USING FOCUS TO GENERATE COMPLEX AND SIMPLE SENTENCES

Marcia A. Derr

Department of Computer Science
Columbia University
New York, NY 10027 USA
and
AT&T Bell Laboratories
Murray Hill, NJ 07974 USA

Kathleen R. McKeown

Department of Computer Science
Columbia University
New York, NY 10027 USA

Abstract

One problem for the generation of natural language text is determining when to use a sequence of simple sentences and when a single complex one is more appropriate. In this paper, we show how focus of attention is one factor that influences this decision and describe its implementation in a system that generates explanations for a student advisor expert system. The implementation uses tests on functional information such as focus of attention within the Prolog Definite Clause Grammar formalism to determine when to use complex sentences, resulting in an efficient generator that has the same benefits as a functional grammar system.

1. INTRODUCTION

Two problems in generation of natural language text are deciding *what to say* and *how to say it*. This research concentrates on the second question by identifying and implementing functional mechanisms (e.g., focus, given/new information, semantic information, user characteristics) for determining how to express an underlying message. One decision a surface generator must face is whether to use a sequence of simple sentences or a single complex one. In this paper we explore the use of focus of attention as the basis on which such decisions can be made and describe its implementation using functional information[1] within the Prolog Definite Clause Grammar (DCG) formalism[2]. This implementation was done as part of an explanation facility for a student advisor expert system developed at Columbia University.

2. CHOOSING SURFACE STRUCTURE

Input to the surface generator is a semantic representation of what is to be said, including focus information. This is represented as a set of logical propositions, where each proposition consists of a predicate relating a group of arguments. Given the semantic content of a message, the generator must produce an appropriate surface structure taking into consideration a wide variety of alternatives. One choice is whether to express each proposition as a simple sentence or to combine propositions into complex sentences. As an example, consider the two propositions in Figure 1. These may be expressed as two simple sentences ((1) below) or as one sentence containing a subordinate clause (sentence (2)). The sentences in (1) and (2) also show that a generation system should be able to choose between definite and indefinite reference and decide when to pronominalize. Another decision is what syntactic structure to use, such as whether to use the active or the passive voice. Thus, the first proposition in Figure 1 may be expressed as any of the sentences shown in (3) - (5).

Proposition 1: predicate = give protagonist = John goal = book beneficiary = Mary Proposition 2: predicate = need protagonist = Mary goal = book	(1) John gave Mary a book. Mary needed the book. (2) John gave Mary a book that she needed. (3) John gave Mary a book. (4) Mary was given a book by John. (5) A book was given to Mary by John.

Figure 1. Logical propositions and possible surface sentences

Given that there are multiple ways to express the same underlying message, how does a text generator choose a surface structure that is appropriate? What are some mechanisms for guiding the various choices? Previous research has identified focus of attention as one choice mechanism. McKeown[3] demonstrated how focus can be used to select sentence voice, and to determine whether pronominalization is called for. Focus can also be used as the basis on which to combine propositions. Suppose that, in the example in Figure 1, focus is on *John* in proposition 1 and *book* in proposition 2. If a third proposition follows with focus returning to *John*, then the surface generator can signal that the shift to *book* is only temporary by combining the two propositions using subordination as in sentence (2). A textual sequence illustrating this possibility is shown in (6) below. On the other hand, if the third proposition continues to focus on *book*, then it is more appropriate to generate the first and second propositions as two separate sentences as in sentence (1) above. It may even be possible to combine the second and third propositions as in the textual sequence shown in (7).

(6) John gave Mary a book that she needed.
He had seen it in the Columbia bookstore.

(7) John gave Mary a book.
Mary needed the book and had been planning on buying it herself.

Argument identity can also serve as a basis for combining propositions. In the first two propositions in Figure 2, the values of predicate, protagonist, and focus match. These propositions can be joined by conjunction, deleting the protagonist and predicate of the second proposition as in sentence (8). Propositions 3 and 4, with identical values for focus can also be combined using conjunction and deleting the focused argument in the second proposition (sentence (9)).

Proposition 1:
predicate = buy
protagonist = John
goal = car
focus = John

Proposition 2:
predicate = buy
protagonist = John
goal = plane
focus = John

Proposition 3:
predicate = catch
protagonist = cat
goal = mouse
focus = cat

Proposition 4:
predicate = give
goal = milk
beneficiary = cat
focus = cat

(8) John bought a car and a plane.
(9) The cat caught a mouse and was given milk.

Figure 2. Combining propositions with argument identities

While some previous generation systems have been capable of combining propositions, the decision to combine was either already made[4] or made on the basis of rhetorical structure[5], also an important basis. Influence of focus on these decisions has not been previously noted.

3. ENCODING CHOICE MECHANISMS IN A FUNCTIONAL GRAMMAR

Certain features of a functional grammar[1] were selected to use within the Prolog DCG formalism[2]. This endows our generator with the best features of both types of grammars: simplification of input through the functional grammar formalism and efficiency of execution through the Prolog unification algorithm.

In a functional grammar, functional information is treated in the same manner as syntactic information. The use of functional information means that the input to the generator is simplified as it need not completely specify all the syntactic information. Rather, tests on functional information (such as focus), for choosing between alternative surface structures, can be encoded in the grammar to arrive at the complete syntactic structure. Both the underlying message and the grammar are specified as functional descriptions. The final surface structure description is derived by unifying the message with the grammar. As an example, consider the proposition encoded as a functional description in Figure 3. When unified with a grammar that contains rules for how *focus* determines sentence voice, sentence (3) of Figure 1 is generated. If *focus* were *book*, the grammar would generate sentence (5).

```
predicate = give
protagonist = John
goal = book
beneficiary = Mary
focus = John
tense = past
```

Figure 3. Functional description of an underlying message

The DCG formalism[2] is a method for expressing grammar rules as clauses of first-order predicate logic. A DCG extends a context-free grammar by allowing nonterminals to have arguments. Since DCGs are executable Prolog programs, text generation is accomplished by unifying a Prolog goal with a set of grammar clauses. The Prolog goal specifies a functional description of the underlying message as input arguments. The generated text is returned in another argument of the goal.

A DCG was used to encode a functional grammar for text generation. Previous implementations of functional grammars have been concerned with the efficiency of the unification algorithm[3,6]. Encoding a functional grammar as a DCG takes advantage of Prolog's efficient unification algorithm and thus, speed is no longer a problem. Functional information, supplied as input arguments, is unified with the grammar to select the rules that will generate an appropriate surface structure. The DCG formalism allows extra conditions to be encoded in the grammar rules, providing the mechanism for testing functional information. Figure 4 shows a proposition encoded as an input argument to a DCG. The arguments of *prop* are the predicate, protagonist, goal, beneficiary, and focus.

```
prop (
      pred (give, past),
      arg (John, _),
      arg (book, sing),
      arg (Mary, _),
      arg (John, _)
)
```

Figure 4. Functional description encoded as input to a DCG

4. IMPLEMENTATION OF CHOICE MECHANISMS

The mechanisms for selecting surface structure and, in particular, combining propositions, were implemented as part of an explanation facility for a student advisor expert system which is implemented in Prolog. One component of the advisor system, the planner, determines a student's schedule of courses for

a particular semester. An explanation of the planning process can be derived from a trace of the Prolog goals that were invoked during planning[7]. Each element of the trace is a proposition that corresponds to a goal. The propositions are organized hierarchically, with propositions toward the top of the hierarchy corresponding to higher level goals. Relationships between propositions are implicit in this organization. For example, satisfying a higher level goal is conditional on satisfying its subgoals. The explanation domain provides a rich testbed to experiment with techniques for combining propositions.

A grammar for automatically generating explanations was implemented which encodes tests for combining propositions. The grammar also includes tests on focus for determining active/passive sentence voice, but does not currently pronominalize on the basis of focus. Because the expert system does not yet automatically generate a trace of its execution, the propositions that served as input to the grammar were hand-encoded from the results of several system executions.

The generator determines that subordination is necessary by checking whether focus shifts over three propositions. A simplified example of a DCG rule *foc_shift*, that tests for this is shown in Figure 5. The left-hand-side of this rule contains three input propositions and an output proposition. Each proposition has five arguments: verb, protagonist, goal, beneficiary, and focus. If the first proposition focuses on *Foc1* and mentions an unfocused argument *Goal1*, the second proposition specifies *Goal1* as its focus,[1] but in the third proposition the focus returns to *Foc1*, then the first and second propositions can be combined using subordination. The combined propositions are returned as a single proposition in the fourth argument; the third proposition is returned, unchanged, in the third argument. Both can be tested for further combination with other propositions. A sample text produced using this rule is shown in (10) below.

1. The right-hand-side of the rule contains a test to check that the focus of the second proposition is different from the focus of the first.

```
foc_shift (
    prop (Verb1, Prot1, Goal1, Ben1, Foc1),
    prop (Verb2, Prot2, Goal2, Ben2, Goal1),
    prop (Verb3, Prot3, Goal3, Ben3, Foc1),
    prop (Verb1, Prot1, np(Goal1, prop(Verb2, Prot2, Goal2, Ben2, Goal1)), Ben1, Foc1))
 ->
    {Goal1 \== Foc1}.
```

(10) Assembly_language has a prerequisite that was taken.
Assembly_language does not conflict.

Figure 5. Combining propositions using subordination

Other tests for combining propositions look for identities among the arguments of propositions. Simplified examples of these rules are *id_del* and *foc_del* in Figure 6. According to *id_del*, if the first and second proposition differ only by the arguments *Goal1* and *Goal2*, these arguments are combined into one *Goal* and returned in the third proposition. The result is a single sentence containing a noun phrase conjunction as sentence (11) illustrates. The other rule, *foc_del*, specifies that if two propositions have the same focus, *Foc*, and in the second proposition, the focus specifies the protagonist, then the two propositions can be joined by conjunction, deleting the focused protagonist of the second proposition. Rather than returning a proposition, *foc_del* in its right-hand-side, invokes rules for generating a compound sentence. Sample text generated by this rule is shown in (12).

```
id_del (
    prop (Verb, Prot, Goal1, Ben, Foc),
    prop (Verb, Prot, Goal2, Ben, Foc),
    prop (Verb, Prot, Goal, Ben, Foc))
 ->
    {Goal1 \== Goal2, append (Goal1, Goal2, Goal)}.

foc_del (
    prop (Verb1, Prot1, Goal1, Ben1, Foc),
    prop (Verb2, Foc, Goal2, Ben2, Foc))
 ->
    sentence (prop(Verb1, Prot1, Goal1, Ben1, Foc)),
    [and],
    {getnum (Foc, Num)},
    verb_phrase (Verb2, Num, Goal2, Bene2, act_gb).
```

(11) Analysis_of_algorithms requires Data_structures and Discrete_math.
(12) Introduction_to_computer_programming does not have prerequisites and does not conflict.

Figure 6. Combining propositions using identity deletion

The grammar also makes use of the organization of the input to indicate causal connectives. An explanation for why a certain course was not scheduled is shown in (13). The antecedent goals are presented in the first part of the explanation; the consequence, introduced by *therefore*, follows.

(13) Modeling_and_analysis_of_operating_systems requires Fundamental_algorithms, Computability_and_formal_languages, and Probability.
Fundamental_algorithms and Computability_and_formal_languages were taken.
Probability was not taken.
Therefore, Modeling_and_analysis_of_operating_systems was not added.

5. FUTURE DIRECTIONS

The current implementation can be extended in a variety of ways to produce better connected text. Additional research is required to determine how and when to use other textual connectives for combining propositions. For example, the second and third sentences of (13) might be better expressed as (14).

(14) Although Fundamental_algorithms and Computability_and_formal_languages were taken, Probability was not taken.

The question of how to organize propositions and how to design the grammar to handle various organizations deserves further attention. In the current implementation, the grammar is limited to handling propositions structured as a list of antecedents and a list of consequences. If propositions were organized in trees rather than lists, as in more complex explanations, the use of additional connectives would be necessary.

The grammar can also be extended to include tests for other kinds of surface choice such as definite/indefinite reference, pronominalization, and lexical choice. As the grammar grows larger and more complex, the task of specifying rules becomes unwieldy. Further work is needed to devise a method for automatically generating DCG rules.

6. CONCLUSIONS

We have shown how focus of attention can be used as the basis for a language generator to decide when to combine propositions. By encoding tests on functional information within the DCG formalism, we have implemented an efficient generator that has the same benefits as a functional grammar:

input is simplified and surface structure can be determined based on constituents' function within the sentence. In addition to producing natural language explanations for the student advisor application, this formalism provides a useful research tool for experimenting with techniques for automatic text generation. We plan to use it to investigate additional criteria for determining surface choice.

REFERENCES

[1] Kay, Martin, "Functional Grammar," *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, (1979).

[2] Pereira, Fernando C. N. and Warren, David H. D., "Definite Clause Grammars for Language Analysis-A Survey of the Formalism and a Comparison with Augmented Transition Networks," *Artificial Intelligence* 13 pp. 231-278 (1980).

[3] McKeown, Kathleen R., *Generating Natural Language Text in Response to Questions about Database Structure*, PhD thesis, University of Pennsylvania (1982).

[4] McDonald, David D., "Description directed control: its implications for natural language generation," *Computers and Mathematics with Applications* 9(1) pp. 111-129 (1983).

[5] Davey, Anthony, *Discourse Production*, Edinburgh University Press (1978).

[6] Appelt, Douglas E., "Telegram: a grammar formalism for language planning," *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pp. 74-78 (1983).

[7] Davis, Randall and Lenat, Douglas B., *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York (1982).