

Using Focus to Generate Complex and Simple Sentences

Marcia A. Derr

AT&T Bell Laboratories
Murray Hill, NJ 07974 USA

and

Department of Computer Science
Columbia University

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 natural language generation are deciding *what to say* and *how to say it*. This paper addresses issues in the second of these tasks, that of *surface generation*. Given a semantic representation of what to say, a surface generator must construct an appropriate surface structure taking into consideration a wide variety of alternatives. When a generator is used to produce *text* and not just single sentences, one decision it must make is whether to use a sequence of simple sentences or a single complex one. We show how focus of attention can be used as the basis on which this decision can be made.

A second goal of this paper is to introduce a formalism for surface generation that uses aspects of Kay's functional grammar (Kay, 1979) within a Prolog definite clause grammar (Pereira and Warren, 1980). This formalism was used to implement a surface generator that makes choices about sentence complexity based on shifts in focus of attention. The implementation was done as part of an explanation facility for a student advisor expert system being developed at Columbia University.

2. Language Generation Model

In our model of natural language generation, we assume that the task of generating a response can be divided into two stages: determining the semantic content of the response and choosing a surface structure¹. One

component makes decisions about which information to include in the response and passes this information to a surface generator. For example, an expert system explanation facility may select part of the goal tree, a particular goal and its antecedent subgoals, to explain a behavior of the system. In the advisor system, the output of this component consists of one or more logical propositions where each proposition consists of a predicate relating a group of arguments. The output includes functional information, such as focus, and some syntactic features, such as number and tense, for convenience. Other information, such as the relationships between propositions, is implicit in the organizational structure of the output.

The output of the semantic component is passed on to another component, the surface generator. The job of generator is to use whatever syntactic and lexical information is needed to translate the logical propositions into English. The generator must be able to make choices concerning various alternatives, such as whether to use active or passive voice, or when to pronominalize. While we have found the explanation facility for the advisor system to be a valuable testbed for the surface generator, the generator is an independent module that can be transported to other domains by changing only the vocabulary.

3. Choosing Surface Structure

Given a set of propositions, one decision a surface generator must make is whether to produce a simple sentence for each proposition or whether to combine propositions to form complex sentences. As an example, consider propositions 1 and 2 below. These may be expressed as two simple sentences (sequence 1) 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

1. In order to concentrate on the task of surface generation, these two stages are totally separate in our system, but we don't dispute the value of interaction between the two (Appelt, 1983).

whether to use the active or the passive voice. Thus, proposition 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.

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 (1982) demonstrated how focus can be used to select sentence voice, and to determine whether pronominalization is called for. In this paper, we will show how focus can also be used as the basis on which to combine propositions.

3.1 Linguistic Background

Grosz (1977) distinguished between two types of focus: *global* and *immediate*. Immediate focus refers to how a speaker's center of attention shifts or remains constant over two consecutive sentences, while global focus describes the effect of a speaker's center of attention throughout a sequence of discourse utterances on succeeding utterances. In this paper, when we refer to "focus of attention," we are referring to immediate focus.

Phenomena reflecting immediate focus in text have been studied by several linguists. Terminology and definitions for these vary widely; some of the names that have emerged include topic/comment, given/new, and theme/rheme. These linguistic concepts describe distinctions between functional roles elements play in a sentence. In brief, they can be defined as follows:

- *Topic*: Constituents in the sentence that represent what the speaker is talking about. Comment labels constituents that represent what s/he has to say about that topic (see SGall, Hajicova, and Benesova, 1973; Lyons, 1968; Reinhart, 1981).
- *Given*: Information that is assumed by the speaker to be derivable from context where context may mean either the preceding discourse or shared world knowledge. New labels information that cannot be derived (see Halliday, 1967; Prince, 1979; and Chafe, 1976).
- *Theme*: The Prague School of linguists (see Firbas, 1966; Firbas, 1974) define the theme

of a sentence as elements providing common ground for the conversants. Rheme refers to elements that function in conveying the information to be imparted. In sentences containing elements that are contextually dependent, the contextually dependent elements always function as theme. Thus, the Prague School version is close to the given/new distinction with the exception that a sentence always contains a theme, while it need not always contain given information².

What is important here is that each of these concepts, at one time or another, has been associated with the selection of various syntactic structures. For example, it has been suggested that new information and rheme usually occur toward the end of a sentence (e.g., Halliday, 1967; Lyons, 1968; SGall et al., 1973; Firbas, 1974). To place this information in its proper position in the sentence, structures other than the unmarked active sentence may be required (for example, the passive). Structures such as *it-extrapolation*, *there-insertion*, *topicalization*, and *left-dislocation*³ have been shown to function in the introduction of new information into discourse (Sidner, 1979; Prince, 1979), often with the assumption that it will be talked about for a period of time (Joshi and Weinstein, 1981). Pronominalization is another linguistic device associated with these distinctions (see Akmajian, 1973; Sidner, 1979).

One major difference between linguistic concepts and immediate focus is that focusing describes an active process on the part of speaker and listener. However, the speaker's immediate focus influences the surfacing of each of the linguistic concepts in the text. It influences topic (and Halliday's theme) in that it specifies what the speaker is focusing on (i.e., talking about) now. But it also influences given information in that immediate focus is linked to something that has been mentioned in the previous utterance and thus, is already present in the reader's consciousness. Since immediate focus is intimately related to the linguistic definitions of functional information, the influence of functional information on the surface structure of the sentence can be extended to immediate focus as well.

2. Halliday also discusses theme (Halliday, 1967), but he defines theme as that which the speaker is talking about now, as opposed to given, that which the speaker *was* talking about. Thus, his notion of theme is closer to the concept of topic/comment articulation. Furthermore, Halliday always ascribes the term theme to the element occurring first in the sentence.

3. Some examples of these constructions are:

1. It was Sam who left the door open. (it-extrapolation)
2. There are 3 blocks on the table. (there-insertion)
3. Sam, I like him. (left-dislocation)
4. Sam I like. (topicalization)

3.2 Focus and Complex Sentences

While previous research in both linguistics and computer science has identified focus as a basis for choosing sentence voice and for deciding when to pronominalize, its influence on selecting complex sentence structure over several simple sentences has for the most part gone unnoticed. If a speaker wants to focus on a single concept over a sequence of utterances, s/he may need to present information about a second concept. In such a case, a temporary digression must be made to the second concept, but the speaker will immediately continue to focus on the first. To signal that s/he is *not* shifting focus, the speaker can use subordinate sentence structure in describing the second concept.

Suppose that, in the previous example, 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 using coordination as in the textual sequence shown in 7 below.

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 with focus as a basis for combining propositions as shown in the example below. In proposition 3, the values of predicate, protagonist, and focus match the values of the corresponding arguments in proposition 4. Thus, the two propositions can be joined by deleting the protagonist and predicate of the second proposition and using conjunction to combine the two goals as in sentence 8. Note that if the focus arguments were different, the propositions could not be combined on this basis. Propositions 5 and 6, with matching values for focus can also be combined by using coordination and deleting the focused protagonist in the second proposition (sentence 9).

proposition 3: predicate = buy protagonist = John goal = book focus = John	proposition 4: predicate = buy protagonist = John goal = cassette focus = John
proposition 5: predicate = read protagonist = Mary goal = book focus = Mary	proposition 6: predicate = play protagonist = Mary goal = cassette focus = Mary
8. John bought a book and a cassette. 9. Mary read the book and played the cassette.	

4. A Formalism for Surface Generation

In this section we discuss the Prolog definite clause grammar (DCG) formalism (Pereira and Warren, 1980) and how it can be used for language generation, as well as recognition. We then review the functional grammar formalism (Kay, 1979) that has been used in other generation systems (e.g., McKeown, 1982; Appelt, 1983). Finally, we describe how aspects of a functional grammar can be encoded in a DCG to produce a generator with the best features of both formalisms.

4.1 Definite Clause Grammars

The DCG formalism (Pereira and Warren, 1980) is based on a method for expressing grammar rules as clauses of first-order predicate logic (Colmerauer, 1978; Kowalski, 1980). As discussed by Pereira and Warren, DCGs extend context-free grammars in several ways that make them suitable for describing and analyzing natural language. DCGs allow nonterminals to have arguments that can be used to hold the string being analyzed, build and pass structures that represent the parse tree, and carry and test contextual information (such as number or person). DCGs also allow extra conditions (enclosed within brackets '{' and '}') to be included in the rules, providing another mechanism for encoding tests. A simple sentence grammar is shown in Figure 1.

Viewed as a set of grammar rules, a DCG functions as a declarative description of a language. Viewed as a set of logic clauses, it functions as an executable program for analyzing strings of the language. In particular, a DCG can be executed by Prolog, a logic programming language that implements an efficient resolution proof procedure using a depth-first search strategy with backtracking and a matching algorithm based on unification (Robinson, 1965). To analyze a sentence, the sentence is encoded as an argument to a Prolog goal. Prolog attempts to prove this goal by matching it against the set of grammar clauses. If the proof succeeds, the sentence is valid and a second argument is instantiated to the parse tree structure. A recognition goal and its resulting parse tree are shown in Figure 1. More extensive examples can be found in Pereira and Warren (1980).

```

sentence(s(NP,VP)) -->
  n_phrase(NP,Num),v_phrase(VP,Num).

n_phrase(np(Noun),Num) -->
  noun(Noun,Num).

noun(n(Root,Num),Num) -->
  [the], [Word], {is_noun(Root,Word,Num)}.

v_phrase(vp(Verb,NP),Num) -->
  verb(Verb,Num), n_phrase(NP,N2).

verb(v(Root,Num,Tense),Num) -->
  [Word], {is_verb(Root,Word,Num,Tense)}.

is_noun(student,student,singular).
is_noun(student,students,plural).

is_noun(answer,answer,singular).
is_noun(answer,answers,plural).

is_verb(give,gives,singular,pres)
is_verb(give,give,plural,pres).
is_verb(give,gave,_,past).

```

Recognition Goal:

```
  sentence(T,[the,student,gave,the,answer],[]).
```

Result:

```
T = s(np(n(student,singular)),
      vp(v(give,singular,past),
          np(n(answer,singular))))
```

Generation Goal:

```
  sentence(s(np(n(student,singular)),
             vp(v(give,singular,past),
                 np(n(answer,singular)))),S,[]).
```

Result:

```
S = [the,student,gave,the,answer]
```

Figure 1. Language recognition and generation using a DCG

While Pereira and Warren concentrate on describing the DCG formalism for language recognition they also note its use for language generation, which is similar to its use for language recognition. The main difference is in the specification of input in the goal arguments. In recognition, the input argument specifies a surface string that is analyzed and returned as a parse tree structure in another argument. In generation, the input goal argument specifies a deep structure and a resulting surface string is returned in the second argument. Though not always practical, grammar rules can be designed to work in both directions (as were the rules in Figure 1). A generation goal and the sentence it produces are shown in Figure 1.

4.2 Functional Grammars

Another formalism that has been used in previous generation systems (McKeown, 1982; Appelt, 1983) is the functional grammar formalism (Kay, 1979)⁴. In a

functional grammar, functional information such as *focus* and *protagonist* are treated in the same manner as syntactic and grammatical information such as *subject* and *NP*. By using functional information, input to the generator is simplified as it need not completely specify all the syntactic details. Instead, tests on functional information, that select between alternative surface structures, can be encoded in the grammar to arrive at the complete syntactic structure from which the string is generated. This formalism is consistent with the assumption that is part of our generation model: that one generation component produces a semantic specification that feeds into another component for selecting the final surface structure.

In the functional grammar formalism, both the underlying message and the grammar are specified as functional descriptions, lists of attribute-value pairs, that are unified⁵ to produce a single complete surface structure description. The text is then derived by linearizing the complete surface structure description. As an example, consider the proposition encoded as a functional description below. When unified with a sentence grammar that contains tests on focus to determine voice and order constituents, sentence 12 is generated. If FOCUS were <GOAL>, instead, sentence 13 would result.

```
CAT = S
PRED = [LEX = give]
TENSE = PAST
PROT = [LEX = student]
GOAL = [LEX = answer]
BENEF = NONE
FOCUS = <PROT>
```

12. The student gave the answer.
13. The answer was given by the student.

Previous implementations of functional grammars have been concerned with the efficiency of the functional grammar unification algorithm. Straightforward implementations of the algorithm have proved too time-consuming (McKeown, 1982) and efforts have been made to alter the algorithm to improve efficiency (Appelt, 1983). Efficiency continues to be a problem and a functional grammar generator that can be used practically has as yet to be developed.

4.3 Combining the Formalisms

We have implemented a surface generator based on both the DCG formalism and the functional grammar

-
4. Functional grammar has also been referred to as *unification grammar* (Appelt, 1983).
 5. The functional grammar unification operation is similar to set union. A description of the algorithm is given in Appelt (1983). It is not to be confused with the unification process used in resolution theorem proving, though a similarity has been noted by Pereira and Warren (1983).

formalism. The result is a generator with the best features of both grammars: simplification of input by using functional information and efficiency of execution through Prolog. Functional information, supplied as part of the generation goal's input argument, is used by the grammar rules to select an appropriate surface structure. The extra conditions and context arguments allowed by the DCG formalism provide the mechanism for testing functional information.

Figure 2 shows a proposition encoded as the input argument to a DCG goal. The proposition specifies, in order, a predicate, protagonist, goal, beneficiary, and focus. In this example, the focus argument is the same as the protagonist. While the proposition also includes tense and number information, less syntactic information is specified compared to the input argument of the generation goal in Figure 1. In particular, no information regarding constituent order is specified. Also shown in Figure 2 are some DCG rules for choosing syntactic structure based on focus. The rules test for number agreement and tense, as well. The *sentence* rule selects the focused argument as the subject noun phrase. The *vp* rule determines that focus is on the protagonist, selects active voice, and puts the goal into a noun phrase followed by the beneficiary in a *to* prepositional phrase. Thus, the order of constituents in the generated sentence is not explicitly stated in the input goal, but is determined during the generation process. The sentence that results from the given proposition is shown at the bottom of Figure 2.

Generation Goal:

```
sentence(prop(pred (give, past),
             arg(student, singular),
             arg(answer, singular),
             arg(nil, _),
             arg(student, singular)),S,[I]).
```

Rules:

```
sentence(prop(Pred,Prot,Goal,Bene,Foc)) -->
    np(Foc),
    vp(Pred,Prot,Goal,Bene,Foc).

vp(pred(Verb,Tense),Prot,Goal,Bene,Prot) -->
    {getnum(Prot,Num)},
    verb(Verb,Tense,Num,active),
    np(Goal),
    pp(to,Bene).
```

Result:

```
S = [the,student,gave,the,answer]
```

Figure 2. DCG rules that use focus to select syntactic structure

5. Surface Generator Implementation

A surface generator, with mechanisms for selecting surface structure and, in particular, combining propositions, was 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 results of the planning process can be derived from a trace of the Prolog goals that were invoked during planning (Davis and Lenat, 1982). 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. This provides a rich testbed on which to experiment with techniques for combining propositions. Because the expert system does not yet automatically generate a trace of its execution, the propositions that served as input to the surface generator were hand-encoded from the results of several system executions. In the current implementation, the grammar is limited to handling input propositions structured as a list of antecedents (subgoals) followed by a single consequence (goal).

A grammar for automatically generating explanations was implemented using the formalism described in the previous section. The grammar encodes several tests for combining propositions. Based on temporary focus shift, it forms complex sentences using subordination. Based on focus and argument identities it uses coordination and identity deletion to combine propositions. The grammar also includes tests on focus for determining active/passive sentence voice, but does not currently pronominalize on the basis of focus.

The generator determines that subordination is necessary by checking whether focus shifts over a sequence of three propositions. A simplified example of a DCG rule *foc_shift*, that tests for this is shown in Figure 3. 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*, and if the second proposition specifies *Goal1* as its focus,⁶ 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 14 below.

6. 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}.

```

14. *Assembly Language* has a prerequisite that was taken. *Assembly Language* does not conflict.

Figure 3. 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 4. 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 15 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 form a coordinate sentence, deleting the focused protagonist of the second proposition. Instead of 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 16.

```

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, Prot2, Goal2, Ben2, Foc))
-->
sentence (prop(Verb1, Prot1, Goal1, Ben1, Foc)),
[and],
verb_phrase (Verb2, Prot2, Goal2, Ben2, Foc).

```

15. *Analysis of Algorithms* requires *Data Structures* and *Discrete Math*.

16. *Introduction to Computer Programming* does not have prerequisites and does not conflict.

Figure 4. Combining propositions using coordination and identity deletion

The generator uses of the organization of the input to show causal connectives. Recall that the input to the generator is a set of propositions divided into a list of antecedents and a single consequence that was derived by the expert system. The generator can identify the consequence for the reader by using a causal connective.

An explanation for why a particular course was not scheduled is shown in 17. The antecedents are presented in the first part of the explanation; the consequence, introduced by *therefore*, follows.

17. *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.

6. Related Work in Generation

There are two basic classes of related work in generation. The first class of systems makes use of functional information in constructing the surface structure of the text and has relatively little to say about how and when to produce complex sentences. The second class of work has addressed the problem of producing complex sentences but does not incorporate functional information as part of this decision making process.

Of the systems which make use of functional information, three (Kay, 1979; McKeown, 1982; Appelt, 1983) have already been mentioned. Kay's work provides the basis for McKeown's and Appelt's and emphasizes the development of a formalism and grammar for generation that allows for the use of functional information. Both McKeown and Appelt make direct use of Kay's formalism, with McKeown's emphasis being on the influence of focus information on syntax and Appelt's emphasis being on the development of a facility that allows interaction between the grammar and an underlying planning component.

Nigel (Mann, 1983) is a fourth system that makes use of functional information and is based on systemic grammar (Hudson, 1974). A systemic grammar contains choice points that query the environment to decide between alternatives (the environment may include functional, discourse, semantic, or contextual information). Mann's emphasis, so far, has been on the development of the system, on the development of a large linguistically justified grammar, and on the influence of underlying semantics on choices. The influence of functional information on syntactic choice as well as the generation of complex propositions are issues he has not yet addressed within the systemic grammar framework.

Of previous systems that are able to combine simple clauses to produce complex sentences, Davey's (1978) is probably the most sophisticated. Davey's system is able to recognize underlying semantic and rhetorical relations between propositions to combine phrases using textual connectives, also an important basis for combining propositions. His emphasis is on the identification of contrastive relations that could be specified by connectives such as *although*, *but*, or *however*. While Davey uses a systemic grammar in his generator, he does not exploit the

influence of functional information on generating complex sentences.

Several other systems also touch on the generation of complex sentences although it is not their main focus. MUMBLE (McDonald, 1983) can produce complex sentences if directed to do so. It is capable of ignoring these directions when it is syntactically inappropriate to produce complex sentences, but it can not decide when to combine propositions. KDS (Mann, 1981) uses heuristics that sometimes dictate that a complex sentence is appropriate, but the heuristics are not based on general linguistic principles. Ana (Kukich, 1983) can also combine propositions, although, like Davey, the decision is based on rhetorical relations rather than functional information.

In sum, those systems that are capable of generating complex sentences tend to rely on rhetorical, semantic, or syntactic information to make their decisions. Those systems that make use of functional information have not investigated the general problem of choosing between complex and simple sentences.

7. 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 17 might be better expressed as 18.

18. 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 input propositions structured as a list of antecedents and a single consequence. 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.

8. 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.

Acknowledgments

We would like to thank Mitch Marcus for his comments on an earlier version of this paper.

References

- Akmajian, A. (1973), "The role of focus in the interpretation of anaphoric expressions," In Anderson and Kiparsky (Ed.), *Festschrift for Morris Halle*, Holt, Rinehart, and Winston, New York, NY, 1973.
- Appelt, Douglas E. (1983), "Telegram: a grammar formalism for language planning," *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 74-78, 1983.
- Chafe, W. L. (1976), "Givenness, contrastiveness, definiteness, subjects, topics, and points of view," In Li, C. N. (Ed.), *Subject and Topic*, Academic Press, New York, 1976.
- Colmerauer, A. (1978), "Metamorphosis grammars," In Bolc, L. (Ed.), *Natural Language Communication with Computers*, Springer, Berlin, 1978.
- Davey, Anthony. (1978), *Discourse Production*, Edinburgh University Press, 1978.
- Davis, Randall and Lenat, Douglas B. (1982), *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.
- Firbas, J. (1966), "On defining the theme in functional sentence analysis," *Travaux Linguistiques de Prague* 1, University of Alabama Press, 1966.
- Firbas, J. (1974), "Some aspects of the Czechoslovak approach to problems of functional sentence perspective," *Papers on Functional Sentence Perspective*, Academia, Prague, 1974.
- Grosz, B. J. (1977), The representation and use of focus in dialogue understanding. Technical note 151, Stanford Research Institute, Menlo Park, CA, 1977.
- Halliday, M. A. K. (1967), "Notes on transitivity and theme in English," *Journal of Linguistics*; 3, 1967.
- Hudson, R.A. (1974), "Systemic generative grammar," *Linguistics*, 139, 5-42, 1974.
- Joshi, A. and Weinstein, S. (1981), "Control of inference: role of some aspects of discourse structure - centering," *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981.
- Kay, Martin. (1979), "Functional grammar," *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, 1979.
- Kowalski, R. A. (1980), *Logic for Problem Solving*, North Holland, New York, NY, 1980.
- Kukich, Karen. (1983), "Design of a knowledge-based report generator," *Proceedings of the 21st Annual Meeting of the Association for Computational*

- Linguistics*, 145-150, 1983.
- Lyons, J. (1968), *Introduction to Theoretical Linguistics*, Cambridge University Press, London, 1968.
- Mann, W.A. and Moore, J.A. (1981), "Computer generation of multiparagraph English text," *American Journal of Computational Linguistics*, 7 (1), 17-29, 1981.
- Mann, William C. (1983), "An overview of the Nigel text generation grammar," *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 79-84, 1983.
- McDonald, David D. (1983), "Description directed control: its implications for natural language generation," *Computers and Mathematics with Applications*, 9 (1), 111-129, 1983.
- McKeown, Kathleen R. (1982), *Generating Natural Language Text in Response to Questions about Database Structure*, Ph.D. dissertation, University of Pennsylvania, 1982.
- Pereira, Fernando C. N. and Warren, David H. D. (1980), "Definite clause grammars for language analysis--a survey of the formalism and a comparison with augmented transition networks," *Artificial Intelligence*, 13, 231-278, 1980.
- Pereira, Fernando C. N. and Warren, David H. D. (1983), "Parsing as deduction," *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 137-144, 1983.
- Prince, E. (1979), "On the given/new distinction," *CLS*, 15, 1979.
- Reinhart, T. (1981), "Pragmatics and linguistics: an analysis of sentence topics," *Philosophica*, 1981.
- Robinson, J. A. (1965), "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, 12 (1), 23-41, 1965.
- Sgall, P., Hajicova, E., and Benesova, E. (1973), *Focus and Generative Semantics*, Scriptor Verlag, Democratic Republic of Germany, 1973.
- Sidner, C. L. (1979), *Towards a Computation Theory of Definite Anaphora Comprehension in English Discourse*, Ph.D. dissertation, MIT, Cambridge, MA, 1979.