Functional Unification Grammar Revisited

Kathleen R. McKeown and Cecile L. Paris
Department of Computer Science
450 Computer Science
Columbia University
New York, N.Y. 10027
MCKEOWN@CS.COLUMBIA.EDU
CECILE@CS.COLUMBIA.EDU

Abstract

In this paper, we show that one benefit of FUG, the ability to state global constraints on choice separately from syntactic rules, is difficult in generation systems based on augmented context free grammars (e.g., Definite Clause Grammars). They require that such constraints be expressed locally as part of syntactic rules and therefore, duplicated in the grammar. Finally, we discuss a reimplementation of FUG that achieves the similar levels of efficiency as Rubinoff's adaptation of MUMBLE, a deterministic language generator.

1 Introduction

Inefficiency of functional unification grammar (FUG, [5]) has prompted some effort to show that the same benefits offered by FUG can be achieved in other formalisms more efficiently [3; 14; 15; 16]. In this paper, we show that one benefit of FUG, the ability to concisely state global constraints on choice in generation, is difficult in other formalisms in which we have written generation systems. In particular, we show that a global constraint can be stated separately from syntactic rules in FUG, while in generation systems based on augmented context free grammars (e.g., Definite Clause Grammars (DCG, [13])) such constraints must be expressed locally as part of syntactic rules and therefore, duplicated in the grammar. Finally, we discuss a reimplementation of FUG in TAILOR [11; 12] that achieves the similar levels of efficiency as Rubinoff's adaptation [16] of MUMBLE [7], a deterministic language generator.

1.1 Statement of Constraints

Language generation can be viewed primarily as a problem of choice, requiring decisions about which syntactic structures best express intent. As a result, much research in language generation has focused on identifying constraints on choice, and it is important to be able to represent these constraints clearly and efficiently. In this paper, we compare the representation of constraints in FUG with their representation in a DCG generation system [3]. We are interested in representing functional constraints on syntactic structure where syntax does not fully restrict expression; that is, constraints other than those coming from syntax. We look at the representation of two specific constraints on syntactic choice: focus of attention on the choice of sentence voice and focus of attention on the choice of simple versus complex sentences.

We claim that, in a FUG, these constraints can be stated separately from rules dictating syntactic structure, thus leading to simplicity of the grammar since the constraints only need to be stated once. This is possible in FUG because of unification and the ability to build constituent structure in the grammar. In contrast, in a DCG, constraints must be stated as part of the individual grammar rules, resulting in duplication of a constraint for each syntactic rule to which it applies.

1.2 Passive/Active Constraint

Focus of attention can determine whether the passive or active voice should be used in a sentence [8]. The constraint dictates that focused information should appear as surface subject in the sentence. In FUG, this can be represented by one pattern indicating that focus should occur first in the sentence as shown in Figure 1. This pattern would occur in the sentence category of the grammar, since focus is a sentence constituent. This constraint is represented as part of an alternative so that other syntactic constraints can override it (e.g., if the goal were in focus but the verb could not be passivized, this constraint would not apply and an active sentence would be generated). The structure of active or passive would be indicated in the verb group as shown in Figure 2.[1] The correct choice of active or passive is made through unification of the patterns: active voice is selected if the focus is on the protagonist (*focus* unifies with *prot*) and passive if focus is on the goal or beneficiary (*focus* unifies with *goal* or *benef*). This representation has two desirable properties: the constraint can be stated simply and the construction of the resulting choice is expressed separately from the constraint.

```
(alt  ((pattern (focus ...))))
```

Figure 1: Constraint on Passive/Active in FUG

In the DCG, the unification of argument variables means a single rule can state that focus should occur first in the sentence. However, the rules specifying construction of the passive and active verb phrases must now depend on which role (protagonist, goal, or beneficiary) is in focus. This requires three separate rules, one of which will be chosen depending on which of the three other case roles is the same as the value for focus. The DCG representation thus mixes information from the constraint, focus of attention, with the passive/active construction, duplicating it over three

[1]This figure shows only the order of constituents for active and passive voice and does not include other details of the construction.

```
(alt
 ((voice active)
  (pattern (prot verb goal)))

 ((voice passive)
  (alt
   ((pattern (goal verb1 verb2 by-pp)))
   ((pattern
     (benef verb1 verb2 by-pp))))))
```

Figure 2: Passive/Active Construction in FUG

rules.

The sentence rule is shown in Figure 3 and the three other rules are presented in Figure 4. The constituents of the proposition are represented as variables of a clause. In Figure 4, the arguments, in order, are verb (V), protagonist (PR), goal (G), beneficiary (B), and focus. The arguments with the same variable name must be equal. Hence, in the Figure, focus of the clause must be equal to the protagonist (PR).

```
sentence(clause
         (Verb,Prot,Goal,Benef,Focus))
 -->
 nplist(Focus),
 verb_phrase(Verb,Prot,Goal,Benef,Focus).
```

Figure 3: Passive/Active Constraint in DCG

1.3 Focus Shift Constraint

This constraint, identified and formalized by Derr and McKeown [3], constrains simple and complex sentence generation. Any generation system that generates texts and not just sentences must determine when to generate a sequence of simple sentences and when to combine simple sentences to form a more complex sentence. Derr and McKeown noted that when a speaker wants to focus on a single concept over a sequence of sentences, additional information may need to be presented about some other concept. In such a case, the speaker will make a temporary digression to the other concept, but will immediately continue to focus on the first. To signal that focus does not shift, the speaker can use subordinate sentence structure when presenting additional information.

The focus constraint can be stated formally as follows: assume input of three propositions, P1, P2, and P3 with

```
/* V = Verb; PR = Prot; G = Goal;
B = Beneficiary; last argument = focus */

verb_phrase(pred(V,NEG,T,AUX),PR,G,B,PR)

   -->verb(V,NEG,T,AUX,N,active),
      nplist(G),
      pp(to,B).

verb_phrase(pred(V,NEG,T,AUX),PR,G,B,G)

   -->verb(V,NEG,T,AUX,N,passive),
      pp(to,B),
      pp(by,PR).

verb_phrase(pred(V,NEG,T,AUX),PR,G,B,B)

   -->verb(V,NEG,T,AUX,N,passive),
      nplist(G),
      pp(by,PR).
```

Figure 4: Passive/Active Construction in DCG

arguments indicating focus F1, F2, and F3.[2] The constraint states that if F1 = F3, F1 does not equal F2 and F2 is a constituent of P1, the generator should produce a complex sentence consisting of P1, as main sentence with P2 subordinated to it through P2's focus, followed by a second sentence consisting of P3. In FUG, this constraint can be stated in three parts, separately from other syntactic rules that will apply:

1. Test that focus remains the same from P1 to P3.
2. Test that focus changes from P1 to P2 and that the focus of P2 is some constituent of P1.
3. If focus does shift, form a new constituent, a complex sentence formed from P1 and P2, and order it to occur before P3 in the output (order is specified by patterns in FUG).

Figure 5 presents the constraint, while Figure 6 shows the construction of the complex sentence from P1 and P2. Unification and paths simplify the representation of the constraint. Paths, indicated by angle brackets (<>), allow the grammar to point to the value of other constituents. Paths and unification are used in conjunction in Part 1 of Figure 5 to state that the value of *focus* of P1 should unify with the

[2]In the systems we are describing, input is specified in a case frame formalism, with each proposition indicating protagonist (*prot*), goal, beneficiary (*benef*), verb, and focus. In these systems, lexical choice is made before entering the grammar, thus each of these arguments includes the word to be used in the sentence.

```
(alt
  % Is focus the same in P1 and P3?
1.((P1 ((focus <^ P3 focus>)))

       % Does not apply if focus
       % stays the same
2. (alt (((P1 ((focus <^ P2 focus>))))

        ( % Focus shifts; Check that P2
          % focus is a constituent of
          % P1.
           (alt
            (((P1 ((prot <^ P2 focus>))))
             ((P1 ((goal <^ P2 focus>))))
             ((P1 ((benef
                    <^ P2 focus>))))))
         % Form  new constituent from P1
         % and P2 and order before P3.
3.    (pattern (P1P2subord P3))
         (P3 (cat s))
         % New constituent is of category
         % subordinate.
         (P1P2subord
          % Place P2 focus into
          % subordinate as it will
          % be head of relative clause.
          (same <^ P2 focus>)
          (cat subordinate)))))
```

Figure 5: Focus Shift Constraint in FUG

```
((cat subordinate)
 % Will consist of one compound sentence
 (pattern (s))
 (s ((cat s)))
  % Place contents of P1 in s.
 (s   <^^ P1>)
  % Add the subordinate as a
  % relative clause modifying SAME.
   (same
   % Place the new subordinate made from
   % P2 after head.
    ((pattern (... head newsubord ...))
   % Form new subordinate clause
     (newsubord
    % It's a relative clause.
      (cat s-bar)
      (head <^ head>)
    % All other constituents in
    % newsubord come from P2.
     (same ((newsubord <^ ^ P2>)
    % Unify same with appropriate
    % constituent of P1 to attach
    % relative clause
  (s
   ((alt (((prot <^ same>))
          ((goal <^ same>))
          ((benef <^ same>)))))))
```

Figure 6: Forming the Subordinate Clause in FUG

value of *focus* of P3 (i.e., these two values should be equal).[3] Unification also allows for structure to be built in the grammar and added to the input. In Part 3, a new constituent *P1P2subord* is built. The full structure will result from unifying *P1P2subord* with the category *subordinate*, in which the syntactic structure is represented. The grammar for this category is shown in Figure 6. It constructs a relative clause[4] from P2 and attaches it to the constituent in P1 to which focus shifts in P2. Figure 7 shows the form of input required for this constraint and the output that would be produced.

[3]A path is used to extract the focus of P3. An attribute value pair such as (focus <P3 focus>) determines the value for *focus* by searching for an attribute P3 in the list of attributes (or Functional Description (FD)) in which *focus* occurs. The value of P3's *focus* is then copied in as the value of *focus*. In order to refer to attributes at any level in the tree formed by the nested set of FDs, the formalism includes an up-arrow (^). For example, given the attribute value pair (attr1 <^ attr2 attr3>), the up-arrow indicates that the system should look for *attr2* in the FD containing the FD of *attr1*. Since P3 occurs in the FD containing P1, an up-arrow is used to specify that the system should look for the attribute P3 in the FD containing P1 (i.e., one level up). More up-arrows can be used if the first attribute in the path occurs in an even higher level FD.

[4]The entire grammar for relative clauses is not shown. In particular, it would have to add a relative pronoun to the input.

In the DCG formalism, the constraint is divided between a rule and a test on the rule. The rule dictates focus remain the same from P1 to P3 and that P2's focus be a constituent of P1, while the test states that P2's focus must not equal P1's. Second, because the DCG is essentially a context free formalism, a duplication of rules for three different cases of the construction is required, depending on whether focus in P2 shifts to protagonist, goal or beneficiary of P1. Figure 8 shows the three rules needed. Each rule takes as input three clauses (the first three clauses listed) and produces as output a clause (the last listed) that combines P1 and P2. The test for the equality of foci in P1 and P3 is done through PROLOG unification of variables. As in the previous DCG example, arguments with the same variable name must be equal. Hence, in the first rule, focus of the third clause (F1) must be equal to focus of the first clause (also F1). The shift in focus from P1 to P2 is specified as a condition (in curly brackets {}). The condition in the first rule of Figure 8 states that the focus of the second clause (PR1) must not be the same as the focus of the first clause (F1).

Note that the rules shown in Figure 8 represent primarily the constraint (i.e., the equivalent of Figure 5).

```
INPUT:
((P1 ((prot ((head === girl)))
      (goal ((head === cat)))
      (verb-group ((verb === pet)))
      (focus <prot>))))

 (P2 ((prot ((head === cat))
      (goal ((head === mouse))
      (verb-group ((verb === caught)))
      (focus <prot>))))

 (P3  ((prot ((head === girl)))
       (goal ((head === happy)))
       (verb-group ((verb === be)))
       (focus <prot>)))))

OUTPUT = The girl pet the cat that caught
         the mouse. The girl was happy.
```

Figure 7: Input and Output for FUG

The building of structure, dictating how to construct the relative clause from P2 is not shown, although these rules do show where to attach the relative clause. Second, note that the constraint must be duplicated for each case where focus can shift (i.e., whether it shifts to prot, goal or beneficiary).

1.4 Comparisons With Other Generation System Grammars

The DCG's duplication of rules and constraints in the examples given above results because of the mechanisms provided in DCG for representing constraints. Constraints on constituent ordering and structure are usually expressed in the context free portion of the grammar; that is, in the left and right hand sides of rules. Constraints on when the context free rules should apply are usually expressed as tests on the rules. For generation, such constraints include pragmatic constraints on free syntactic choice as well as any context sensitive constraints. When pragmatic constraints apply to more than one ordering constraint on constituents, this necessarily means that the constraints must be duplicated over the rules to which they apply. Since DCG allows for some constraints to be represented through the unification of variables, this can reduce the amount of duplication somewhat.

FUG allows pragmatic constraints to be represented as meta-rules which are applied to syntactic rules expressing ordering constraints through the process of unification. This is similar to Chomsky's [2] use of movement and focus rules to transform the output of context free rules in order to avoid rule duplication. It may be possible to factor out constraints and represent them as meta-rules in a DCG, but this would involve a non-standard implementation of the DCG (for example, compilation of the DCG to another grammar formalism which is capable of representing constraints as meta-rules).

```
/* Focus of P2 is protagonist of P1 (PR1)
Example:  the cat was petted by the girl
  that brought it.  the cat purred */

foc_shift(clause(V1,PR1,G1,B1,F1),
  clause(V2,PR2,G2,B2,PR1),
  clause(V3,PR3,G3,B3,F1),
  clause(V1,
       [np(PR1,clause(V2,PR2,G2,B2,PR1))],
       G1,B1,F1))
  /* Test: focus shifts from P1 to P2 */
  {PR1 \== F1}

/* Focus of P2 is  goal of P1 (G1)
Example:  the girl pet the cat that
caught the mouse.  the girl was happy */

foc_shift(clause(V1,PR1,G1,B1,F1),
  clause(V2,PR2,G2,B2,G1),
  clause(V3,PR3,G3,B3,F1),
  clause(V1,PR1,
       [np(G1,clause(V2,PR2,G2,B2,G1))],
       B1,F1))
  /* Test: focus shifts from P1 to P2 */
  {G1 \== F1}

/* Focus of P2 is  Beneficiary of P1 (B1)
Example: the mouse was given to the cat
that was hungry. the mouse was not
happy */

foc_shift(clause(V1,PR1,G1,B1,F1),
  clause(V2,PR2,G2,B2,B1),
  clause(V3,PR3,G3,B3,F1),
  clause(V1,PR1,G1,
       [np(B1,clause(V2,PR2,G2,B2,B1))],
       F1))
  /* Test: focus shifts from P1 to P2 */
  {B1 \== F1}
```

Figure 8: Focus Shift Constraint in DCG

Other grammar formalisms that express constraints through tests on rules also have the same problem with rule duplication, sometimes even more severely. The use of a simple augmented context free grammar for generation, as implemented for example in a bottom-up parser or an augmented transition network, will require even more duplication of constraints because it is lacking the unification of variables that the DCG includes. For example, in a bottom-up generator implemented for word algebra problem generation by Ment [10], constraints on wording of the problem are expressed as tests on context free rules and natural language output is generated through actions on the rules. Since Ment controls the linguistic difficulty of the generated word algebra problem as well as the algebraic difficulty, his constraints determine when to generate particular syntactic constructions that increase wording difficulty. In the bottom-up generator, one such instructional constraint must be duplicated over six different syntactic rules, while in FUG it could be expressed as a single constraint. Ment's work points to interesting ways instructional constraints interact as well, further complicating the problem of clearly representing constraints.

In systemic grammars, such as NIGEL [6], each choice point in the grammar is represented as a *system*. The choice made by a single system often determines how choice is made by other systems, and this causes an interdependence among the systems. The grammar of English thus forms a hierarchy of systems where each branch point is a choice. For example, in the part of the grammar devoted to clauses, one of the first branch points in the grammar would determine the voice of the sentence to be generated. Depending on the choice for sentence voice, other choices for overall sentence structure would be made. Constraints on choice are expressed as LISP functions called *choosers* at each branch point in the grammar. Typically a different chooser is written for each system of the grammar. Choosers invoke functions called *inquiry operators* to make tests determining choice. Inquiry operators are the primitive functions representing constraints and are not duplicated in the grammar. Calls to inquiry operators from different choosers, however, may be duplicated. Since choosers are associated with individual syntactic choices, duplications of calls is in some ways similar to duplication in augmented context free grammars. On the other hand, since choice is given an explicit representation and is captured in a single type of rule called a system, representation of constraints is made clearer. This is in contrast to a DCG where constraints can be distributed over the grammar, sometimes represented in tests on rules and sometimes represented in the rule itself. The systemic's grammar use of features and functional categories as opposed to purely syntactic categories is another way in which it, like FUG, avoids duplication of rules.

It is unclear from published reports how constraints are represented in MUMBLE [7]. Rubinoff [16] states that constraints are local in MUMBLE, and thus we suspect that they would have to be duplicated, but this can only be verified by inspection of the actual grammar.

2 Improved Efficiency

Our implementation of FUG is a reworked version of the tactical component for TEXT [9] and is implemented in PSL on an IBM 4381 as the tactical component for the TAILOR system [11; 12]. TAILOR's FUG took 2 minutes and 10 seconds of real time to process the 57 sentences from the appendix of TEXT examples in [9] (or 117 seconds of CPU time). This is an average of 2.3 seconds real time per sentence, while TEXT's FUG took, in some cases, 5 minutes per sentence.[5] This compares quite favorably with Rubinoff's adaptation [16] of MUMBLE [7] for TEXT's strategic component. Rubinoff's MUMBLE could process all 57 sentences in the appendix of TEXT examples in 5 minutes, yielding an average of 5 seconds per sentence. Thus our new implementation results in yet a better speed-up (130 times faster) than Rubinoff's claimed 60 fold speed-up of the TEXT tactical component.

Note, however, that Rubinoff's comparison is not at all a fair one. First, Rubinoff's comparisons were done in real times which are dependent on machine loads for time-sharing machines such as the VAX-780, while Symbolics real time is essentially the same as CPU time since it is a single user workstation. Average CPU time per sentence in TEXT is 125 seconds.[6] This makes Rubinoff's system only 25 times faster than TEXT. Second, his system runs on a Symbolics 3600 in Zetalisp, while the original TEXT tactical component ran in Franzlisp on a VAX 780. Using Gabriel's benchmarks [4] for Boyer's theorem proving unification based program, which ran at 166.30 seconds in Franzlisp on a Vax 780 and at 14.92 seconds in Symbolics 3600 CommonLisp, we see that switching machines alone yields a 11 fold speed-up. This means Rubinoff's system is actually only 2.3 times faster than TEXT.

Of course, this means our computation of a 130 fold speed-up in the new implementation is also exaggerated since it was computed using real time on a faster machine too. Gabriel's benchmarks are not available for PSL on the IBM 4381,[7] but we are able to make a fair comparison of the two implementations since we have both the old and new versions of FUG running in PSL on the IBM. Using CPU times, the new version proves to be 3.5 times faster than the old tactical component.[8]

Regardless of the actual amount of speed-up achieved, our new version of FUG is able to achieve similar speeds to MUMBLE on the same input, despite the fact that FUG uses a non-deterministic algorithm and MUMBLE uses a deterministic approach. Second, regardless of comparisons between systems, an average of 2.3 seconds real time per sentence is quite acceptable for a practical generation system.

We were able to achieve the speed-up in our new version of FUG by making relatively simple changes in the unification algorithm. The first change involved immediately selecting the correct category for unification from the grammar whenever possible. Since the grammar is represented as a list of possible syntactic categories, the first stage in unification involves selecting the correct category to unify with the input. On first invoking the unifier, this means selecting the sentence level category and on unifying each constituent of the input with the grammar, this means selecting the category of the constituent. In the old grammar, each category was unified successively until the correct one was found. In the current implementation, we retrieve the correct category immediately and begin

[5] We use real times for our comparisons in order to make an analogy with Rubinoff [16], who also used real times.

[6] This was computed using TEXT's appendix where CPU time is given in units corresponding to 1/60 second.

[7] Gabriel's benchmarks are available only for much larger IBM mainframes.

[8] The new version took 117 CPU seconds to process all sentences, or 2 CPU seconds per sentence, while the old version took 410 CPU seconds to process all sentences, or 7 CPU seconds per sentence.

unification directly with the correct category. Although unification would fail immediately in the old version, directly retrieving the category saves a number of recursive calls.

Unification with the lexicon uses the same technique in the new version. The correct lexical item is directly retrieved from the grammar for unification, rather than unifying with each entry in the lexicon successively.

Another change involved the generation of only one sentence for a given input. Although the grammar is often capable of generating more than one possible sentence for its input[9], in practice, only one output sentence is desired. In the old version of the unifier, all possible output sentences were generated and one was selected. In the new version, only one successful sentence is actually generated.

Finally, other minor changes were made to avoid recursive calls that would result in failure. Our point in enumerating these changes is to show that they are extremely simple. Considerably more speed-up is likely possible if further implementation were done. In fact, we recently received from ISI a version of the FUG unifier which was completely rewritten from our original code by Jay Myers. It generates about 6 sentences per seconds on the average in Symbolics Commonlisp. Both of these implementations demonstrate that unification for FUG can be done efficiently.

3 Conclusions

We have shown how constraints on generation can be represented separately from representation of syntactic structure in FUG. Such an ability is attractive because it means that the constraint can be stated once in the grammar and can be applied to a number of different syntactic rules. In contrast, in augmented context free based generation systems, constraints must be stated locally as part of individual syntactic rules to which they apply. As a result, constraints must be duplicated. Since a main focus in language generation research has been to identify constraints on choice, the ability to represent constraints clearly and efficiently is an important one.

Representing constraints separately is only useful for global constraints, of course. Some constraints in language generation are necessarily local and must be represented in FUG as they would in augmented context free based systems: as part of the syntactic structures to which they apply. Furthermore, information for some constraints may be more easily represented outside of the grammar. In such cases, using a function call to other components of the system, as is done in NIGEL, is more appropriate. In fact, this ability was implemented as part of a FUG in TELEGRAM [1]. But for global constraints for which information is available in the grammar, FUG has an advantage over other systems.

Our reimplementation of FUG has demonstrated that efficiency is not as problematic as was previously believed. Our version of FUG, running in PSL on an IBM 4381, runs faster than Rubinoff's version of MUMBLE in Symbolics 3600 Zetalisp for the same set of input sentences. Furthermore, we have shown that we were able to achieve a slightly better speed-up over TEXT's old tactical component than Rubinoff's MUMBLE using a comparison that takes into account different machines. Given that FUG can produce sentences in time comparable to a deterministic generator, efficiency should no longer be an issue when evaluating FUG as a generation system.

[9]Often the surface sentences generated are the same, but the syntactic structure built in producing the sentence differs.

Acknowledgements

The research reported in this paper was partially supported by DARPA grant N00039-84-C-0165, by ONR grant N00014-82-K-0256 and by NSF grant IST-84-51438. We would like to thank Bill Mann for making a portion of NIGEL's grammar available to us for comparisons.

References

[1] Appelt, D. E.
TELEGRAM: A Grammar Formalism for Language Planning.
In *Proceedings of the Eigth National Conference on Artificial Intelligence*, pages 595 - 9. Karlsruhe, West Germany, August, 1983.

[2] Chomsky, N.
Essays on Form and Interpretation.
North-Holland Publishing Co., Amsterdam, The Netherlands, 1977.

[3] Derr, M.A. and McKeown, K. R.
Using Focus to Generate Complex and Simple Sentences.
In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 501-4. Stanford, Ca., July , 1984.

[4] Gabriel, R. P.
Performance and Evaluation of Lisp Systems.
MIT Press, Cambridge, Mass., 1985.

[5] Kay, Martin.
Functional Grammar.
In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*. Berkeley Linguistics Society, 1979.

[6] Mann, W.C. and Matthiessen, C.
NIGEL: A Systemic Grammar for Text Generation.
Technical Report ISI/RR-85-105, Information Sciences Institute, February, 1983.
4676 Admiralty Way, Marina del Rey, California 90292-6695.

[7] McDonald, D. D.
Natural Language Production as a Process of Decision Making under Constraint.
PhD thesis, MIT, Cambridge, Mass, 1980.

[8] McKeown, K. R.
Focus Constraints on Language Generation.
In *Proceedings of the Eight International Conference on Artificial Intelligence*. Karlsruhe, Germany, August, 1983.

[9] McKeown, K.R.
Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.
Cambridge University Press, Cambridge, England, 1985.

[10] Ment, J.
From Equations to Words. Language Generation and Constraints in the Instruction of Algebra Word Problems.
Technical Report, Computer Science Department, Columbia University, New York, New York, 10027, 1987.

[11] Paris, C. L.
Description Strategies for Naive and Expert Users.
In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics.* Chicago, 1985.

[12] Paris, C. L.
Tailoring Object Descriptions to the User's Level of Expertise.
Paper presented at the International Workshop on User Modelling, Maria Laach, West Germany.
August, 1986

[13] Pereira, F.C.N. and Warren, D.H.D.
Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Network.
Artificial Intelligence :231- 278, 1980.

[14] Ritchie, G.
The Computational Complexity of Sentence Derivation in Functional Unification Grammar.
In *Proceedings of COLING '86.* Association for Computational Linguistics, Bonn, West Germany, August, 1986.

[15] Ritchie, G.
Personal Communication.

[16] Rubinoff, R.
Adapting MUMBLE: Experience with Natural Language Generation.
In *Proceedings of the Fifth Annual Conference on Artificial Intelligence.* American Association of Artificial Intelligence, 1986.