MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working paper 103 June 1975

A Preliminary Report on a Program for Generating Natural Language

by

David McDonald

Abstract

A program framework has been designed in which the linguistic facts and heuristics necessary for generating fluent natural language can be encoded. The linguistic data is represented in annotated procedures and data structures which are designed to make English translations of already formulated messages given in a primary program's internal representation. The messages must include the program's intentions in saying them, in order to adequately specify the grammatical operations required for a translation.

The pertinant questions in this research have been: what structure does natural language have that allows it to encode mutifaceted messages; and how must that structure be taken into account in the design of a generation facility for a computer program.

This paper describes the control and data structures of the design and and their motivation. It is a condensation of my Master's Thesis <1>, to which the reader is refered for further information. Work is presently underway on implementing the design in LISP and developing a grammar for use in one or more of the domains given below.

This paper will appear in the Proceedings of the Fourth International Joint Conference on Artificial Intelligence, September 3-8, 1975, Tibilisi, USSR.

This report describes research done at the Artificial Inteligence Laboratory of the Massachusetts Institute of Technology. Suport for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research contract N00014-70-A-0362-0003.

Introduction

At the present time, there are intelligent, interactive programs under development which will require greater fluency in generating natural language than any current system can offer. Three such programs, in particular, are a personal scheduling program (Goldstein <2>), a programmer's assistant (Rich and Shrobe <3>), and the MACSYMA advisor (Geneserith <4>).

A characteristic of all these programs is that they will employ models of their users - of their habits, and the things they are likely to know in various situations. They will also maintain models of themselves and their intentions as they reason and interact with their users. This will have a large effect on the design of a suitable generating facility for them.

Level of Fluency Desired

The sort of conversations we hope these programs will eventually be able to have are typified by the short example below, between a scheduling program (P) playing the part of a secretary scheduling appointments for a professor, and a student (S).

(S) I want to see Professor Winston sometime in the next few days.
(P) He's pretty busy this week. Can it wait?
(H) No, I need his signature on my petition before Friday.
(P) Well, maybe he can squeeze you in tomorrow morning. Give me your name and check back in an hour.

These are very fluent answers by human standards. The wording is colloquial - "pretty busy", "squeeze you in" - but at the same time, it deliberately conveys useful information; casual speech can give an impression of flexibility. Impressions like this can be as important to the total message as the sentence's propositional content. Similarly, the use of "well" at the beginning of a reply can signal an admission on the part of the speaker that the answer which follows may not be adequate. This realizes a need within the discourse situation that is

properly part of conversations among people, and should be included in conversations between computers and people.

A Separate Linguistic Component

The total process of generating language involves making a large variety of decisions and having available the information on which to base them. The initial urge to speak comes from somewhere to fulfill some need which must then be made more precise. Models of the audience, their present knowledge and expectations, must be consulted. The available procedures for putting together an utterance will never be totally adequate and therefore compromises must be reached. Finally, the most appropriate linguistic representations must be found and organized, probably requiring more compromise. Despite the intricacy of this process, I believe it is both meaningful and profitable to divide it two: decision making which requires cognitive/domain knowledge, versus decisions requiring linguistic knowledge.

In some early communicating programs (Winograd <5>, Woods <6>), linguistic and domain knowledge were freely mixed. This was possible because the programs only worked in a very small number of situations where the relevant linguistics could be "built in". However, if, as seems to be the case, an extensive amount of "linguistic reasoning" is required to meet the needs of more sophisticated programs (and people), then the programming difficulties of designing a mixed system become insurmountable. (n.b. the more recent systems, Goldman <7>, and Slocum <8>, incorporate essentially the same division as my own.)

A possible objection to this compartmentalization is that people don't work that way. A grammar that is organized in a different way than human grammar may have difficulties

representing the same rules. People often consider the potential impact on their audience of their use of particular words, or of the ordering of their phrases. Poets, in particular, are certainly as conscious of choosing syntax and meter as they are of choosing cognitive content.

When I say that the generation process should be divided in two, I do intend the strongest interpretation: namely, that the "cognition" will have totally developed the message to be communicated before any linguistic processing is done. Futhermore, the linguistic processing should not change the meaning in any way that the cognitive domain cares about, and the message, once determined, should not be modified. This design will, indeed, not be able to behave as people do on the tasks above.

However, I believe that computer programs will not be able to motivate such behavior for a long time, and that the division in the design is a useful one for the purpose of current research. (In my thesis, <1>, I describe how the design might be upgraded to handle increasingly sophisticated human linguistic behavior, and what additional information such behavior would require the cognitive "component" to have.)

The Translation Process

In this design, the generation of English utterances in context is seen as involving two, computationally separate components. (All of this work has been done in English, though there should not be any difficulty in applying it in other languages.) Since there are two components, there must be a communications channel between them and a language which the cognitive component uses to describe to the linguistics component what it wants said (the use of the term "linguistics component" in this paper only refers to generation processes and not to interpretive processes, see <1>).

Messages

Before the linguistic component is called in, the main program (for example <2>,<3>, or <4>) has as total a picture as it needs of what is to be said. It knows that it wants to mention certain particular entites and certain relations among them, and to achieve a certain effect upon its audience. To communicate with the linguistic component, this information is collected into a data structure, a "message". A message can be viewed as consisting of two sorts of things.

1. A collection of pointers to the internal objects that are to be talked about. The pointers are annotated to describe how the objects relate to each other and the message as a whole.
2. A list of features which characterize the primary program's communicative intent in making that message.

The following structure is an example of what a message might look like. The word on the left of each pair is the annotation, and the phrases on the right in angle-brackets represent objects in the main program with roughly the meaning of those phrases.

```
message-1
  features  (prediction)
  proposition
     <status-of-person>
          |type <busy>
          |of  <Winston>
  hedge  <70% chance>
  time-predicted  <12:00-17:00>
```

This message may be translated into the sentence: "Professor Winston will probably be busy all afternoon".

The character of the translation

Translation from the internal representation of a computer program to natural language is very much like translating from one language to another, and the same problems arise.

Basically, the problems are that the same concepts may not exist as primitives in each language, and that the conventions of the target language may require additional information that was not in the source language. Translation, therefore, cannot be simply one for one.

What English phrase should be used for a particular element in the program's message will vary as a function of what is in the rest of the message and of what the external context is. To allow these factors to be adequately considered, the translation of an element is carried out by special procedures called "composers" which may take into account a wide variety of phenomena as they do their translation.

Every concept, name, structure, process, or other entity which the main program might employ in a message will be associated with such a describing procedure. The association might be a direct pointer from a unique name in the program to its composer, or it might be derived by examining "is-a" links or "type" features associated with the object (e.g. all things of type "event" might share a composer). Composers are run at predetermined points in the translation process and are designed to expect a particular computational/grammatical context at that point.

The translation to English, then, does not employ a single, unified grammar, such as an ATN (<7>,<8>). Instead, the grammatical information is distributed among the individual composers. This is in part a matter of programming aesthetics, and in part due to a belief that the attendant increase in modularity and flexibility will make the grammar more tractable and easier to improve.

Control Structure

With the grammar distributed among a very large number of separate processes, the task of coordinating their actions becomes of paramount importance. Roughly speaking, the translation process has this character: the intentions and objects in the message will suggest (via their composers) strategies for realizing themselves in English. However, these strategies may be blocked or modified by general linguistic information in "the grammar", or by the effects of decisions made by earlier strategies. (The term "the grammar" refers to the collective information in the composers and their data structures, rather than some central body of constraints.)

The control structures required to implement this process are themselves, very simple. This is because the descriptive apparatatus of the grammar can provide a rich enough description of the situation to direct the actions of the composers and keep them, in effect, from tripping over each other's feet.

This would not be possible if it were not for the fact that natural languages are very complex entities with rich structures. In more concrete terms, this is to say that languages are made up of a relatively large number of types of structures (noun phrases, function words, inflectional endings, modifiers, etc.) and that the possible arrangements of these structures are very highly restricted - only a few combinations are possible. By encoding this information into a system of features and data structures, and then writing a grammar for the composers in terms of that system, a tremendous, implicit coordination should be achieved.

This coordination does not come automatically of course. Since situations are defined in terms of features, a composer will recognize where it is by using conditional statements

involving those features. The larger the number of possible situations that a particular composer may be run in, the more intricate its conditionals will be, and the harder the composer will be for the human designer to write.

Part of the job of cutting down on the complexity can be done in the grammar by increasing each feature's descriptive power (and probably adding to their total number). The more information that a given feature codes for, then the more decisions that can be made solely on its basis. An even more effective technique is to closely control when the individual composers will be run. This can provide implicit situational information. Also, if it can be arranged that a decision, once made, seldom has to be reconsidered, then a considerable overhead in mechanism and composer code will be saved.

The requirement of linear order

One of the fundamental characteristics of natural language is that utterances are necessarily made up of linear strings of words. This requirment is an inescapable fact of the "physics" of natural language and accordingly, it has been given a large role in conveying information. It can realize propositional meaning and rhetorical intent, and permit abbreviations throughout the utterance for example.

Fortunately for the program designer and the linguist, those things that are ordered are not arbitrary clumps of words, but rather structural units (noun phrase, adverb, etc.) with two important characteristics.

1. They seem (not coincidentally) to describe categories of experience which are natural to us as people and which we will probably want to introduce into our computer programs.
2. Linguistically speaking, they are very modular and can usually be "moved" to several

different positions within an utterance to achieve rhetorical effects, and require only minimal, well specified structural changes in each position.

The majority of the descriptive composers in the lexicon will describe their corresponding internal entities with just such coherent grammatical structures. Most of the syntactic details of these structures vary with position in the sentence. If these composers can be given a guarantee that the position of their object will not be shifted as the utterance is further developed, there will be a considerable savings in the complexity of individual composers and in the overhead required to manage them.

Two Phases

To provide this quarantee, the translation process is divided into two phases. During the first phase, the message as a whole is examined according to the intentions given for it and the annotation for each object that it mentions. A "plan" is selected for it (see below) which embodies the syntactic structure of the ultimate utterance and which has "slots" in it into which the largely "unexpanded" objects of the message are transfered. In this phase, all of the elements in the message which will involve ordering conventions in their realization (translation) are found and the plan modified to accomodate them.

During the second phase, the developed plan, which is essentially a constituent structure tree with the possible positions explicitly labeled, is "walked" from left to right and top down - as it would be spoken - and the objects in it are described by their composers as they are encountered. With the "proto-utterance" represented in its surface structure form during this phase, relationships become apparant which could not otherwise be seen. These include the

possibilites for pronominalizing elements, and the actual scope of quantifiers. These can be dealt with by syntactic procedures associated with the features and grammatical units in the plan.

Data-directed processing

In both phases, control is data-directed. In the first phase, the data structure being interpreted is the message, and in the second, it is the plan that was chosen and filled in during the first phase. This is another source of coordination for the composers since the control of their order of execution is now governed by structures which can be written to be very rich in grammatical information.

Let me summarize what has been said so far. This design proposes that a very loose, modular framework can be used in language generation; that it will be most convenient if the domain and audience specialists in the program be allowed to work out their message independently of its ultimate linguistic details; and that when the time comes to consider the linguistics, the process should be viewed as a translation which is performed by a large number of specialist procedures associated with the possible things which may appear in a message. The operation of these specialists can be coordinated implicitly by the grammar and data structures that are developed.

In the rest of this paper, I will describe some of operations and structures of this design in more detail.

Plans

All natural language programs and linguistic theories employ, in one form or another, a tree structured constituent analysis of their sentences in terms of the traditional grammatical units. My design is no different, except that I have found it necessary to augment the usual descriptive framework to make it capable of the task at hand. This has resulted in the data structure I call a "plan".

The principle function of plans is to mark the possible positions in a grammatical unit, in terms of a fixed vocabulary of slots such as "subject", "main verb", "post-verb-modifiers", and so on. The slots in a plan are arranged in a fixed order corresponding to the normal English surface structure. For example, if we used the grammar developed by Winograd in his SHRDLU program <5>, the slots in a noun phrase would be as follows.

```
NG - |determiner
     |ordinal
     |number
     |adjectives
     |clasifiers
     |head noun
     |qualifiers
```

Since the slots are named, they can be referred to directly from within the grammar, rather than requiring some complicated tree walk and string matching operation as in PROGRAMMAR <5>, or in transformational grammars. The grammar can, for example when it is doing verb agreement, ask what is the number of the object in the subject slot, or, when considering using extraposition, ask if the subject will be described using a clause.

The major motivation for naming the possible positions within a grammatical unit involves more than convenience in writing grammatical rules. The basic operation during the first

phase is to insert another element from the message into an established plan. To do this, the composing procedure must know: 1) what positions are open in this plan where it is grammatically feasable to put this element; and 2) if more than one position is available, in what ways do their properties vary so that a reasoned decision can be made as to which one is best in this case.

When the possible positions in an utterance are marked with unique names, it becomes possible to associate grammatical information with them to use in situations such as above. Most of this information will probably reside directly in the relevant composers, but some will be used by functions which mediate the insertion of an object into a slot.

The function of such mediation is to relieve the composers of the need to know low level syntactic information. The verb group is a prime example. Because of the intricacy of its syntax, it will be convenient to have only one slot, VG, in a clause or verb phrase plan, and let a function associated with VG manage a full verb group plan below it. The function determines what sub-slot should be filled (perhaps even changing the actual configuration of the slots) and adjusts the features of the group if necessary. This way, the bulk of the composers no longer need to know about details such as: "if you add a modal verb ("would") to a verb group, you have to add a marker to the main verb to inhibit the later morphological expression of tense".

Plans are associated with grammatical units, with possibly a separate plan for each set of grammatical features that a unit might have, reflecting the different slots that may be present in each case. By knowing the features of a unit, a composer will know exactly what slots to expect it to have. The next section has examples of how plans are used.

Translating a Message

Most of the work in the first phase is done by organizational composers associated with the intentional features on the messages. Each such composer will include code which understands the possible annotations that typically are mentioned with such intentions, and which will govern their insertion into a plan at the proper time. Consider the example message given earlier and repeated here.

```
message-1
  features (prediction)
  proposition
      <status-of-person>
          |type <busy>
          |of  <Winston>
  hedge  <70% chance>
  time-predicted  <12:00-17:00>
```

Here, the organizing composer will be associated with the feature "prediction". Typically, one element of the message will be most important and is translated first. The others will probably refer to it and may need to be realized inside the plan that it was translated into. In this case, the prime element is the one annotated "proposition", an object of the sort "status of a person". Plan selection is done by the descriptive composer for this sort and is guided by further characteristics of the object. The lexicon will record that the type property <busy> must be realized as a predicate adjective. This leads to the following, partially filled in plan.

```
node-1
  features (clause major copular pred-adj)
  slots   |pre-sentential-modifiers <>
          |subject <Winston>
          |vg BE
          |pred-adj "busy"
          |complement <>
          |post-sentential-modifiers <>
```

The rest of the message is transferred by the prediction composer chunk by chunk. Predictions are of future events, so "will" is added to the verb group; the "hedge", <70% chance> will be realized as an adverb, say, "probably", and so it is added to the adverb slot in the verb group; and "time-predicted" is a time modifier to the clause, making it part of the post-sentential-modifiers. With the entire message transfered, the plan looks like this.

```
node-1
  features (clause major copular pred-adj)
  slots  |pre-sentential-modifiers <>
         |subject <Winston>
         |vg node-2
              features (verb-group future modified)
              slots  |modal "will"
                     |pre-vb-adv "probably"
                     |mvb BE
         |pred-adj "busy"
         |complement <>
         |post-sentential-modifiers
              <12:00-17:00>
```

Annotating Composers

To properly fit the pointers/objects in a message into a plan, the organizing composer must know what sort of grammatical object they will be. This can not always be directly deduced from the nature of the annotation on the message. For example, in this sentence, the "hedge" might well have been an object corresponding to the phrase "unless something comes up", which is a bound clause and would have to go at the end of the post-sentential-modifiers.

The necessary information can be maintained by each descriptive composer as a permanant annotation in the form of a feature list which describes what sort of grammatical unit it constructs. To check this for an object in the message, an organizing composer will look up the object in the lexicon to see what composer will describe it, and then read the annotation on that composer. In this case, the features might be "(adverb event-modifer)",

versus "(clause bound conditional)".

An Example of a Composer

Each of the objects in a message will eventually be described by a general descriptive composer which is keyed to the sort of object that they are, plus additional information associated with the names in each object. As there are "sorts" of objects in a program, there will be descriptive composers in the lexicon. Some sorts might be reasons, actions, people, appointments, activities, times of the day, and so on. This design makes no restrictions on the possible composers; only that they should reflect what properties objects have in common and common ways that they can be described. Individual objects will usually only supply parameters to their composers, but some may be idiosyncratic and instead point to complete words or phrases or to specially tailored composing procedures.

Actions

Actions are things that something does: "making an appointment", "evaluating a procedure", "defending a chess piece", etc. The function of "the action composer" is to set up the syntactic environment that all actions have in common. In this analysis, actions are realized as verb phrases, with the internal name of the action indicating in the lexicon what the verb should be, and the objects asociated with the name (if any) becoming its syntactic objects in the phrase.

An example of an action in a likely internal representation might be the following (in a programmer's assistant)

```
<action-427>
    action  set-value-of
    variable  switch1  ;a name from a program
    set-to  nil
```

We might see this in an utterance like "it is necessary to set the value of switch1 to nil before leaving this routine".

The first thing any composer does when it begins to run is find out where it is. In the above utterance, the location would be in the second phase, with the action-object in the "complement" slot. Other slots where actions could occur are "subject", and as the main-proposition in an answer to a question. With the situation known, the composer might dispatch to a particular block of code which handles that situation, but in this case, the only difference is that complements must have infinitival verbs. This is done by adding a feature to the verb group at the end of the operation.

All actions yield verb groups, so the composer begins by replacing the pointer in the complement slot with a syntactic node for a verb group. It must then get a plan for this verb group, and fill in the appropriate slots of that plan with the subcomponents of the object. Then it is finished and the node and plan are in turn refined by their own composers as the second phase controler walks along the plan to them.

The information on what plan to use and what transfers to make is part of an object's specific lexical entry. To get at it, the action composer must know what property of the object describes its structure (of course, the programmer must see to it that such a property exists) and then follow it into the lexicon for a plan and a mapping of properties on the object to slots in the plan. For the example action this entry is given below.

```
set-value-of
   plan (vg SET  ;a pointer into a morphological lexicon
         object1 (noun group
                  det "the"
                  head "value"
                  of-group
                     (prep "of"
                      np <> ))
         prep "to"
         object2 <> )
   mapping
         ((variable object1.of-group.np)
          (set-to object2))
```

Note that this plan is not so much a grammatical skeleton as a variablized English phrase. With such information in the lexicon, the action composer can employ straightforward pattern substitution functions to finish its job.

The Syntactic Environment

The primary operation in the second phase is to describe the chunks of the message that have been embedded in the plan. This is done by walking the plan with a simple controler to run the composer for each object as it is encountered; print out the words given literally, and ignore any empty slots. Since a plan is esentially a constituent structure tree, walking it topdown, from left to right results in words being uncovered (and "spoken") in the same order as would occur if a human were making the utterance. At the same time, parts of the plan further on, which have not yet been walked, retain their unexpanded character, presenting those characteristics which may be important to know in decisions involving the whole utterance while hiding those details that are unimportant.

This points out that plans can be viewed as providing an environment that composers can

ask questions of. Some questions are easy because their answers are represented directly ("what is the transitivity of the main verb?") and others are much harder because they must be computed ("are there any intervening noun groups between me and that previous occurance of me way back there?" - needed for reflexive pronouns). However, in environments with different structures than that of plans, answering such question could become simple.

Such additional environments could be created as a side-effect of the construction and walking of a plan. Because the walking follows the temporal order of the generation of an utterance, it readily marks what the audience can be presumed to know at any given point.

I have not yet done any work on determining just what such parallel environments should look like. That will come as grammars are written for this design. However, it is clear that they must encode some very subtle aspects of what the audience knows, and should describe the syntactic situation in such a way as to guide pronominalization and "deletion" of later structures.

Pronominalization

Pronominalization is only one instance of a very general phenomena in language which "encourages" the speaker to abbreviate their utterance wherever possible. Languages contain conventional structures which themselves mark the relationships that are going on, so that the actual words need not be physically present (e.g. equi-np-deletion: "John is ready to please" - the subject "John" does not need to be repeated with each verb).

Often conventions which allow potential descriptions to be omitted take into account semantic information that the audience is assumed to share. For example consider the sentence "White's knight can take a pawn". What is interesting here is that there is no need

to say "... take a black pawn". Presumably, it is what we know about the semantics of "take" in chess games - that its subject and object will be pieces of opposite colors - which has taken effect here.

Every composer describing an object will have to examine the "discourse" environment to see if it would be most appropriate to use a pronoun or otherwise cut down on the normal amount of description.

Quantifier Scope

Certain relationships become apparent during the second phase that can not be seen at other times. One very important one is quantifier scope. Certain accidental misreadings can be generated as the plan is walked, precisely because the individual composers work independantly of each other. This can be corrected by introducing "global" syntactic processes associated with the grammatical units, which can "monitor" the activities of the composers and insert corrective patches when necessary.

Situations in the grammar where such accidents are possible must be identified and routines designed for them. Then, when any syntactic unit is entered by the second phase controler, a check will first be make for any monitoring routines, which are then run before going on.

One situation that would be checked for would be that of a verb followed by a conjoined object. The monitor would be associated with the verb group and go to work if it saw that a conjoined noun phrase followed. The problem is that the structure "(are (not A) and (B))" is usually misinterpreted by people as "(are not (A and B))" with the scope of "not" inadvertently taking in B as well. The monitor must watch as the first conjunct is described, and if it begins with "not", it should patch the construction by copying the "are" after the "and" - "(are not A

and are B)". A repetoire of such monitors and patches will be required in the grammar.

Present Directions

The design that I have described here (see <1> for greater detail) represents some contentions about what a very fluent "generation grammar" for English must deal with, and what control and data structures will be convenient to write that grammar in. At this writing (June 1975), a LISP implementation of the design is well under way, and it is anticipated that part of a grammar can be completed before the end of the summer. However, until a working grammar exists, and the generator has been interfaced with some primary program, many of the things described in this paper remain contentions which I believe to be true, but which may turn out to be without substance, necessitating possibly drastic changes in the design. In particular, the program has made these assumptions.

1. That the candidate primary programs will have a sufficiently rich organizing structure that very general composers can be written, cutting down on the bulk of the lexicon, and that associating objects with composers will be a straightforward thing to do.
2. That the messages constructed by a main program will naturally be translatable without editing. Some problems in a message could be patched by the grammar, but others, like too much necessary embedding, could not be fixed without going directly back to the program and "explaining" that some material must be cut, letting the main program decide what is to be left out.
3. The proposed grammar depends on having good information at all times, otherwise, the composers may thrash and will continually find themselves in unanticipated situations.

This information will be encoded in a system of features and possible plans and slots. It must be possible to devise an adequate grammatical system, or else the resulting inefficiences may swamp the generator.

4. The grammar will organize linguistic constructions in terms of the reasons why speakers use them. However, the reasons for using the bulk of the constructions in English are poorly understood. It is hoped that a combination of the fact that programs are presently rather simple minded compared to humans, and that initial hunches about the use of those grammatical constructions which are called for will be close to correct, will make it possible to write a grammar without unmanagable gaps in it.

Some people in A.I. have said that language generation is "easy". Basically I agree with them. I think that the structure of language is well enough understood that we should be able to have our programs speak in very fluent English without excessive research. As in many things, however, to make a system "easy" to work with seems to require first introducing a rather complicated structuring framework in order to separate out its component influences into managable chunks, and let the messy interfacing details work themselves out, away from our view.

Bibliography

<1> McDonald, D. (in press) The Design of a Program for Generating Natural Language, Masters thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, Mass.

<2> Goldstein I. et al. (in press) "Progress on the Personal Assistant Project", working paper MIT A.I. Lab.

<3> Rich C. and Shrobe H. (1974) "Understanding LISP Programs: Towards a Programming Apprentice", working paper 82 MIT A.I. lab.

<4> Genesereth M. (1975) "A MACSYMA Advisor", memo, Project MAC, MIT.

<5> Winograd T. (1972) Understanding Natural Language, in Cognitive Psychology, 3, 1, 1-191.

<6> Woods W. (1972) "The Lunar Sciences Natural Language System", BBN report 2378, Bolt Beranek and Newman, Cambridge, Mass.

<7> Goldman N. (1974) "Computer Generation of Natural Language from a Deep Conceptual Base", memo AIM-247, Stanford Artificial Intelligence Lab., Stanford, Calif.

<8> Slocum J. (1973) "Question Answering via Cannonical Verbs and Semantic Models: Generating English from the Model", technical report NL 13, Department of Computer Science, University of Texas, Austin, Texas