An Efficient Chart Generator for (Semi-)Lexicalist Grammars

John Carroll
johnca@cogs.susx.ac.uk
Cognitive and Computing Sciences
University of Sussex
Falmer, Brighton BN1 9QH, UK

Ann Copestake
aac@csli.stanford.edu
CSLI, Stanford University
Stanford, CA 94305, USA

Dan Flickinger
danf@csli.stanford.edu
CSLI, Stanford University
Stanford, CA 94305, USA

Victor Poznański
vp@sharp.co.uk
Sharp Laboratories of Europe
Edmund Halley Road
Oxford OX4 4GA, UK

Abstract

We describe a generator for rule-based grammars which are primarily lexicalist but may introduce some semantics via constructions. By combining chart generation with a treatment of modification by adjunction, we obtain substantial performance improvements over standard lexically-driven chart-generation.

1 Introduction

There are a range of approaches to tactical generation in which the input to the generator can be thought of as a bag of lexical items with semantic relationships captured by an appropriate instantiation of variables. The instantiated lexical items may either be constructed from a logical form, or generated directly, for instance by lexicalist transfer in a machine-translation (MT) system. Examples of such approaches are Shake-and-Bake (Whitelock, 1992), bag generation from logical form (Phillips, 1993) and some versions of chart generation (e.g., Kay (1996)). We will refer to these approaches collectively as lexically-driven generation.

The basic ideas behind lexically-driven generation and its advantages over alternative approaches such as semantic head-driven generation have been extensively discussed (e.g., by Whitelock (1992), Beaven (1992), Brew (1992) and Trujillo (1994)) so we will just give a brief overview here. Lexically-driven generation is potentially highly suitable for grammar frameworks such as HPSG (Pollard and Sag, 1994) and categorial grammar, where the majority of the information is encoded in lexical entries (or in lexical rules) as opposed to being represented in constructions (i.e., grammar rules operating on phrases). Lexically-driven generation has two major advantages for a generator which must operate with a range of grammars and applications:

1. Apart from the requirements for lexicalism and semantic monotonicity (discussed below), the generator imposes few constraints on the grammar. This is especially significant for systems which must work with a range of large-scale grammars, especially ones which are also used for parsing, since developing and maintaining such grammars is very time-consuming. It is thus very important that any constraints that the generation algorithm imposes fit in naturally with the grammar formalism and are intuitive and easy to test.

2. Syntactic requirements on generator input are minimized. There is a well-known problem in guaranteeing that a grammar will generate from a particular input, sometimes described as the problem of logical form equivalence (Shieber, 1993). The importance of this issue with respect to MT has been extensively discussed (e.g., Whitelock (1992), Beaven (1992)), but even for other applications it is desirable to minimize the extent to which construction of the input requires knowledge of the grammar. With lexically-driven approaches, there is no requirement for the strategic generator to impose a hierarchical organization mirroring the grammar.

Thus lexically-driven generation is inherently more flexible than alternative approaches. However, there are some disadvantages to the existing algorithms:

1. Generation is generally less efficient than in algorithms which use structure to guide processing, such as semantic head-driven generation (Shieber et al, 1990). Chart generation and the original

Shake-and-Bake approach are both exponential in worst-case complexity. As we will discuss below, the main problems arise in intersective modification, which is in sharp contrast to most other algorithms, where efficiency issues tend to arise with non-local dependencies.

2. Some systems which generate from a logical form have used inadequate semantic representations, avoiding internal structure by omitting a representation of scope.

3. Absolute lexicalism is difficult to sustain in grammars. It is often desirable to introduce semantics via constructions, for instance for bare NP formation, even in primarily lexicalist frameworks such as HPSG (e.g., Sag (1997) on relative clause constructions).

In this paper, we describe a generator which avoids these problems. We address the efficiency issue by combining chart generation with a special treatment of modification reminiscent of the approach taken in Poznański et al (1995). We use a flat semantic representation which can express scope or underspecified scopal relationships. Finally, although we assume a primarily lexicalist grammar, we allow the use of constructions which introduce semantics.

The generator works with grammars in which the semantics of a phrase can be described as a list of relations with coindexed variables. This list must be constructed monotonically from a bag of non-overlapping sublists, each sublist corresponding to the semantic contribution of a lexical entry, or lexical or grammar rule (though we assume most of the semantics originates lexically). One suitable formalism is Minimal Recursion Semantics (MRS: Copestake et al (1995), Copestake et al (1997)), which we will use for concreteness here. Our experiments have all been carried out on grammars encoded in a type feature structure formalism, but this is not a requirement of the algorithm.

The generator involves three phases: lexical lookup, chart generation and adjunction of modifiers. We will discuss each of these components in detail, but we start with a brief introduction to MRS.

2 MRS

The MRS representation technique has been described in detail in Copestake et al (1997). Its salient features are:

Flatness The representation consists of a list of labeled relations, plus a list of relationships between labels which constrain scope. In MRS, labels on relations are referred to as handles.

Underspecifiability Scope may be underspecified.

Representable using typed feature structures MRS structures and the composition operations on them can be straightforwardly encoded in a typed feature structure framework. However, for reasons of space and readability, we will not show feature structures here.

To illustrate MRS, we will consider the example sentence in (1a) which we will assume has the two possible scoped representations shown in (1b) and (1c). In both cases, we show a conventional representation, followed by the MRS representation, in which the information usually carried by nesting relations is instead encoded by the use of handles (e.g., h1, h3 etc). In MRS, conjunction is implicit: relations with the same handle are assumed to be conjoined. The toplevel handle in the MRS representation (e.g., h1), indicates the outermost relation(s).

(1) a Every manager interviewed a big German consultant.

b a(y, consultant(y) ∧ German(y) ∧ big(y), every(x, manager(x), interview(e,x,y) ∧ past(e)))
h1:[h1:a(y,h3,h5), h3:consultant(y), h3:German(y), h3:big(y), h5:every(x,h6,h8), h6:manager(x), h8:interview(e,x,y), h8:past(e)]

c every(x, manager(x), a(y, consultant(y) ∧ German(y) ∧ big(y), interview(e,x,y) ∧ past(e)))
h1:[h2:a(y,h3,h8), h3:consultant(y), h3:German(y), h3:big(y), h1:every(x,h6,h2), h6:manager(x), h8:interview(e,x,y), h8:past(e)]

In a fully scoped representation, the handles are constants, but if we allow the labels in argument positions in relations to be variables over handles, we can underspecify scope. The MRS representation which corresponds to the generalization of the two readings above can be written as (2), where the uppercase H4 etc correspond to variables over handles.

(2) H1 [h2:a(y,h3,H4), h3:consultant(y), h3:German(y), h3:big(y), h5:every(x,h6,H7), h6:manager(x), h8:interview(e,x,y), h8:past(e)]

In general, we may require constraints which restrict the assignment of handles to variables. For instance, if we add the information that h2 outscopes h5 to (2), we have the reading in (1b). For applications such as MT, it is often desirable not to have to fully specify quantifier scope, though there are also cases where scope must be represented and MRS is an improvement over the versions of flat semantics described by Phillips (1993) and by Trujillo (1994) since it allows this. However, for simplicity in the current paper, we will generally ignore quantifier scope and omit the handles in the examples where it is unimportant.

3 Lexical lookup

The first phase in generation is lexical lookup, which is analogous to morphological processing during parsing. Lexical entries, lexical rules and grammar rules are all indexed by the relations which they contain. In order to find the lexical entries with which to populate the chart, the input semantics is checked against the indexed lexicon. When a lexical entry is retrieved, the variable positions in its relations are instantiated in one-to-one correspondence with the variables in the input MRS.

For instance, given the MRS in (2c), the instantiated lexical entry for *interview* would contain interview(e,x,y). Here and below, lowercase e, x, y, etc indicate instantiated variables. Note that, as far as generation is concerned, these are really constants, e.g., x and y can never be equated. Handle variables (i.e., label arguments not coindexed with relations) are left uninstantiated.

Lexical and morphological rules are applied to the instantiated lexical entries in this phase. If the lexical rules introduce relations, their application is only allowed if these relations correspond to parts of the input semantics. In this case, their relations are also instantiated. The chart is populated with edges containing instantiated lexical items or structures derived by lexical rule(s), each with pointers to the semantic relations which they cover. For expository purposes, we represent edges as a tuple of index, relation list, syntactic category label and orthography. For instance, we can represent the edge corresponding to *interview* after the application of the past rule as:

(3)

e	interview(e,x,y), past(e)	V	*interviewed*

Lexical lookup and instantiation would be trivial to implement if there were a one-to-one mapping between relations and lexical entries or lexical rules. Several things may complicate this picture:

Relations corresponding to more than one lexical item For instance, a grammar might use the same semantic relation for synonyms such as *autumn* and *fall*. This is straightforwardly analogous to lexical ambiguity during parsing and requires that we include multiple lexical edges in the chart. How common this is depends on the particular grammar's approach to synonymy and to lexical encoding: for example, one grammar might use a single structure to encode multiple subcategorization possibilities for a single sense, while another might require multiple structures. In the second case, the lexical lookup phase will produce multiple edges spanning the same relation (even if the structures are related by lexical rules).

This situation can also occur because the typed feature structure encoding of MRS allows for underspecification of relations. For instance, the temporal location senses of *in* and *on* could have different relations both of which are subsumed by a temporal location relation. It can be desirable to specify such a general relation in the input to the generator, for instance to allow the grammar to make the distinction between *in* and *on* in examples like *on Tuesday morning* vs. *in the morning*.

Lexical items containing more than one relation Compare parsing, where single lexical items may include multiple 'words', e.g., *ice cream*. For instance, *anything* could be represented as containing two relations, any and thing. This leads to edges in the chart which cover more than one relation, and possibly overlap with other lexical edges. Thus an edge corresponding to *anything* could overlap with one for *any*.

Grammar rules which introduce relations An example of this is a rule which licenses bare noun phrases such as *cats* and *dirty water*, which might plausibly apply a quantifier or generic operator. Previous lexically-driven approaches have not allowed such rules. In our approach, any potentially applicable

Lexical edges (inactive)

x	manager(x)	N	*manager*	
x	the(x)	Det	*the*	
e	work(e,x), past(e)	VP	*worked*	

Active edges constructed — Required edge

x	manager(x)	NP	*manager*	x		Det	
e	work(e,x), past(e)	S	*worked*	x		NP	

Inactive edges constructed

x	manager(x), the(x)	NP	*the manager*	
e	work(e,x), past(e), manager(x), the(x)	S	*the manager worked*	

Figure 1: Example of a chart for generating *the manager worked*. This assumes there are two rules S $\rightarrow$ NP VP (VP head) and NP $\rightarrow$ Det N (N head).

grammar rules are instantiated in the lexical lookup phase and passed to the chart generator. Grammar rules which introduce relations which are not in the input will be excluded.

Lexical items which do not introduce relations In principle, all lexical items without relations have to be added to the chart when generating any sentence (compare empty categories in parsing). Although some grammar writers insist that all words carry semantic information, in other grammars words like *do* and (infinitival) *to* have no relations. Even a few such items will seriously affect efficiency. However, for the lexicalist grammars we have examined, it is relatively straightforward to provide filters which avoid unnecessary postulation of such items, since the contexts in which they may occur can be distinguished on the basis of the input semantics, though the actual filters are necessarily highly grammar-specific.

For instance, in the LinGO grammar (discussed in §6), in a sentence such as *Kim had slept*, the auxiliary *had* does not introduce a relation. The tense is indicated by the event variable. In this case, *had* should only be postulated if there is an event variable marked as having tense **pastperf**. This sort of filter can be implemented using rules defined by the grammar writer, where the antecedent of the rule is a pattern which is checked against the input semantics, and the consequent is an identifier for a lexical entry. If a lexical entry has no relations, it is only added if it is licensed by a rule. For example, the rule for *had* can be expressed as follows:

```
< [ EVENT [ TENSE pastperf ]] > => had_aux
```

Here the antecedent matches any semantic expression which contains a verb with a pastperf event variable.

4 Chart generation

Once the chart is instantiated, the chart generation phase is invoked. Chart generation is very similar to chart parsing, but what an edge covers is defined in terms of the semantics, rather than orthography. Each edge is associated with the set of relations it covers. We assume an edge is indexed by a semantic index (following Kay (1996)). The strategy we use is bottom-up, head-first: i.e., an active edge is created from an inactive one by instantiating the head daughter of a rule. When an active edge is applied, a check is made to ensure that the daughters do not overlap: i.e., that they do not include the same relation(s). Figure 1 shows a tiny chart.

Intuitively, a semantic index is a variable denoting an individual entity in a logical form. However, this concept is to some extent grammar-specific. For the purposes of chart generation, what is required is just that, when constructing an active edge, an index for the missing component can be determined, and that each inactive edge can be checked to see if it has a compatible index. A good indexing scheme will be one in which this is maximally discriminating: however in our implementation it is not essential that every edge have a 'sensible' index, since an edge for which an actual index cannot be identified will be given a generic index, which is compatible with all other indices (though this naturally reduces efficiency).

However, in general, there are problematic cases for efficiency when edges have the same index. In particular, in intersective modification, an indefinite number of modifiers may apply and application order is not locally determined. For instance, when generating from (4) the edges in (5) will all be constructed.

(4) big(x), German(x), consultant(x)

(5)

x	consultant(x)	N	*consultant*
x	consultant(x), big(x)	N	*big consultant*
x	consultant(x), German(x)	N	*German consultant*
x	consultant(x), big(x), German(x)	N	*big German consultant*

Note that, if *German big consultant* is excluded by the grammar, the second edge is wasted.

In general, chart generation may be exponential for intersective modification, even when chart parsing is polynomial (see Kay (1996) and, for the equivalent problem in Shake-and-Bake, Brew (1992)). Of course, if there are no constraints on the linear order of modifiers, there will be $n!$ possible phrases incorporating n modifiers. However, even if the grammar constrains modifiers so there is only one valid ordering, the number of edges built by a chart generator will still be 2^n, because of the subphrases. E.g., with the input in (3a), edges such as *every manager interviewed a German consultant* will be generated, omitting *big*.

Kay's partial solution to this problem is to propose that edges be checked before they are created to see if they would seal off access to a semantic index for which there is still an unincorporated modifier. In his example:

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

the index for *athlete* is not available outside the phrase *the tall young Polish athlete ran fast*. This at least prevents the exponentiality proliferating. However this criterion can be expensive to check and may be of limited utility. For instance, in (7) the index for *run* must be available to *how*:

(7) How did the newspapers say the athlete ran?

For Shake-and-Bake, where a similar problem arises, a variety of alternative solutions have been proposed (e.g., Brew (1992), Trujillo (1994)). None of these approaches changes the worst case complexity, however.

For most grammars, this problem inherently involves intersective modification, because it can only arise when both the syntactic category and the semantic index are compatible in the structures before and after rule application. With non-intersective modification, the semantics will prevent the modifiers applying in the wrong order (assuming their scope is specified in the input). E.g., in (8), *probably run* cannot be generated:

(8) he probably did not run.
h1[h1:probably(h2), h2:not(h3), h3:run(e,x)]

With subcategorized complements, the syntax prevents spurious subphrases, even if complement satisfaction is binary branching, as long as the order of complements is fixed. For instance, given the input in (9), the edge (10) will never be generated (without a gap) because the grammar will not license attachment of the complement *money* before *Kim*.

(9) sandy(x), bet(e,x,y,z), kim(y), money(z)

(10)

e	bet(e,x,y,z), money(z)	V	*bet money*

Our approach to the problem is to treat intersective modification in a separate phase, after all possible edges that do not involve potentially recursive intersective modification have been constructed by chart generation. This reduces worst-case complexity to polynomial in the case where modifier order is constrained, and also helps efficiency when it is free.

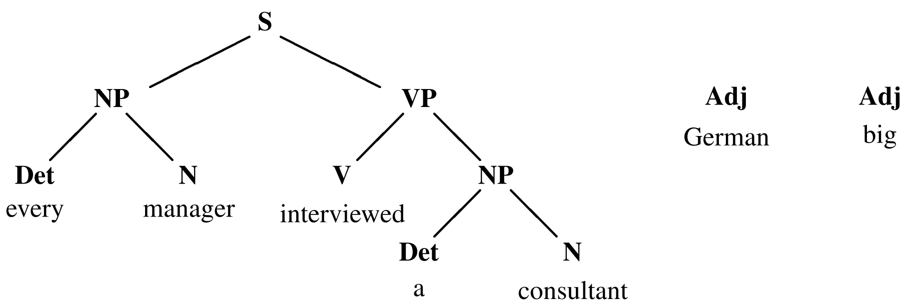


Figure 2: Fragments without intersective modifier rule

5 Modification

The properties of intersective modification that cause these efficiency problems also mean that it can be delayed until after the rest of the structure of the sentence is generated. At this point, intersective modifiers can be added by adjunction to the partial structures. This is possible just because intersective modification does not involve changes in category (other than specialization), and therefore new rule applications will never be licensed by the application of an intersective modifier lower in the tree. (An exception to this generalization is if the modifier contains a gap, as we will discuss below.) Structures on the edges above the adjunction site must be recalculated after adjunction, since modification may specialize the structure, but at worst this leads to trees being excluded.

For example, when generating from (3c), if intersective modification is excluded during the standard chart generation phase, we obtain fragments as shown in Figure 2, corresponding to *every manager interviewed a consultant*, *big* and *German*. The inactive edges which correspond to modifiers (the edges for *big* and *German*) are converted to active edges using the previously excluded intersective modifier rules. These edges are grouped into partitions, corresponding to sets of relations with the edges that cover them. The modifier edges from each partition are applied to the tree by adjunction, as shown in Figure 3, until all relations in the partition have been added. In this case, the result corresponds to *every manager interviewed a big German consultant*, which covers the complete input.

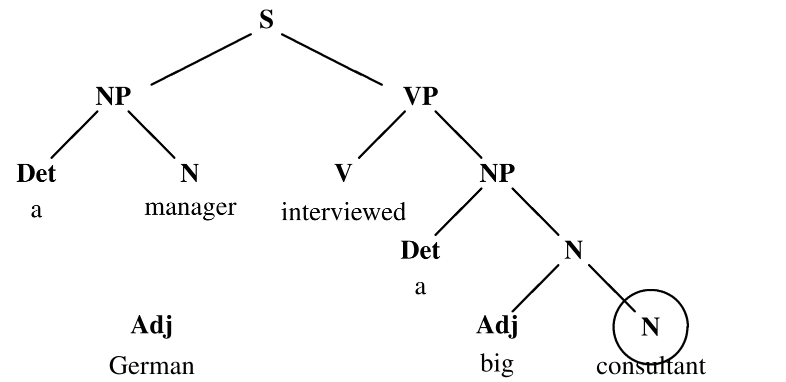
This technique has two substantial efficiency advantages:

Unwanted edges reduced Assume that precedence constraints are specified so any ordering constraint between two modifiers can be determined when adjunction of the second modifier is attempted. For instance, if A, B and C are possible modifiers of N, any ordering constraint between A and B is determinable without reference to C. In this case, partial phrases are only constructed as an intermediate step to building correct structures. Thus, if linear order is fixed, the number of edges constructed will be equal to the number of modifiers, in sharp contrast with chart generation. The difference arise because modifiers may be adjoined above or below a previous adjunction site, whereas chart generation cannot insert constituents into edges.

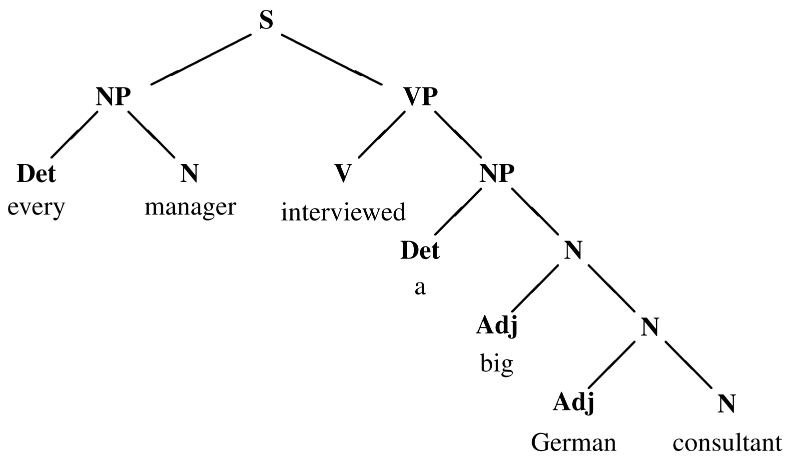
Proliferation delayed Because modification is applied at the end of the generation process, any proliferation of modifier edges due to lack of constraints on word order only multiples the final edges, not the intermediate results.

Our use of adjunction is similar to the locate-adjoin mechanism in Poznański et al (1995), although we apply it to the results of chart generation rather than make use of the general tree reconstruction method described there. Poznański et al (1995) only describe how to construct a single string, although the approach was extended to generate all strings using an agenda (Poznański, personal communication). The advantage of the current algorithm is that it exploits the natural properties of modification, imposes less stringent constraints on the grammar and input, and utilizes the chart's efficiency for dealing with ambiguity.

For our two-phase generation algorithm to work, the grammar must satisfy the condition that modification does not alter the properties of the partial phrases in such a way as to license the application of non-modifier rules. Modifiers with gaps are a counter-example. For instance, consider the example in Figure 4. Here, the modification of the VP by the PP/NP is required to license *which office*. The



Tree after adjunction of *big* (possible adjunction site for *German* is circled).



Tree after adjunction of *German*.

Figure 3: Adjunction of modifiers

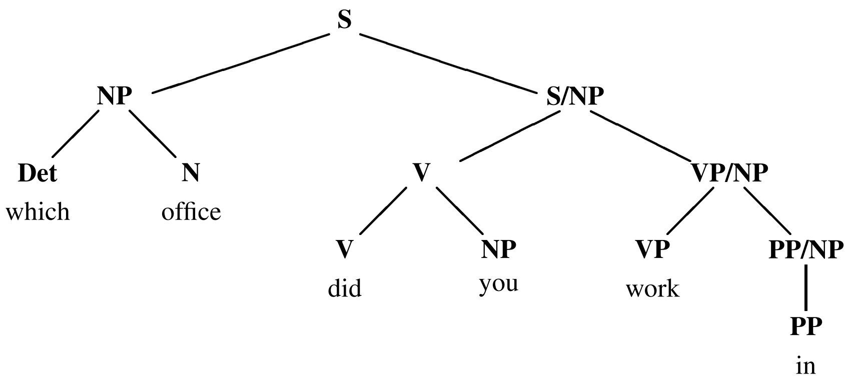


Figure 4: Extraction out of a modifier (*which office did you work in?*)

phrase *which office did you work* is not a constituent, thus will not be generated in the first phase, and the attachment of *which office* to *did you work in* is not modification, so will not occur in the second phase. The solution is simply to treat modification by slashed constituents during the normal chart generation phase, rather than by adjunction. This is reasonable, because extraction out of adjuncts is limited to a single constituent, and doesn't present the complexity problems we have discussed.

	mean edges generated	mean CPU time (secs)
standard chart generation	856	5.4
two phase generation	501	3.3

Table 1: Generation performance on 44 dialogue examples. Mean number of strings generated per input was 5.0, mean string length 5.4 words.

Once all modifiers are adjoined, the complete edges are checked to see whether they cover all the input relations and whether they are compatible with the input semantics. Compatibility of underspecified representations is a non-trivial notion, since in principle it is desirable to generate sentences which preserve ambiguity. However space limitations preclude discussion of this here.

6 Evaluation

So far the generation algorithm has been tested with two pre-existing grammars for English, both of which were developed by grammar writers who had access to parsers but not generators. The main test was on the large-scale English grammar developed by the Linguistic Grammars Online (LinGO) project.[1] This grammar has a wide coverage of linguistic phenomena, including conjunction, extraposition, ellipsis etc, and some more peripheral constructions, such as time expressions and tag questions, which are required for processing spoken dialogue. It produces quite detailed analyses using MRS for the semantics. Although we made some minor changes to this grammar for generation, these were mostly concerned with regularizing the semantic representation to make implementation of the lexical lookup phase more straightforward, and they had the independent benefit of making the grammar more consistent and easier to understand. The second grammar is smaller and much less complex and required no changes for the generator to work.[2]

We tested the performance of the two-phase algorithm against the standard chart generator using the LinGO grammar. On a logical form corresponding to sentence (11a) (with all PPs attached to nouns), the two-phase system took 1.8 seconds and constructed 314 edges, compared to 5.6 seconds and 923 edges for the standard. In (11a) the modifiers are in a fixed order: in contrast, in (11b) the modifier order is not constrained by the grammar and 48 strings are generated ($4! \times 2$, because of topicalization). The two-phase system requires 4.3 seconds and 776 edges, compared to 54.8 seconds and 4710 edges for the standard.

(11) a the manager in that office interviewed a new consultant from Germany
b our manager organized an unusual additional weekly departmental conference

Of course, these examples are artificial. We also tested the system by generating from the first (underspecified) logical forms produced by parsing some collected dialogue sentences. These were part of a test set used to evaluate parsing and were thus not in any way tuned for this evaluation: indeed a considerable percentage contained no modifiers. The results in Table 1 illustrates the performance on 44 examples which went through in both runs. An additional 4 examples which created too many edges for the standard generator succeeded on the two-phase system. As this illustrates, one of the major advantages of the new algorithm is that naturally occurring sentences which had pathological complexity behavior with previous approaches to lexically-driven generation become tractable.

7 Conclusions

The generation algorithm described here has several key advantages for processing lexicalist grammars such as HPSGs:

1. It makes very few assumptions about the grammar, and most of those conditions can be violated without affecting coverage (though at a cost in efficiency).

[1] `http://hpsg.stanford.edu/hpsg/lingo.html`

[2] Of course, in both cases the generator revealed some bugs in the grammar which were fixed.

2. The logical form equivalence problem is reduced without impoverishing the representation. Although we assumed the use of MRS here, it is also possible to generate with a grammar using a non-flat representation, provided efficient procedures are available to:
   (a) retrieve and instantiate lexical entries and rules based on generator input
   (b) determine coverage of edges
   (c) check compatibility of complete edges with the input.

   The great advantage of flat representations for generation with this type of algorithm is that such procedures are trivial to implement and can be written to be usable by a variety of grammars, given suitable parameterization.
3. Although the algorithm is lexically driven, semantics may be contributed by grammar rules.
4. Efficiency is improved compared to chart generation and the original Shake-and-Bake approach by taking advantage of the natural properties of most grammars.

We also believe our approach has considerable promise in that we can generate from logical form, instantiated lexical entries, or a combination of two, which is useful for flexible templatic generation. We expect to demonstrate this last point in future work.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant number IRI-9612682.

References

John L. Beaven (1992) *Lexicalist unification-based machine translation,* PhD thesis, University of Edinburgh.

Chris Brew (1992) 'Letting the cat out of the bag: Generation for Shake-and-Bake MT', *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92),* Nantes, France, pp. 29–34.

Ann Copestake, Dan Flickinger, Robert Malouf, Susanne Riehemann, Ivan Sag (1995) 'Translation using Minimal Recursion Semantics', *Proceedings of the The Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI95),* Leuven, Belgium.

Ann Copestake, Dan Flickinger and Ivan Sag (1997) *Minimal Recursion Semantics: an introduction,* ms. CSLI, Stanford University. `ftp://ftp-csli.stanford.edu/linguistics/sag/mrs.ps.gz`.

Martin Kay (1996) 'Chart Generation', *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96),* Santa Cruz, CA, pp. 200–204.

John D. Phillips (1993) 'Generation of text from logical formulae', *Machine Translation, vol.8 (4),* 209–235.

Pollard, Carl and Ivan Sag (1994) *Head-Driven Phrase Structure Grammar,* Chicago University Press, Chicago and CSLI Publications, Stanford.

Victor Poznański, John L. Beaven and Pete Whitelock (1995) 'An efficient generation algorithm for lexicalist MT', *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95),* Cambridge, Mass., pp. 261–267.

Ivan Sag (1997) 'English relative clause constructions', *Journal of Linguistics, vol.33,* 431–484.

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

Stuart Shieber, Gertjan van Noord, Fernando Pereira, and Robert Moore (1990) 'Semantic-Head-Driven Generation', *Computational Linguistics, vol.16 (1),* 30–43.

Arturo Trujillo (1994) *Lexicalist Machine Translation of Spatial Prepositions,* PhD dissertation, University of Cambridge.

Pete Whitelock (1992) 'Shake-and-Bake translation', *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92),* Nantes, France, pp. 610–616.