf{np}_x)/\mathsf{np}_y}$ a dog$_d$ and [Gil$_g$ a cat$_c$]$_{\mathsf{s}\backslash((\mathsf{s}\backslash\mathsf{np}_g)/\mathsf{np}_c)}$

(33) *and* $\vdash$ $(((\mathsf{s}_s\backslash\mathsf{np}_{x_1})\backslash((\mathsf{s}_e\backslash\mathsf{np}_x)/\mathsf{np}_y))\backslash\mathsf{np}_{y_1})/(\mathsf{s}\backslash((\mathsf{s}\backslash\mathsf{np}_{x_2})/\mathsf{np}_{y_2}))$

Category (33) combines first with the pair of NPs *Gil a cat* on the right, then successively with the NP *a dog*, the transitive verb *received* and the NP *Ted* on the left. As such, it handles gapping without appealing to reanalysis, as in Steedman (2000b), though at the expense of requiring *and* to coordinate unlike categories, suggesting that (33) should be viewed as a compiled-out version of Steedman's (2000b) approach to gapping.

6 Efficiency

As Moore (2002) notes, it appears that the realization problem is inherently exponential in worst case complexity unless one is willing to rely on

[9] Cf. (Prevost, 1995) for a related use of unreduced λ-terms in the context of representing information structural units.

[10] In the collective reading, also plausible in (27), x can simply be unified with a set-valued referent for *Ted and Gil*; with *Ted or Gil*, in contrast, only the distributive reading is possible.

	First	All
Avg	**0.19**	1.32
Max	0.98	13.0

Table 1: Realizer Timing (in seconds)

	First	All
Avg	0.50	13.3
Max	**3.84**	349

Table 2: Realizer Timing Without Index Filter

the potentially arbitrary order of LF conjuncts. In practice, as Carroll et al. (1999) explain, the main complexity issue is the factorial number of possible word orders that can arise when the grammar leaves modifier order relatively unconstrained. Our current strategy to address this issue is to concentrate on reliably finding good realizations in a reasonably short time span when running the algorithm in anytime mode, rather than worrying about the amount of time it might take on occasion to find all possible realizations. We suggest that this anytime focus is appropriate for practical use in dialog systems.

To test whether our realizer's speed is in the right ballpark for dialog applications, we have measured its performance on a pre-existing set of test phrases — namely all those discussed in Baldridge (2002) — using a small but linguistically rich grammar covering heavy NP shift, non-peripheral extraction, parasitic gaps, particle shift, relativization, right node raising, topicalization, and argument cluster coordination. On this test suite, the performance is reasonably promising, averaging under 200 ms. until the first realization is found, on a Linux PC. Table 1 shows the average and maximum times until the first realization is found and until all realizations are found.

Even with this small test suite, it is clear that the index filter is essential for efficient realization. Table 2 shows the comparable realization times with the index filter turned off. As the table shows, the average time until the first realization more than doubles, and the maximum time until the first realization is nearly four times worse. The expected exponential increase in realization times (cf. §5) can be seen in the times to find all realizations.

To increase performance, there is ample room to make improvements to the unification algorithm. While the index filter reduces the number of unification operations attempted, unification still dominates the realization time. The implementations of the combinatory rules have been optimized as described in Baldridge (2002), but unification is otherwise naive and performs more copying than necessary.

Employing packing and pruning strategies could also improve performance. Currently, there is no structure sharing among edges, and no means to prune low ranked edges from the chart.

7 Conclusions and Future Work

Our approach to chart realization with CCG is most closely related to Carroll et al. (1999), which in turn builds upon much earlier work cited therein, such as Kay (1996). Moore (2002) presents a related algorithm for a broad class of context free grammars.

Compared to Carroll et al. (1999), we have employed a similar but more straightforward approach to semantic construction than described in Copestake et al. (2001), since we do not allow underspecification of the logical scope of quantifiers,[11] and since there is no need for special treatment of external arguments to handle control phenomena in CCG. We also have not tried delaying the insertion of intersective modifiers (Carroll et al., 1999), in part because doing so would complicate the use of n-gram ranking strategies.

The primary novel contribution of our approach is showing how to efficiently realize a wide range of coordination phenomena with CCG. In particular, we have shown how to use an index filter sensitive to paired entities in the input LF in order to handle argument cluster coordination and gapping.

In future work, we plan to take several steps to make the realizer more practical. As already mentioned, we are currently exploring strategies for ranking partial solutions based on n-gram measures, and we plan to improve efficiency via enhancements to our unification algorithm. We are also currently investigating techniques for handling Steedman's (2000a) approach to information

[11] Cf. Steedman (1999) for discussion.

structure and intonation. In addition, we plan to bootstrap a wide coverage grammar for English from the CCG Bank (Hockenmaier and Steedman, 2002), and to develop improved XML grammar management tools.

Beyond these practically-oriented steps, we also plan to investigate new techniques for coupling CCG realization with higher level planning components. A particularly appealing direction is to see whether the present approach to coordination can simplify the treatment of aggregation in higher level planning components used in conjunction with the realizer. Since current bottom-up approaches to aggregation such as Dalianis (1996) and Shaw (1998) combine simple syntactic phrases into more complex ones by looking for patterns of related semantic material, they do not fit naturally into applications where it makes sense to group semantic material during content planning, based on intentions or information structural considerations. In contrast, working with our realizer, content planning components could specify their aggregation decisions via distinctions made at the level of logical form, taking advantage of the realizer's ability to use differences in the input LF to control the choice of coordination options made available by the grammar.

Acknowledgements

We would like to thank Mark Steedman, Geert-Jan Kruijff, Johanna Moore, Jon Oberlander and Mary Ellen Foster for helpful discussions. This work was supported in part by the COMIC (IST-2001-32311) and ROSIE (Edinburgh-Stanford Link R36763) projects.

References

Jason Baldridge and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proc. of 40th Annual Meeting of the Association for Computational Linguistics*, pages 319–326.

Jason Baldridge and Geert-Jan Kruijff. 2003. Multi-Modal Combinatory Categorial Grammar. In *Proc. of 10th Annual Meeting of the European Association for Computational Linguistics*.

Jason Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh.

Patrick Blackburn. 2000. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–625.

John Carroll, Ann Copestake, Dan Flickinger, and Victor Poznański. 1999. An efficient chart generator for (semi-) lexicalist grammars. In *Proc. of the 7th European Workshop on Natural Language Generation*, pages 86–95.

Ann Copestake, Alex Lascarides, and Dan Flickinger. 2001. An algebra for semantic construction in constraint-based grammars. In *Proc. of the 39th Annual Meeting of the Association of Computational Linguistics*, pages 132–139.

Hercules Dalianis. 1996. *Concise Natural Language Generation from Formal Specifications*. Ph.D. thesis, Royal Institute of Technology, Stockholm.

Julia Hockenmaier and Mark Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proc. of the Third International Conference on Language Resources and Evaluation*.

Beryl Hoffman. 1995. *Computational Analysis of the Syntax and Interpretation of 'Free' Word-order in Turkish*. Ph.D. thesis, University of Pennsylvania. IRCS Report 95-17.

Martin Kay. 1996. Chart generation. In *Proc. of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 200–204.

Geert-Jan M. Kruijff. 2001. *A Categorial Modal Architecture of Informativity: Dependency Grammar Logic & Information Structure*. Ph.D. thesis, Charles University.

Robert C. Moore. 1989. Unification-based semantic interpretation. In *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 33–41.

Robert C. Moore. 2002. A complete, efficient sentence-realization algorithm for unification grammar. In *Proc. of the 2nd International Natural Language Generation Conference*.

Scott Prevost. 1995. *A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation*. Ph.D. thesis, University of Pennsylvania. IRCS TR 96-01.

James Shaw. 1998. Clause aggregation using linguistic knowledge. In *Proc. of the Ninth International Workshop on Natural Language Generation*, pages 138–148.

Stuart Shieber. 1993. The problem of logical-form equivalence. *Computational Linguistics*, 19(1):179–190.

Mark Steedman. 1999. Quantifier Scope Alternation in CCG. In *Proc. of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 301–308.

Mark Steedman. 2000a. Information structure and the syntax-phonology interface. *Linguistic Inquiry*, 31(4):649–689.

Mark Steedman. 2000b. *The Syntactic Process*. MIT Press.

Sebastian Varges. 2001. Instance-based natural language generation. In *Proc. of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 1–8.