Bidirectional Incremental Generation and Analysis with Categorial Grammar and Indexed Quasi-Logical Form.

Torbjörn Lager
Department of Linguistics
University of Göteborg
E-mail: lager@ling.gu.se

William J Black
Centre for Computational Linguistics
UMIST, Manchester
E-mail: bill@ccl.umist.ac.uk

Abstract

We describe an approach to surface generation designed for a "pragmatics-based" dialogue system. The implementation has been extended to deal with certain well-known difficulties with the underlying linguistic formalism (Categorial Grammar) at the same time yielding a system capable of supporting incremental generation as well as interpretation. Aspects of the formalism used for the initial description that constitutes the interface with the planning component are also discussed.

1. Introduction

In a monolingual dialogue system, strong arguments are needed for generation not to reversibly use the same linguistic resources as parsing. We examine several characteristics of an implemented surface generation component deriving from the needs of this application. The generator uses as its linguistic resource a lexicon encoded in a version of Categorial Grammar (CG), the extension of which with rules of function composition gives rise to a problem of spurious overgeneration. As in analysis, these extensions permit incremental processing, and the amelioration of spurious overgeneration is demonstrated to follow identical lines to that in analysis. Interpretation in the PLUS[1] system (supporting dialogues about Yellow Pages information) was carried out abductively (cf. Guessoum et al 1993) starting from an underspecified quasi-logical form. Reversibility required the same formalism to be used for surface generation, the feasibility of which was demonstrated by Phillips (1993). We improve on his earlier version, solving nontermination with modifiers, interfacing to a structured morphological lexicon with efficient lookup, eliminating spurious overgeneration arising from CG's rules of function composition, and enabling incremental generation.

2 Generation from indexed QLF

A working hypothesis of the PLUS project was that strict compositionality provides too many meanings for efficient interpretation. The alternative is to rely on defeasible reasoning over an underspecified (w.r.t lexical, referential, quantificational and attachment ambiguities) representation. On the generation side, we adopt a 3-way split between content (i.e. application dictated) planning with output expressed in terms of standard logical forms (LF), linguistic planning (i.e. "how to say it"), with output expressed in QLF, and realisation. Here, we only discuss the last (Jokinen, 1993 describes the second). The two planning components between them need to be able to exercise full control of the linguistic choices, and do so through the QLF, which includes linguistic features as well as predicate-argument structures derived from the LF via the lexical choice process.

We might conclude from this reasoning that what we really need as surface generator input is the level of description found in a typical feature structure analysis assigned by a formalism like LFG/HPSG/FUG. Many systems in the NLG literature have adopted this kind of initial description language in preference to logical languages. Our QLF contains the same kind of information as this, encoded in a "flat" representation comprising a set of first order Prolog terms. The flat QLF notation means that the planner need not 'know' about the syntactic form of feature structures as defined by a particular grammar, but simply decide which grammatical constraints hold of each logical element's realisation. That QLF is a *quasi* logical form can be seen from two properties:

(a) It is less expressive in that it lacks scope constructs.

(b) It contains "non-semantic" information, such as grammatical or pragmatic properties of linguistic elements corresponding to logical individuals and variables.

The latter distinguishes our QLF from the better known one of Alshawi. The non-semantic predicates comprise a closed class and are filtered from the QLF during lexicon lookup. In the example below, `past_time(94)` and `num_sing(96)` are examples of non-semantic annotations.

```
[def(95),name(95,bill),book(96),num_sing(96),
long(s(96),96),very(s(96)),indef(96),
past_time(94),write(94), subj(94,95),obj(94,96)]
```

The generator is also constrained by a syntactic description of the target phrase, but only at the top level.

The only properties of QLF relevant to the generation algorithm are that it should be a conjunction of literals, with instantiated arguments, and that each word in the lexicon has at least one QLF term associated with it.[2] From the perspective of the inferential components in the dialogue system, this is a proto-logical form and the relationship between it and LF is beyond the scope of this paper. A benefit of this formalism in relation to our generation algorithm is the simplicity of its manipulation. Since QLF statements are unordered sets, set theoretic operators (e.g. membership, union) suffice for information extraction. Fedder (1991) used a similar algorithm to generate from

[1] PLUS: A Pragmatics-based Language Understanding System. Part-funded by the Commission of the European Communities. Project N° 5284. See Black *et al* (1993) for an overview.

[2] This is a defect of the notation, requiring that particles have a 'semantics'. This can be remedied pragmatically by either a procedural attachment to the lexical entry of the subcategorising item (which sacrifices bidirectionality) or by a dummy semantics which can be inserted at the what to say stage.

scoped logical forms, flattening them to a notation like the one used here.

2.1 Lexicon lookup

Lexical lookup from QLF begins by filtering out the predicates that do not correspond to lexemes.

```
(1)  sleep(1) & past_time(1) & name(2,john) &
     def(2) & arg0(1,2)
```

In (1), the non-lexical elements `arg0(1,2)` and `past_time(1)`, are ignored in accessing the lexicon, but after retrieval of the relevant lexical entries, play their part in filtering out inappropriate forms. The functors of the remaining predications are used to index into the lexicon: `sleep(1)`'s functor `sleep` corresponds to the lexeme or citation form for the lexical entry, and the non-logical annotation `past_time(1)` will after lookup select the correct form `slept`. The indexes (1,2 in the example) are co-instantiated between the semantics and the syntax in the individual lexical and phrasal categories, so as to produce a string corresponding to the correct argument bindings. (This does not happen correctly if the indexes are uninstantiated variables, as in a parse result.)

2.2 Categorial Grammar

The generation algorithm discussed in the next section is not tied to a particular linguistic formalism, but favours a lexicalist formalism with as few rules as possible. This is especially true of CG in which most constituent structure is captured by the two rules of function application. The CG rules of forward and backward function application can be stated as follows in the parsing grammar:

```
%%%%  Forward application
f :: Root/Arg:FunSem + Arg:ArgSem =>
Root:MotherSem :-
    append(ArgSem,FunSem,MotherSem).
%%%%  Backward application
b :: Arg:ArgSem + Root\Arg:FunSem =>
Root:MotherSem :-
    append(ArgSem,FunSem,MotherSem).
```

(Their statement in the generation grammar is slightly longer). In either case, the rules are matched by categories recursively defined over the basic categories s,np and n and the directional slash operators / and \. Briefly, an expression of category A/B combines with an expression of category B to form a phrase of category A. An instance is a determiner, category `np/n` combining with a common noun, category `n`, to its right, forming a noun phrase, category `np`. All expressions in the lexicon belong to either basic or derived categories. To take a complex example, the verb "bet" requires a subject, two object nps and a further sentential object, and hence has category `s\np/s/np/np`.

3 The surface generation algorithm

Initial edges are asserted into a chart, for each word in the lexicon whose semantics is subsumed by the target expression's semantics. As each edge is added to the chart, combinations are made with existing edges, as licensed by the rules, and new spanning edges added.

While this description may make the algorithm to appear something of a blind search, it is in fact strongly directed by the elements present in the QLF, supported by an inversion of the indexing used in parsing.

The lexicon match is not based on direct unification of the target phrase's semantics with that of its head, a fundamental requirement of the bottom-up head-driven algorithm of Shieber et al (1989) and Van Noord (1990). Relaxing this requirement enables semantically equivalent QLFs (arising from commutativity of &) to be handled directly without any special mechanism. The top-level procedure is stated as follows in Prolog:

```
generate(_Syntax:Semantics,_Text) :-
    abolish(edge,1),
    generate_lex_lookup(Semantics,Word,Syn,Sem),
    acceptable(Sem,Semantics,Compl),
    add_edge(Syn:[Word|R]:R:Sem:Compl,Word).
generate(Syntax:_Semantics,Text) :-
    edge(Syntax:Text:[]:_:[]).
```

The lookup procedure retrieves a word whose semantics is a subset of that in `Semantics`, returning the word and ts syntactic and semantic description. `acceptable/3` ensures that the semantics of the word is a subset of the target semantics, and also returns the "unused" part of the semantics in `Compl`. Subsequent recursive calls work on `Compl`, ensuring that constituents are not generated more times (perhaps infinitely) than specified in the target semantics. The second `generate/2` clause requires that all elements in the target semantics are consumed. `Add_edge` is a recursive procedure that does the main work.

```
addEdge(Category1) :-
        \+(edge(Category1)),
        assert(edge(Category1)),
        foreach((edge(Category2),
          addEdge(Mother)).
```

`addedge/1` operates just as it would in a parser: after adding edges to the chart, any combinations permitted with it are applied recursively. `xapply/3` applies the grammatical rules, in this case the rules of categorial function application.

4 Type raising and Composition

A forward composition rule and a type raising rule have been added to those of function application, both to the parser and to the generator. Also, topicalization has been added to the parser. In the parsing grammar (for brevity) these rules are stated as follows:

```
%%%%  Functional Composition
fc :: A/B:FunSem1 + B/C:FunSem2 => A/C:MotherSem
:-  append(FunSem1,FunSem2,MotherSem).
%%%%  Type Raising
ft :: np(Agr)#L:Sem =>
s(Form)#S/(s(Form)#S\np(Agr)#L):Sem.
%%%%  Topic Type Raising
tt :: C#L:Sem => s(top)#S/(s(fin)#S/C#L):Sem :-
    member(C,[np(Agr)]).
```

There have been two motivations for adding these rules to a CG. Firstly, without them, certain co-ordinate and gapping constructs cannot be described neatly. Secondly, they permit incremental interpretation, said to be motivated on psychological grounds. Examples in Section 4.1 illustrate the co-ordinate and gapping constructions that can be treated. With respect to generation, we also find incremental processing well-motivated for interactive systems. Firstly, in the context of the PLUS project, corpus studies (particularly in French) revealed a great deal of overlap between the turns of the two parties in human-simulated machine dialogues, and hence the generator needs to be able to begin realisation before the content is fully planned. Secondly, this enables the generator to be incorporated into a distributed or multi-agent architecture, since partial results are available to external evaluation. Thirdly, interleaving interpretation and planning with generation may create in the user a more favourable impression of response time.

However, the benefits of incrementality are not without their costs. In using rules of function composition, we encounter a *spurious ambiguity problem.* This refers to the multiplicity of derivation paths that are semantically equivalent (and therefore spurious), for the same string, and was first discussed by Wittenburg (1987). This causes multiple generation of identical strings with the same analysis, and an exponential increase in the search space. Fortunately, this problem is already known in the domain of parsing and what we have discovered is that its solution carries over to generation more or less unaltered.

The method of Hepple and Morrill (1989) has been used, in both parser and generator, to cope with spurious ambiguity. The main idea is to enforce normal form proofs by cutting the current branch in the search space when a sequence of rule invocations known to lead to non-normal form derivations is about to be made.

4.1 Coverage of the Grammar

We begin this section with some illustrative constructs and their representation in the lexicon and in QLF, concluding with an illustration of the non-constituent co-ordination and gapping constructs that specifically motivate the rules of function composition. *Intensifier adverbs* such as *very*, *quite*, *really* enable sentences like (2) to be parsed or generated. The QLF corresponding to adjectives is a two-place predicate where the first argument is a state-variable. The connection between a state and an object X in that state is denoted using a skolem function s applied to X. Thus, *long(X)* in a classical logic translation becomes *long(s(X),X)* in the new representation. The full lexical entry for adjectives is given as (3), and (4) is the lexical entry for intensifier adverbs. (5) shows one of the definitions for "and" which enables sentences like (6) and (7) to be parsed and generated.

```
(2)   Bill wrote a very long book.
(3)   non_infl_lex(Word,n(Agr)#X/n(Agr)#X,[QLF])
        :- adj(Word), QLF=.. [Word,s(X),X].
(4)   non_infl_lex(Word,(n(Agr)#X/n(Agr)#X)/
        (n(Agr)#X/n(Agr)#X),[QLF]) :-
        adverb(Word,grad), QLF =.. [Word,s(X)].
(5)   non_infl_lex(and,C#Res\C#Left/C#Right,
        [conj(Res,Left,Right)]).
(6)   Bill and Kristiina wrote a very short
        book and a long letter today.
(7)   Bill saw and heard Kristiina.
(8)   Bill heard and Nancy saw Kristiina.
(9)   Bill walks and Nancy runs today.
(10)  Bill saw the man who John heard.
(11)  Bill saw the man who heard John.
```

Forward composition and type raising rules cover non-constituent co-ordination as shown in (8) and (9). They also permit analysis of WH-movement as shown in (10) and (11).

5. Incremental Generation

Incremental generation has been introduced (Kempen and Hoenkamp 1982) on psychological grounds, and several reports of surface generators have emphasised this property (e.g. Reithinger, 1991, de Smedt and Kempen, 1991, van de Veen forthcoming). In practical terms, the idea is that we should be able to throw logical statements at the generator, one at the time, as soon as they become available (as a product of a reasoning process in a background application, perhaps), and that the generator should be able to start generating right away, without having to wait for the stream of semantic representations to end.

Here we argue: 1) QLF is suitable for specifying the content of the target to be generated incrementally, 2) a chart-based generation algorithm is suitable for incremental generation, and 3) CG rules used can determine the level of 'talkativeness' of an incremental generation system .

QLF is a suitable formalism for this kind of job since it is designed especially with the representation of partial information in mind. QLFs can, while still being well-formed in a syntactic sense, codify such things as a predicate-argument structure where one argument is not yet specified, or a lack of knowledge concerning the properties of another argument, and afterwards, at another time, when it becomes available, the missing information can be given.

The main strengths of the chart-based algorithm used are that QLF terms are not required in a particular order, or all at once.

The only addition to the original CKY generation algorithm is that when no more edges can be added to the chart, the string(s) corresponding to all the QLF given so far is printed; more QLF is requested from the background process; It is then added as 'still to be consumed', and the generation process is called recursively from there.

To see the role of the CG rules for regulating the talkativeness of the generator, note that edges that have consumed all semantic input at a given point in time, and therefore deserve to be printed, must always correspond to *constituents* given the grammar. Now, while a CG with only forward and backward application (FA and BA), implies a standard notion of constituency, rules like type raising (TR) and functional composition (FC) give rise to a more generous notion of constituency (this is what makes 'non-

constituent co-ordination' possible). This means that an incremental generation system of the kind sketched above, employing FA, BA, TR and FC, will be 'chattier' than the very same system employing only FA and BA.

For example, assuming only FA and BA, and QLF = {indef(x), black(s(x),x)}, no string would be generated, since np/n and n/n do not form a constituent. Assuming FA, BA, FC and TR, and the same QLF, the string *a black* would be generated, since np/n and n/n can be composed into the constituent np/n. The string *the black cat* would be generated under both circumstances, if cat(x) was added to the above set.

As another example, consider how the incremental version of the generator, which uses FA, BA, FC and TR, interacts with a user (where the user - input in boldface - plays the role of the QLF producing background process):

```
?- generate.
QLF term: def(x).
[the]
QLF term: man(x).
[the,man]
QLF term: write(e).
QLF term: subj(e,x).
QLF term: obj(e,y).
QLF term: pres(e).
[the,man,writes]
QLF term: long(s(x),x).
[the,long,man,writes]
QLF term: indef(y).
[the,long,man,writes,a]
QLF term: short(s(y),y).
[the,long,man,writes,a,short]
QLF term: letter(y).
[the,long,man,writes,a,short,letter]
```

In the same circumstances, but given only FA and BA, neither [the,man,writes] nor [the,long,man,writes,a], or [the,long,man,writes,a,short] would be generated.

6. Conclusion

A system for parsing and generation based on combinatory categorial grammar and quasi-logical form has been presented. The system seems to score high on at least the following points:

- bi-directionality
- (potential) capability of handling a large repertoire of grammatical phenomena
- incrementality

The system is bi-directional in the sense that given a quasi-logical form, that the parser would have produced had it been given the same string, the generator will produce the same string. Of course, this is the case only if we choose to use exactly the same rules (and lexicon) for both parsing and generation.

The large repertoire of grammatical phenomena that can (potentially) be handled in the system is due to the fact that it was possible, without much performance penalty (due to the use of the Hepple-Morrill method of eliminating spurious ambiguity), to implement, on top of forward and backward application, rules such as type raising and functional composition. This enables many forms of discontinuity phenomena to be treated.

The framework used also seems to offer some interesting possibilities for incremental generation, which is particularly pertinent for surface generation within the context of dialogue systems.

References

Black, William J, Nancy Underwood, Hamish Cunningham and Kristiina Jokinen, "Dialogue Management in a Pragmatics-Based Language Understanding System" In Eds. McEnery, T and Paice, C. Proc 14th Information Retrieval Colloquium, Lancaster 1992, 79-88, Springer-Verlag, 1993.

Guessoum, A, Black, W J, Gallagher, J and Wachtel, T J. "Abduction for Pronoun Resolution", Proc ICLP Workshop on Abduction, Budapest, 1993.

Hepple, Mark and Morrill, Glyn. "Parsing and Derivational Equivalence". Proc. 4th European ACL, 10-18, 1989.

Jokinen, K. "Reasoning about Coherent and Co-operative System Responses", Proc. 5th European Workshop on Natural Language Generation, Pisa, April 1993.

Kempen, Gerard and Edward Hoenkamp. "Incremental sentence generation: implications for the structure of a syntactic processor." In COLING-82, 151-156, 1982.

Phillips, J. D. "Generation of text from Logical Formulae" Machine Translation 8(4), 209-236, 1993.

Reithinger, Norbert. "POPEL: A Parallel and Incremental Natural Language Generation System" In Eds. Paris, Swartout and Mann Natural Language Generation in Artificial Intelligence and Computational Linguistics Kluwer, 179-200, 1991.

Shieber, Stuart M, Gertjan van Noord, Robert C Moore and Fernando C N Pereira, "A Semantic Head-Driven Generation Algorithm for Unification Grammars", Proc. ACL, 27th Annual Meeting, 7-17, 1989.

de Smedt, Koenraad and Gerard Kempen, "Segment Grammar: A Formalism for Incremental Sentence Generation" In Eds. Paris, Swartout and Mann Natural Language Generation in Artificial Intelligence and Computational Linguistics Kluwer, 329-350, 1991.

van de Veen, Evelyn. "Incremental generation for highly interactive dialogues" Proc. Workshop on Pragmatics in Dialogue Management, XIV Scandinavian Conference of Linguistics, 16-21 August, 1993, (forthcoming).

van Noord, Gertjan. "An Overview of Head-Driven Bottom-Up Generation" In Eds. Dale, Mellish and Zock, Current Research in Natural Language Generation. Academic Press, 141-166, 1990.

Wittenburg, Kent. "Predictive combinators: a method for efficient processing of combinatory categorial grammars." In ACL Proceedings, 25th Annual Meeting, 73-80, 1987.