CCG Chart Realization from Disjunctive Inputs

Michael White
Department of Linguistics
The Ohio State University
Columbus, OH 43210 USA
http://www.ling.ohio-state.edu/~mwhite/

Abstract

This paper presents a novel algorithm for efficiently generating paraphrases from disjunctive logical forms. The algorithm is couched in the framework of Combinatory Categorial Grammar (CCG) and has been implemented as an extension to the OpenCCG surface realizer. The algorithm makes use of packed representations similar to those initially proposed by Shemtov (1997), generalizing the approach in a more straightforward way than in the algorithm ultimately adopted therein.

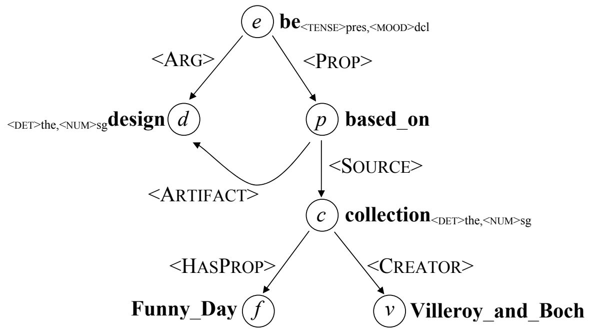
1 Introduction

In recent years, the generate-and-select paradigm of natural language generation has attracted increasing attention, particularly for the task of surface realization. In this paradigm, symbolic methods are used to generate a space of possible phrasings, and statistical methods are used to select one or more outputs from this space. To specify the desired paraphrase space, one may either provide an input logical form that underspecifies certain realization choices, or include *explicit disjunctions* in the input LF (or both). Our experience suggests that disjunctive LFs are an important capability, especially as one seeks to make grammars reusable across applications, and to employ domain-specific, *sentence-level* paraphrases (Barzilay and Lee, 2003).

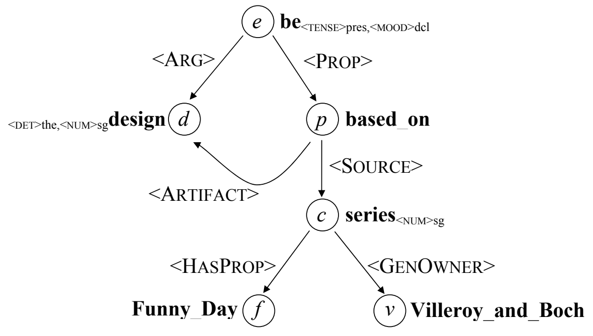
Prominent examples of surface realizers in the generate-and-select paradigm include Nitrogen/Halogen (Langkilde, 2000; Langkilde-Geary, 2002) and Fergus (Bangalore and Rambow, 2000). More recently, generate-and-select realizers in the *chart realization* tradition (Kay, 1996) have appeared, including the OpenCCG (White, 2004) and LinGO (Carroll and Oepen, 2005) realizers. Chart realizers make it possible to use the same reversible grammar for both parsing and realization, and employ well-defined methods of semantic composition to construct semantic representations that can properly represent the scope of logical operators.

In the chart realization tradition, previous work has not generally supported disjunctive logical forms, with (Shemtov, 1997) as the only published exception (to the author's knowledge). Arguably, part of the reason that disjunctive LFs have not yet been embraced more broadly by those working on chart realization is that Shemtov's solution, while ingenious, is dauntingly complex. Looking beyond chart realizers, both Nitrogen/Halogen and Fergus support some forms of disjunctive input; however, in comparison to Shemtov's inputs, theirs are less expressive, in that they do not allow disjunctions across different levels of the input structure.

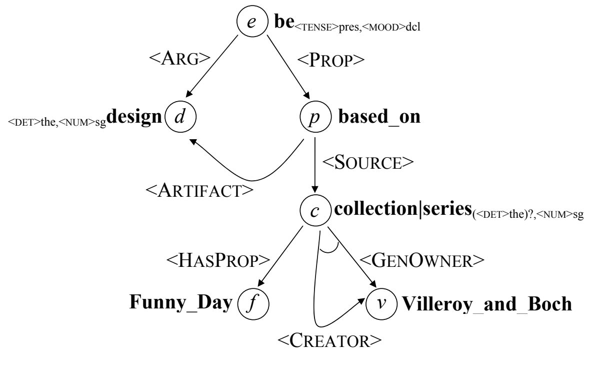
As an alternative to Shemtov's method, this paper presents a chart realization algorithm for generating paraphrases from disjunctive logical forms that is more straightforward to implement, together with an initial case study of the algorithm's efficiency. As discussed in Section 5, the algorithm makes use of packed representations similar to those initially proposed by Shemtov, generalizing the approach in a way that avoids the problems that led Shemtov to reject his preliminary method. The algorithm is couched in the framework of Steedman's (2000) Combinatory Categorial Grammar (CCG) and has been implemented as an extension to the OpenCCG surface realizer. Though the algorithm is well suited to CCG, it is expected to be applicable to other constraint-based grammatical frameworks as well.



(a) Semantic dependency graph for *The design (is|'s) based on the Funny Day collection by Villeroy and Boch.*



(b) Semantic dependency graph for *The design (is|'s) based on Villeroy and Boch's Funny Day series.*



(c) Disjunctive semantic dependency graph covering (a)-(b), i.e. *The design (is|'s) based on (the Funny Day (collection|series) by Villeroy and Boch | Villeroy and Boch's Funny Day (collection|series)).*

Figure 1: Example semantic dependency graphs from the COMIC dialogue system.

$$
\begin{aligned}
&@_e(\textbf{be} \wedge \langle\textsc{tense}\rangle\text{pres} \wedge \langle\textsc{mood}\rangle\text{dcl} \wedge \\
&\quad \langle\textsc{Arg}\rangle(d \wedge \textbf{design} \wedge \langle\textsc{det}\rangle\text{the} \wedge \langle\textsc{num}\rangle\text{sg}) \wedge \\
&\quad \langle\textsc{Prop}\rangle(p \wedge \textbf{based\_on} \wedge \\
&\qquad\quad \langle\textsc{Artifact}\rangle d \wedge \\
&\qquad\quad \langle\textsc{Source}\rangle(c \wedge \textbf{collection} \wedge \langle\textsc{det}\rangle\text{the} \wedge \langle\textsc{num}\rangle\text{sg} \wedge \\
&\qquad\qquad\quad \langle\textsc{HasProp}\rangle(f \wedge \textbf{Funny\_Day}) \wedge \\
&\qquad\qquad\quad \langle\textsc{Creator}\rangle(v \wedge \textbf{V\&B}))))
\end{aligned}
$$

(a)

⋮

$$
\begin{aligned}
&@_e(\textbf{be} \wedge \langle\textsc{tense}\rangle\text{pres} \wedge \langle\textsc{mood}\rangle\text{dcl} \wedge \\
&\quad \langle\textsc{Arg}\rangle(d \wedge \textbf{design} \wedge \langle\textsc{det}\rangle\text{the} \wedge \langle\textsc{num}\rangle\text{sg}) \wedge \\
&\quad \langle\textsc{Prop}\rangle(p \wedge \textbf{based\_on} \wedge \\
&\qquad\quad \langle\textsc{Artifact}\rangle d \wedge \\
&\qquad\quad \langle\textsc{Source}\rangle(c \wedge \langle\textsc{num}\rangle\text{sg} \wedge (\langle\textsc{det}\rangle\text{the})? \wedge \\
&\qquad\qquad\quad (\textbf{collection} \veebar \textbf{series}) \wedge \\
&\qquad\qquad\quad \langle\textsc{HasProp}\rangle(f \wedge \textbf{Funny\_Day}) \wedge \\
&\qquad\qquad\quad (\langle\textsc{Creator}\rangle\boxed{v} \veebar \langle\textsc{GenOwner}\rangle\boxed{v})))) \\
&\wedge\ @_v(\textbf{Villeroy\_and\_Boch})
\end{aligned}
$$

(c)

Figure 2: HLDS for examples in Figure 1.

2 Disjunctive Logical Forms

As an illustration of disjunctive logical forms, consider the semantic dependency graphs in Figure 1, which are taken from the COMIC[1] multimodal dialogue system.[2] Graphs such as these constitute the input to the OpenCCG realizer. Each node has a lexical predication (e.g. **design**) and a set of semantic features (e.g. ⟨NUM⟩sg); nodes are connected via dependency relations (e.g. ⟨ARTIFACT⟩).

Given the lexical categories in the COMIC grammar, the graphs in Figure 1(a) and (b) fully specify their respective realizations, with the exception of the choice of the full or contracted form of the copula. To generalize over these alternatives, the disjunctive graph in (c) may be employed. This graph allows a free choice between the domain synonyms *collection* and *series*, as indicated by the vertical bar between their respective predications. The graph also allows a free choice between the ⟨CREATOR⟩ and ⟨GENOWNER⟩ relations—lexicalized via *by* and the possessive, respectively—connecting the head c (*collection* or *series*) with the dependent v (for

[1]`http://www.hcrc.ed.ac.uk/comic/`

[2]To simplify the exposition, the features specifying information structure and deictic gestures have been omitted, as have the semantic sorts of the discourse referents.

$$\begin{array}{l}
@_e(\textbf{see} \wedge \langle \textsc{Arg0} \rangle (m \wedge \textbf{man}) \wedge \langle \textsc{Arg1} \rangle (g \wedge \textbf{girl}) \wedge \\
\quad @_o(\textbf{on} \wedge \langle \textsc{Arg1} \rangle (h \wedge \textbf{hill})) \wedge @_w(\textbf{with} \wedge \langle \textsc{Arg1} \rangle (t \wedge \textbf{telescope})) \wedge \\
\quad (((\langle \textsc{Mod} \rangle o \wedge @_h(\langle \textsc{Mod} \rangle w)) \mathbin{\underline{\vee}} \\
\quad\; (@_g(\langle \textsc{Mod} \rangle o) \wedge (@_g(\langle \textsc{Mod} \rangle w) \mathbin{\underline{\vee}} @_h(\langle \textsc{Mod} \rangle w))) \mathbin{\underline{\vee}} \\
\quad\; (\langle \textsc{Mod} \rangle w \wedge (\langle \textsc{Mod} \rangle o \mathbin{\underline{\vee}} @_g(\langle \textsc{Mod} \rangle o)))))
\end{array}$$

Figure 3: Disjunctive LF for 5-way ambiguity in *A man saw a girl on the hill with a telescope.*

Villeroy and Boch); this choice is indicated by an arc between the two dependency relations. Finally, the determiner feature ($\langle$DET$\rangle$the) on c is indicated as optional, via the question mark.

It is worth pausing at this point to observe that in designing the COMIC grammar, the differences between (a) and (b) could perhaps have been collapsed. However, such a move would make it more difficult to reuse the grammar in other applications—and indeed, the core of the grammar is shared with the FLIGHTS system (Moore et al., 2004)—as it would presuppose that these paraphrases should always available in the same contexts. An example of a *sentence-level* paraphrase, whose context of applicability is more clearly limited, appears in (1):

(1) (This design | This one | This) (is|'s) (classic | in the classic style) | Here we have a (classic design | design in the classic style).

This example shows some of the phrasings that may be used in COMIC to describe the style of a design that has not been discussed previously. The example includes a top-level disjunction between the use of a deictic NP *this design* | *this one* | *this* (with an accompanying pointing gesture) followed by the copula, or the use of the phrase *here we have* to introduce the design. While these alternatives can function as paraphrases in this context, it is difficult to see how one might specify them in a single underspecified (and application-neutral) logical form.

Graphs such as those in Figure 1 are represented internally using Hybrid Logic Dependency Semantics (HLDS), as in Figure 2. HLDS is a dependency-based approach to representing linguistic meaning developed by Baldridge and Kruijff (2002). In HLDS, hybrid logic (Blackburn, 2000) terms[3] are used to describe dependency graphs. These graphs have been suggested as representations for discourse structure, and have their own underlying semantics (White, 2006).

In HLDS, as can be seen in Figure 2(a), 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. Modal relations are also used to represent semantic features. In (c), two new operators are introduced to represent periphrastic alternatives and optional parts of the meaning, namely $\underline{\vee}$ and $(\cdot)?$, for exclusive-or and optionality, respectively. To indicate that a nominal represents a reference to a node that is considered a shared part of multiple alternatives, the nominal is annotated with a box, as exemplified by $\boxed{v}$. As will be discussed in Section 3.1, this notion of shared references is needed during the logical form flattening stage of the algorithm in order to determine which elementary predications are part of each alternative.

As mentioned earlier, disjunctive LFs may contain alternations that are not at the same level. To illustrate, Figure 3 shows the representation (minus semantic features) for the 5-way ambiguity in *A man saw a girl on the hill with a telescope* (Shemtov, 1997, p. 45); in the figure, the nominal o (for *on*) can be a dependent of e (for *see*) or g (for *girl*), for example. As Shemtov explains, such packed representations can be useful in machine translation for generating ambiguity-preserving target language sentences. In a straight generation context, disjunctions that span levels enable one to compactly represent alternatives that differ in their head-dependent assumptions; for instance, to express contrast, one might employ the coordinate conjunction *but* as the sentence head, or the subordinate conjunction *although* as a dependent of the main clause head.

3 The Algorithm

As with the other chart realizers cited in the introduction, the OpenCCG realizer makes use of a chart and an agenda to perform a bottom-up dynamic programming search for signs whose LFs

[3]Hybrid logic extends modal logic with *nominals*, a new sort of basic formula that explicitly names states/nodes. Like propositions, nominals are first-class citizens of the object language, and thus formulas can be formed using propositions, nominals, and standard boolean operators. They may also employ the *satisfaction operator*, @. A formula $@_i(p \wedge \langle F \rangle (j \wedge q))$ indicates that the formulas p and $\langle F \rangle (j \wedge q)$ 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.

completely cover the elementary predications in the input logical form. The search for complete realizations proceeds in one of two modes, anytime or two-stage packing/unpacking. This section focuses on how the two-stage mode has been extended to efficiently generate paraphrases from disjunctive logical forms.

3.1 LF Flattening

In a preprocessing stage, the input logical form is flattened to an array of elementary predications (EPs), one for each lexical predication, semantic feature or dependency relation. When the input LF contains no exclusive-or or optionality operators, the list of EPs, when conjoined, yields a graph description that is equivalent to the original one. With disjunctive logical forms, however, more needs to be said. Our strategy is to keep track of the elementary predications that make up the alternatives and optional parts of the LF, as specified by the exclusive-or or optionality operators, and use these to enforce constraints on the elementary predications that may appear in any given realization. These constraints ensure that only combinations of EPs that describe a graph that is also described by the original LF are allowed.

To illustrate, the results of flattening the LF in Figure 2(c) are given below:

(2) $0{:}\ @_e(\textbf{be}),\ 1{:}\ @_e(\langle\text{TENSE}\rangle\text{pres}),$
$2{:}\ @_e(\langle\text{MOOD}\rangle\text{dcl}),\ 3{:}\ @_e(\langle\text{ARG}\rangle d),$
$4{:}\ @_d(\textbf{design}),\ 5{:}\ @_d(\langle\text{DET}\rangle\text{the}),$
$6{:}\ @_d(\langle\text{NUM}\rangle\text{sg}),$
$7{:}\ @_e(\langle\text{PROP}\rangle p),\ 8{:}\ @_p(\textbf{based\_on}),$
$9{:}\ @_p(\langle\text{ARTIFACT}\rangle d),\ 10{:}\ @_p(\langle\text{SOURCE}\rangle c),$
$11{:}\ @_c(\langle\text{NUM}\rangle\text{sg}),\ 12{:}\ @_c(\langle\text{DET}\rangle\text{the}),$
$13{:}\ @_c(\textbf{collection}),\ 14{:}\ @_c(\textbf{series}),$
$15{:}\ @_c(\langle\text{HASPROP}\rangle f),\ 16{:}\ @_f(\textbf{Funny\_Day}),$
$17{:}\ @_c(\langle\text{CREATOR}\rangle v), 18{:}\ @_c(\langle\text{GENOWNER}\rangle v),$
$19{:}\ @_v(\textbf{Villeroy\_and\_Boch})$

(3) $\mathsf{alt}_{0,0} = \{13\};\ \mathsf{alt}_{0,1} = \{14\}$
$\mathsf{alt}_{1,0} = \{17, 19\};\ \mathsf{alt}_{1,1} = \{18, 19\}$
$\mathsf{opt}_0 = \{12\}$

In (2), the EPs are shown together with their array positions. Since the EPs are tracked positionally, it is possible to use bit vectors to represent the alternatives and optional parts of the LF. In (3), the first line shows the bit vectors[4] for the choice between **collection** (EP 13) and **series** (EP 14), as alternatives 0 and 1 in alternative group 0. On the second line, the bit vectors for the ⟨CREATOR⟩ (EP 17) and ⟨GENOWNER⟩ (EP 18) alternatives appear; note that both of these options also involve the shared EP 19. The bit vector for the optional determiner (EP 12) is shown on the third line.

[4]Only the positive bits are shown, via their indices.

The constraint associated with each group of alternatives is that in order to be valid, a collection of EPs must not intersect with the non-overlapping parts of more than one alternative. For example, for the second group of alternatives in (3), a valid collection could include EPs 17 and 19, or EPs 18 and 19, but it could not include EPs 17 and 18 together.

Flattening an LF to obtain the array of EPs, as in (2), just requires a relatively straightforward traversal of the HLDS formula. Obtaining the alternatives and optional parts of the LF is a bit more involved. To do so, during the traversal, the exclusive-or and optionality operators are handled by introducing a new alternative group or optional part, and then keeping track of which elementary predications fall under each alternative or under the optional part. Subsequently, the alternatives and optional parts are recursively propagated through any nominals marked as shared, collecting any further EPs that turn up along the way.[5] For example, with the second alternative group (second line) of (3), the initial traversal creates EPs 17 and 18 under alts $\mathsf{alt}_{1,0}$ and $\mathsf{alt}_{1,1}$, respectively. Since EPs 17 and 18 both include a nominal dependent v marked as shared in Figure 2(c), both alternatives are propagated through this reference, and thus EP 19 ends up as part of both $\mathsf{alt}_{1,0}$ and $\mathsf{alt}_{1,1}$. Determining which EPs have shared membership in multiple alternatives is essential for accurately tracking an edge's coverage of the input LF, a topic which will be considered next.

3.2 Edges

In the OpenCCG realizer, an *edge* is a data structure that wraps a CCG sign, which itself consists of a word sequence paired with a category (syntactic category plus logical form). An edge has bit vectors to record its coverage of the input LF and its indices, i.e. syntactically available nominals. In packing mode, a *representative* edge also maintains a list of alternative edges whose signs have equivalent categories (but different word sequences), so that a representative edge may effec-

[5]Though space precludes discussion, it is worth noting that the same propagation of membership applies to the LF chunks described in (White, 2006).

tively stand in for the others during chart construction.

To handle disjunctive inputs, an edge additionally maintains a list of active (i.e., partially completed) LF alternatives. It also makes use of a revised notion of input coverage and a revised equivalence relation. As in Shemtov's (1997, Section 3.3.2) preliminary algorithm, an edge is considered to cover an entire disjunction (alternative group) if it covers all the EPs of one of its alternatives. With optional parts of an LF, an edge that does not cover any EPs in the optional part can be extended to a new edge (using the same sign) that is additionally considered to cover all the EPs in the optional part. In this way, an edge can be defined to be *complete* with respect to the input LF if it covers all its EPs. For example, an edge for the sentence in Figure 1(b) would be considered complete, since (i) it would cover all the EPs in (2) except for 12, 13 and 17; (ii) 12 is optional; (iii) 14 completes $\mathsf{alt}_{0,1}$, and thus counts as covering 13, the other EP in the group; and (iv) 18 and 19 complete $\mathsf{alt}_{1,1}$, and thus count as covering EP 17.

As Shemtov points out, this extended notion of input coverage provides an appropriate way to form edge equivalence classes, as it can gather edges together that realize different alternatives in the same group. Thus, in OpenCCG, edge equivalence classes have been modified to include edges with the same syntactic category and coverage bit vector, but different word sequences and/or logical forms (as the latter varies according to which alternative is realized). The appropriate equivalence checks are efficiently carried out using a hash map with a custom hash function and equals method.

3.3 Lexical Instantiation

Once the input LF has been flattened, and the alternatives and optional parts have been identified, the next step is to access and instantiate lexical items. For each elementary predication, all lexical items indexed by the EP's lexical predicate or relation are retrieved from the lexicon.[6] Each such lexical item is then instantiated against the input EPs, starting with the one that triggered its retrieval, and incrementally extending successful instantiations until all the lexical item's EPs have been instantiated (otherwise failing). The lexical instantiation routine returns all instantiations that satisfy the alternative exclusion constraints. Associated with each instantiation is a bit vector that encodes the coverage of the input EPs. From each bit vector, the active (partially completed) LF alternatives are determined, and the bit vector is updated to include the EPs in any completed disjunctions. Finally, edges are created for the instantiated lexical items, which include the active alternatives and the updated coverage vector.

Continuing with example (2)-(3), the selected lexical edges in (4) below illustrate how lexical instantiation interacts with disjunctions:

(4)
a. $\{11,13,14\}$ *collection* $\vdash \mathsf{n}_c$: $@_c(\textbf{collection}) \wedge @_c(\langle\textsc{num}\rangle\mathrm{sg})$

b. $\{11,13,14\}$ *series* $\vdash \mathsf{n}_c$: $@_c(\textbf{series}) \wedge @_c(\langle\textsc{num}\rangle\mathrm{sg})$

c. $\{17\}$ $\mathsf{alt}_{1,0}$ *by* $\vdash \mathsf{n}_c\backslash\mathsf{n}_c/\mathsf{np}_v$: $@_c(\langle\textsc{Creator}\rangle v)$

d. $\{18\}$ $\mathsf{alt}_{1,1}$ *'s* $\vdash \mathsf{np}_c/\mathsf{n}_c\backslash\mathsf{np}_v$: $@_c(\langle\textsc{GenOwner}\rangle v)$

e. $\{19\}$ $\mathsf{alt}_{1,0}; \mathsf{alt}_{1,1}$ *Villeroy_and_Boch* $\vdash \mathsf{np}_v$: $@_v(\textbf{V\&B})$

The nouns in (a) and (b) complete $\mathsf{alt}_{0,0}$ and $\mathsf{alt}_{0,1}$, respectively, and thus they each count as covering EPs 11, 13 and 14. In (c) and (d), *by* and *'s* partially cover $\mathsf{alt}_{1,0}$ and $\mathsf{alt}_{1,1}$, respectively, and thus these alternatives are active for their respective edges. In (e), *V&B* partially covers both $\mathsf{alt}_{1,0}$ and $\mathsf{alt}_{1,1}$, and thus both alternatives are active.

3.4 Derivation

Following lexical instantiation, the lexical edges are added to the agenda, as is usual practice with chart algorithms, and the main loop is initiated. During each iteration of the main loop, an edge is moved from the agenda to the chart. If the edge is in the same equivalence class as an edge already in the chart, it is added as an alternative to the existing representative edge. Otherwise, it is combined with all applicable edges in the chart (via the grammar's combinatory rules), as well as with the grammar's unary rules, where any newly created edges are added to the agenda. The loop terminates when no edges remain on the agenda.

Before edge combinations are attempted, a number of constraints are checked, as detailed in (White, 2006). In particular, the edges' coverage bit vectors are required to not intersect, which ensures that they cover disjoint parts of the input LF. Since the coverage vectors are updated to cover all the EPs in a disjunction when one of the alternatives is completed, this check also ensures that the

[6]See (White, 2004; White, 2006) for discussion of how semantically null lexical items and unary type changing rules are handled.

1. $\{8\text{-}10\}$ *based_on* $\vdash \mathsf{s}_p\backslash\mathsf{np}_d/\mathsf{np}_c$
2. $\{12\}$ *the* $\vdash \mathsf{np}_c/\mathsf{n}_c$
3. $\{15, 16\}$ *Funny_Day* $\vdash \mathsf{n}_c/\mathsf{n}_c$
4. $\{11, 13, 14\}$ *collection* $\vdash \mathsf{n}_c$
   $\{11, 13, 14\}$ *series* $\vdash \mathsf{n}_c$
5. $\{17\}\,\mathsf{alt}_{1,0}$ *by* $\vdash \mathsf{n}_c\backslash\mathsf{n}_c/\mathsf{np}_v$
6. $\{18\}\,\mathsf{alt}_{1,1}$ *'s* $\vdash \mathsf{np}_c/\mathsf{n}_c\backslash\mathsf{np}_v$
7. $\{19\}\,\mathsf{alt}_{1,0};\mathsf{alt}_{1,1}$ *Villeroy_and_Boch* $\vdash \mathsf{np}_v$
8. $\{11, 13\text{-}16\}$ *FD [collection]* $\vdash \mathsf{n}_c$ $(3\ 4\ >)$
9. $\{17\text{-}19\}$ *by V&B* $\vdash \mathsf{n}_c\backslash\mathsf{n}_c$ $(5\ 7\ >)$
10. $\{17\text{-}19\}$ *V&B 's* $\vdash \mathsf{np}_c/\mathsf{n}_c$ $(7\ 6\ <)$
11. $\{11, 13\text{-}19\}$ *FD [coll.] by V&B* $\vdash \mathsf{n}_c$ $(8\ 9\ <)$
12. $\{11, 13\text{-}19\}$ *V&B 's FD [coll.]* $\vdash \mathsf{np}_c$ $(10\ 8\ >)$
13. $\{11\text{-}19\}$ *the FD [coll.] by V&B* $\vdash \mathsf{np}_c$ $(2\ 11\ >)$
    $\{11\text{-}19\}$ *V&B 's FD [coll.]* $\vdash \mathsf{np}_c$ (12 optC)
14. $\{8\text{-}19\}$ *b._on [the FD [coll.] ...]* $\vdash \mathsf{s}_p\backslash\mathsf{np}_d$ $(1\ 13\ >)$

Figure 4: Part of realization chart for Figure 1(c).

exclusion constraints for the disjunction continue to be enforced. Thus, for example, no attempt will be made to combine the edges for *collection* and *series* in (4a) and (4b), since they both express EP 11 and since they contribute to different alternatives in group 0.

To enforce the constraints associated with active alternatives, a compatibility check is made to ensure that if the input edges have active alternatives in the same group, the intersection of these alternatives is non-empty. To illustrate, consider the edges for *by* and the possessive *'s* in (4c) and (4d). Since these edges have different alternatives active within group 1, the compatibility check fails, and thus their combination is not attempted. By contrast, the edge for *Villeroy and Boch* in (4e) will pass the compatibility check with both (4c) and (4d), as it shares an active alternative in common with each of these. When two edges succeed in combining, a new edge is constructed from the resulting sign by taking the union of the coverage bit vectors, determining the active alternatives, and updating the coverage vector to include the EPs in any completed disjunctions.

When the grammar's unary rules are applied to an edge, an operation is also invoked for creating an edge (for the same sign) with one or more optional parts marked as completed. This operation is invoked when it would complete the input LF, complete an alternative, or complete an LF chunk.[7] A constraint on its application is that the optional parts must be wholly missing from the input edge; additionally, in the case of completing an alternative or LF chunk, the optional parts must be part of the alternative or chunk in question.

Figure 4 demonstrates how the lexical edges in (4) are combined in the chart.[8] These lexical edges appear on lines 4-7. Note that the edge for *series* is added as an alternative edge to the one for *collection*, which acts as a representative for both; to highlight its role as a representative, *collection* is shown in square brackets from line 8 onwards. At the end of each line, the derivation of each (non-lexical) edge is shown in parentheses, in terms of its input edges and combinatory rule. On line 13, observe that the NP using the possessive is added as an alternative to the one using the *by*-phrase; the possessive version becomes part of the same equivalence class when the optional determiner is marked as covered, via the optional part completion operation.

3.5 Unpacking

Once chart construction has finished, the complete realizations are recursively unpacked bottom-up in a way that generalizes the approach of (Langkilde, 2000). Unpacking proceeds by multiplying out the alternative edges stored with the representative input edges; filtering out any duplicate edges resulting from spurious ambiguities; scoring the new edges with the scoring method configured via the API; and pruning the results with the configured pruning strategy. Note that since there is no need for checking grammatical or other constraints during the unpacking stage, new edges can be quickly and cheaply constructed using structure sharing.

To briefly illustrate the process, consider how the *Funny_Day collection* edge in line 8 of Figure 4 is unpacked. While the *Funny_Day* input edge has no alternative edges, the *collection* input edge has the *series* edge as an alternative, and thus a new *Funny_Day series* edge will be created and scored; as long as the pruning strategy keeps more than the single-best option, this edge will be added as an alternative, and both combinations will be propagated upwards through the edges in lines 11

[7]LF chunks serve to avoid propagating semantically incomplete phrases; see (White, 2006) for discussion.

[8]To save space, the figure only shows part of the normal form derivation, and the logical forms for the categories have been suppressed.

	10-best two-stage		1-best anytime	
	time	edges	time	edges
disjunctive	1.1	602	0.5	281
sequential	5.6	3550	4.1	2854

Table 1: Comparison of average run times (in seconds) and edges created vs. sequential realization

and 12.

4 Case Study

To examine the potential of the algorithm to efficiently generate paraphrases, this section presents a case study of its run times versus sequential realization of the equivalent top-level LF alternatives in disjunctive normal form. The study used the COMIC grammar, a small but not trivial grammar that suffices for the purposes of the system. In this grammar, there are relatively few categories per lexeme on average, but the boundary tone categories engender a great deal of non-determinism. With other grammars, run times can be expected to vary.

In anticipation of the present work, Foster and White (2004) generated disjunctive logical forms during sentence planning, then (as a stopgap measure) multiplied out the disjunctions and sequentially realized the top-level alternatives until an overall time limit was reached. Taking the previous logical forms as a starting point, 104 sentences from the evaluation in (Foster and White, 2005) were selected, and their LFs were manually augmented to cover a greater range of paraphrases allowed by the grammar.[9] To obtain the corresponding top-level LF alternatives, 100-best realization was performed, and the unique LFs appearing in the top 100 realizations were gathered; on average, there were 29 such unique LFs. We then compared the present algorithm's performance against sequential realization in producing 10-best outputs and single-best outputs. In the 10-best case, we used the two-stage packing/unpacking mode; for the single-best case, we used the anytime mode with 3-best pruning. With both cases, the run times include scoring with a trigram language model, and were measured on a 2.8GHz Linux PC. Realization quality was not assessed as part of the study, though manual inspection indicated that it was very high.

Table 1 shows the results of the comparison. The average run times of the present algorithm, with disjunctive LFs as input, appear on the first line, along with the average number of edges created; on the second line are the average aggregate run times and num. edges created of sequentially realizing the top-level alternatives (not including the time taken to produce these alternatives). As can be seen, realization from disjunctive inputs yields a 5-fold and 8-fold speedup over the sequential approach in the two cases, with corresponding reductions in the number of edges created. Additionally, the run times appear to be adequate for use in interactive dialogue systems (especially in the anytime, single-best case).

[9]Extending the COMIC sentence planner to produce these augmented LFs is left for future work.

5 Comparison to Shemtov (1997)

The present approach differs from Shemtov's in two main ways. First, since Shemtov developed his approach with the task of ambiguity preserving translation in mind, he framed the problem as one of generating from *ambiguous* semantic representations, such as one might find in a parse chart with unresolved ambiguities. Consequently, he devised a method for converting the meanings in a packed parse chart into an encoding where each fact (here, EP) appears exactly once, together with an indication of the meaning alternatives it belongs to, expressed as propositional formulas. While this *contexted facts* encoding may be suitable for MT, it is not very convenient as an input representation for systems which generate from non-linguistic data, as the formulas representing the contexts only make sense in reference to a parse chart. By contrast, the present approach takes as input disjunctive logical forms that should be reasonably intuitive to construct in dialogue systems or other NLG applications, since they are straightforwardly related to their non-disjunctive counterparts.

The second way in which the approach differs concerns the relative simplicity of the algorithms ultimately adopted. As part of his preliminary algorithm (Shemtov, 1997, Section 3.3.2), Shemtov proposed the extended use of coverage bit vectors that we embraced in Section 3.2. He then developed a refined version to handle disjunctions with intersecting predicates. However, he concluded that this refined version was arc-consistent but not path-consistent (p. 65, fn. 10), given that it checked combinations of contexted facts pairwise, without keeping track of which alternations such combinations were committed to. By contrast, the present approach does not suffer from this defect, because it checks the alternative exclusion constraints on all of a lexical edge's EPs at once (using bit vectors for both edge coverage and alternative membership), and also ensures that the active alternatives are compatible before combining edges during derivations. Shemtov does not appear to have considered a solution along the lines proposed here; instead, he went on to develop a sound but considerably more complex algorithm (his Section 3.4), where an edge's coverage bit vector is replaced with a contexted coverage array (an array of boolean conditions). With these arrays, it is no longer easy to group edges into equivalence classes, and thus during chart construction Shemtov is forced to group together edges which are not derivationally equivalent. Consequently, to prevent overgeneration, his algorithm has to solve during the enumeration phase a system of constraints (potentially exponential in size) formed from the conditions in the contexted coverage arrays—a process which is far from straightforward.

6 Conclusions

This paper has presented a new chart realization algorithm for efficiently generating surface realizations from disjunctive logical forms, and has argued that the approach represents an improvement over that of (Shemtov, 1997) in terms of both usability and simplicity. The algorithm has been implemented as an extension to the OpenCCG hybrid symbolic/statistical realizer, and has recently been employed to generate n-best realization lists for reranking according to their predicted synthesis quality (Nakatsu and White, 2006), as well as to generate dialogues exhibiting individuality and alignment(Brockmann et al., 2005; Isard et al., 2005). An initial case study has shown that the algorithm works many times faster than sequential realization, with run times suitable for use in dialogue systems; a more comprehensive study of the algorithm's efficiency is planned for future work.

Acknowledgements

The author thanks Mary Ellen Foster, Amy Isard, Johanna Moore, Mark Steedman and the anonymous reviewers for helpful feedback and discussion, and the University of Edinburgh's Institute for Communicating and Collaborative Systems for partially supporting this work.

References

Jason Baldridge and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proc. ACL-02*.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proc. COLING-00*.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proc. of NAACL-HLT*.

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

Carsten Brockmann, Amy Isard, Jon Oberlander, and Michael White. 2005. Modelling alignment for affective dialogue. In *Proc. UM-05 Workshop on Adapting the Interaction Style to Affective Factors*.

John Carroll and Stefan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proc. IJCNLP-05*.

Mary Ellen Foster and Michael White. 2004. Techniques for Text Planning with XSLT. In *Proc. 4th NLPXML Workshop*.

Mary Ellen Foster and Michael White. 2005. Assessing the impact of adaptive generation in the COMIC multimodal dialogue system. In *Proc. IJCAI-05 Workshop on Knowledge and Representation in Practical Dialogue Systems*.

Amy Isard, Carsten Brockmann, and Jon Oberlander. 2005. Individuality and alignment in generated dialogues. In *Proc. INLG-06*. To appear.

Martin Kay. 1996. Chart generation. In *Proc. ACL-96*.

Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proc. INLG-02*.

Irene Langkilde. 2000. Forest-based statistical sentence generation. In *Proc. NAACL-00*.

Johanna Moore, Mary Ellen Foster, Oliver Lemon, and Michael White. 2004. Generating tailored, comparative descriptions in spoken dialogue. In *Proc. FLAIRS-04*.

Crystal Nakatsu and Michael White. 2006. Learning to say it well: Reranking realizations by predicted synthesis quality. In *Proc. of COLING-ACL-06*. To appear.

Hadar Shemtov. 1997. *Ambiguity Management in Natural Language Generation*. Ph.D. thesis, Stanford University.

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

Michael White. 2004. Reining in CCG Chart Realization. In *Proc. INLG-04*.

Michael White. 2006. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language & Computation*, on-line first, March.