WORKING PAPER 73

ANOTHER APPROACH TO ENGLISH

by

MARTIN BROOKS

Massachusetts Institute of Technology
Artificial Intelligence Laboratory
June 1974

Abstract

A new approach to building descriptions of English is outlined and programs implementing the ideas for sentence-sized fragments are demonstrated.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advance Research Projects Agency of the Department of Defense and monitored by the office of Naval Research under Contract Number N00014-70-A-0362-0005.

Working Papers are informal papers intended for internal use.
This is a progress report on the natural language system I have been building during the last few months. I now have almost-debugged LISP programs that build descriptions of English inputs over a small vocabulary. All examples are taken from the programs' output.

The principal strategy behind my programs is that one should teach the computer about concepts in a frame-like manner and this knowledge should include information about communicating the concept in English. (I have only worked on understanding, not generation.) This is similar to recent vision theories where one teaches concepts including how they are to be parsed out of a visual scene. This approach indicates that grammar consists of analogies over the similarities of linguistic presentations of different concepts. As such, it must be built upon many specific instances and constructing some of these instances is my concern now.

I have written frame-program-definitions of ten concepts: three actions, four entities and three descriptive concepts.
For the most part they are limited to the world of text-editing. My image of the semantics of this world is that there are entities of two types: places (e.g. pages, lines, spaces) and text (e.g. paragraphs, diagrams, sentences, words, letters, punctuation). Each type admits relationships within, e.g. is-part-of and is-part*of, the transitive closure of is-part-of, and there are relationships between the classes, e.g. containment; paragraphs usually occupy a portion of a page, sentences are contained in several lines, letters are contained in spaces, etc. One can structure them via a geometry that is between one and two dimensional, not too complex but not trivial. There are actions which a person (another type of entity) may perform on and/or with these entities, e.g. inserting, deleting, moving, underlining, making-room-for, etc.

For the purposes of perceiving and communicating aspects of the actions and entities one builds descriptive systems such as size and the geometry already mentioned. One might also include a system for indicating identifications between concepts referred to in sentence or extended discourse.

Of the concepts mentioned I have programmed insert (F8-INSERT), delete (F1-DELETE), move (F5-MOVE), sentence (F3-SENTENCE paragraph (F4-PARAGRAPH), page (F6-PAGE), size (F7-SIZE), Martin (F2-MARTIN), and Barbara (F2*5-BARBARA), (two people
distinguishable only by name). I have also programmed two STATIVE-DESCRIPTORS, have (F9-HAVE) and be (F10-BE), devices for attaching properties and descriptions to actions and entities.

I will now discuss some aspects of my implementation of frame-like structures. A frame has two distinguished parts, a control section and a slot section. The control section is a program that receives messages and responds to them, perhaps by binding one of its slots to another structure. The slot section is the data structure with those slot bindings. When one talks about a specific frame-description of a concept one must talk about an instance of that frame description. In my scheme different instances of (what has been decided for now to be) the same concept share the control section but have different slot structures. This is done by making the control section be a program of two inputs: a message and an atom. The atom's property list has all the slot bindings for that instance of the frame-description. The pair of the control section and the slot section together make an instantiated frame, which is given a name.

My frames each have several parts, also implemented as frames. The basic structure is that all messages sent to an instantiated frame that come from outside the frame pass through a clearinghouse. Depending on the kind of message (to find out – ask it) it is relayed to one or another of the frame's
parts. There are three important kinds of messages:

INFO-MSG, MOD-MSG, RSVP-MSG.

An important part of a frame-system is a language internal to the system describing the frames themselves. For instance, F3-SENTENCE and F4-PARAGRAPH are ENTITIES and TEXT, F2-MARTIN is an ENTITY and a PERSON, F10-BE is a STATIVE-DESCRIPTOR, F8-INSERT is an ACTION. Often a frame will want to know whether another frame referred to in a message is a PLACE, or whatever. It does this by asking the frame itself whether it is a PLACE. It asks this with an INFO-MSG. Thus when a frame receives an INFO-MSG it sends it off to the part of itself that knows these sorts of things.

When a string of words is read it is converted to a list of lists of morphemically decomposed words each of which is then replaced with whatever the DICTIONARY says it should be replaced with. For prepositions and conjunctions (and eventually determiners) this is the token itself. For others, a newly instantiated frame for the concept the word stands for will replace the word. This newly instantiated frame will also contain information about the affixes or irregular presentation (e.g. is) of the concept's word. I call this list of instantiated frames and tokens a CONTEXT. Now that we have this CONTEXT the problem is to interconnect the frames with the relationships indicated (to a reader of English) by the string. The knowledge needed for determining these relationships is in the frames. The top
level interpreter in my system sets up "conversations" between adjacent frames to see if they should be interconnected. (This interconnection is entirely controlled by the frames, the possibility of the connection is controlled by the interpreter, it recognizes the frames to be adjacent, furthermore it does not set up these conversations in random order (see below)). When two adjacent frames do become interconnected the pair is removed from the CONTEXT (along with any prepositions, conjunctions, etc., that the frames want) and a new CONTEXT with the interconnected structure or whatever the frames designate, as the result of their conversation (see discussion of F10-BE and F9-HAVE below) replacing the pair is made (i.e. given a name). Further processing proceeds on this new CONTEXT. When there is only one structure in the context, processing is successfully completed.

The order in which the interpreter sets up the conversations between frames is determined so as to make a non-deterministic version of the control structure Vaughan Pratt espouses in CGOL (Reference: Vaughan Pratt, [1973]) with operator precedence replaced by the conversation. This guarantees (assuming certain restrictions on the frames' behavior) that all modifications among frames are nested and that no frame gets taken as a modifier to another until it has received all of its modifiers.
Even though all relationships between frames are done via two way links (i.e. if an instance of F3-SENTENCE knows it is BEFORE an instance of F4-PARAGRAPH then the F4-PARAGRAPH knows it is AFTER the F3-SENTENCE) the acquisition of this relationship is asymmetric. When a conversation between FRAME 1 and FRAME 2 is set up (either by the interpreter or an instance of F10-BE or F9-HAVE, see below) it may typically proceed as follows:

A message is sent to FRAME 1 asking it if it is interested in taking FRAME 2 as a modifier. This is a MOD-MSG, and being a MOD-MSG tells FRAME 1 what the situation is. If FRAME 1 does not want FRAME 2 then the conversation is restarted by asking FRAME 2 if it would like FRAME 1 as a modifier. If FRAME 1 does want FRAME 2 (perhaps along with some tokens in the CONTEXT (prepositions, conjunctions, etc.) it sends a message to FRAME 2 saying that it wants it and in what capacity it will be used. This is done with an RSVP-MSG and being an RSVP-MSG is what clues FRAME 2 in to what is happening. FRAME 2 may now reject the whole idea, in which case FRAME 1 is not allowed to take FRAME 2, or if FRAME 2 approves the suggestion, it will record the relationship in its slot structure and send a message of approval back to FRAME 1 which will then record the relationship in its slot structure
and decide what structure should replace the two in the CONTEXT. Very often this is simply itself, although in the cases of subordinate clauses and most usages of F10-BE and F9-HAVE this will not be the case.

I should mention that the interpreter is a small recursive algorithm (less than one page of LISP). All the work is done by the frames since they have the knowledge to do it. This also results in there being no concept of sentence.

What kinds of knowledge do frames use in responding to a MOD-MSG? A MOD-MSG has two slots, called BODY and LEX. BODY contains the frame which is the proposed modifier. LEX says where the proposed modifier is coming from. This might be LEFT, RIGHT, F10-BE, or F9-HAVE. LEFT and RIGHT indicate that the proposed modifier was lying in the CONTEXT (immediately) to the frame's left or right. The frames F10-BE and F9-HAVE each take two frames as slot-fillers, upon receiving the second one it sends a MOD-MSG to one asking if it would like the other as a possible modifier. In this case LEX of this MOD-MSG would be F10-BE or F9-HAVE. (F9-HAVE and F10-BE perform some other functions, discussed below).

Where the proposed modifier comes from (i.e. what LEX is) is very important. Other important things are the way the
proposed modifier responds to certain questions about itself and what prepositions and conjunctions (i.e. tokens, in general) appear in various places in the CONTEXT.

In Example 1, F3-SENTENCE uses the fact that LEX = RIGHT, that the proposed modifier (F4-PARAGRAPH) is TEXT and something it could be PART* of (transitive closure of PART) (F3-SENTENCE knew that itself), and that there was a CONTAINMENT-PREP (i.e. one of \{ in, on, of \}) in between them (see Example 1) (I have not done determiners yet, read them in as you please).

These structures are printed so that the name (the second name of a frame is the name of the atom with the slot-structure in its property list) appears above a vertical list of its bound slots. (There are no slots bound by default for these purposes). The contents of the bound slot is printed recursively after the colon following the slot's name. To avoid problems with circular pointers any structure to be printed which is already being printed has its name printed only. The slot called MOD-ORDER is a list of the frame's modifiers in the order they were received (most recent on the top), including the slot the modifier fills and where it comes from. (LEFT, RIGHT, F10-BE, F9-HAVE as before, or RSVP if it was taken as a result of an RSVP-MSG). This is very useful for understanding a frame's history which is in turn useful for debugging and parsing CONTEXTS with conjunctions.
Example 1:

(FR#1156 IS-PART* OF F4-PARAGRAPH FR#1153)

HAS-AS-PART* : F3-SENTENCE FR#1156

MOD-ORDER : F3-SENTENCE FR#1156

HAS-AS-PART* RSVP

MOD-ORDER : F4-PARAGRAPH FR#1153

IS-PART* OF RIGHT
As well as initiating relationships between the two frames filling their slots F10-BE and F9-HAVE determine the time relationships between these two frames, and in case one is an action, inform it of its VOICE and ASPECT. ASPECT (either COMPLETED or ONGOING) is not important to the action at this point, however VOICE determines which of the two forms of presentation (PASSIVE and ACTIVE) it should use as a first assumption about its presentation in the CONTEXT. This decision relates to what modifiers it expects to receive from where.

When one makes a description of something, it is a description of the something a certain time. Furthermore, a description of X at time T1 may include that at time T2 X will have a certain property. I have observed that English transmits descriptions of this form via the instances of F10-BE and/or F9-HAVE between an entity (or in some cases this will be an action) and an action the entity is involved in, as follows (this is not all these "auxiliary verbs" determine): they determine the relationship (either BEFORE, AT, or AFTER) between TIME-OF-UTTERANCE and the time of the description of the entity and the relationship between the time of the description of the entity and the time of the description of the action. For example, compare the TIME slots in F2-MARTIN and F8-INSERT in examples 2, 3 and 4. There are nine such pairs of relationships and it is easy to figure out how to
Example 2:

(MARTIN WILL HAVE INSERTED)

FB-INSERT FR#4816
TIME : BEFORE
WHO
VOICE : ACTIVE
SUFFIX : ED
ASPECT : COMPLETE
WHO : F2-MARTIN FR#4817

TIME : AFTER
TIME-OF-UTTERENCE
ACTIONS-INVOLVED-IN : F8-INSERT FR#4816

MOD-ORDER : F8-INSERT FR#4816
ACTIONS-INVOLVED-IN
RSVP

MOD-ORDER : F2-MARTIN FR#4817
WHO
LEFT

Example 3:

(MARTIN HAS INSERTED)

FB-INSERT FR#5066
TIME : BEFORE
WHO
VOICE : ACTIVE
SUFFIX : ED
ASPECT : COMPLETE
WHO : F2-MARTIN FR#5067

TIME : AFTER
TIME-OF-UTTERENCE
ACTIONS-INVOLVED-IN : F8-INSERT FR#5066

MOD-ORDER : F8-INSERT FR#5066
ACTIONS-INVOLVED-IN
RSVP

MOD-ORDER : F2-MARTIN FR#5067
WHO
LEFT

Example 4:

(MARTIN HAD INSERTED)

FB-INSERT FR#5281
TIME : BEFORE
WHO
VOICE : ACTIVE
SUFFIX : ED
ASPECT : COMPLETE
WHO : F2-MARTIN FR#5282

TIME : AFTER
TIME-OF-UTTERERENCE
ACTIONS-INVOLVED-IN : F8-INSERT FR#5281

MOD-ORDER : F8-INSERT FR#5281
ACTIONS-INVOLVED-IN
RSVP

MOD-ORDER : F2-MARTIN FR#5282
WHO
LEFT
say each in almost any voice or aspect
in English, using the "auxiliaries" have, be and going.

This theory can be extended to include the appropriate
filler for the TIME slot (i.e. time of description) for every
frame in the CONTEXT. It has interesting aspects when considering
subordinate clauses, however, I have not yet implemented this
extension.

The actions in Examples 5 and 6 use knowledge about the
VOICE of their presentation (in 5 deduced by default, in 6
informed by F10-BE), where they receive the proposed modifier
from (RIGHT, LEFT or F10-BE), the way the proposed modifier
answer their questions and the prepositions occurring in order
to fill in the appropriate slots.

To be precise F10-BE and F9-HAVE expect one modifier from
the LEFT and another either also from the LEFT or from the RIGHT.
In the latter case it sends a MOD-MSG to the one from the LEFT
asking if it would like the one from the RIGHT as a modifier.
If the one from the RIGHT is an action and the one from the
LEFT an entity then the entity will know (since the message it
receives in from F10-BE or F9-HAVE) that it should send a
MOD-MSG to the action proposing itself as a possible modifier.
If the one from the RIGHT is not an action then the one from the
Example 5:

```plaintext
Example 5:

F5-MOVE FR#5722
WHAT : F4-PARAGRAPH FR#5706
  ACTIONS-INVOLVED-IN : F5-MOVE FR#5722
  SIZE : F7-SIZE FR#5707
    PARTICULAR-VALUE : LONG
    WHAT : F4-PARAGRAPH FR#5706
    MOD-ORDER : F4-PARAGRAPH FR#5706
    WHAT
    RSVP
    MOD-ORDER : F5-MOVE FR#5722
    ACTIONS-INVOLVED-IN
    RSVP
    F7-SIZE FR#5707
    SIZE
    LEFT

TIME : AT
WHO
VOICE : ACTIVE
SUFFIX : ED
ASPECT : COMPLETED
WHO : F2-MARTIN FR#5735
  TIME : BEFORE
  TIME-OF-UTTERENCE
  ACTIONS-INVOLVED-IN : F5-MOVE FR#5722
  MOD-ORDER : F5-MOVE FR#5722
  ACTIONS-INVOLVED-IN
  RSVP

TO-WHERE : F6-PAGE FR#5723
  ACTIONS-INVOLVED-IN : F5-MOVE FR#5722
  MOD-ORDER : F5-MOVE FR#5722
  ACTIONS-INVOLVED-IN
  RSVP

MOD-ORDER : F4-PARAGRAPH FR#5706
WHAT
RIGHT
F6-PAGE FR#5723
TO-WHERE
LEFT
F2-MARTIN FR#5735
WHO
LEFT
```
Example 6:

**WHO**: F2-MARTIN FR#688  
**ACTIONS-INVOLVED-IN**: F8-INSERT FR#691  
**MOD-ORDER**: F8-INSERT FR#691  
**TIME**: AT WHAT  
**ASPECT**: COMPLETE  
**SUFFIX**: ED  
**VOICE**: PASSIVE  
**WHAT**: F3-SENTENCE FR#719  
**TIME**: BEFORE TIME-OF-UTTERANCE  
**ACTIONS-INVOLVED-IN**: F8-INSERT FR#691  
**MOD-ORDER**: F8-INSERT FR#691  
**WHERE**: IN F4-PARAGRAPH FR#692  
**CONTAINED-IN**: F6-PAGE FR#696  
**CONTAINS**: F4-PARAGRAPH FR#692  
**MOD-ORDER**: F4-PARAGRAPH FR#692  
**CONTAINS**: RSVP  
**ACTIONS-INVOLVED-IN**: F8-INSERT FR#691  
**MOD-ORDER**: F8-INSERT FR#691  
**ACTIONS-INVOLVED-IN**: RSVP  
**F6-PAGE FR#696**: CONTAINED-IN RIGHT  
**MOD-ORDER**: F2-MARTIN FR#688  
**WHO**: RIGHT  
**IN F4-PARAGRAPH FR#692**  
**WHERE**: RIGHT  
**F3-SENTENCE FR#719**  
**WHAT**: LEFT
left will consider it as a possible modifier, (see Examples 7, 8, 9 and 10. One might also wish to communicate the relationship in 10 but with more emphasis on F4-PARAGRAPH (perhaps for the purpose of further modification). This is done as in Example 11. In this example F10-BE receives both of its modifiers on the LEFT, which is what distinguishes it from Example 10.

When it is presented in the ACTIVE voice F8-INSERT "expects" to fill its WHAT slot with some TEXT from its RIGHT as in Example 12. However, if it receives some TEXT from the LEFT it will use it as WHAT but also take it as an indication that it should not return itself as a replacement for the pair in the CONTEXT, but rather it should return the TEXT, as in Example 13. This is the general idea behind how I do subordinate clauses, having "expectations" that are broken. (See Examples 14, 15, and 16).

More complex constructions are easily recognized by looking for a preposition in the right place. Compare Examples 17 and 18. Example 17 involves a change, in F8-INSERT, F4-PARAGRAPH is taken as WHAT until F3-SENTENCE is considered as a possible modifier. Then, since the preposition comes after the occurrence of F3-SENTENCE, a switch is made, IN F4-PARAGRAPH becomes WHERE and F3-SENTENCE becomes WHAT.
Example 7:

Example 8:

Example 9:
Example 10:

Example 11:
Example 12:

(BARBARA HAS BEEN INSERTING SENTENCE)
FR#684
WHAT : F3-SENTENCE FR#684
  ACTIONS-INVOLVED-IN : F8-INSERT FR#684

MOD-ORDER : F8-INSERT FR#684
  ACTIONS-INVOLVED-IN : RSVP

WHO : F2o5-BARBARA FR#686
  ACTIONS-INVOLVED-IN : F8-INSERT FR#684

TIME : AT
  TIME-OF-UTTERENCE

ASPECT : ONGOING
  SUFFIX : ING
  VOICE : ACTIVE
  TIME : BEFORE

WHO
MOD-ORDER : F3-SENTENCE FR#684
  WHAT
  RIGHT
  F2o5-BARBARA FR#686
  WHO
  LEFT

Example 13:

(FR#684)
FR#1189
ACTS-INVOLVED-IN : F8-INSERT FR#1189

SUBORDINATE : WHAT
  TIME : BEFORE
  WHO
  VOICE : ACTIVE
  SUFFIX : ING
  ASPECT : ONGOING

WHO : F2o5-BARBARA FR#1121
  TIME : AT
  TIME-OF-UTTERENCE

ACTS-INVOLVED-IN : F8-INSERT FR#1189

MOD-ORDER : F3-SENTENCE FR#1189
  WHAT
  LEFT
  F2o5-BARBARA FR#1121
  WHO
  LEFT

MOD-ORDER : F8-INSERT FR#1189
  ACTIONS-INVOLVED-IN : RSVP
Example 14:

```
(MARTIN DELETED LONG SENTENCE FROM PARAGRAPH BARBARA INSERTED)

F1-DELETE FR#1284
FROM-WHERE : F4-PARAGRAPH FR#1233
ACTIONS-INVOLVED-IN : F1-DELETE FR#1264
F8-INSERT FR#1234
WHAT : F4-PARAGRAPH FR#1233
WHO : F2-MARTIN FR#1298
ACTIONS-INVOLVED-IN : F8-INSERT FR#1234
TIME : BEFORE
TIME-OF-UTTERENCE
MOD-ORDER : F8-INSERT FR#1234
ACTIONS-INVOLVED-IN
RSVP

ASPECT : COMPLETED
SUFFIX : ED
VOICE : ACTIVE
TIME : AT
WHO

SUBORDINATE : WHAT
MOD-ORDER : F4-PARAGRAPH FR#1233
WHAT
LEFT
F2-MARTIN FR#1298
WHO
LEFT

MOD-ORDER : F1-DELETE FR#1284
ACTIONS-INVOLVED-IN
RSVP
F8-INSERT FR#1234
ACTIONS-INVOLVED-IN
RSVP

TIME : AT
WHO
VOICE : ACTIVE
SUFFIX : ED
ASPECT : COMPLETED
WHO : F2-MARTIN FR#1298
TIME : BEFORE
TIME-OF-UTTERENCE
ACTIONS-INVOLVED-IN : F1-DELETE FR#1284
MOD-ORDER : F1-DELETE FR#1264
ACTIONS-INVOLVED-IN
RSVP
WHAT : F3-SENTENCE FR#1265
SIZE : F7-SIZE FR#1269
WHAT : F3-SENTENCE FR#1265
PARTICULAR-VALUE : LONG
MOD-ORDER : F3-SENTENCE FR#1265
WHAT
RSVP
ACTIONS-INVOLVED-IN : F1-DELETE FR#1264
MOD-ORDER : F1-DELETE FR#1264
ACTIONS-INVOLVED-IN
RSVP
F7-SIZE FR#1269
SIZE
LEFT
MOD-ORDER : F4-PARAGRAPH FR#1233
FROM-WHERE
RIGHT
F3-SENTENCE FR#1265
WHAT
RIGHT
F2-MARTIN FR#1298
WHO
LEFT
```
Example 15:

(Sentence was inserted on page paragraph was moved to)

F8-INSERT FR#2459
WHERE : ON F6-PAGE FR#2427

F5-MOVE FR#2428
TO-WHERE : F6-PAGE FR#2427
WHAT : F4-PARAGRAPH FR#2430

F8-INSERT FR#2459
WHERE : ON F6-PAGE FR#2427

F5-MOVE FR#2428
TO-WHERE : F6-PAGE FR#2427
WHAT : F4-PARAGRAPH FR#2430

F5-MOVE FR#2428
TIME : BEFORE
TIME-OF-UTTERENCE
MOD-ORDER : F5-MOVE FR#2428

RSVP

VOICE : PASSIVE
SUFFIX : ED
ASPECT : COMPLETE
TIME : AT

WHAT
SUBORDINATE : TO-WHERE
MOD-ORDER : F6-PAGE FR#2427
TO-WHERE
LEFT
F4-PARAGRAPH FR#2430
WHAT
LEFT

MOD-ORDER : F8-INSERT FR#2459

F5-MOVE FR#2428
ACtIONS-INVOLVED-IN
RSVP

F8-INSERT FR#2459
WHAT : F3-SENTENCE FR#2461

ACtIONS-INVOLVED-IN
RSVP

TIME : BEFORE
TIME-OF-UTTERENCE
MOD-ORDER : F8-INSERT FR#2459

RSVP

VOICE : PASSIVE
SUFFIX : ED
ASPECT : COMPLETE
TIME : AT

WHAT
MOD-ORDER : ON F6-PAGE FR#2427
WHERE
RIGHT
F3-SENTENCE FR#2461
WHAT
LEFT
Example 16:

*****************
******
(PARAGRAPH BEING DELETED HAS SHORT SENTENCE)
*****************

F4-PARAGRAPH FR#4334
HAS-AS-PART OF: F3-SENTENCE FR#4318
IS-PART OF: F4-PARAGRAPH FR#4334

SIZE: F7-SIZE FR#4319
PARTICULAR-VALUE: SHORT
WHAT: F3-SENTENCE FR#4318

MOD-ORDER: F3-SENTENCE FR#4318
WHAT: RSVP

MOD-ORDER: F4-PARAGRAPH FR#4334
IS-PART OF: RSVP
F7-SIZE FR#4319
SIZE: LEFT

ACTIONS-INVOLVED-IN: F1-DELETE FR#4336
WHAT: F4-PARAGRAPH FR#4334
TIME: AT
WHAT: RSVP
VOICE: PASSIVE
SUFFIX: ED
ASPECT: ONGOING
SUBORDINATE: WHAT
MOD-ORDER: F4-PARAGRAPH FR#4334
WHAT: LEFT

TIME: AT
TIME-OF-UTTERENCE
MOD-ORDER: F3-SENTENCE FR#4318
HAS-AS-PART OF:
F9-HAVE
F1-DELETE FR#4336
ACTIONS-INVOLVED-IN
RSVP
Example 17:

*Example Text*

- ACTIONS_INVOLVED_IN: F8-INSERT FR#296
  - WHERE: IN F4-PARAGRAPH FR#328
  - WHO: F2=5-BARBARA FR#307
    - ACTIONS_INVOLVED_IN: F8-INSERT FR#296
      - TIME: BEFORE TIME_OF_UTTERENCE
        - MOD_ORDER: F8-INSERT FR#296
          - ACTIONS_INVOLVED_IN: RSVP
            - ASPECT: COMPLETED
              - SUFFIX: ED
              - VOICE: ACTIVE
              - TIME: AT
                - WHO
                  - MOD_ORDER: F3-SENTENCE FR#297
                    - ACTIONS_INVOLVED_IN: F8-INSERT FR#296
                      - MOD_ORDER: F3-SENTENCE FR#297
                        - WHAT
                          - RIGHT
                          - IN F4-PARAGRAPH FR#320
                          - CHANGED_TO
                            - WHERE
                            - F4-PARAGRAPH FR#320
                            - WHAT
                              - LEFT
                              - F2=5-BARBARA FR#307
                              - WHO
                              - LEFT
Example 18. Notice the ambiguity. The paragraph could be the one that Martin inserted which is on the page, or it could be the one Martin inserted on the page.
When filling certain of its slots, a frame will make sure that the proposed modifier has certain properties as I have been describing and then, before binding it to the appropriate slot, will look in that slot and make sure that it is empty. If not it does two things. It looks for a conjunction immediately before or after (depending on whether the proposed modifier comes from the RIGHT or the LEFT) the proposed modifier in the CONTEXT and it looks at its MOD-ORDER slot to see if the most recent modification was the modifier already in the slot in question. If both conditions hold, the conjunction is removed from the CONTEXT and a new structure which indicates the conjunction is put in the slot. (See Example 19). Simple conditions like these can also be used to parse more complex conjunctions involving deletions, etc., however, I have not implemented this.

The only serious problem I have encountered in my approach is that my programs run terribly slowly. Their time basically increases exponentially with the number of frames in the original CONTEXT. For a medium length sentence, e.g. Example 6, the running-time is about thirty seconds, not counting time spent garbage collecting, with most of my programs compiled. This is a result of some inefficient programming on my part and the non-determinism of the control structure. If other procedures intervened at the branch points within the non-deterministic
Example 19:

***************
***************
(MARTIN OR BARBARA MOVED PARAGRAPH)
***************
FS-MOVE FR#3267
WHAT : F4-PARAGRAPH FR#3264
ACTIONS-INVOLVED-IN : FS-MOVE FR#3267
MOD-ORDER : FS-MOVE FR#3267
ACTIONS-INVOLVED-IN
RSVP

WHO : OR
F2-MARTIN FR#3278
ACTIONS-INVOLVED-IN : FS-MOVE FR#3267
MOD-ORDER : FS-MOVE FR#3267
ACTIONS-INVOLVED-IN
RSVP
F2aS-BARBARA FR#3279
ACTIONS-INVOLVED-IN : FS-MOVE FR#3267
TIME : BEFORE
TIME-OF-UTTERENCE
MOD-ORDER : FS-MOVE FR#3267
ACTIONS-INVOLVED-IN
RSVP

ASPECT : COMPLETED
SUFFIX : ED
VOICE : ACTIVE
TIME : AT
WHO
WHO
MOD-ORDER : F4-PARAGRAPH FR#3264
WHAT
RIGHT
OR
F2-MARTIN FR#3278
F2aS-BARBARA FR#3279
WHO
LEFT
structure and made decisions based on expectations, preferences, and perhaps a global concept of grammar I suspect the slowness would be relieved.

The first logical continuation at this point is a larger vocabulary of concepts, some expansion of linguistic presentations of concepts including more complicated conjoined structures, negation, etc. Introduction of means for specifying identifications between concepts, i.e. determiners, numbers, and ordinals, and implementation of my extended time-reference theory would also be easy; all these things should be straightforward constructions within the structure I have already built.

A more important continuations is to build a mechanism for integrating new information into a body of information already received in discourse. Aside from deductions, etc., that are specific to the reader's purposes, a most important function of such a mechanism is to make identifications between actions or entities in the information being integrated and those already processed. Such a mechanism will also be very useful for determining pronoun references, opaque references, and the like. Beyond that, generation of English is a good goal.
BIBLIOGRAPHY
