Research on Language and Computation (2006) 4:39–75
DOI 10.1007/s11168-006-9010-2

Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar

MICHAEL WHITE
School of Informatics, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, UK (*Web: http://www.iccs.informatics.ed.ac.uk/~mwhite/*)

Abstract. We describe a chart realization algorithm for 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 incorporates three novel methods for improving the efficiency of chart realization: (i) using rules to chunk the input logical form into sub-problems to be solved independently prior to further combination; (ii) pruning edges from the chart based on the *n*-gram score of the edge's string, in comparison to other edges with equivalent categories; and (iii) formulating the search as a best-first anytime algorithm, using *n*-gram scores to sort the edges on the agenda. The algorithm has been implemented as an extension to the OpenCCG open source CCG parser, and initial performance tests indicate that the realizer is fast enough for practical use in natural language dialogue systems.

Key words: Combinatory Categorial Grammar, computational semantics, natural language generation, surface realization.

1. Introduction

Chart realization algorithms (Shieber, 1988; Kay, 1996; Shemtov, 1997; Carroll et al., 1999; Moore, 2002) perform the inverse task of chart parsing algorithms: that is, rather than transducing strings to logical forms (LFs), they transduce LFs to strings, a task often called surface (or linguistic or syntactic) realization.[1] Many variations on the chart realization scheme are possible in principle, depending in part on the grammatical framework one adopts. In this paper, we describe a chart realization algorithm for Steedman's Combinatory Categorial Grammar (CCG; Steedman, 1999, 2000a, b). Since CCG is particularly known for its unique account of coordination, as a case study, we show how the algorithm can be used to efficiently realize a wide range of coordination phenomena, including argument cluster coordination and gapping. Along the way, we also show how the algorithm – together with the approach to semantic construction we adopt – makes it possible to use differences in the input LF to control the choice of coordination options made available by the grammar.

The algorithm incorporates three novel methods for improving the efficiency of chart realization. The first method makes use of a small set of rules, written by the grammar author, for chunking input LFs into sub-problems to be solved independently prior to further combination. This technique addresses the problem, noted by Kay (1996), that chart realization algorithms can waste a great deal of time on generation paths containing semantically incomplete phrases. While the method is not fully automatic, as it requires the insights of the grammar author, it is more flexible than Kay's approach, and also extends to cases not considered by Kay; in particular, it can help to more efficiently realize coordinated constituents, including non-standard ones.

The second and third methods involve integrating *n*-gram scoring of possible realizations into the chart realization algorithm, as proposed by Varges and Mellish (2001), rather than ranking all complete realizations by their *n*-gram score as a post-process, as in e.g. (Langkilde and Knight, 1998; Langkilde-Geary, 2002). Specifically, the second method involves pruning edges from the chart based on the *n*-gram score of the edge's string – in comparison to other edges with equivalent categories – while the third method involves formulating the search as a best-first anytime algorithm, using *n*-gram scores to sort the edges on the agenda. These two easy-to-implement methods partially address the problem that the grammar may license an exponential number of possible realizations for a given input, especially with modifiers whose linear order is left relatively unconstrained (Kay, 1996; Carroll et al., 1999). They do so by helping to ensure that a good realization can be found quickly, even when it would take a long time to find the best realization or all possible realizations. We suggest that this emphasis is appropriate for the needs of natural language dialogue systems, where response times must be kept short in order to achieve sufficient interactivity.

Although the three efficiency methods are particularly well suited to CCG, they are potentially applicable to other grammatical frameworks as well. While the first method depends in part on the details of our approach to semantic construction, the second method just requires a means of identifying edges with equivalent derivational potential, and the third method should be directly applicable to other grammatical frameworks.

The algorithm has been implemented as an extension to the OpenCCG[2] open source CCG parser, and takes advantage of the multi-modal extensions to CCG developed by Baldridge (2002) and Baldridge and Kruijff (2003), as well as their dependency-based approach to representing linguistic meaning, Hybrid Logic Dependency Semantics (HLDS; Kruijff, 2001, 2003; Baldridge and Kruijff, 2002). Initial performance tests indicate that the realizer is fast enough for practical use in natural language dialogue systems. To date, the OpenCCG realizer has been deployed in two

prototype dialogue systems (den Os and Boves, 2003; Moore et al., 2004), where realization times have been satisfactory.

The rest of the paper is organized as follows. In section 2, we provide background on CCG and HLDS. In section 3, we present our approach to semantic construction and discuss how it facilitates realization. In section 4, we present the algorithm itself, including its novel methods. In section 5, we show how the realizer handles a wide range of coordination phenomena, then in section 6, we clarify the semantics of the HLDS representations, via a translation to the discourse representations structures of Discourse Representation Theory (DRT; Kamp and Reyle, 1993). In section 7, we describe our initial performance tests. Finally, in section 8, we discuss related work and conclude with a discussion of future directions.

2. Background

2.1. Combinatory categorial grammar

The CCG is a unification-based categorial framework that is both linguistically and computationally attractive. 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 verb form, agreement, etc.) and sub-categorization information. Some simplified lexical entries are given below:

(1) a. $a \vdash \mathsf{np/n}$
b. $\mathit{musician} \vdash \mathsf{n}$
c. $\mathit{that} \vdash \mathsf{(n\backslash n)/(s}_{vform=fin}\mathsf{|np)}$
d. $\mathit{Bob} \vdash \mathsf{np}$
e. $\mathit{saw} \vdash \mathsf{s}_{vform=fin}\mathsf{\backslash np/np}$

The slashes indicate the direction in which an argument category is sought in the string: the forward slash (/) specifies an argument appearing to the right; the backward slash (\) specifies an argument appearing to the left; and the vertical slash (|) specifies an argument appearing in either direction. Note that the argument category always appears on the right-hand side of the slash, with the result category on the left-hand side of the slash. A convention of left associativity is assumed, so that a category such as $\mathsf{s\backslash np/np}$ is equivalent to $\mathsf{(s\backslash np)/np}$.

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:

(2) $(>) \quad \mathsf{X/Y} \quad \mathsf{Y} \quad \Rightarrow \quad \mathsf{X}$
$(<) \quad \mathsf{Y} \quad \mathsf{X\backslash Y} \quad \Rightarrow \quad \mathsf{X}$

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:[3]

(3)
$$\begin{array}{llllll} (>\mathbf{B}) & X/Y & Y/Z & \Rightarrow & X/Z \\ (<\mathbf{B}) & Y\backslash Z & X\backslash Y & \Rightarrow & X\backslash Z \\ (>\mathbf{T}) & X & & \Rightarrow & Y/(Y\backslash X) \\ (<\mathbf{T}) & X & & \Rightarrow & Y\backslash(Y/X) \end{array}$$

These rules are crucial for building the "non-standard" constituents that are the hallmark of categorial grammars, 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 *that Bob saw*, as a post-modifier of *musician*:

(4)
$$\dfrac{\dfrac{\textit{a}}{\mathsf{np/n}} \quad \dfrac{\textit{musician}}{\mathsf{n}} \quad \dfrac{\textit{that}}{\mathsf{(n\backslash n)/(s|np)}} \quad \dfrac{\dfrac{\dfrac{\textit{Bob}}{\mathsf{np}}}{\mathsf{s/(s\backslash np)}}\,{>}\mathbf{T}}{\mathsf{s/np}}\,{>}\mathbf{B}}{\mathsf{n\backslash n}}\,{>}}{\mathsf{n}}\,{<}}{\mathsf{np}}\,{>}$$

As the derivation in (4) shows, type raising and composition allow substrings like *Bob saw* to be analyzed as full-fledged constituents, with category s/np. They can therefore undergo coordination, enabling a movement- and deletion-free account of right node raising:

(5)
$$\dfrac{\dfrac{\textit{Ted adores}}{\mathsf{s/np}} \quad \dfrac{\dfrac{\textit{but}}{\mathsf{s\$\backslash s\$/s\$}} \quad \dfrac{\textit{Gil detests}}{\mathsf{s/np}}}{\mathsf{(s/np)\backslash(s/np)}}\,{>}}{\mathsf{s/np}}\,{<} \quad \dfrac{\textit{a musician that Bob saw}}{\mathsf{np}}}{\mathsf{s}}\,{>}$$

The derivation in (5) relies on the category s$\s$/s$ for the conjunction *but*, where s$ unifies with either s or any function category into s. In section 5, we will see how this category for conjunctions enables the realizer to handle a variety of clausal coordination phenomena.

In addition to the combinatory rules like those in (3), a CCG grammar may also contain a handful of unary type changing rules. For example, a unary type changing rule similar to the category for *that* in (1) can be used for reduced relatives, as in *a musician Bob saw*:

(6) $\mathsf{s}_{vform=fin}|\mathsf{np} \Rightarrow \mathsf{n}\backslash\mathsf{n}$

As Hockenmaier and Steedman (2002) suggest, unary rules such as (6) can be thought of as corresponding to zero morphemes in the lexicon. In particular, note that unary type changing rules – unlike the combinatory rules in (2) and (3) – may introduce their own semantics, in which case the realizer must track their semantic contribution, as with ordinary lexical items.

OpenCCG uses a multi-modal version of CCG (Baldridge, 2002; Baldridge and Kruijff, 2003), where modalities on the slashes enable fine-grained lexicalized derivational control over the re-ordering and re-bracketing effects of the various type raising and composition operators. The modalities furthermore make it possible to employ a fully universal rule component and to write more efficient unification schemes for rule application than for the original CCG framework. An advantage of the present approach to realization is that it directly reuses the derivational machinery originally developed for the OpenCCG parser, making the multi-modal improvements to CCG completely orthogonal to the realizer's concerns. For this reason, we will pass over the details of multi-modal CCG here.

2.2. 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 Minimal Recursion Semantics (MRS) framework (Copestake et al., 2001) used for HPSG. In the context of categorial grammar, Kruijff (2001, 2003) 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, 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.

As Blackburn (2000) explains, hybrid logic provides a language for representing relational structures that overcomes standard modal logic's inability to directly refer to states in a model – or equivalently, nodes in a graph. It does so by using *nominals*, a new sort of basic formula with which we can explicitly name states/nodes. Like 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 $@_i(p \wedge \langle \mathrm{F} \rangle (j \wedge q))$ 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, where q holds, is reachable via the modal relation F; equivalently, it states that node i is labeled by p, and that node j, labeled by q, is reachable from i via an arc labeled F.

In HLDS, hybrid logic is used as a language for describing discourse representation structures – which have their own underlying semantics, as we explain further in section 6 – 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 *Ted adores a musician that Bob saw* receives the represention in (7):

(7) $@_e(\mathbf{adore} \wedge \langle \textsc{tense} \rangle \mathrm{pres} \wedge$
$\langle \textsc{Exp} \rangle (t \wedge \mathbf{Ted}) \wedge$
$\langle \textsc{Cont} \rangle (m \wedge \mathbf{musician} \wedge \langle \textsc{def} \rangle \text{-} \wedge$
$\langle \textsc{GenRel} \rangle (e_2 \wedge \mathbf{see} \wedge \langle \textsc{tense} \rangle \mathrm{past} \wedge$
$\langle \textsc{Perc} \rangle (b \wedge \mathbf{Bob}) \wedge$
$\langle \textsc{Phen} \rangle m)))$

In this example, e is a discourse referent for the event of adoring, which takes place in the present. It is related to t, the discourse referent for Ted, by the Exp(eriencer) role, and to m, the indefinite discourse referent for the musician, via the Cont(ent) role. The referent m is in turn related to e_2, the discourse referent for the past event of seeing, via the Gen(eral)Rel(ation) dependency role. Finally, the referent e_2 is related to b, for Bob, and back to m, by the dependency roles Perc(eiver) and Phen(omenon), respectively.[4]

The HLDS term in (7) is isomorphic to the underlying graph depicted in Figure 1 (see section 6 for discussion of such graphs). It may also be flattened to an equivalent (but no longer isomorphic) conjunction of fixed-size

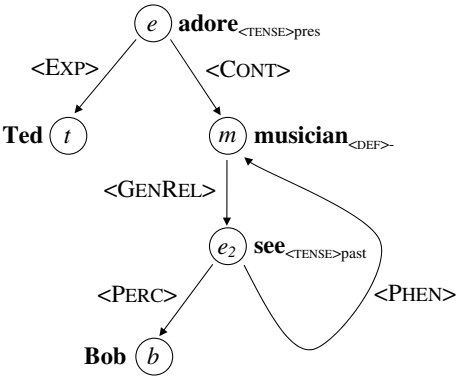

Figure 1. Underlying structure of *Ted adores a musician that Bob saw*.

elementary predications (EPs), which together describe the same underlying graph structure:

(8) $@_e\mathbf{adore} \wedge @_e\langle\textsc{tense}\rangle\text{pres} \wedge @_e\langle\textsc{Exp}\rangle t \wedge @_e\langle\textsc{Cont}\rangle m \wedge$
$@_t\mathbf{Ted} \wedge @_m\mathbf{musician} \wedge @_m\langle\textsc{def}\rangle\text{-} \wedge @_m\langle\textsc{GenRel}\rangle e_2 \wedge$
$@_{e_2}\mathbf{see} \wedge @_{e_2}\langle\textsc{tense}\rangle\text{past} \wedge @_{e_2}\langle\textsc{Perc}\rangle b \wedge @_{e_2}\langle\textsc{Phen}\rangle m \wedge$
$@_b\mathbf{Bob}$

As (8) shows, EPs come in three varieties: lexical predications, e.g. $@_e$ **adore**; semantic features, e.g. $@_e\langle\textsc{tense}\rangle$pres; and dependency relations, e.g. $@_e\langle\textsc{Exp}\rangle t$. When flattened, the HLDS representations resemble the neo-Davidsonian ones used in e.g. Kay's (1996) work on chart realization:

(9) $\text{adore}(e) \wedge \text{tense}(e, \text{pres}) \wedge \text{Exp}(e, t) \wedge \text{Cont}(e, m) \wedge$
$\text{Ted}(t) \wedge \text{musician(m)} \wedge \text{def}(m, \text{-}) \wedge \text{GenRel}(m, e_2) \wedge$
$\text{see}(e_2) \wedge \text{tense}(e_2, \text{past}) \wedge \text{Perc}(e_2, b) \wedge \text{Phen}(e_2, m) \wedge$
$\text{Bob}(b)$

In contrast to (9), the HLDS representation in (8) makes the head-dependent distinction explicit; (8) also has an equivalent hierarchical representation, (7), which the neo-Davidsonian one lacks.

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 (8). In our revised approach, categories consist of syntactic categories (atomic or complex) paired with logical forms, in the form of a conjunction of elementary predications; such categories are then paired with strings in the lexicon to form signs, as shown in (10). 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 (with feature names suppressed).

(10) a. $\textit{a} \vdash \mathsf{np}_x/\mathsf{n}_x : @_x\langle\textsc{def}\rangle\text{-}$
b. $\textit{musician} \vdash \mathsf{n}_m : @_m\mathbf{musician}$
c. $\textit{that} \vdash (\mathsf{n}_x\backslash\mathsf{n}_x)/(\mathsf{s}_{e,\,fin}|\mathsf{np}_x) : @_x\langle\textsc{GenRel}\rangle e$
d. $\textit{Bob} \vdash \mathsf{np}_b : @_b\mathbf{Bob}$
e. $\textit{saw} \vdash \mathsf{s}_{e,\,fin}\backslash\mathsf{np}_x/\mathsf{np}_y : @_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\text{past} \wedge$
$@_e\langle\textsc{Perc}\rangle x \wedge @_e\langle\textsc{Phen}\rangle y$

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, as in derivation (4), the sign for *Bob* in (10) can type-raise and compose with the one for

saw to yield (11), where x has been coindexed with b, and where the EPs have been conjoined. The sign for *that* can then apply to (11), yielding (12), where the EPs have again been conjoined and y has been coindexed with x.[5] Finally, successive applications of (12) to the sign for *musician*, and of the sign for *a* to the intermediate result, yield (13):

(11) *Bob saw* $\vdash \mathsf{s}_{e,\,fin}/\mathsf{np}_y : @_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\mathrm{past} \wedge @_e\langle\textsc{Perc}\rangle b \wedge @_e\langle\textsc{Phen}\rangle y \wedge @_b\mathbf{Bob}$

(12) *that Bob saw* $\vdash \mathsf{n}_x\backslash\mathsf{n}_x : @_x\langle\textsc{GenRel}\rangle e \wedge @_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\mathrm{past} \wedge @_e\langle\textsc{Perc}\rangle b \wedge @_e\langle\textsc{Phen}\rangle x \wedge @_b\mathbf{Bob}$

(13) *a musician that Bob saw* $\vdash \mathsf{n}_m : @_m\mathbf{musician} \wedge @_m\langle\textsc{def}\rangle\text{-} \wedge @_m\langle\textsc{GenRel}\rangle e \wedge @_e\mathbf{see} \wedge @_e\langle\textsc{tense}\rangle\mathrm{past} \wedge @_e\langle\textsc{Perc}\rangle b \wedge @_e\langle\textsc{Phen}\rangle m \wedge @_b\mathbf{Bob}$

An important property of additive approaches to semantic construction, such as ours, is that no semantic information can be dropped during the course of a derivation – that is, semantic construction is guaranteed to be *monotonic*. In the absence of this property, it becomes more difficult to ensure that one's realization algorithm is complete, as Copestake et al. (2001) discuss.

A benefit of employing flat representations in semantic construction is that it becomes easier to perform equality tests on signs, since the flat conjunctions of EPs, once sorted into a canonical order, can be straightforwardly employed in a hashing scheme. In our case, we use such equality tests 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 unwanted copying of predications in coordinate constructions.[6] For example, the EPs for the shared argument *a musician that Bob saw* in the right node raising example (5) appear just once in the resulting logical form, shown in hierarchical form below in (14) for readability, and with most of the predications of (13) suppressed (the dependency roles are Pos(itive), Neg(ative), Exp(eriencer), Cont(ent)):

(14) $@_s(\mathbf{but} \wedge$
$\quad\langle\textsc{Pos}\rangle(e_1 \wedge \mathbf{adore} \wedge \langle\textsc{tense}\rangle\mathrm{pres} \wedge$
$\quad\quad\langle\textsc{Exp}\rangle(t \wedge \mathbf{Ted}) \wedge$
$\quad\quad\langle\textsc{Cont}\rangle m) \wedge$
$\quad\langle\textsc{Neg}\rangle(e_2 \wedge \mathbf{detest} \wedge \langle\textsc{tense}\rangle\mathrm{pres} \wedge$
$\quad\quad\langle\textsc{Exp}\rangle(g \wedge \mathbf{Gil}) \wedge$
$\quad\quad\langle\textsc{Cont}\rangle m)) \wedge$
$@_m(\mathbf{musician} \wedge \ldots)$

In contrast, Baldridge and Kruijff's (2002) approach yields duplicate predications in such examples. In their approach, *Ted adores but Gil detests* ends up with the semantics given in (15) below, where the proposition p appears twice; consequently, when the coordinated phrase is combined with the shared argument *a musician that Bob saw*, the predications in (13) are copied into the two locations where p appears.

(15) $@_s(\mathbf{but} \wedge$
$\langle \textsc{Pos} \rangle (e_1 \wedge \mathbf{adore} \wedge \langle \textsc{tense} \rangle \text{pres} \wedge$
$\langle \textsc{Exp} \rangle (t \wedge \mathbf{Ted}) \wedge$
$\langle \textsc{Cont} \rangle (m \wedge p)) \wedge$
$\langle \textsc{Neg} \rangle (e_2 \wedge \mathbf{detest} \wedge \langle \textsc{tense} \rangle \text{pres} \wedge$
$\langle \textsc{Exp} \rangle (g \wedge \mathbf{Gil}) \wedge$
$\langle \textsc{Cont} \rangle (m \wedge p)))$

As we will show in section 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.

4. The Algorithm

4.1. Input

The input to the algorithm is a logical form encoded as an HLDS term, such as the one for *a musician that Bob saw* below, together with a function that computes n-gram scores of possible realizations. Optionally, the algorithm may also take a function implementing a custom pruning strategy as an argument; the default, n-best pruning strategy keeps the n best-scoring edges per group of edges with equivalent categories in the chart, for a configurable parameter n.

(16) $@_m(\mathbf{musician} \wedge \langle \textsc{def} \rangle \text{-} \wedge$
$\langle \textsc{GenRel} \rangle (e \wedge \mathbf{see} \wedge \langle \textsc{tense} \rangle \text{past} \wedge$
$\langle \textsc{Perc} \rangle (b \wedge \mathbf{Bob}) \wedge$
$\langle \textsc{Phen} \rangle m))$

Although in principle the contents of HLDS terms like (16) may vary considerably in the extent to which they precisely specify particular surface forms, we have chosen to employ a relatively low level of abstraction in our input LFs, in order to provide a high degree of control over the results of realization. As Elhadad and Robin (1998) point out, practical experience with reusable realizers such as their FUF/SURGE and Lavoie and Rambow's (1997) RealPro has shown that applications often demand considerable control over lexical and syntactic choices, mostly leaving the realizer to just handle inflection, agreement, word order and insertion of function

words. In accord with this observation, in our approach, lexical and syntactic choices are primarily governed by the lexical predicates and semantic features that the client application has selected for inclusion in the realizer's input logical form.[7] While it could prove desirable with some applications to defer more decision making to the realizer – perhaps via greater use of underspecification in the input, or explicitly listed alternatives – we leave the question of how to do so in a practical way as a topic for future research.

4.2. Chunking and flattening

In a preprocessing stage, chunking rules are applied to the input LF, which is subsequently flattened. The chunking rules serve to identify sub-problems to be solved independently prior to further combination. They address the problem, noted by Kay (1996), that chart realization algorithms can waste a great deal of time on generation paths containing semantically incomplete phrases (see section 4.6 for discussion).

The chunking rules together specify a transformation, κ, for adding chunks to an HLDS term. The default rules appear in (17) below, in order of increasing priority; grammar-specific exceptions to the default rules are discussed in section 4.6.

(17) a. $\kappa(x) = x$
$\kappa(\mathbf{label}) = \mathbf{label}$
$\kappa(\langle \textsc{attr} \rangle \mathrm{val}) = \langle \textsc{attr} \rangle \mathrm{val}$

b. $\kappa(@_x(\ldots)) = @_x(\kappa(\ldots))$
$\kappa(\langle \textsc{Rel} \rangle(\ldots)) = \langle \textsc{Rel} \rangle(\kappa(\ldots))$
$\kappa(\ldots_1 \wedge \ldots \wedge \ldots_n) = \kappa(\ldots_1) \wedge \ldots \wedge \kappa(\ldots_n)$

c. $\kappa(\ldots_1 \wedge \ldots \langle \textsc{Rel} \rangle(x \wedge \ldots) \ldots \wedge \ldots_n)$
$= [_c \kappa(\ldots_1) \wedge \ldots \kappa(\langle \textsc{Rel} \rangle(x \wedge \ldots)) \ldots \wedge \kappa(\ldots_n)]$

The first two clauses of (17) just recursively copy the input term. The third clause takes precendence over the second one, and introduces a chunk for non-trivial subtrees.

When applied to the LF in (16), the chunking rules identify the predicates that specify the relative clause as forming an independent sub-problem, chunked as c_1 below:

(18) $@_m(\mathbf{musician} \wedge \langle \textsc{def} \rangle\text{-} \wedge$
$\langle \textsc{GenRel} \rangle([_{c_1} e \wedge \mathbf{see} \wedge \langle \textsc{tense} \rangle \mathrm{past} \wedge$
$\langle \textsc{Perc} \rangle(b \wedge \mathbf{Bob}) \wedge$
$\langle \textsc{Phen} \rangle m]))$

More complex logical forms will often be chunked into multiple independent sub-problems, some of which may be nested.

After the chunking rules have been applied, the logical form is flattened to a list of EPs, so that the extent to which partial realizations cover the input LF can be tracked positionally. In this process, the set of EPs that make up each chunk is also determined. For example, the LF in (18) is flattened to (19), with the constituents of chunk c_1 identified in (20):

(19) $0{:}@_m\mathbf{musician}$, $1{:}@_m\langle\text{DEF}\rangle$-, $2{:}@_m\langle\text{GENREL}\rangle e$, $3{:}@_e\mathbf{see}$, $4{:}@_e\langle\text{TENSE}\rangle\text{past}$, $5{:}@_e\langle\text{PERC}\rangle b$, $6{:}@_e\langle\text{PHEN}\rangle m$, $7{:}@_b\mathbf{Bob}$

(20) $c_1{:}\{3, 4, 5, 6, 7\}$

4.3. Data structures

The algorithm makes use of four principal data structures: edges, unary rule instances, an agenda, and a chart. An *edge* is a CCG sign plus the following bookkeeping data structures: a bit vector which records the sign's coverage of the input LF; a bit vector with the sign's indices (nominals) that are syntactically available; and a list of incomplete chunks. The 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. The list of incomplete chunks is used to prevent combinations with edges that do not contribute to completing these chunks. That is, if an edge has one or more incomplete chunks listed for it, then it is only allowed to combine with edges whose predicates (if any) intersect with all the incomplete chunks; combinations with semantically null edges are also permitted.

To illustrate these data structures, 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, and the list of incomplete chunks (here, just c_1) in square brackets:

(21) $\{3, 4, 5, 6\}$, $\{e, b, m\}$, $[c_1]$,
$\mathit{saw} \vdash \mathsf{s}_{e,\,fin}\backslash\mathsf{np}_b/\mathsf{np}_m : @_e\mathbf{see} \wedge @_e\langle\text{TENSE}\rangle\text{past} \wedge @_e\langle\text{PERC}\rangle b \wedge @_e\langle\text{PHEN}\rangle m$

(22) $\{3, 5, 6\}$, $\{e, b, m\}$, $[c_1]$,
$\mathit{see} \vdash \mathsf{s}_{e,\,nonfin}\backslash\mathsf{np}_b/\mathsf{np}_m : @_e\mathbf{see} \wedge @_e\langle\text{PERC}\rangle b \wedge @_e\langle\text{PHEN}\rangle m$

Unary rule instances are instantiations of the unary type changing rules for particular input EPs. Like edges, they have bit vectors for the EPs and indices, but there is no need for a list of incomplete chunks:

(23) $\{2\}$, $\{e, m\}$, $\mathsf{s}_{e,\,fin}|\mathsf{np}_m \Rightarrow \mathsf{n}_m\backslash\mathsf{n}_m : @_m\langle\text{GENREL}\rangle e$

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 scoring function.

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 the extent of partial parses, a chart for realization uses an edge's coverage vector to track partial realizations. Similarly, whereas a chart for parsing uses string adjacency to determine which edges to try to combine, a chart for realization uses intersecting indices to implement a corresponding notion of graph adjacency in the input logical forms.

To improve efficiency, the chart maintains a hash set of edges in order to check in constant time whether a new edge is equivalent to one already in the chart. It also maintains a hash map from categories to sets of edges that share the same category but differ in their surface strings, for pruning purposes. Note that category equivalence requires that two categories have exactly matching syntactic categories and matching conjunctions of elementary predications in their LFs.

4.4. Lexical lookup

In the first phase of the algorithm, for each EP in the flattened input LF, relevant lexical entries are accessed according to an indexing scheme which records the most informative part of each lexical item's semantics. 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 6 in Figure 2, p. 51).[8] To improve performance, the semantically null entries are pre-filtered for relevance; in particular, 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 first unified with one of the lexical entry's EPs, and then unification of the remaining EPs in the lexical entry is attempted against the remaining EPs in the input LF.[9] If the lexical entry contains an incompatible semantic feature or relation, then instantiation fails. If instantiation succeeds, an edge is created for the instantiated entry and added to the agenda. In creating the edge, the chunks of EPs for the input LF are consulted, and any chunks which partially overlap with the edge's coverage vector are listed as incomplete chunks for the edge.

Until the agenda is empty:

1. Check whether the time limit for the anytime search has been exceeded. If so, return the best complete edge found so far (if any, otherwise continue).
2. Remove the first edge from the agenda and set it to be the current edge.
3. Check whether the current edge is equivalent to one already in the chart, or fails to meet the pruning threshold. If so, skip the rest of the loop.
4. 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 tuple (cf. section 5 for discussion).
   (c) Check that all incomplete chunks for the current edge intersect with the chart edge's coverage vector, and vice-versa. If not, skip the chart edge.
   (d) Combine the current edge with the chart edge using all available binary combinatory rules.
   (e) Add any resulting new edges to the agenda, updating the reference to the best complete edge found so far (if any).
5. Apply all unary combinatory rules to the current edge, likewise adding any resulting new edges to the agenda. With the unary rule instances, the same checks on the coverage vector, the index vector and the incomplete chunks are performed, as if the rule instance were a chart edge.
6. Combine the current edge with edges for all semantically null lexical items, as if these were chart edges (except that there are no EPs to check), likewise adding any resulting new edges to the agenda.
7. Add the current edge to the chart. If the number of edges with the same category exceeds the pruning threshold, prune the lowest scoring edge in this group from the chart.

Return the best scoring edge.

Figure 2. Main loop.

With unary type changing rules, lookup and instantiation is handled in much the same way. For each EP in the input logical form, instantiation is attempted with any unary rules that are indexed by this EP. Successfully instantiated rules, together with those unary type changing rules that have empty semantics, are added to the list of unary rules to be used in the derivation (see step 5 in Figure 2).

Continuing our example, the predicational EP $@_e$**see** triggers the lookup of the lexical entries for the edges shown in (21) and (22). Note that the present tense form *sees* is accessed as well, but instantiation fails due to its incompatible tense value (whereas the non-finite form *see* has no tense value). The predicational EPs $@_m$**musician** and $@_b$**Bob** likewise trigger entries for *musician* and *Bob*. The relational EP $@_m\langle$GenRel$\rangle e$ triggers the lookup and instantiation of the edge for the relative pronoun shown in

(24) below, in addition to the unary rule instance seen earlier in (23). Similarly, the featural EPs $@_e\langle\text{TENSE}\rangle\text{past}$ and $@_m\langle\text{DEF}\rangle$-trigger the introduction of the auxiliary *did* and the indefinite determiner *a*.

(24) $\{2\},\ \{e, m\},\ [\ ],\ \textit{that} \vdash (\mathsf{n}_m\backslash\mathsf{n}_m)/(\mathsf{s}_{e,\,fin}|\mathsf{np}_m) : @_m\langle\text{GENREL}\rangle e$

In its current form, the lexical lookup phase of the algorithm assumes that all the EPs in a lexical entry can be uniquely instantiated starting from the indexed EP. This simplifying assumption imposes some minor limitations on the EPs that are allowed to appear in a lexical entry; in particular, the EPs are required to specify a connected sub-graph with unique dependency roles.

4.5. Application of 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, or until the time limit for the anytime search is exceeded. In terms of standard chart parsing/realization terminology, all edges are passive (rather than active), since they represent complete constituents in CCG. Figure 2 describes the main loop in full detail, including the various constraints used to cut down the search space.

Some of the edges generated during the combinatory rule phase are shown in (25)–(35) below, with the EPs suppressed. The edge for *Bob* type-raises, yielding (25), and the edge for *a* applies to *musician* to yield (26). The edge for *see* (22) combines with the semantically null infinitival *to*, yielding (27); (25) then forward composes with both *saw* (21) and *to see* (27), yielding (28) and (29). Since *Bob to see* (29) is marked syntactically as infinitival rather than finite, it will not unify with the relative pronoun edge (24), which instead applies only to *Bob saw* (28), yielding (30). The unary rule instance for reduced relatives (23) also applies to (28), yielding (31). Finally, the relative clause (30) and reduced relative (31) apply to *musician* yielding (32) and (33), which are then combined with *a* to yield the complete edges in (34) and (35), with the choice between these forms left to the *n*-gram scoring function to decide.

(25) $\{7\},\ \{b\},\ [c_1],\ \textit{Bob} \vdash \mathsf{s}_1/(\mathsf{s}_1\backslash\mathsf{np}_b)$

(26) $\{0, 1\},\ \{m\},\ [\ ],\ \textit{a musician} \vdash \mathsf{np}_m$

(27) $\{3, 5, 6\},\ \{e, b, m\},\ [c_1],\ \textit{to see} \vdash (\mathsf{s}_{e,\,inf}\backslash\mathsf{np}_b)/\mathsf{np}_m$

(28) $\{3, 4, 5, 6, 7\},\ \{e, m\},\ [\ ],\ \textit{Bob saw} \vdash \mathsf{s}_{e,\,fin}/\mathsf{np}_m$

(29) $\{3, 5, 6, 7\},\ \{e, m\},\ [c_1],\ \textit{Bob to see} \vdash \mathsf{s}_{e,\,inf}/\mathsf{np}_m$

(30) $\{2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *that Bob saw* $\vdash \mathsf{n}_m \backslash \mathsf{n}_m$

(31) $\{2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *Bob saw* $\vdash \mathsf{n}_m \backslash \mathsf{n}_m$

(32) $\{0, 2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *musician that Bob saw* $\vdash \mathsf{n}_m$

(33) $\{0, 2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *musician Bob saw* $\vdash \mathsf{n}_m$

(34) $\{0, 1, 2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *a musician that Bob saw* $\vdash \mathsf{np}_m$

(35) $\{0, 1, 2, 3, 4, 5, 6, 7\}$, $\{m\}$, [ ], *a musician Bob saw* $\vdash \mathsf{np}_m$

As the example shows, the rules apply in bottom-up fashion to reproduce the same derivation that would be used in parsing the sentence. The bit vectors help to improve efficiency by avoiding certain combinations without attempting unification; for example, no attempt is made to combine the edges for *saw* and *to see*, since these overlap semantically, and no attempt is made to combine *Bob* and *a musician*, since their index vectors do not intersect.

The chunking constraints introduce an element of top-down prediction to the algorithm, by avoiding combinations that are not compatible with the constituent boundaries implicit in the grouped EPs. For example, since the edge for the verb *saw* has c_1 as an incomplete chunk, it is not allowed to combine with the edge for *a musician*. It is possible to improve efficiency further via more top-down prediction, e.g. to avoid generating the edges *to see* and *Bob to see*, which do not participate in any complete derivation; to that end, a more advanced scheme for using features to license and instantiate semantically null categories is proposed in White (2004).

4.6. The problem of semantically incomplete phrases

The main purpose of the chunking rules is to avoid a proliferation of semantically incomplete edges. As Kay (1996) points out, chart realization in its naïve form will generate sentences for all subsets of the predicates corresponding to syntactically optional modifiers, only one of which is semantically complete. For example, with his sentence (36), there will be edges for syntactically complete sentences corresponding to all subsets of the modifiers *newspaper*, *fast*, *tall*, *young*, and *Polish*, yielding a grand total of 32 strings, 31 of which are useless:

(36) Newspaper reports said that the tall young Polish athlete ran fast.

Kay's (1996) approach to this problem is to avoid creating edges that would prevent access to a semantic index for which there remain unincorporated modifiers. For example, the index for *athlete* becomes unavailable once the embedded subject combines with *ran*, so edges like *the athlete ran*, which do not include all the modifiers of *athlete*, will be avoided. As a

result, this strategy helps to ensure that only semantically complete phrases are incorporated into larger phrases.

As an alternative to Kay's approach, Carroll et al. (1999) propose to delay the insertion of all intersective modifiers until the rest of the chart has been completed, and then to add them via adjunction. An advantage of their solution over Kay's is that it further reduces unwanted edges by avoiding extra intermediate results. However, it is unclear how well delaying the handling of intersective modifiers until the end would fit with our anytime approach to realization, and for this reason we have not pursued their solution.

The chunking rules similarly keep semantically incomplete phrases from proliferating, though in a more fine-grained and flexible way that is better suited to CCG than Kay's approach. As we saw in the preceding section, the chunking rules can cut down the search space not just with syntactically optional modifiers, but also with "non-standard" constituents such as $\mathsf{s/np}$ – both in relative clauses and coordinate structures (cf. section 5). They can also cut off the proliferation of semantically incomplete phrases earlier than would Kay's approach. For example, in CCG, subject NPs can type raise and compose with pre-verbal modifiers, such as *also*, yielding a category where the subject's semantic index is still accessible. In this case, Kay's filter would not prevent a semantically incomplete subject NP from combining with *also*; in contrast, the chunking rules could easily do so, by identifying the subject's EPs as a chunk.

As mentioned in section 4.2, the default chunking rules must be accompanied by a handful of grammar specific exceptions, in order to handle certain cases where the semantic and syntactic structures diverge. For example, an exception must be made with VP negation. Consider the following LF for *Bob did not see the musician*:

(37) $@_s(\mathbf{not} \wedge \langle \text{TENSE} \rangle \text{past} \wedge$
$\langle \text{SCOPE} \rangle (e \wedge \mathbf{see} \wedge$
$\langle \text{PERC} \rangle (b \wedge \mathbf{Bob}) \wedge$
$\langle \text{PHEN} \rangle (m \wedge \mathbf{musician} \wedge \langle \text{DEF} \rangle +)))$

Here the subtree under ⟨SCOPE⟩ cannot be chunked, as this would prevent *not* from combining with the verb phrase *see the musician*, whose semantics does not include the EP for the subject *Bob*. Exceptions to the default rules are typically based on particular predicates or dependency relations, and are given higher priority. The rule for VP negation appears in (38) below; it overrides (17c) and simply reverts to the basic term copying behavior, as in (17b):

(38) $\kappa(\ldots_1 \wedge \mathbf{not} \wedge \ldots \langle \text{SCOPE} \rangle (x \wedge \ldots) \ldots \wedge \ldots_n) =$
$\kappa(\ldots_1) \wedge \mathbf{not} \wedge \ldots \langle \text{SCOPE} \rangle (x \wedge \kappa(\ldots)) \ldots \wedge \kappa(\ldots_n)$

In our experience to date, it has not been too onerous to identify the handful of cases where exceptions to the default chunking rules have been required. Using the OpenCCG regression testing tool and a reasonably comprehensive regression test suite, one can look for examples which parse successfully but fail to realize with chunking turned on. Then, for each problematic example, one can examine a detailed realization trace to see where the desired derivation has been blocked, and add an exception rule like (38) to avoid inserting a chunk in the problematic context.

4.7. The problem of relatively free word order

While the chunking rules cut down the search space by keeping semantically incomplete phrases from proliferating, a grammar may still license an exponential number of phrases for a given input – especially when, for engineering reasons, the grammar is intentionally allowed to overgenerate, in order to take advantage of an *n*-gram scoring function's ability to select preferred word orders. For example, with prenominal modifiers such as *tall*, *young*, and *Polish*, one can employ the simple syntactic category n/n and leave the preferred ordering to the *n*-grams, rather than laboriously encoding order preferences via syntactic features. However, this means that the realizer will create 3! semantically complete phrases for the embedded subject in Kay's example (36) – including *the tall young Polish athlete* as well as *the Polish young tall athlete* – which will of course multiply the number of larger phrases that incorporate these various possibilities.

The edge pruning and anytime search methods are designed to help keep the realizer from getting bogged down in the face of relatively free word order – which more typically occurs with intersective modifiers, but may also arise from different lexico-syntactic choices, when these are left underdetermined by the grammar and input LF. For example, assuming that the *n*-gram scorer has access to appropriate training data, preferred phrases such as *the tall young Polish athlete* will be prioritized ahead of variants like *the Polish young tall athlete* on the agenda, leading eventually to the earlier emergence of complete realizations incorporating the preferred phrase. In fact, the dispreferred variants may never even reach the front of the agenda, if the anytime search limit is exceeded prior to this point, and a complete realization has already been found. Edge pruning helps to further improve the situation, as it lessens the multiplicative effects of the dispreferred variants (or better, their earlier sub-phrases).

It is worth pointing out that since edge pruning only takes place within groups of edges sharing the same syntactic and semantic category, there is no way that edge pruning can prevent the search from turning up any complete realizations. In contrast, in our earlier attempts to prune edges from the chart as a whole – without grouping edges in this way – we found it

difficult to find a threshold that led to substantial pruning without causing realization to fail on some examples.

5. Coordination

5.1. Clausal coordination

CCG's flexible approach to constituency delivers derivations for a wide variety of coordinate structures, including those involving the coordination of "non-standard" constituents such as $\mathsf{s/np}$, as we saw in the derivation of the right node raising example (5), repeated in abbreviated form below:

(39) [Ted adores]$_{\mathsf{s/np}}$ but [Gil detests]$_{\mathsf{s/np}}$ a musician that Bob saw.

Examples like (39) can be handled using the category for *but* given in (40), where $\mathsf{s\$}$ schematizes over functions into s:

(40) *but* $\vdash$ $(\mathsf{s}_s\$1\backslash\mathsf{s}_{e_1}\$1)/\mathsf{s}_{e_2}\$1 : @_s\mathbf{but} \wedge @_s\langle\textsc{Pos}\rangle e_1 \wedge @_s\langle\textsc{Neg}\rangle e_2$

Category (40) enables *Ted adores* and *Gil detests* to coordinate as follows, where x fills the Cont(ent) role for both e_1 and e_2:

(41) *Ted adores but Gil detests* $\vdash$ $\mathsf{s}_s/\mathsf{np}_x$:
$@_s\mathbf{but} \wedge @_s\langle\textsc{Pos}\rangle e_1 \wedge @_s\langle\textsc{Neg}\rangle e_2 \wedge$
$@_{e_1}\mathbf{adore} \wedge \ldots @_{e_1}\langle\textsc{Cont}\rangle x \wedge$
$@_{e_2}\mathbf{detest} \wedge \ldots @_{e_2}\langle\textsc{Cont}\rangle x$

The coordinated constituent (41) can then combine with *a musician that Bob saw*, unifying x with m, and yielding a flat conjunction of EPs equivalent to (14):

(42) *Ted adores but Gil detests a musician that Bob saw* $\vdash$ s_s :
$@_s\mathbf{but} \wedge @_s\langle\textsc{Pos}\rangle e_1 \wedge @_s\langle\textsc{Neg}\rangle e_2 \wedge$
$@_{e_1}\mathbf{adore} \wedge \ldots @_{e_1}\langle\textsc{Cont}\rangle m \wedge$
$@_{e_2}\mathbf{detest} \wedge \ldots @_{e_2}\langle\textsc{Cont}\rangle m \wedge$
$@_m\mathbf{musician} \wedge \ldots$

Since the present approach to semantic construction does not produce duplicate EPs for the shared argument *a musician that Bob saw*, the output of the OpenCCG parser for (39) is the same as what the realizer expects as input. In contrast, the duplicate EPs that would arise with Baldridge and Kruijff's Baldridge and Kruijff's (2002) approach to semantic construction – cf. (15) on p.10 – would cause problems for the realizer's tracking of input LF coverage. Indeed, the LF that would arise from (15) is perhaps more similar to the one for the clause-level coordination in (43) below than it is to (42):[10]

(43) *Ted adores a musician that ... but Gil detests a musician that ...*
$\vdash \mathsf{s}_s : @_s\mathbf{but} \wedge @_s\langle\textsc{Pos}\rangle e_1 \wedge @_s\langle\textsc{Neg}\rangle e_2 \wedge$
$@_{e_1}\mathbf{adore} \wedge \ldots @_{e_1}\langle\textsc{Cont}\rangle m_1 \wedge$
$@_{e_2}\mathbf{detest} \wedge \ldots @_{e_2}\langle\textsc{Cont}\rangle m_2 \wedge$
$@_{m_1}\mathbf{musician} \wedge \ldots \wedge @_{m_2}\mathbf{musician} \wedge \ldots$

The HLDS terms in (42) and (43) are not interchangeable as inputs to the realizer – which is as desired, since (42) requires the same musician to be adored and detested, whereas (43) strongly implicates that two different musicians are involved. As such, these examples show how differences in the realizer's input logical form – which are reminiscent of the differences between reduced and unreduced λ-terms[11] – can be used to control the choice of coordination options made available by the grammar.[12]

The distinction between (42) and (43) naturally raises the question as to what happens in similar examples involving quantification, first noted by Geach (1972) and more recently discussed by Steedman (1999, 2003):

(44) [Every boy adores]$_{\mathsf{s/np}}$ but [every girl detests]$_{\mathsf{s/np}}$ a musician that Bob saw.

In one reading of (44), there is a possibly different musician for each boy or girl that is adored or detested. There is also another reading of (44) where there is a specific musician that is adored or detested by each boy or girl. There does not, however, seem to be a "mixed" reading, where a specific musician is adored by each boy but a possibly different musician is detested by each girl, or vice versa.

There is no problem with parsing or realizing (44); all that is needed is the following category for *every*, which is semantically and syntactically type-raised in the lexicon:[13]

(45) *every* $\vdash \mathsf{s}_s/(\mathsf{s}_e\backslash\mathsf{np}_x)/\mathsf{n}_x : @_s\mathbf{every} \wedge @_s\langle\textsc{Restr}\rangle x \wedge @_s\langle\textsc{Scope}\rangle e$

Category (45) leads to the following HLDS representation for (44), shown in hierarchical form for readability:

(46)
$@_s(\mathbf{but} \wedge$
$\quad\langle\textsc{Pos}\rangle(s_1 \wedge \mathbf{every} \wedge$
$\quad\quad\langle\textsc{Restr}\rangle(b \wedge \mathbf{boy}) \wedge$
$\quad\quad\langle\textsc{Scope}\rangle(e_1 \wedge \mathbf{adore} \wedge \langle\textsc{tense}\rangle\text{pres} \wedge$
$\quad\quad\quad\langle\textsc{Exp}\rangle b \wedge$
$\quad\quad\quad\langle\textsc{Cont}\rangle m)) \wedge$
$\quad\langle\textsc{Neg}\rangle(s_2 \wedge \mathbf{every} \wedge$
$\quad\quad\langle\textsc{Restr}\rangle(g \wedge \mathbf{girl}) \wedge$
$\quad\quad\langle\textsc{Scope}\rangle(e_2 \wedge \mathbf{detest} \wedge \langle\textsc{tense}\rangle\text{pres} \wedge$
$\quad\quad\quad\langle\textsc{Exp}\rangle g \wedge$

$$\langle \textsc{Cont} \rangle m))) \wedge$$
$$@_m(\textbf{musician} \wedge \ldots)$$

At first glance, the HLDS term in (46) would seem to require that there is a specific musician m that is adored or detested by all boys and girls, raising the question as to how the non-specific reading (with a possibly different musician) is to be handled. However, in section 6, we will show how a proper treatment of scope in the underlying graph structures enables (46) to represent the non-specific reading, despite making use of a single musician discourse referent m. We will also discuss how Geurts's (2002) proposal for handling specific indefinites in DRT could be used to derive the specific reading, as well as how the shared musician node could prevent the mixed reading from arising.

5.2. Distributive 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 *Bob and Gil* in (47) below pose a challenge for unification-based approaches to semantic construction, since the index x cannot be unified with the referents for both *Bob* and *Gil*:[14]

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

Following Moore's strategy, we tackle this problem by introducing a bound variable into the semantic representation for (47):

(48)
$$@_s(\textbf{and} \wedge$$
$$\langle \textsc{BoundVar} \rangle x \wedge$$
$$\langle \textsc{List} \rangle (e_1 \wedge \text{elem} \wedge$$
$$\langle \textsc{Item} \rangle (b \wedge \textbf{Bob}) \wedge$$
$$\langle \textsc{Next} \rangle (e_2 \wedge \text{elem} \wedge$$
$$\langle \textsc{Item} \rangle (g \wedge \textbf{Gil}))) \wedge$$
$$\langle \textsc{Pred} \rangle (e \wedge \textbf{adore} \wedge \langle \textsc{tense} \rangle \text{pres} \wedge$$
$$\langle \textsc{Exp} \rangle (t \wedge \textbf{Ted}) \wedge$$
$$\langle \textsc{Cont} \rangle x))$$

In (48), the relations ⟨List⟩, ⟨Item⟩, and ⟨Next⟩ encode a linked list. Each item in the list is wrapped by an elem(ent) nominal, where ⟨Item⟩ points to the item itself, and ⟨Next⟩ points to the next element in the list, if any. ⟨List⟩ points to the first element in the list.

As we will explain further in section 6, the HLDS term in (48) is intended to be equivalent to the conjunction of the terms formed by distributing the predicate *Ted adores x* across each member of the two item list *Bob and Gil*, as indicated in (49):

(49) $\bigwedge_{x \in [b:\text{Bob},\ g:\text{Gil}]} e : \text{Ted adores } x$

In parallel fashion, the HLDS term for *Ted adores Bob or Gil* – i.e., (48) with **and** replaced by **or** – is is intended to be equivalent to the disjunction of the terms formed by distributing the predicate *Ted adores x* across each member of the two item list *Bob and Gil*:

(50) $\bigvee_{x \in [b:\text{Bob},\ g:\text{Gil}]} e : \text{Ted adores } x$

With (48) as the target semantics, example (47) can be parsed and realized using the category given in (51), which takes the two coordinated NPs and forms a (backwards) type-raised NP reminiscent of the category for *every* seen in (45):

(51) $\textit{and} \vdash \mathsf{s}_s\$\backslash(\mathsf{s}_e\$/\mathsf{np}_x)\backslash\mathsf{np}_{x_1}/\mathsf{np}_{x_2} : @_s\mathbf{and} \wedge @_s\langle\textsc{BoundVar}\rangle x \wedge @_s\langle\textsc{List}\rangle e_1 \wedge @_s\langle\textsc{Pred}\rangle e \wedge @_{e_1}\text{elem} \wedge @_{e_1}\langle\textsc{Item}\rangle x_1 \wedge @_{e_1}\langle\textsc{Next}\rangle e_2 \wedge @_{e_2}\text{elem} \wedge @_{e_2}\langle\textsc{Item}\rangle x_2$

As an alternative to using the category in (51), one could in principle locate the distributivity in the semantics of verbs taking plural NPs as arguments, as proposed in Steedman (2003), and treat conjoined NPs as simply introducing a set referent. By doing so, one could prevent scope-inverting readings for sentences like *Some fan adores Bob and Gil* – in which a possibly different fan adores each of the musicians Bob and Gil – from arising, which do not seem to be available empirically. However, since such an approach would make it more difficult to come up with a parallel treatment of *or*, we have chosen to implement a more traditional approach to distributive NP coordination here.

It is possible to generalize the category in (51) to handle lists of arbitrary length. One way to do so is to split (51) into two parts, one that connects the last two items in the list and adds the boolean operator, and another that uses a unary rule to close off the list and type-raise the result. In addition, a category for the comma may be used to connect the remaining items in the list, as well as to thread the nominal for the boolean operator down to the conjunction. To accomplish this threading, we introduce an op-index feature to hold another nominal – the one for the boolean operator – in addition to the one held by the usual index feature. In the categories below, npconj is used as the category for incomplete lists; also, the values for the index feature are paired with the values for the op-index feature (with feature names still suppressed):[15]

(52) $\textit{and} \vdash \mathsf{npconj}_{e_1,s}\backslash\mathsf{np}_{x_1}/\mathsf{np}_{x_2} : @_s\mathbf{and} \wedge @_{e_1}\text{elem} \wedge @_{e_1}\langle\textsc{Item}\rangle x_1 \wedge @_{e_1}\langle\textsc{Next}\rangle e_2 \wedge @_{e_2}\text{elem} \wedge @_{e_2}\langle\textsc{Item}\rangle x_2$

(53) $, \vdash \mathsf{npconj}_{e_1,s}\backslash\mathsf{np}_{x_1}/\mathsf{npconj}_{e_2,s} : @_{e_1}\text{elem} \wedge @_{e_1}\langle\textsc{Item}\rangle x_1 \wedge @_{e_1}\langle\textsc{Next}\rangle e_2$

(54) $\mathsf{npconj}_{e_1,s} \Rightarrow \mathsf{s}_s\backslash(\mathsf{s}_e\$/\mathsf{np}_x) : @_s\langle\textsc{BoundVar}\rangle x \wedge @_s\langle\textsc{List}\rangle e_1 \wedge @_s\langle\textsc{Pred}\rangle e$

A similar approach can also be taken to generalize conjunction at the clause level to lists of arbitrary length, using sconj as the category for incomplete lists.

5.3. Argument clusters and gapping

The approach to distributive NP coordination presented above can be extended to handle argument clusters – as in (55) below – without the need to invoke otherwise unnecessary deletion operations.

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

To handle (55), we introduce tup(le) elements to connect pairs of NP referents and bound variables, in the following category for *and*:

(56) *and* $\vdash$
$(\mathsf{s}_s\$\backslash(\mathsf{s}_e\$/\mathsf{np}_y/\mathsf{np}_x))\backslash(\mathsf{s}\$\backslash(\mathsf{s}\$/\mathsf{np}_{y_1}/\mathsf{np}_{x_1}))/(\mathsf{s}\$\backslash(\mathsf{s}\$/\mathsf{np}_{y_2}/\mathsf{np}_{x_2})) :$
$@_s\textbf{and} \wedge @_s\langle\textsc{BoundVar}\rangle t \wedge @_s\langle\textsc{List}\rangle e_1 \wedge @_s\langle\textsc{Pred}\rangle e \wedge$
$@_t\text{tup} \wedge @_t\langle\textsc{Item1}\rangle x \wedge @_t\langle\textsc{Item2}\rangle y \wedge$
$@_{e_1}\text{elem} \wedge @_{e_1}\langle\textsc{Item}\rangle t_1 \wedge @_{e_1}\langle\textsc{Next}\rangle e_2 \wedge$
$@_{t_1}\text{tup} \wedge @_{t_1}\langle\textsc{Item1}\rangle x_1 \wedge @_{t_1}\langle\textsc{Item2}\rangle y_1 \wedge$
$@_{e_2}\text{elem} \wedge @_{e_2}\langle\textsc{Item}\rangle t_2 \wedge$
$@_{t_2}\text{tup} \wedge @_{t_2}\langle\textsc{Item1}\rangle x_2 \wedge @_{t_2}\langle\textsc{Item2}\rangle y_2$

Category (56) enables (55) to be parsed into a semantic representation analogous to (48). The resulting logical form is intended to be equivalent to the conjunction of the terms formed by distributing the predicate *Bob gave xy* across each pair $\langle x, y\rangle$ in the list consisting of the pairs $\langle$*Ted*, *a dog*$\rangle$ and $\langle$*Gil*, *a cat*$\rangle$, as sketched in (57):

(57) $\bigwedge_{\langle x,y\rangle\in[\langle t:\text{Ted},\ d:\text{a dog}\rangle,\ \langle g:\text{Gil},\ c:\text{a cat}\rangle]} e : \text{Bob gave } x\ y$

The derivation of (55) 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 could let all NPs combine via type raising and composition in all possible orders, we instead require the indices to be in an appropriate tuple in the input LF in order for the NPs to combine (cf. step 4b in Figure 2).

To handle gapping examples like (58), a similar category can be supplied for *and*, as shown in (59) without the semantics, which remains unchanged:

(58) Ted_t $\text{received}_{\mathsf{s}_e\backslash\mathsf{np}_x/\mathsf{np}_y}$ a dog_d and $[\text{Gil}_g\ \text{a cat}_c]_{\mathsf{s}\backslash(\mathsf{s}\backslash\mathsf{np}_g/\mathsf{np}_c)}$

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

Category (59) 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 (59) should be viewed as a compiled-out version of Steedman's approach to gapping.

The argument cluster category (56) can be generalized to handle longer lists of tuples, following the patterns in (52)–(54), and using the category $\mathsf{sconj}_{e_1,s}\$\backslash(\mathsf{s}\$/\mathsf{np}/\mathsf{np})$ as the category for incomplete lists. The gapping category (59) can be similarly generalized, though the first tuple requires special treatment, in that it must be handled in the unary rule analogue of (54).

5.4. Chunking and coordination

There are two ways in which the chunking rules interact with coordination. First, in order to realize examples involving shared arguments in coordinate structures, the shared arguments must be raised to the same level as the node for the conjunction. For example, with the right node raising example (5), *Ted adores but Gil detests a musician that Bob saw*, the predications for the shared argument *a musician that Bob saw* must appear at the same level as the ones for *but*, as shown in (14), repeated below as (60):

(60)
$$
\begin{aligned}
&@_s(\mathbf{but} \wedge \\
&\quad \langle \textsc{Pos} \rangle (e_1 \wedge \mathbf{adore} \wedge \langle \textsc{tense} \rangle \text{pres} \wedge \\
&\qquad \langle \textsc{Exp} \rangle (t \wedge \mathbf{Ted}) \wedge \\
&\qquad \langle \textsc{Cont} \rangle m) \wedge \\
&\quad \langle \textsc{Neg} \rangle (e_2 \wedge \mathbf{detest} \wedge \langle \textsc{tense} \rangle \text{pres} \wedge \\
&\qquad \langle \textsc{Exp} \rangle (g \wedge \mathbf{Gil}) \wedge \\
&\qquad \langle \textsc{Cont} \rangle m)) \wedge \\
&@_m(\mathbf{musician} \wedge \ldots)
\end{aligned}
$$

If instead the predications on the nominal m appeared under e_1 (or similarly for e_2), the default subtree chunking rule would prevent *Ted adores* from combining with *Gil detests*, as the edge for *Ted adores* would be considered incomplete; in this situation, the sub-tree under e_1 would only be considered complete when the edge for *Ted adores a musician that Bob saw* was generated – but this edge does not participate in the requisite derivation. In contrast, with (60) as the input (as desired), the chunk-

ing rules ensure that the edge for *Ted adores but Gil detests* is complete before allowing it to combine with the edge for *a musician that Bob saw*, and prevent unwanted edges such as *Ted adores a musician that Bob saw* from being generated.[16]

The second way in which the chunking rules interact with coordination involves the gapping category (59). Since this category requires the first pair in a list of paired items to be realized discontinuously, an exception rule must be added that avoids chunking the first pair of items into an independent sub-problem. Like (38), this rule overrides (17c) and just reverts to the basic term copying behavior.

6. Translation to DRT

The HLDS representations proposed in the preceding section rely on quantificational structures involving bound variables. To clarify the intended semantics of these representations, we provide below a translation from the graph structures underlying the HLDS terms – where the requisite notion of logical scope becomes clear – to the more familiar discourse representations structures (DRSs) of DRT (Kamp and Reyle, 1993). In principle, of course, a semantics for the structures described by the HLDS terms could be given directly, rather than indirectly via DRT. To do so, however, one would need to extend the model-theoretic semantics of HLDS given by Kruijff (2001, 2003) to handle quantificational structures, a topic which is beyond the scope of this paper.

6.1. Basic translation

The graph in Figure 1 (p. 44) depicts the structure underlying the logical forms in (7) and (8) for the sentence *Ted adores a musician that Bob saw*. Such graphs include nodes with labeled, directed arcs between them. Nodes themselves have an ID, a label and a set of features.[17]

The graphs of interest may be considered essentially acyclic, if we view cycles as involving arcs that point backwards from the perspective of a distinguished root node. For example, in Figure 1, the Phen(omenon) role may be viewed as a back reference to m from e_2. While the graphs are primarily tree structured, they may involve shared nodes, as we will see below.

The DRT representation for the graph in Figure 1 appears in (61) below. The DRS in (61) uses the concise linear notation for DRSs found in e.g. Muskens's compositional reformulation of DRT (Muskens, 1996), rather than the bulkier, two-dimensional boxes in Kamp and Reyle (1993). In this notation, a box is represented by a pair of square brackets, and contains a list of discourse referents separated by a vertical bar from a list of conditions. The merge of two boxes may also be specified using a semi-colon.

(61) $[e, t, m, e_2, b|$
$\text{adore}(e),\ \text{tense}(e, \text{pres}),\ \text{Exp}(e, t),\ \text{Cont}(e, m),$
$\text{Ted}(t),$
$\text{musician(m)},\ \text{def}(m, \text{-}), \text{GenRel}(m, e_2),$
$\text{see}(e_2),\ \text{tense}(e_2, \text{past}),\ \text{Perc}(e_2, b),\ \text{Phen}(e_2, m),$
$\text{Bob}(b)]$

The graph in Figure 1 may be translated to the DRS in (61) using the function τ defined in (62) below. The function τ takes as arguments a node x and a reference (not shown) to the location where the resulting DRS is to appear, so that the currently accessible discourse referents can be determined. To depict the relevant aspects of a node in the clauses for τ, we use (parts of) the node's HLDS description. The first clause covers the case of a node corresponding to an accessible discourse referent; it just returns an empty DRS. The second clause covers the case where there is no corresponding accessible discourse referent. In this clause, the resulting DRS is defined as a box that introduces x as new discourse referent, $[x|]$, merged with the translation of the node itself, $\tau_n(x)$, and the recursive translation of the node's relations, $\tau_r(x)$. The third clause defines the translation of a node itself, $\tau_n(x)$, as the straightforward translation of the node's label and features into DRS conditions. The fourth clause defines the recursive translation of the node's relations, $\tau_r(x)$, as a box containing the DRS conditions for the arcs merged with the translations of each of the nodes reached via these arcs.[18]

(62) a. $\tau(x) = [|]$, if x is accessible
b. $\tau(x) = [x|]; \tau_n(x); \tau_r(x)$, if x is not accessible
c. $\tau_n(x \wedge \mathbf{label} \wedge \langle \text{ATTR}_1 \rangle \text{val}_1 \wedge \ldots \wedge \langle \text{ATTR}_n \rangle \text{val}_n) =$
$[|\text{label}(x), \text{attr}_1(x, \text{val}_1), \ \ldots, \ \text{attr}_n(x, \text{val}_n)]$
d. $\tau_r(x \wedge \langle \text{REL}_1 \rangle x_1 \wedge \ldots \wedge \langle \text{REL}_n \rangle x_n) =$
$[|\text{Rel}_1(x, x_1), \ \ldots, \ \text{Rel}_n(x, x_n)]; \tau(x_1); \ldots; \tau(x_n)$

To illustrate the role of accessibility in the definition of τ, let us consider how the back reference to the musician node m in Figure 1 is handled. In traversing the graph beginning with the root e, no nodes with accessible referents are encountered until m is visited for the second time. At this point, the translation determined so far is equal to (61), as shown below:

(63) $\tau(e) = (61); \tau(m)$

Since m is accessible in (61), the first clause is chosen, and $\tau(m)$ in (63) is simply translated as an empty box, which disappears when merged with (61).

To translate examples involving negation, universal quantification, and coordination, additional clauses are required which take priority over the basic translation clause (62). The clause for negation appears in (64); the

clauses for universal quantification and coordination are given in the next two subsections.

(64) $\tau(s \wedge \mathbf{not} \wedge \langle \text{SCOPE} \rangle e) = [|\neg \tau(e)]$

6.2. Universal quantifiers

Translation of sentences involving universal quantifiers, such as (65), requires transforming the structures underlying HLDS representations like (66) into DRSs with nested boxes, as in (67):

(65) Every boy adores a musician.

(66) $@_s(\mathbf{every} \wedge$
$\langle \text{RESTR} \rangle(b \wedge \mathbf{boy}) \wedge$
$\langle \text{SCOPE} \rangle(e \wedge \mathbf{adore} \wedge \langle \text{TENSE} \rangle \text{pres} \wedge$
$\langle \text{EXP} \rangle b \wedge$
$\langle \text{CONT} \rangle(m \wedge \mathbf{musician} \wedge \langle \text{DEF} \rangle \text{-})))$

(67) $[|[b|\text{boy}(b)] \Rightarrow [e, m|\text{adore}(e),\ \text{tense}(e, \text{pres}),$
$\text{Exp}(e, b),\ \text{Cont}(e, m),$
$\text{musician}(m),\ \text{def}(m, \text{-})]]$

The desired result may be obtained using the following clause:

(68) $\tau(s \wedge \mathbf{every} \wedge \langle \text{RESTR} \rangle x \wedge \langle \text{SCOPE} \rangle e) = [|\tau(x) \Rightarrow \tau(e)]$

Note that the discourse referent x introduced in the restriction becomes accessible in the translation of the scope e, and thus the predicates attached to x end up only in the antecedent of the resulting DRS.

The reading of example (65) shown in the resulting DRS (67) is the one where there is a possibly different musician adored by each boy, as the discourse referent m appears in the consequent box of the universal, and is thus dependent on the discourse referent b appearing in the antecedent box. Likewise, the translation of the Geach example (44) yields the reading where there is a possibly different musican adored by each boy and detested by each girl. Figure 3 shows the structure described by the HLDS representation (46) for (44); the result of the translation is shown in (69) (treating *but* as a simple conjunction, for simplicity):

(69) $[|[b|\text{boy}(b)] \Rightarrow [e_1, m|\text{adore}(e_1),\ \text{tense}(e_1, \text{pres}),$
$\text{Exp}(e_1, b),\ \text{Cont}(e_1, m),$
$\text{musician}(m),\ \text{def}(m, \text{-}), \ldots],$
$[g|\text{girl}(g)] \Rightarrow [e_2, m|\text{detest}(e_2),\ \text{tense}(e_2, \text{pres}),$
$\text{Exp}(e_2, g),\ \text{Cont}(e_2, m),$
$\text{musician}(m),\ \text{def}(m, \text{-}), \ldots]]$

Since the scopes of the two universal quantifiers are independent, the musician node gets translated twice, despite appearing just once in the graph. The partial translation in (70) below shows that the discourse referent m in the consequent of the first universal is not accessible at the point where m is encountered for the second time:

(70) $\tau(s) =$
$[|[b|\text{boy}(b)] \Rightarrow [e_1, m|\text{adore}(e_1),\ \text{tense}(e_1, \text{pres}),$
$\text{Exp}(e_1, b),\ \text{Cont}(e_1, m),$
$\text{musician}(m),\ \text{def}(m, \text{-}), \ldots],$
$[g|\text{girl}(g)] \Rightarrow [e_2|\text{detest}(e_2),\ \text{tense}(e_2, \text{pres}),$
$\text{Exp}(e_2, g),\ \text{Cont}(e_2, m)]; \tau(m)]$

In contrast, with our original right node raising example (5), *Ted adores but Gil detests a musician that Bob saw*, the musician node m is accessible upon its second encounter, and thus the resulting DRS ends up with just a single musican adored by Ted but detested by Gil:

(71) $\tau(s) =$
$[e_1, t, m, e_2, g|$
$\text{adore}(e_1),\ \text{tense}(e_1, \text{pres}), \text{Exp}(e_1, t),\ \text{Cont}(e_1, m),$
$\text{Ted}(t),\ \text{musician}(m),\ \text{def}(m, \text{-}),\ \ldots,$
$\text{detest}(e_2),\ \text{tense}(e_2, \text{pres}), \text{Exp}(e_2, g),\ \text{Cont}(e_2, m),$
$\text{Gil}(g)]; \tau(m)$

If the translations of the structures underlying (65) and (44) yield nonspecific readings, the question arises as to how the specific readings can be accounted for. One possibility is offered by Geurts's (2002) proposal to handle both specificity and presupposition through a common pragmatic process of backgrounding. In his proposal, both the raising of specific indefinites and presupposition projection are governed by the buoyancy principle, whereby backgrounded material tends to float up to the main DRS (subject to various constraints and preferences). From the perspective of interpretation, the backgrounding process begins with an initial DRS computed by the grammar, such as (67) or (69), and optionally floats specific indefinites upwards, yielding wide or intermediate scope readings, if pragmatically plausible. For example, the result of floating the indefinite in (67) up to the main DRS appears below:

(72) $[m|\text{musician}(m),$
$[b|\text{boy}(b)] \Rightarrow [e|\text{adore}(e),\ \text{tense}(e, \text{pres}),$
$\text{Exp}(e, b),\ \text{Cont}(e, m)]]$

Geurts's proposal appears to fit reasonably well with our approach, since he takes specificity to be an essentially pragmatic phenomenon that is outside of the syntax–semantics interface, where the scopes of true quantifiers are

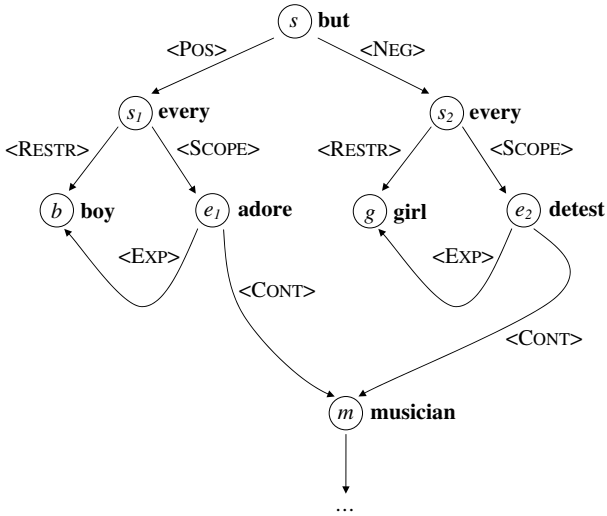

Figure 3. Underlying structure of *Every boy adores but every girl detests a musician ...* (*that Bob saw*).

determined. As such, his proposal could potentially be combined with our approach to semantic composition to yield an account of the different readings of indefinites; while the resulting account would not make use of Steedman's (1999, 2003) anytime skolemization operation, it would otherwise be essentially compatible with Steedman's approach to handling quantifiers in the syntax/semantics interface. One issue, however, is that applying Geurts's backgrounding process to the DRS translation in (69) of the Geach example (44) could plausibly yield the missing "mixed" reading, whereby a specific musician is adored by each boy but a possibly different musician is detested by each girl, or vice versa. This suggests that backgrounding should instead be applied to a discourse representation structure like the one in Figure 3, where there is a single shared musician node, which must either be raised or left *in situ*, with no possibility of a mixed reading arising.

6.3. List structures

Conjunctive list structures, such as the one seen in (48) for sentence (47), *Ted adores Bob and Gil*, could be translated similarly to universal quantifiers, as shown below (assuming an appropriate DRT element-of construct):

(73) $[|[x, b, g|\mathrm{Bob}(b),\ \mathrm{Gil}(g),\ x \in \{b, g\}] \Rightarrow$
$[e, t|\mathrm{adore}(e),\ \mathrm{tense}(e, \mathrm{pres}),\ \mathrm{Exp}(e, t),\ \mathrm{Cont}(e, x),\ \mathrm{Ted}(t)]]$

As an alternative, the predication can be spelled out for each member of the list, as in (74); this latter scheme has the advantage that it will also work for disjunctive lists:[19]

(74) $[x, b, e, t | x = b,\ \mathrm{Bob}(b),\ \mathrm{adore}(e),\ \mathrm{tense}(e, \mathrm{pres}),$
$\mathrm{Exp}(e, t),\ \mathrm{Cont}(e, x),\ \mathrm{Ted}(t)];$
$[x, g, e, t | x = g,\ \mathrm{Gil}(g),\ \mathrm{adore}(e),\ \mathrm{tense}(e, \mathrm{pres}),$
$\mathrm{Exp}(e, t),\ \mathrm{Cont}(e, x),\ \mathrm{Ted}(t)]$

To translate (48) into (74), the following clause may be used:

(75) $\tau(s \wedge \mathbf{and} \wedge$
$\langle \textsc{BoundVar} \rangle x \wedge$
$\langle \textsc{List} \rangle (e_1 \wedge \mathrm{elem} \wedge$
$\langle \textsc{Item} \rangle x_1 \wedge$
$\langle \textsc{Next} \rangle (\ldots e_n \wedge \mathrm{elem} \wedge \langle \textsc{Item} \rangle x_n \ldots))$
$\langle \textsc{Pred} \rangle e) =$
$[x | x = x_1]; \tau(x_1); \tau(e); \ldots; [x | x = x_n]; \tau(x_n); \tau(e)$

For disjunctive lists, a similar clause may be given that connects subordinate boxes with $\vee$.

Finally, to translate the pairs that arise with argument clusters and gapping, as in examples (55) and (58), clause (75) may be augmented as follows:

(76) $\tau(s \wedge \mathbf{and} \wedge$
$\langle \textsc{BoundVar} \rangle x \wedge$
$\langle \textsc{List} \rangle (e_1 \wedge \mathrm{elem} \wedge$
$\langle \textsc{Item} \rangle (t_1 \wedge \mathrm{tup} \wedge \langle \textsc{Item1} \rangle x_1 \wedge \langle \textsc{Item2} \rangle y_1) \wedge$
$\langle \textsc{Next} \rangle (\ldots e_n \wedge \mathrm{elem} \wedge$
$\langle \textsc{Item} \rangle (t_n \wedge \mathrm{tup} \wedge$
$\langle \textsc{Item1} \rangle x_n \wedge \langle \textsc{Item2} \rangle y_n) \ldots))$
$\langle \textsc{Pred} \rangle e) =$
$[x, y | x = x_1,\ y = y_1]; \tau(x_1); \tau(y_1); \tau(e); \ldots;$
$[x, y | x = x_n,\ y = y_n]; \tau(x_n); \tau(y_n); \tau(e)$

7. Efficiency

To test whether the realizer's speed is in the right ballpark for dialogue applications, we have measured its performance on a pre-existing set of 46 test phrases and accompanying logical forms – including 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, topicalization, NP coordination, clausal coordination (including verb clusters and right node raising), and argument cluster coordination. The phrases average 8.3 words in length, and vary from a minimum of four words to a maximum of 16 words. The number of nodes in the input logical forms averages 6.7.

Table I. Realizer timing (in seconds)

	Mean			Max		
	First ($\pm\sigma$)	Best ($\pm\sigma$)	All ($\pm\sigma$)	First	Best	All
−Ind,−Chunk,−Pru	0.382 (±0.694)	0.385 (±0.693)	10.5 (±37.4)	3.82	3.82	**235**
+Ind,−Chunk,−Pru	0.132 (±0.136)	0.134 (±0.137)	1.07 (±1.94)	0.554	0.554	10.9
+Ind,+Chunk,−Pru	**0.067** (±0.044)	**0.067** (±0.044)	0.327 (±0.527)	0.211	0.211	3.33
+Ind,−Chunk,+Pru3	0.132 (±0.136)	0.134 (±0.137)	0.919 (±1.45)	0.561	0.561	6.77
+Ind,+Chunk,+Pru3	0.066 (±0.042)	0.067 (±0.042)	0.266 (±0.326)	0.206	0.206	1.83
+Ind,+Chunk,+Pru2	0.066 (±0.041)	0.066 (±0.041)	0.250 (±0.283)	0.189	0.189	1.44
+Ind,+Chunk,+Pru1	0.065 (±0.038)	0.065 (±0.038)	0.174 (±0.153)	**0.152**	**0.152**	0.809

For each test phrase, we timed how long it took on a 2.2 GHz Linux PC to realize each logical form using various realizer configurations.[20] To rank candidate realizations, we used a modified version[21] of the Bleu n-gram precision metric (Papineni et al., 2001) employed in machine translation evaluation, using 1- to 4-grams, with the longer n-grams given more weight. The n-gram precision scores were computed against just the target phrase, a technique which we have found to be very useful for regression testing the grammar. In practice, of course, one is not likely to have n-grams available that so precisely guide realization; towards the end of this section, we address the question of whether similar performance can be expected in more realistic settings.

The results of timing the realizer on this test suite appear in Tables I and II. Table I shows the realization times, in seconds, under a series of configurations, while Table II shows how the phrases involving coordination compare to those not involving coordination. Note that under all configurations, the best scoring realization exactly matched the target phrase in every case; in the general case though, pruning can prevent the target phrase from ever being found, if applied too aggressively. Additionally, and somewhat surprisingly, we found that under all configurations, the best scoring realization was either found first or shortly thereafter, with little difference between the time to find the first complete realization and the time to find the best realization.

Each row of Table I shows the amount of time it takes to find the first complete realization, the best scoring realization, and all realizations, both on average and in the worst case.[22] With the means, the standard deviations are also given in parentheses. The first two rows show the effect of indexing. The index filter cuts the mean time until the first realization is found by nearly a third, and lessens the maximum time until the first

Table II. Realizer timing: coordination comparison

+Ind,+Chunk,+Pru3	Mean			Max		
	First ($\pm\sigma$)	Best ($\pm\sigma$)	All ($\pm\sigma$)	First	Best	All
All ($\overline{n}=6.7$)	0.066 ($\pm$0.042)	0.067 ($\pm$0.042)	0.266 ($\pm$0.326)	0.206	0.206	1.83
Coord only ($\overline{n}=10.1$)	0.103 ($\pm$0.046)	0.103 ($\pm$0.046)	0.236 ($\pm$0.122)	0.218	0.218	0.503
No coord ($\overline{n}=5.2$)	0.060 ($\pm$0.038)	0.061 ($\pm$0.038)	0.300 ($\pm$0.380)	0.169	0.169	1.81

realization is found by about a factor of seven. As expected, the index filter also drastically reduces the time to find all realizations, by preventing unrelated noun phrases from combining in a factorial number of ways (cf. section 5.3); in particular, the maximum time to find all realizations drops from 235 s to under 11s.

The second two rows show that the chunking rules and edge pruning independently reduce the realization times. In particular, the third row shows that the chunking rules reduce the mean time to find the first realization from 0.132 ($\pm$0.136) to 0.067 ($\pm$0.044) s. The fourth row shows that *n*-best edge pruning with a pruning value of 3 – i.e., with no more than three signs kept in the chart per equivalent category – has a negligible effect on the time to find the first realization, but does noticeably reduce the time to find all realizations. The final three rows demonstrate that edge pruning can work well in combination with the chunking rules, with the last row showing that the maximum time to find the first realization goes down to 0.152 s when chunking is used in combination with a pruning value of 1.

Even with edge pruning turned on, there remain substantial differences between the mean and maximum times to find the best realization and the mean and maximum times to find all realizations. Thus, to keep realization times consistently low, the anytime search can be stopped well prior to the completion of the chart, as long as at least one complete realization has been found. One way to do so is to employ a *new best* time limit, which caps the amount of time to look for a better scoring realization after finding the first complete one.

In Table II, the realization times for the configuration with indexing, chunking and three-best pruning are compared against those obtained in the same configuration, but on just the examples involving coordination, and just those not involving coordination.[23] There are 14 test phrases involving some form of coordination, with an average of 10.2 words per phrase, a range of 6–16 words, and an average of 10.1 input nodes ($\overline{n}$ in the table). The remaining 32 test phrases average 7.5 words in length, with a range of 4–15 words, and an average of 5.2 input nodes. Taking the

difference in the average number of input nodes into account, the presence of coordination appears to have little effect on realization times. For example, for the coordination cases, the mean time to find the best realization divided by the mean number of input nodes is 10.2 ms per input node, while for the cases not involving coordination, the number is 11.5 ms per node. If anything, the phrases involving coordination do appear to show less variance than the phrases of comparable size that do not involve coordination. That performance should be roughly comparable irrespective of the presence of coordination is not surprising, since the efficiency methods are all quite general. Nevertheless, it is a welcome result that coordinate structures – even those involving non-standard constituents – can be realized as efficiently as ones not involving coordination.

As mentioned earlier in this section, the use of *n*-gram precision scores derived from the target phrase is not realistic from an application perspective – if the target were already known, there would seem to be little point in generating it. In White (2004), we have used such *n*-gram precision scores as a topline, and examined whether similar performance can be obtained with *n*-gram models derived in a cross-validation setup. Using 5-gram backoff models with semantic class replacement, created with the SRI language modeling toolkit (Stolcke, 2002), we observed much of the same performance gains as with the topline scoring method, and substantially better performance than either of two baselines employing no *n*-gram scoring (see White, 2004 for details).

The performance figures in Table I suggest that the realizer is fast enough for practical use in natural language dialogue systems; indeed, the OpenCCG realizer has been deployed in two prototype dialogue systems (den Os and Boves, 2003; Moore et al., 2004) to date, where realization times have been satisfactory. While we expect that performance may vary substantially with different grammars, the empirical observation that the best scoring realizations appear first – or soon after – suggests that one could realize sentences quickly enough for interactive use even with wider coverage grammars. Whether the approach would continue to work equally well with less fully specified input logical forms, however, is less clear.

There are several ways in which the realizer's efficiency could be further improved. First, the unification of feature structures could be optimized along the lines of Malouf et al. (2000). While the implementations of the combinatory rules have been optimized (Baldridge, 2002), unification is otherwise naïve and performs more copying than necessary. In principle, it should also be possible to employ techniques for "packing" local ambiguities (Shemtov, 1997; Langkilde-Geary, 2002); in White (2004), a simpler alternative is proposed, namely using cached category combinations. Finally, the algorithm could benefit from employing more top-down constraints, as in semantic head–driven approaches to realization (Shieber

et al., 1990; Hoffman, 1995). To that end, as noted in section 4.5, a more advanced scheme for using features to license and instantiate categories is proposed in White (2004), which yields some of the benefits of such top-down constraints, without changing the essentially bottom-up nature of the chart realization algorithm.

8. Conclusion

Our approach to chart realization with CCG is closely related to that of Carroll et al. (1999), which in turn builds upon Kay (1996) and earlier work cited therein. Compared to Carroll et al., we have employed a similar but more straightforward approach to semantic construction than the one formalized in Copestake et al. (2001), since we do not allow underspecification of the logical scope of quantifiers,[24] and since there is no need for special treatment of external arguments to handle control phenomena in CCG. Additionally, rather than delaying the insertion of intersective modifiers until a second realization phase, as in Carroll et al.'s approach, we have proposed the use of LF chunking rules to reduce the proliferation of semantically incomplete phrases, a technique which fits well with our novel edge pruning and anytime search methods.

In this article, we have presented a case study showing how our algorithm can be used to efficiently realize a wide range of coordination phenomena, including argument cluster coordination and gapping. We have also presented initial performance tests indicating that the realizer is fast enough for practical use in dialogue systems. To date, the OpenCCG realizer has been deployed in two prototype dialogue systems (den Os and Boves, 2003; Moore et al., 2004), where realization times have been satisfactory.

In ongoing work, we are investigating techniques for handling Steedman's (2000a) approach to information structure and intonation. We also plan to investigate new techniques for coupling CCG realization with higher level planning components. One appealing direction is to see whether the present approach to coordination can simplify the treatment of aggregation in content planning components used in conjunction with the realizer. Since current bottom-up approaches to aggregation such as (Dalianis, 1996; 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 can 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. Initial steps in this direction are reported in Foster and White (2004).

Acknowledgements

Many thanks to Jason Baldridge for help in getting started with OpenCCG, and for co-authoring (White and Baldridge, 2003), which the present article builds upon. Thanks also to Mark Steedman, Geert-Jan Kruijff, Johanna Moore, Jon Oberlander, Mary Ellen Foster, Ann Copestake, John Beavers, Johan Bos, and the anonymous reviewers for helpful discussion. This work was supported in part by the COMIC (IST-2001-32311) and FLIGHTS (EPSRC-GR/R02450/01) projects.

Notes

[1] We prefer the term chart realization over chart generation, which has also appeared in the literature, since surface realization is just one part of the overall task of natural language generation.
[2] `http://openccg.sourceforge.net/`
[3] In practice, the type raising rules are constrained to apply to certain specific categories, depending on the grammar, such as np and pp.
[4] The particular choice of dependency roles does not matter to the realization algorithm.
[5] Here we are glossing over the detail that logic variables in lexical entries need to be replaced with fresh variables to avoid name clashes across entries, e.g. between the two appearances of x in (10).
[6] With λ-semantics, the same result can be achieved by selectively leaving λ-terms for coordinate structures unreduced.
[7] At present, certain syntactic choices may be left open by underspecifying semantic features, and the realizer can make some lexical choices, e.g. choosing between lexicalizing a combination of EPs such as the predicate **see** and the past tense feature as the single word *saw*, or using the auxiliary and base form, *did … see*.
[8] Alternatively, edges for semantically null lexical items could be added to the chart, but doing so would require adding special cases for these items in steps 4b and c in Figure 2.
[9] For unification purposes, nominals are treated as atoms during realization, rather than as logic variables.
[10] Note that each use of a lexical item gives rise to a distinct index nominal (similarly to DRT), so (43) involves isomorphic EPs, rather than duplicate ones.
[11] Cf. (Prevost, 1995) for a related use of unreduced λ-terms in the context of representing information structural units.
[12] The same control over the realizer's coordination decisions remains even when equivalent logical forms are involved, as would be the case if *a musician that …* were replaced with *the musican that …*
[13] This category is forwards type-raised, for subjects; there is also a backwards type-raised version, for objects.
[14] In the collective reading, also plausible in (47), x can simply be unified with a set-valued referent for *Bob and Gil*; with *Bob or Gil*, in contrast, only the distributive reading is possible.
[15] To make the EPs for the conjunction form a connected sub-graph, as required by the realizer's lexical lookup algorithm, the nominals s and x_2 may be connected by a relation marking the end of the list.

[16] To facilitate the creation of LFs such as (60), OpenCCG includes a routine to convert the flat list of EPs returned by the parser into hierarchical form, together with a customizable transformation for raising shared arguments in coordinate structures up to their desired location.

[17] The features may also be considered simple nodes with no arcs emanating from them, to bring these structures closer to Blackburn's (2000) relational structures for hybrid modal logic.

[18] With this basic clause, the order in which the relations are translated is assumed to be inconsequential. Also, to make the translation process easier to understand, we tolerate the use of the discourse referents $x_1 \ldots x_n$ in the translation of the node's relations, even though these referents may not become accessible until the recursive translations $\tau(x_1); \ldots ; \tau(x_n)$ are merged.

[19] The DRS in (74) requires a semantics that allows reassignment of discourse referents, as in Muskens (1996).

[20] Running the tests under different Linux and Windows Java virtual machines did not appear to change the relative timings.

[21] Our version did not include the bells and whistles intended to make cheating the Bleu metric more difficult. Also, the individual n-gram scores were combined using rank-order centroid weights, rather than the geometric mean, so as to avoid problems with combining precision scores of zero.

[22] More precisely, the "All" columns indicate the amount of time until the agenda is emptied, since pruning may prevent some realizations from ever being found.

[23] Since these timings were obtained in separate runs, there is some minor variation in the observed maximum times. Also, we chose to use a pruning value of 3, which we expect to be more typically employed than a pruning value of 1.

[24] Cf. Steedman (1999, 2003) for discussion.

References

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

Baldridge J., Kruijff G.-J. (2002) Coupling CCG and Hybrid Logic Dependency Semantics. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pp. 319–326.

Baldridge J., Kruijff G.-J. (2003) Multi-Modal Combinatory Categorial Grammar. In *Proceedings of 10th Annual Meeting of the European Association for Computational Linguistics*.

Blackburn P. (2000) Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto. *Logic Journal of the IGPL*, 8/3, 339–625.

Carroll J., Copestake A., Flickinger D., Poznański V. (1999) An Efficient Chart Generator for (Semi-) Lexicalist Grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pp. 86–95.

Copestake A., Lascarides A., Flickinger D. (2001) An Algebra for Semantic Construction in Constraint-based Grammars. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics*, pp. 132–139.

Dalianis H. (1996) Concise Natural Language Generation from Formal Specifications. PhD. thesis, Royal Institute of Technology, Stockholm.

den Os E., Boves L. (2003) Towards Ambient Intelligence: Multimodal Computers that Understand our Intentions. In *Proceedings of eChallenges e-2003*.

Elhadad M., Robin J. (1998) SURGE: A Comprehensive Plug-in Syntactic Realization Component for Text Generation. http://www.cs.bgu.ac.il/surge/.

Foster M. E., White M. (2004) Techniques for Text Planning with XSLT. In *Proceedings of NLPXML-2004*.

Geach P. (1972) A Program for Syntax. In Davidson D., Harman G. (eds.), *Semantics of Natural Language*. Reidel, Dordrecht, pp. 483–497.

Geurts B. (2002) Specific Indefinites, Presupposition and Scope. In Bäuerle R., Reyle U., Zimmerman T. E. (eds.), *Presuppositions and Discourse*. Elsevier, Oxford.

Hockenmaier J., Steedman M. (2002) Acquiring Compact Lexicalized Grammars from a Cleaner Treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation*.

Hoffman B. (1995) Computational Analysis of the Syntax and Interpretation of 'Free' Word-order in Turkish. PhD. thesis, University of Pennsylvania. IRCS Report 95–17.

Kamp H., Reyle U. (1993) *From Discourse to Logic*. Kluwer, Dordrecht.

Kay M. (1996) Chart Generation. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. pp. 200–204.

Kruijff G.-J. M. (2001) A Categorial Modal Architecture of Informativity: Dependency Grammar Logic & Information Structure. PhD. thesis, Charles University.

Kruijff G.-J. M. (2003) Binding Across Boundaries. In Kruijff G.-J. M., Oehrle R. T. (eds.), *Resource-Sensitivity in Binding and Anaphora*. Kluwer Academic Publishers, pp. 123–158.

Langkilde I., Knight K. (1998) The Practical Value of *n*-grams in Generation. In *Procedings of the Ninth International Workshop on Natural Language Generation*.

Langkilde-Geary I. (2002) An Empirical Verification of Coverage and Correctness for a General-Purpose Sentence Generator. In *Proceedings of the Second International Natural Language Generation Conference*.

Lavoie B., Rambow O. (1997) RealPro – A Fast, Portable Sentence Realizer. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*.

Malouf R., Carroll J., Copestake A. (2000) Efficient Feature Structure Operations Without Compilation. *Natural Language Engineering,* 6/1, pp. 29–46.

Moore J., Foster M. E., Lemon O., White M. (2004) Generating Tailored, Comparative Descriptions in Spoken Dialogue. In *Proceedings of FLAIRS-04*.

Moore R. C. (1989) Unification-based Semantic Interpretation. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pp. 33–41.

Moore R. C. (2002) A Complete, Efficient Sentence-Realization Algorithm for Unification Grammar. In *Proceedings of the 2nd International Natural Language Generation Conference*.

Muskens R. (1996) Combining Montague Semantics and Discourse Representations. *Linguistics and Philosphy,* 19/2, 143–186.

Papineni K., Roukos S., Ward T., Zhu W.-J. (2001) Bleu: a Method for Automatic Evaluation of Machine Translation. Technical Report RC22176, IBM.

Prevost S. (1995) A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation. PhD. thesis, University of Pennsylvania. IRCS TR 96–01.

Shaw J. (1998) Clause Aggregation Using Linguistic Knowledge. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, pp. 138–148.

Shemtov H. (1997) Ambiguity Management in Natural Language Generation. PhD. thesis, Stanford University.

Shieber S. (1988) A Uniform Architecture for Parsing and Generation. In *Proceedings of the 14th International Conference on Computational Linguistics*, pp. 614–619.

Shieber S., van Nord G., Pereira F., Moore R. (1990) Semantic-head–driven generation. *Computational Linguistics,* 16/1, 30–42.

Steedman M. (1999) Quantifier Scope Alternation in CCG. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 301–308.

Steedman M. (2000a) Information Structure and the Syntax-Phonology Interface. *Linguistic Inquiry*, 31/4, 649–689.

Steedman M. (2000b) *The Syntactic Process*. MIT Press, Cambridge.

Steedman M. (2003) Scope Alternation and the Syntax/Semantics Interface. Manuscript, draft 4.1.

Stolcke A. (2002) SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of ICSLP-02*.

Varges S., Mellish C. (2001) Instance-based Natural Language Generation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics,* pp. 1–8.

White M. (2004) Reining in CCG Chart Realization. In *Proceedings of the Third International Natural Language Generation Conference*.

White M., Baldridge J. (2003) Adapting Chart Realization to CCG. In *Proceedings of the 9th European Workshop on Natural Language Generation*.