SPEECH UNDERSTANDING SYSTEMS

William A. Woods, et al

Bolt Beranek and Newman, Incorporated

Prepared for:

Office of Naval Research
Advanced Research Projects Agency

August 1975

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BBN Report No. 3115	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SPEECH UNDERSTANDING SYSTEMS Quarterly Technical Progress Report No. 3 1 May 1975 to 1 August 1975		5. TYPE OF REPORT & PERIOD COVERED Quarterly Tech. Prog. Rep. 1 May 1975 to 1 Aug. 1975 6. PERFORMING ORG. REPORT NUMBER BBN Report No. 3115
7. AUTHOR(s) William A. Woods, Richard M. Schwartz, Craig C. Cook, John W. Klovstad, Lyn A. Bates, Bonnie Nash-Webber, Bertram C. Bruce, John I. Makhoul		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0533
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, MA 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 5D30
11. CONTROLLING OFFICE NAME AND ADDRESS ONR Department of the Navy Arlington, VA 22217		12. REPORT DATE August 1975 13. NUMBER OF PAGES 50
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Acoustic-phonetics, acoustics, acoustic segmentation, augmented transition network, constituent boundaries, data base, dip detector, formant tracking, formant smoothing, fundamental frequency contours, parsing, partial matches phonological rules, property checking, pragmatics, prosodics, retrieval, semantic networks, semantics, speech recognition, speech understanding,

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report covers research and development work done from 1 May 1975 to 1 August 1975 under the Speech Understanding Systems Contract No. N00014-75-C-0533. Areas included in this work are acoustic-phonetics, lexical retrieval, lexical verification, and natural language syntax, semantics, prosodics, and pragmatics. The report consists of two parts -- a brief survey of progress containing a few paragraphs describing the major progress in the individual components of the project, and a Technical Notes

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. Key Words - cont'd.

speaker normalization, syntax analysis, synthesis, synthesis-by-rule, vocal tract length.

---

20. Abstract - cont'd.

section containing detailed specifications of experiments performed, programs implemented, design studies, and, where appropriate, supporting data and appendices. This third QTPR contains such technical notes on handling of time expressions, procedural semantics for the travel budget management system, and control primitives for speech understanding strategies.

BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

BBN Report No. 3115
A.I. Report No. 34

SPEECH UNDERSTANDING SYSTEMS

Quarterly Technical Progress Report No. 3
1 May 1975 to 1 August 1975

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2904

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-75-C-0533.

SPEECH UNDERSTANDING SYSTEMS

Quarterly Technical Progress Report No. 3
1 May 1975 to 1 August 1975

ARPA Order No. 2904

Program Code No. 5D30

Name of Contractor:
Bolt Beranek and Newman Inc.

Effective Date of Contract:
30 October 1974

Contract Expiration Date:
29 October 1975

Amount of Contract: $1,041,261

Contract No. N00014-75-C-0533

Principal Investigator:
William A. Woods
(617) 491-1850 x361

Scientific Officer:
T. H. Lautenschlager

Title:
SPEECH UNDERSTANDING SYSTEMS

QTPR Editor:
Bonnie Nash-Webber
(617) 491-1850 x227

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2904

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-75-C-0533.

Table of Contents

I. PROGRESS OVERVIEWS

A. Acoustic Analysis

One of the main problems in the accurate estimation of formants and signal energy is the variability in the pitch of an individual speaker as well as its variability across speakers. The autocorrelation method of linear prediction, which we have been using so far, has the disadvantage that it is sensitive to wide variations in pitch, due to the interaction between the analysis window and the pitch period. The covariance method does not use a window and hence does not exhibit the same degree of sensitivity to pitch variations. However, it has the disadvantage that the stability of the computed model is not assured. We are now working on a class of methods (due primarily to Itakura and Burg) which do not require windowing and yet do preserve stability. We hope to settle on one method which will prove optimal for speech analysis.

B. Acoustic-Phonetic Segmentation and Labeling

This quarter we extended the first-pass segmentation process described in the last quarterly progress report [Woods et al., 1975b] to the point where it produces segment lattices which are suitable for input to the word matcher.

In this process, the APR component starts by applying dip detection routines to three different energy parameters to produce three sets of boundaries of different types. Dips in the parameter LEZ (smoothed low-frequency energy from 120-440 Hz) indicate likely obstruents or obstruent sequences. Dips in MEPZ (smoothed mid-frequency energy from the preemphasized signal between 640-2800 Hz) which occur within sonorant sequences indicate nasals, back semivowels [W,L], flaps or intervocalic obstruents (e.g., [HH,V,DH,D]). Dips in HEPZ (smoothed high-frequency energy from the preemphasized signal between 3400-5000 Hz) that occur within sonorant sequences indicate [R] or flaps and sometimes nasals, [W] or intervocalic obstruents. Dips in HEPZ within obstruent sequences indicate silences or weak fricatives.

Merging these results yields a lattice of regions of nine different types, each corresponding to one or more phonemes. The remainder of the program consists of rules that are region- and context-specific. One typical rule looks at regions that were classified as intervocalic sonorants or glides, and by looking for rapid changes in the formants (mainly F1) determines whether or not it is a nasal. Another rule looks at an obstruent or silence region followed by a short frication region, and decides whether the frication is the aspiration from an unvoiced plosive or represents a strident fricative. Within vowel regions, the three formants are each described in terms of a series of

canonical shapes. Based on these representations, formant extrema or plateaus are identified as possible vowel targets. Vowel identity is determined using the values of the three formants as normalized by the average fundamental frequency for the utterance [Schwartz, 1971] along with durational constraints. Those rules that are optional add branches to the lattice, while others make a narrower specification of the labels on existing segments.

In addition to the vowels, the program currently recognizes individual unvoiced plosives and fricatives. It also uses formant transitions to classify voiced plosives and nasals in a rough way. Prevocalic [W,R,L,Y] are detected and identified from formant transitions. Unreleased plosive-plosive pairs are detected based on the duration of silences. In all, the program uses 60 different symbols to label the segments of the lattice.

Using the Acoustic-Phonetic Experiment Facility [Woods et al., 1975a, pp. 20-32; Schwartz, 1975], we can compare the labels in the hand-labeled files (correct) with those chosen by the program, in order to create a confusion matrix which is used by the word matcher in scoring possible words. The performance of the small set of algorithms is very encouraging, producing lattices with small branching ratios and relatively few errors. More work will be needed to improve the specificity and accuracy of the segment labels.

We will also be spending more effort on devising algorithms to correctly segment sonorant sequences.

C. Lexical Retrieval Component

In the past quarter, work was done on four areas of lexical retrieval relating to:

1) Generation of appropriate input for the tree. compiler.
2) Modification of the tree compiler.
3) Extension of the Matcher's capabilities.
4) Use of APR statistics for segment lattice generation.

On the first point, the tree compiler requires as input a BCPL-readable version of the expanded dictionary (see Section I.D.) in order to build an appropriate tree structure for the Matcher. To this end, LISP programs were written to produce a BCPL-readable text file with the appropriate information.

Secondly, we extended the BCPL program which reads this text file and creates the appropriate tree structure to recognize and encode certain inflectional endings, to associate a probability with every pronunciation, and to construct two separate tree structures, one for scanning left-to-right, the other right-to-left.

Thirdly, we extended the Matcher to use the appropriate tree structure for a given scan. As a result, phonological context can now be taken into account when scanning to the right of, left of, or between groups of specified word matches. Furthermore the set of words sought can be specified by explicate enumeration, membership in some designated class, by phonetic length, or by any combination of these three. As a further extension to the Matcher, a special control language was designed for efficient interfork communication with the LISP world. The language has been implemented as a BCPL program which reads LISP generated commands and creates LISP readable output.

Finally, programs have been written to collect statistics as well as to pad, adjust, and normalize them in creating a log confusion matrix. This matrix is then used with the present APR output to create segment lattices with probability vectors as segment descriptors. Any improvements resulting from the modification and extension of the APR component can now be quickly realized and evaluated.

D. Dictionary Expansion

During the past quarter, the Bobrow-Fraser rule compiler was extended and the set of phonological rules used

for dictionary expansion refined to a point of relative stability. Changes to the rule compiler included the incorporation of likelihood numbers with optional rules indicating the relative goodness of applying the rule versus not applying it, and similar numbers associated with the alternative base form pronunciations of words in the dictionary. These numbers are multiplied together as the words are expanded so that each expanded form carries with it a likelihood number which is the product of the likelihood associated with its base form, the likelihoods of application of all the optional rules applied to it, and the likelihoods of not applying of all optional rules which matched but were not applied to it.

An additional extension to the rule expansion facility now allows one to associate a predicate with a rule, making the applicability of the rule conditional on arbitrary features of the word to which it is being applied. (For example, features such as syntactic part of speech, length of the word, and geographical or foreign origin of a word could be used to determine the applicability of a rule, thus permitting the inclusion of rules that apply only to special classes of words, such as short function words, words of Latin origin, etc.)

The current set of rules that is being used to expand the dictionary consists of 48 rules that cover regular inflections, vowel reduction, consonant syllabification, palatalization, stop insertion and deletion, as well as dental flapping.

E. Verification

During the past quarter, we have devoted considerable effort to debugging and improving our synthesis-by-rule program. This allows us to synthesize in near real-time a parametric representation of any word, given its phonetic transcription. The addition of a sophisticated phonological component to the program has greatly improved the quality of the synthesis output, by allowing us to take into account phonological effects across word boundaries, altering the parameterization according to the context in which the hypothesized word may occur. In addition, it negates the need to store separately a parameterization for each entry in the lexicon.

Time normalization is done using a dynamic programming algorithm based on a method first developed by Itakura [1975]. The technique involves a non-linear time warping based on the registration of the error metric, in this case, the ratio of the linear prediction residuals. We

have modified his method to allow limited misalignment in time between the hypothesized word parameterization and that portion of an unknown utterance onto which we wish to match it.

In actually computing the 'distance' between a hypothesized word and a portion of the unknown utterance, we sum the prediction residuals between corresponding segments of the two, the correspondence having already been determined by the time normalization technique. Comparing the linear prediction residuals is a method of spectral matching, specifically the spectra of the all-pole models.

At this time, we have integrated the synthesis-by-rule program with the time normalization algorithm and the parametric word matcher into a single component which runs interactively from a console. Given a phonetic spelling of a hypothesized word together with available context, the component synthesizes the parameterization of the word for that context and matches it onto the specific region of the parameterization of the unknown utterance specified by the user. The verification component returns a score, normalized with the duration of the hypothesized word, indicating the likelihood of that word occurring at the given position in the utterance. The interactive implementation operates presently in 3-4 times real time.

F. Syntax

This quarter saw the completion of a major report on the syntactic component of the BBN speech understanding system, which is to be printed in early fall.

With respect to the grammar, we began to design a sub-grammar for adverbial time modifiers (see Section II.A.). An additional change to the grammar was the addition of an expanded proper noun network to parse people's names (first or first and last) and place names (e.g., "Pittsburgh, Pennsylvania," "Paris, France," etc.). With respect to the parser, two flags were added so that a human simulator (or control program) has available options to force the following of all parse paths instead of just the current best ones, and to cause proposals to be made for all monitored syntactic classes rather than those classes with a small number of members.

G. Semantics

In the past quarter, we have brought up a new version of the semantic network package (SEMNET) which supports the maintaining of a semantic network on an external disk file instead of in-core. This has several advantages for us, including lower in-core storage requirements, increased filing speed, and better control over multiple-user access.

The primary impetus for this new implementation was the growing size of the SPEECHLIS semantic network, which is being used to store all our semantic, pragmatic and data base information about the travel budget management domain. At this writing, the number of nodes in the common network has reached 875, with 1484 two-way links and 2379 one-way links. This is a large drain on storage, and we believe that not all these nodes will ever be accessed in any one session by any one knowledge source. The cumbersomeness of merging networks (cf., MERGESEMNET in QTPR 2), after several users' simultaneous changes have resulted in several slightly different versions of the network, was another reason for desiring a new implementation, one which would make impossible simultaneous changes to the network.

In the new implementation, only a few things about the network are initially loaded into core:

1. The set of terms and each one's SREF property whose value is the semantic network node to which the term refers.

2. The semantic network array, containing not the set of links and properties associated with each node, but rather pointers into a separate file (the "guts" file) in which this information is stored.

3. The set of global variables used in network manipulation; e.g., a pointer to the free list (FREELIST), a pointer to the highest network array cell thus far used (NITEMS), the name of the guts file, the size of the network array, etc. When a node is accessed, its corresponding array cell is checked. If it still contains a pair of file pointers, the link information about the node is read in from the guts file before processing continues as usual. If the network is edited and refiled, information about nodes

that have not been loaded in-core is copied quickly and directly from the previous guts file.

One has the option of opening the guts file either "thawed" or protected. If one chooses the latter, one can prevent simultaneous changes to the network by more than one user. If one chooses the former, however, one can thereby allow run-time access to the net by several different processes.

H. User and Discourse Model

An augmented transition network (ATN) grammar is being used to represent some of the common modes of interaction found in travel budget management dialogues. A modified ATN parser has been written that steps through this grammar on the basis of the input sentence structure and the then-current state of the data base. At any given state the parser can predict the most likely next intent and hence such things as the mood and import of the next utterance.

We are also exploring the use of a more flexible discourse model that may partially or wholly supplant the current ATN model. This latter model is based on the notion of a set of pending "demands" and "counter-demands." (A sketch of this model is presented in Section II.B.3.)

I. The Travel Budget Manager's Assistant

The current task domain of the BBN speech project is that of assisting a travel budget manager. It is meant to help the manager keep a record of trips taken or proposed and to produce summary information such as the total money allocated. It is a simplified example of many other resource management problems of essentially the same type and is an initial step toward an intelligent manager's assistant. The data base management facilities of the system are accessed through a formal command language (see II.B.), into which spoken requests will be translated. The command language operates on a set of data base structures representing such things as trips, contracts, budgets, conferences, dates, fares, and lengths of time. The structures for times are discussed in II.A.

In the past quarter, we have developed a preliminary set of programs to allow the travel budget manager's assistant to respond to the manager in an English-like language. For example, it may describe a trip as:

> John Makhoul is going to Pittsburgh from Monday, the 30th of June to Wednesday, the 2nd of July, 1975.

We have also been running simulations to develop and circumscribe the travel budget manager's assistant. In the simulations, one person, sitting at terminal A, plays the

role of the travel budget manager, typing in sentences as if he were talking to a complete travel budget manager's assistant. Another person, at terminal B, intercepts his sentences and translates them into the formal command language mentioned above and passes the translations to the retrieval component for execution. These simulations also provide a source for dialog protocols and new words that should be included in the lexicon.

J. Control

During the past quarter, we continued our work in developing specific control strategies that would take suitable advantage of the different capabilities of the individual high level components. For example, our increased confidence in the results of lexical retrieval, brought about by great improvements in the match component, is leading to strategies which rely more on that component in assembling and evaluating theories. We are working in a mode of incremental simulation in order to recognize and develop these strategies, and to ease the task, a set of "primitive" control operations have been isolated and implemented, which are discussed in detail in Section II.C.

References

Itakura, F. (1975)
"Minimum Prediction Residual Principle Applied to Speech Recognition," IEEE Trans. on Audio, Speech, and Signal Processing, Vol. ASSP-23.

Schwartz, R. (1971)
"Automatic Normalization for Recognition of Vowels of All Speakers," S.B. Thesis, M.I.T., Cambridge, Ma.

Schwartz, R. (1975)
"Acoustic Phonetic Experimental Facility," Report No. 3107, Bolt Beranek and Newman Inc., Cambridge, Ma. (forthcoming).

Woods, W. A., R. M. Schwartz, J. W. Klovstad, C. C. Cook, J. J. Wolf, M. A. Bates, B. L. Nash-Webber, B. C. Bruce, and V. W. Zue (1975a)
"Speech Understanding Systems, Quarterly Technical Progress Report No. 1, 1 November 1974 to 1 February 1975," Report No. 3018, Bolt Beranek and Newman Inc., Cambridge, Ma.

Woods, W. A., R. M. Schwartz, C. C. Cook, D. H. Klatt, J. J. Wolf, M. A. Bates, B. L. Nash-Webber, B. C. Bruce, and J. I. Makhoul (1975b)
"Speech Understanding Systems, Quarterly Technical Progress Report No. 2, 1 February 1975 to 1 May 1975," Report No. 3080, Bolt Beranek and Newman Inc., Cambridge, Ma.

II. TECHNICAL NOTES

A. Time Expressions

Lyn Bates
Bertram C. Bruce

References to dates and time are frequent in the travel budget management domain. The manager needs to know what trips are scheduled for the current budget period, how long a given trip is (and therefore, how expensive), and what conflicts there may be among conference dates or planned trips. In order to understand these time expressions and to process them correctly, we have written a set of programs which (1) parse time and date expressions, (2) convert the parse structures into structures well suited for inference and retrieval, (3) calculate ordering relations among (perhaps incompletely specified) time points, (4) calculate lengths of time from (perhaps incompletely specified) time periods, and (5) generate English descriptions of the time information. This technical note is a discussion of the scope of the problem we are working on, the time/date grammar, the parse structures and the data base representation for the time information. Thus it is basically a presentation of the above points (1) and (2).

Before discussing the details of representation, we should point out what is not being attempted. We are not trying to handle every conceivable expression which has any temporal import. We do not expect the mechanisms discussed

here to process "the driest season in the last ten years," "the last week of our most recent contract year," or even "the week of April 10."

Instead of attempting total generality, we are deliberately isolating a rather large subset of English time phrases which can be processed in a somewhat isolated and efficient manner without recourse to extensive semantic analysis. Thus phrases like:

Late last week John left for Chicago.
We spent money during July.
Will Bill go to Washington in April?
How much did we spend this last quarter?
He is going late in the fall.
Lyn is going to Colorado on August fifteenth.

can be parsed independently of the rest of the parsing and packaged for the data base without the usual semantic processing. Besides being efficient, this allows us to concentrate on other syntactic-semantic problems, and does not preclude handling a more general class of time expressions in the normal way.

The augmented transition network (ATN) for the time grammar is shown in Figure 1. In the figure, WEEKDAY and MONTH are syntactic categories. ORD/ and NUMBER/ are ATN's themselves that are not shown which recognize ordinal and cardinal numbers respectively. The tests on the arcs preclude expressions like "Thursday the five," "thirty January fifteenth," and "the ten of June."

Syntax uses the time grammar of Figure 1 to build a special parse structure which is generally a substructure of that shown in Figure 2. This structure leaves out function words within the time phrases whose meaning is only to determine which structure is to be built. The whole time parse structure can serve as the object of a preposition, as an adverb, or as an adjective.

We currently allow at most one ordinal and one adjective for the phrase as a whole, e.g., "the last of the year." There may also be an ordinal and adjective on each subunit of the structure. For each subunit, e.g. MONTH, there may or may not be a third link pointing to the value. For example, "next May" has the structure.

(TIME (MONTH (ORD NEXT) MAY))

whereas "next month" is simply,

(TIME (MONTH (ORD NEXT)))

Some representative phrases with their parse structures are shown in Figure 3.

Once the parse structure is completed, it becomes a sort of "black box" marked as a time expression. That is, Semantics does not need to analyze it nor bother with connections between elements outside the time expression and subunits of the time expression. Instead, a data base function, TIMEBUILD, takes the parsed time expression directly and builds the appropriate data base structures.

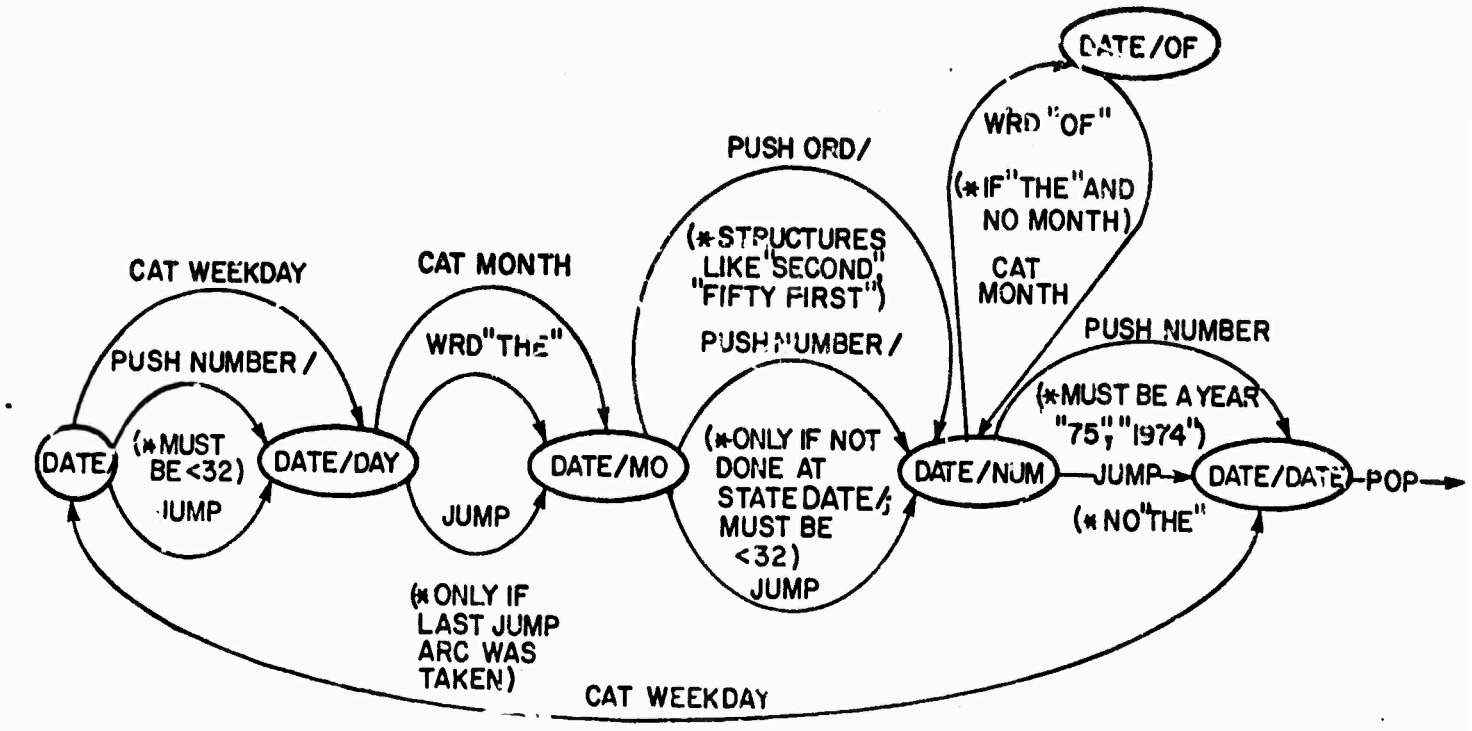


Figure 1. The ATN grammar fragment for time expressions.

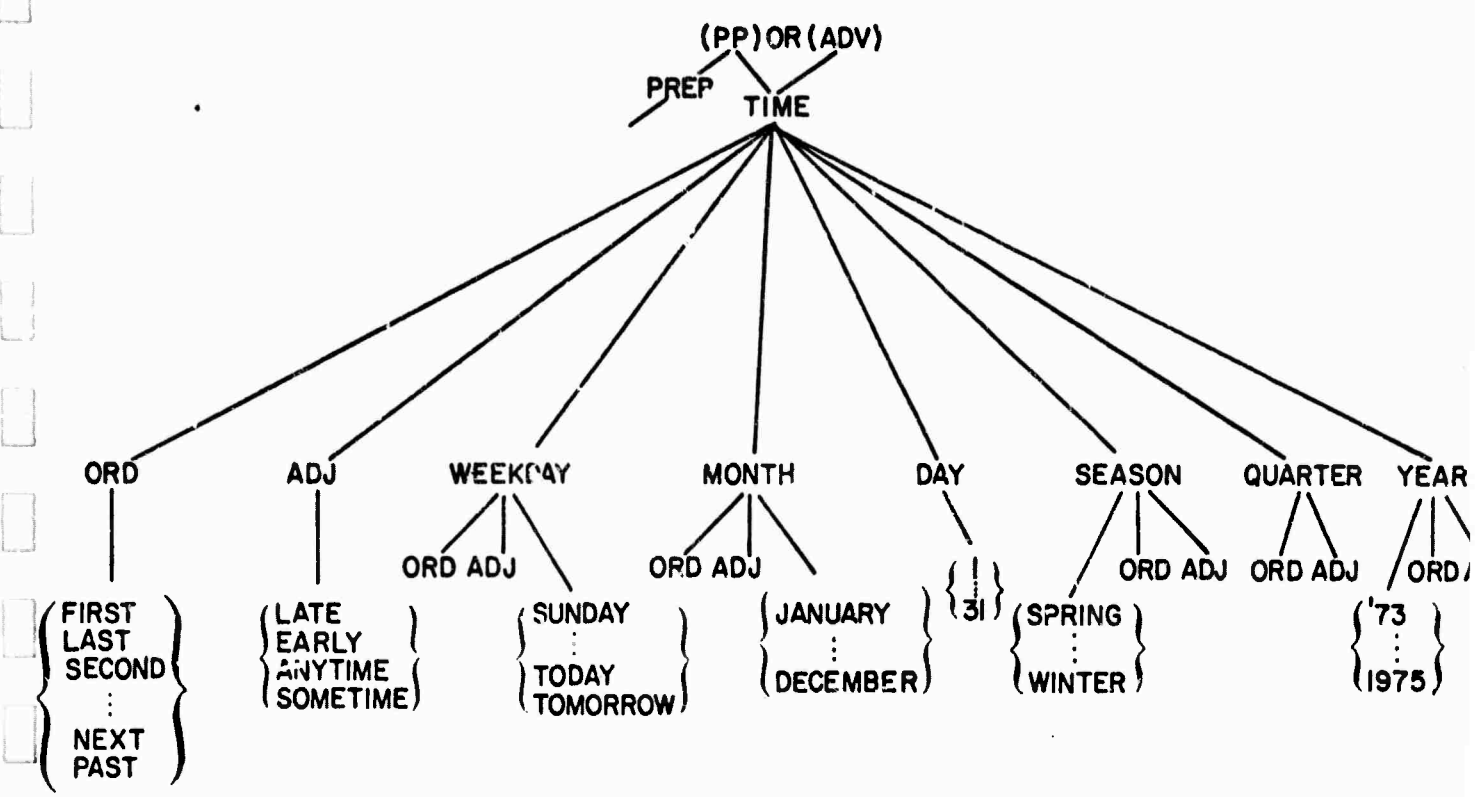
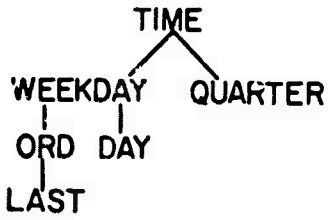
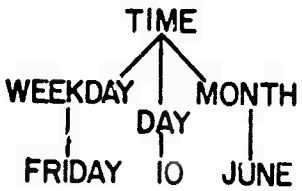


Figure 2. Potential structures built by the parser for time expressions.

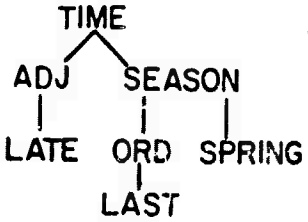
THE LAST DAY OF THE QUARTER



FRIDAY THE 10th OF JUNE



LATE LAST SPRING, LATE IN LAST SPRING



EARLY IN THE FIRST QUARTER OF THE FISCAL YEAR

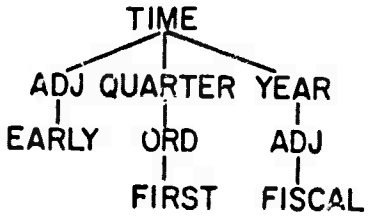


Figure 3. Examples of parse structures.

For example, the phrase, "on a Tuesday in June, 1975" is parsed into the structure,

(TIME (WEEKDAY TUESDAY)(MONTH JUNE)(YEAR 1975))

TIMEBUILD uses this parse structure to build a data base structure such as shown in Figure 4. In order to make the data base structure, TIMEBUILD must consider the ordinals and adjectives and perform appropriate transformations on the values for each subunit.

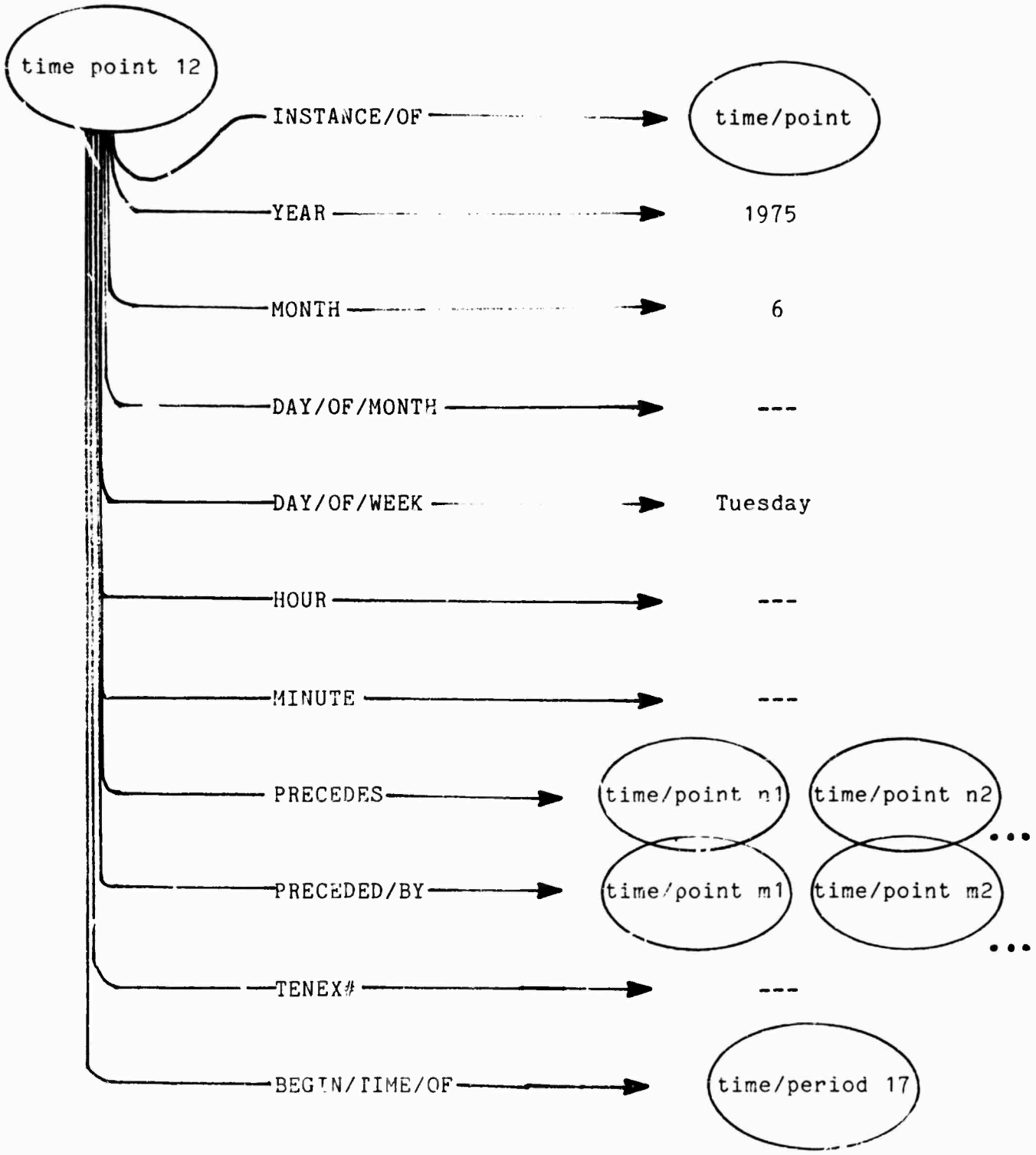


Figure 4. Structure of a time point for "on a Tuesday in June, 1975."

The current version of TIMEBUILD processes the MONTH portion of a TIME parse structure as follows: If there is no ORD (ordinal) link then the MONTH number for the month (e.g., August -> 8) is stored as is. If there is no month value, as in "this month", then the ORD link is used to calculate the appropriate MONTH and YEAR from the corresponding values on NOW, where NOW is a globally accessible time point which represents the current time. Note that for time expression we are treating "this," "next," and "last" as ordinals. If there is both an ORD link and a month value, then each type of ordinal has its own interpretation. For instance, "next" is interpreted as the first future occurrence of the specified month, e.g., if NOW is June, 1975, then "next August" means August, 1975 and "next May" means May, 1976. Other portions of the TIME parse structure are processed in a similar way. The current interpretations are only approximate and need to be buttressed by a consideration of tense, topic and discourse structure.

In addition to time points, the data base has representations for time periods and lengths of time. Examples of these are shown in Figures 5 and 6. The time period is the highest level time structure and may be linked by TIME/OF to some event. (In Figure 5, the time period is associated with a trip). The time period has a CREATE/TIME which is the time the node was added to the data base.

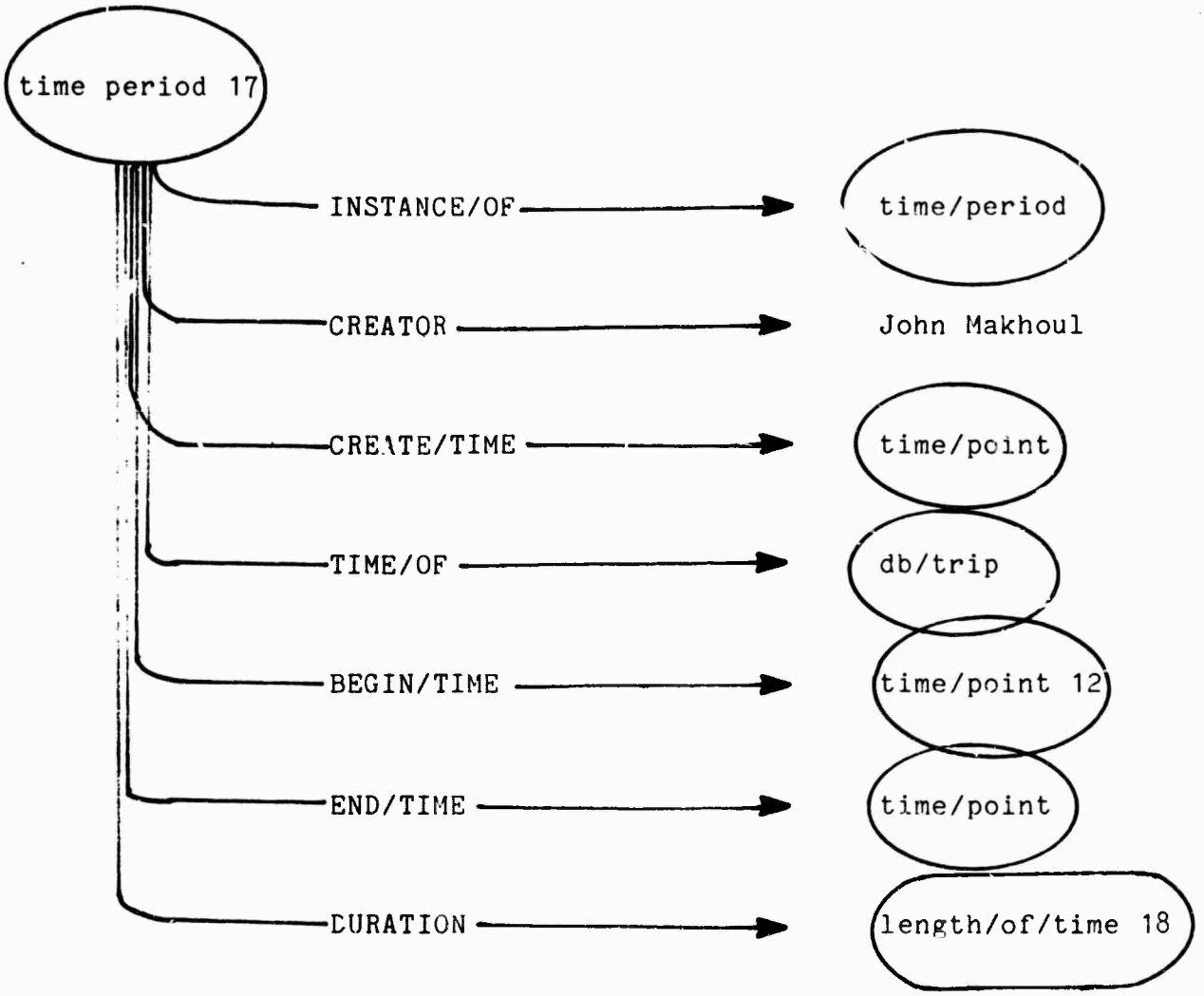


Figure 5. Structure of a time period.

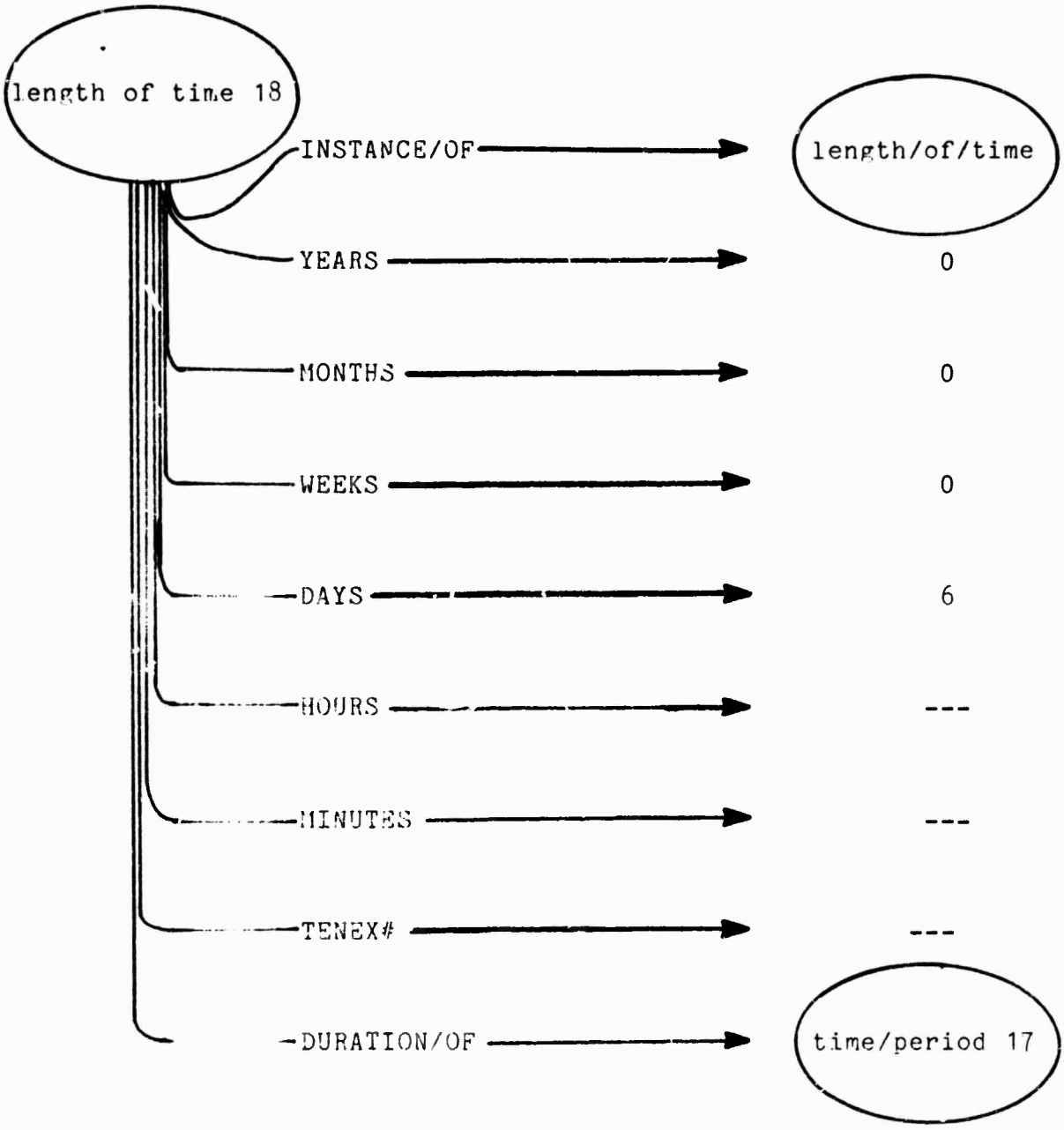


Figure 6. Structure of a length of time.

The time period may be completely or partially specified. For example, one might know only the length of a trip and not its starting time; or one might know when it starts but not when it ends (or its length). The inference routines (see Section II.B.) are able to process such information in various, incomplete forms and produce results at the maximum possible information level.

Time points and lengths of time are the main components of a time period. Both have year, month, day, hour and minute values. In addition time points have links to those time points which they precede or follow (only if that can not be determined directly from their values).

We plan to continue work on the time grammar in several areas. One is to make the grammar more selective especially about prepositions. For example, one says

late on the first Tuesday in June

but

late in the last week of June.

The time grammar also needs to be better integrated with the rest of the travel grammar. In particular the use of nominal time phrases as adverbials (e.g., John is going to California next week.) needs to be worked out more fully.

B. Procedural Semantics in the Travel System

Bertram C. Bruce
Gregory Harris

1. The Command Language

A formal command language for interacting with the travel budget management data base has now been incorporated into SPEECHLIS. On its own it functions as a tool to build and access the travel data base. In the context of processing utterances, it will serve as the goal language of semantic interpretation applied to the parse trees built by Syntax and the caseframes built by Semantics.

Sentences of the command language consist of expressions built out of operators and their arguments. The operators specify operations to be performed on the data base or interactions with the user. The arguments may either refer to elements of the semantic network or be arbitrary constant expressions. In the first case, the argument may be specified by its print name, its index in the network, or by a LISP expression to be evaluated. Some examples of English sentences and their expression in the command language are given below:

(a) Bill is going to Chicago on March 15th.

```
(BUILD: DB/TRIP
        (TRAVELER (FIND: PERSON (FIRSTNAME 'BILL)))
        (DESTINATION 'CHICAGO)
        (BEGIN/TIME '(TIME(MONTH MARCH)(DAY 15))))
```

(b) Estimate a cost for his trip.

```
(FOR: THE X1 / (FIND: DB/TRIP
                    (TRAVELER (FIND: PERSON (GENDER MALE))))
              : T ; (GENERATE: (COMPUTE: X1 'COST)))
```

(c) Print out all scheduled trips.

```
(FOR: EVERY X1 / (FIND: DB/TRIP  (MODALITY 'SCHEDULED))
                      : T ; (GENERATE: X1))
```

(d) When is Lyn going to London?

```
(FOR: THE X1 /(FIND: DB/TRIP
                   (TRAVELER (FIND: PERSON (FIRSTNAME 'LYN))))
                   (DESTINATION 'LONDON))
                 : (AFTER? (GET: X1 'TIME) NOW)
                 ; (GENERATE: (GET: X1 'TIME)))
```

(e) How long is Bill's trip to Chicago?

```
(FOR: THE X1 / (FIND: DB/TRIP
                    (TRAVELER (FIND: PERSON (FIRSTNAME 'BILL)))
                    (DESTINATION 'CHICAGO))
                  : (AFTER? (GET: X1 TIME) NOW)
                  ; (GENERATE: (GET: X1 '#DAYS)))
```

(f) When is the first trip Chip is taking next year?

```
(FOR: (ORD 1) X1 / (FIND: DB/TRIP
                       (TRAVELER (FIND: PERSON (FIRSTNAME 'CHIP))))
                     : (DURING (GET: X1 'BEGIN/TIME)
                               (TIMEBUILD '((ORD NEXT)(YEAR))))
                     ; (GENERATE: X1))
```

The present implementation of some of the functions used in the command language is described below:

(ADD: node link value)

Adds value to node under the attribute link. If link has an ADDFN property associated with it, then the value of that property is a procedure which is executed to add value. Otherwise, SEMNET primitives are used to make the appropriate relational or property

connections.

(COMPUTE: node link)

A command to compute, as opposed to just find, a value for node and link; equivalent to (GET: node link ? DEFAULT). (See below.)

(GET: node link value flag)

If value is NIL or ? then GET: follows link from node and returns value. Otherwise it verifies (or denies) that the specified value is stored. Flag determines the extent of search if the value is not stored explicitly. Currently, if flag is T then no search is done. If it is DEFAULT, then inferences are done as determined by METHODS associated with the link name (see II.B.2). If no METHOD succeeds, then the speaker is asked for help. If flag is NIL then the speaker is again consulted.

(BUILD: item-type (link1 value1) (link2 value2) ...)

Builds an item which is an instance of item-type and has the specified link-value pairs. Uses ADD: for each pair and also adds DB/CREATOR and CREATE/TIME links.

(FIND: item-type (link1 value1) (link2 value2) ...)

Finds an item which is an instance of item-type and has the specified link value pairs. Uses GET: for each pair. FIND: is an enumeration function which can be used with FOR: (see below) to produce elements one at a time.

(FOR: quantifier variable / class : restriction ; command)

Applies command to elements of class for which restriction holds and as determined by quantifier. Variable is bound to elements of the restricted class and is a free variable in command. (The permissible values for quantifier have been generalized from those in LUNAR [Woods, W. A., R. M. Kaplan and B. Nash-Webber, 1972] system to allow specification of cardinals by (THE <number>). Also, FOR: now distinguishes the universal quantifier, EACH, which requires that at least one item belong to class from its counterpart, every which does not.)

(COMPLETE: item)

Special command which searches through item description and asks for missing values. It stops when the

description is complete or when the user says "stop". COMPLETE: also allows the user to say "unknown" to any question.

(GENERATE: arg)

Examines arg to determine appropriate form of printout. If arg is an item whose item-type has a PRINTFN then that procedure is used to print. If there is no PRINTFN then SEMNET printing primitives are used. GENERATE: also prints strings and lists.

2. Inference Done in the Course of Retrieval

Inference in this system can be viewed as a natural generalization of the notion of structures with slots and default values for each slot. Here, instead of being values, defaults are procedures for determining the appropriate value whenever a slot filler is missing. These procedures may require the values of other slots, which in turn may require activating other default procedures. The inference process also contains an advice-passing mechanism that gives it a modicum of control and an ultimate default, which is to ask the speaker.

The current inference process is implemented via the function, GET:. GET: can be used to find the value for a node-link pair or to verify that a specified value is there. A flag can be set that determines the depth of search and whether or not the speaker is to be asked in the event of failure. The effect of the GET: implementation is that the basic operation of requesting the value of an attribute of an object is not conditioned by (perhaps arbitrary) data

base constructions. It also means that whereas a new attribute must be structurally defined, there does not need to be a special set of functions for retrieving its value under an indefinitely large assortment of situations.

For example, the call (GET: <trip> 'COST) is produced as part of the interpretation of "the cost for that trip." If "cost" were stored explicitly, then no deduction would be required. If not, a cascade of calls to GET: can result, based on the default functions for "cost" (see Figure 7). Advice can be passed from higher to lower level calls to guide and constrain the inference process.

The recursive mechanism of GET: is driven by the property, METHODS on the link name specified in the call to GET:. Each METHOD consists of an APPLICABILITY/TEST which restricts the application of the method, a FUNCTION naming an operation to be performed, and ARGUMENT/PATHS which specify, for each argument of the FUNCTION, what links to follow (via GET:) from the present node to get the desired values. As each method is applied, it builds a GENERATE-able trace of its computation tree such as that shown in Figure 7. (The actual printing of this tree is not yet implemented.) The tree enables the system, after estimating a cost, for example, to answer the question "How did you get that?" It also can set monitors on questions about trivially different computations, e.g., "What if the

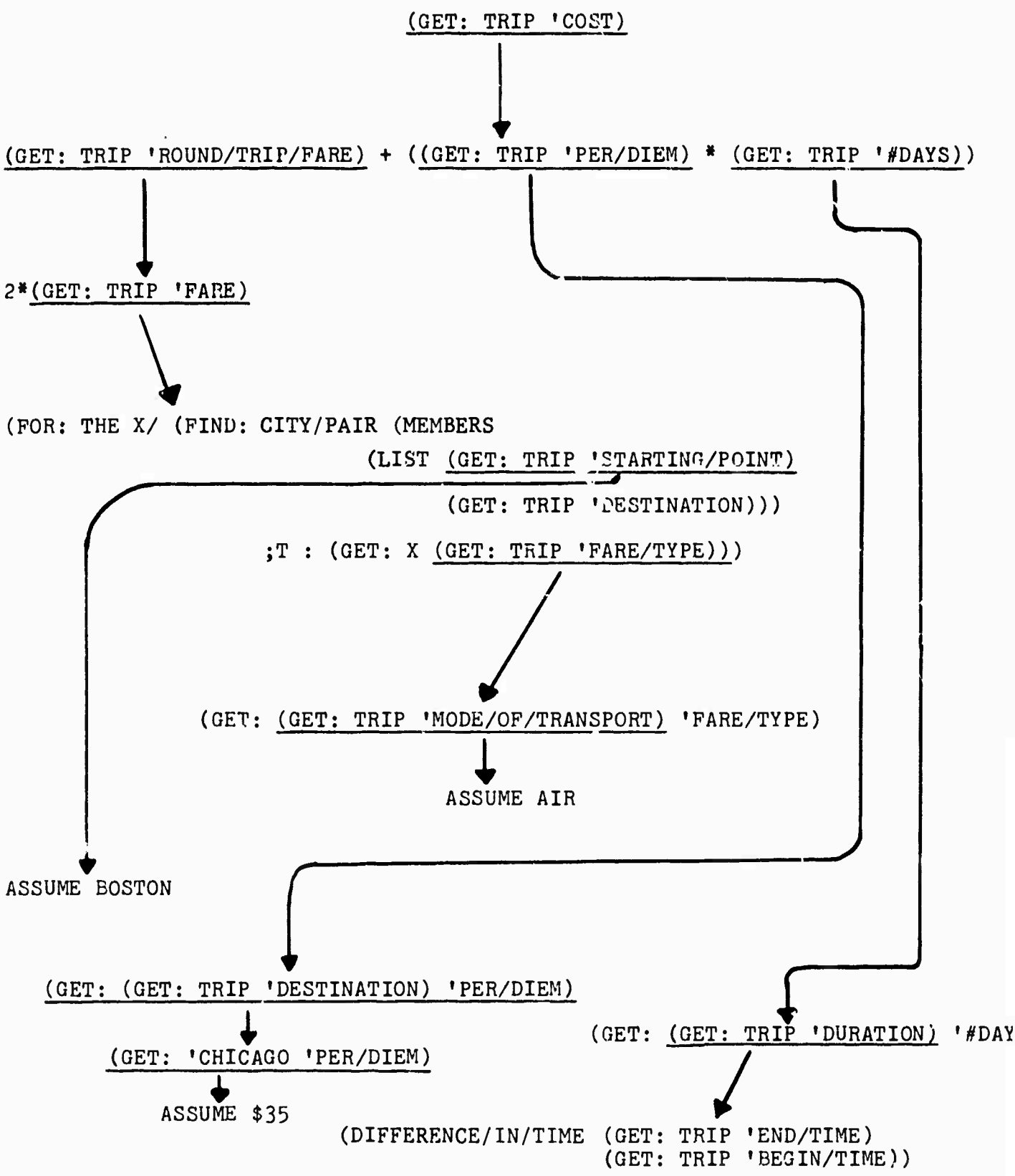


Figure 7. A trace of the inference process obtaining the cost of a trip.

per diem were twenty dollars?" "What if it were for six days?"

3. Coordinating Execution with the Discourse Model

The formal command language has been designed to permit a fairly direct mapping of an input English sentence into its underlying concepts without regard for how information is actually stored. Thus we have (TRAVELER (FIND* PERSON (FIRSTNAME 'BILL))) and (GET: <trip> 'TIME) even though discourse context must be used to pick which "BILL" is meant, trips have their times associated with their individual legs.

We have found the notion of a demand queue model useful in accounting for discourse reference. It also helps to explain how one computation of a response can be pushed down, while a whole dialogue takes place to obtain missing information, and how a computation can spawn subsequent expectations or digressions. Some elements of this demand model are explained below:

(a) Demands: These are demands for service of some sort made upon the system by the user or by the system itself. An active unanswered question is a typical demand with high priority. The fact that some questions cannot be answered without more information leads to the

```
user-makes-query
system-asks-question
user-clarifies
system-answers-query
```

kind of embedding we have been calling a "mode of interaction." Demands of lower priority include such things as a notice by the system that the manager is over his budget. Such a notice might not be communicated until after direct questions had been answered.

(b) Counter-demands: These are questions the system has explicitly or implicitly asked the user. While it should not hold on to these as long as it does to demands, nor expect too strongly that they will be met, the system can reasonably expect that most counter-demands will be resolved in some way. This is an additional influence on the discourse structure.

(c) Current topic: This is the active focus of attention in the dialogue. It could be the actual budget, a hypothetical budget, a particular trip, or a conference. The current topic is used as an anchor point for resolving references and deciding how much detail to give in responses. Again, this structure leads to certain modes of interaction. For example, if the manager says "Enter a trip," the system notes that the current topic has changed to an incompletely described tri. This results in demands that cause standard fill-in questions to be asked. If the manager wants to complete the trip description later, then

the completion of the trip description becomes a low priority demand.

(d) Miscellaneous deictic structures: The discourse area of the data base also contains an assortment of items strongly linked to the here and now, including:

a) NOW, a pointer to the current time and date,

b) SPEAKER, a pointer to the current speaker,

c) the last mentioned person, place, time, trip, budget, conference, etc.

We are designing a preliminary, one-queue implementation of this "demand model." This queue will consist of forms such as (DO (FOR: --) --) which represent the speaker's previous queries and commands as well as commands initiated by the system to examine the consequences of its actions, give information to the user, or check for data base consistency. These forms are related by functional dependencies and relative priorities. At the present time, there are only a few demand types: DO means execute the specified command. TEST means evaluate the form to NIL or non-NIL and answer "no" or "yes" accordingly. RESPOND means give the user some information (which may or may not be part of an answer to a direct query). PREVENT means monitor for a subsequent possible action and block its normal execution (as in "Do not allow more than three trips to Europe.").

References

Woods, W. A., R. M. Kaplan and B. Nash-Webber (1972) "The Lunar Sciences Natural Language Information System: Final Report," BBN Report No. 2378, Bolt Beranek and Newman inc., Cambridge, Ma.

C. Control Primitives

Bonnie Nash-Webber

To aid us in devising and simulating possible control strategies, we have isolated into separate user-callable functions those operations which our current set of data structures suggest to be primitive, plus other functions that have seemed useful. The current set of such functions is undoubtedly incomplete. New data structures and other ways of relating instances of current ones to each other will most likely lead to new control primitives. We have been using the following set of primitives in the incremental simulations of the speech system run in the last quarter. (Excerpts from one such incremental simulation session follow this section.)

1. A function for reading in a new utterance: SENTENCE!
2. A function for creating a lattice of the highest lexically scoring wordmatches: SCAN
3. A function for making a theory of a set of wordmatches: MKTHRY
4. A function for refining a theory with a new wordmatch: REFTHRY
5. Functions for evaluating a theory: SEMVAL, SYNVAL, PRAGVAL
6. Functions for constructing user-made proposals: WORDPSL, CATPSL, BOTHPSL
7. Functions for doing the proposals (i.e., sending them down to the lexical retrieval fork): DOPSL, DOPSLS
8. A function for removing a proposal from the proposals list without having done it: REMOVEPSL

9. Functions for doing the events made by some component: DOEVT, DOEVTS, DOWORDEVTS

10. A function for printing any of the control data structures: P

11. A function for doing a more thorough scan on a region: SCANREGION

12. A function for ascertaining the existence of better matches of a particular word, given one match for that word has already been found: BETTERMATCH?

13. A function for creating fuzzy wordmatches: FUZZ?

14. Functions for talking to the various forks directly: MATCHCONTROL, SYNCONTROL

Notice several things about the above groups of functions. First, we have kept separate the notion of creating a proposal from that of actually doing it. This allows proposals to be queued and selected later. Secondly, proposals can now be made by either a SPEECHLIS component or the user. This allows the user to make proposals, while postponing the decision about which component should have had the smarts to do so itself. Thirdly, we have tried to be somewhat consistent in naming conventions, e.g., anything with PSL in its name refers to proposals, THRY to theories, EVT to events, VAL to evaluation. Finally, evaluation of a theory by a component may involve that component's making a hypothesis about how the words fit together, as well as comparing that hypothesis against information already in the theory. For example, if Syntax goes first, the consistency of Syntax and Semantics is part of the SENVAL evaluation, while if Semantics evaluates the theory first, the

consistency check is part of SYNVAL.

Functions

1. Reading in a sentence.

[SENTENCE! <utterance> <suffix>]

An NLAMBDA which sets up the lexical retrieval fork on the given utterance and also upper level internal structures which depend on the utterance. <suffix> refers to the suffix on the segment label file for the utterance, which is a code for the type of segment labelling. Every time SENTENCE! is called, the lexical component is set up anew.

e.g. [SENTENCE! JJW110 C]

2. Creating a lattice of good words.

[SCAN]

A function of no args which requests the lexical retrieval fork to find the n best wordmatches in the given utterance (currently, n=15), which it then puts into the word lattice, without doing anything else to them. [They are no longer automatically sent to Semantics for evaluation, as they had been in the original SPEECHLIS control strategy.]

3. Making a theory.

[MKTHRY <args>]

MKTHRY is an NLAMBDA nospread which can take any number of arguments. Each argument is a wordmatch handle, i.e. either a number, corresponding to a wordmatch index, or a function which evaluates to a wordmatch, either simple or fuzzy. See 13. for a description of FUZZ?, which will create a fuzzy wordmatch around a given simple wordmatch, if "like" matches exist.

MKTHRY creates a theory data structure and records it on THEORYTBL. It also calls for a lexical evaluation of the theory, which may result in the spawning of son theories whose fuzzy wordmatches have been reduced or even replaced by simple word matches. Other componential evaluations (i.e. Syntactic, Semantic and Pragmatic) can be called for separately. (See 5.)

e.g. [MKTHRY 11 3 (FUZZ? 14) 18]

4. Refining a theory (user-called).

[REFTHRY <theory number> <wordmatch handle>]

REFTHRY is an NLAMBDA which takes a theory number (e.g. 1,2,3,...) and a wordmatch handle, i.e. either a number, corresponding to a wordmatch index, or a function which evaluates to a wordmatch, either simple or fuzzy. Its output is a new theory, a son of the original one, containing the augmented list of wordmatches. It is, in a sense, acting like a user-created event. Like MKTHRY, lexical evaluation is also done on the theory, which again may result in the spawning of refined son theories.

e.g. [REFTHRY 5 6] or [REFTHRY 5 (FUZZ? 14)]

5. Evaluating a theory.

[SEMVAL <theory number>]
[SYNVAL <theory number>]
[PRAGVAL <theory number>]

Each of these functions may be called with either a theory number (an integer) as argument or no argument at all. In the latter case, it is assumed that an evaluation of the last theory created (LASTHEORY) is desired. Each of these functions does one specific kind of evaluation: semantic, syntactic, or pragmatic.

Semantic evaluation of a theory is performed by SEMVAL. It is assumed that the theory has not previously been seen by Semantics, which tries to both construct one or more consistent semantic hypotheses for the set of wordmatches contained in the theory and evaluate those hypotheses. When SEMVAL is given a theory containing more than one wordmatch, it is as if Semantics had taken over control from the Control component. That is, local monitors are set and local events processed as each word in the theory is considered, until either a set of consistent hypotheses is established for the entire wordmatch set or no local events remain to be processed. After the theory is processed, what remains are external monitors for other concepts which could be of use to the theory. SEMVAL is not fully worked out for multi-word theories yet. That is, it is not clear whether the local monitors should disappear after processing or whether they should remain to reduce Semantics' load when given

another theory to evaluate, containing some of the same wordmatches. If they remain, much care will have to be taken to avoid making inappropriate associations.

Syntactic evaluation (SYNVAL) involves constructing partial parses for the set of wordmatches in the theory. The syntactic part of a theory is currently kept down in the lower fork housing syntax, so the only obvious effect of SYNVAL on a theory data structure is the replacement of its syntactic score with the value returned from SYNVAL.

With respect to pragmatic evaluation of a theory, it is not currently clear whether Pragmatics will play a separate role in evaluating a theory which does not span the entire utterance. In any case, when implemented, PRAGVAL will call for the pragmatic evaluation of a theory.

e.g. (SYNVAL 7) or (SEMVAL)

We have not made the lexical evaluation of a theory a user callable control primitive. Since lexical evaluation depends only on wordmatch scores, no real "knowledge source" needs be called upon to compute it. The lexical score for a theory is currently set to the sum of the scores of theory wordmatches. For a fuzzy wordmatch, the score of its best member wordmatch is taken as the score for the whole fuzzy.

When a theory contains fuzzy wordmatches, its lexical evaluation may result in its abandonment in favor of sons spawned during the evaluation process. These sons differ from their father in having more clearly defined fuzzy wordmatches, or no fuzzies at all. The reason for so refining a theory is that, to Syntax, each of the sons will now have a clearly defined character, which their father lacks because it is "too fuzzy". Refinements are created when the following situation arises: the best wordmatch in a fuzzy is separated from its neighbor to the left or right by a one segment gap. If by considering some other match in the fuzzy, this gap could be eliminated, two new son theories are created: one which contains the gap and one which doesn't. The resulting theories are quite different to Syntax, since adjacency is its strongest constraint on how a set of wordmatches can be parsed.

6. Making proposals.

[WORDPSL <wordlist> <direction>
<boundary or context>]
[CATPSL <category list> <direction>
<boundary or context>]
[BOTHPSL (<wordlist><category list>)
<direction> <boundary or context>]

Proposals can arise automatically during the evaluation of a theory by one of the components. In addition, there are three control primitives which allow the user to make proposals too. Note that "making" a proposal is different from actually "doing" it, i.e., sending it down to the lexical retrieval fork for execution. "Making" a proposal just puts it on the appropriate proposal queue. Queuing proposals this way allows for merging similar ones, i.e. ones with similar direction and intersecting contexts, and also for deciding which ones to do when.

Here <wordlist> is a list of words like (GO TRAVEL VISIT), <category list> is a list of word classes like (AUX V ADV). The PSL functions all make appropriate checks that each member of <wordlist> is in the dictionary and each member of <category list> is a valid word class, as supported by the lexical retrieval component. The value of CATEGORIES is the current list of valid categories. <direction> is either RIGHT/OF, LEFT/OF or BETWEEN, indicating the words or categories should be searched for to the left, right or between the given segment boundaries or wordmatches. <boundary or context> then is either a single number, indicating a segment boundary, a dotted pair, indicating two segment boundaries (used with BETWEEN); a list of wordmatch indices; or a double list of wordmatch indices (again, used with BETWEEN, for left and right context). Either of these latter two options may be prefaced by "WM", if the user wants to make sure he does not make a mistake and type a boundary number when he means a wordmatch and vice versa. Matches resulting from proposals, either user or component made, will be anchored at the boundary or context.

e.g. (WORDPSL (ANYWHERE) RIGHT/OF (2 9))
(CATPSL (ADJ QUANT ART) LEFT/OF 7]
(CATPSL (PREP) BETWEEN ((7 11)(5)))
(BOTHPSL ((IJCAI)(ADJ ADV)) RIGHT/OF (WM 17))

7. Doing the proposals.

[DOPSLS]
[DOPSL <args>]

Doing proposals involves sending them down to the lexical retrieval fork for execution. The first function, DOPSLS, does all the proposals currently waiting to be done. The second function DOPSL can be called with a list of numbers, corresponding to the numbers of those proposals the user wishes to have done, or no arguments at all, indicating the user wishes to have the last proposal he made done. One gets the numbers associated with proposals by printing them out with (P PROPOSALS).

e.g. (DOPSLS) or (DOPSL 3 1 4) or (DOPSL)

8. Removing a proposal.

[REMOVEPSL <n>]

In debugging, one may find that an incorrectly formatted proposal has gotten on the list of proposals. To avoid the chance of sending it down to the lexical retrieval fork, one can use the function REMOVEPSL to remove it. Its argument is the number of the proposal one wishes to have removed. Again, one gets the number by printing out the proposals with (P PROPOSALS).

9. Doing events.

[DOEVTS]
[DOEVT <args>]
[DOWORDEVTS]

These functions allow the user to select a specific set or type of component generated events to have done (DOEVT, DOWORDEVTS) or to do them all (DOEVTS). (There is currently no simple way for the user to create his own events and then have them done.) DOEVT takes as its input a list of event numbers which can be gotten by printing out the eventqueue with (P EVENTS). The corresponding events are then removed from the eventqueue and executed in the specified order. DOWORDEVTS calls for the processing of "word" events created by Semantics, which result in the construction of "multi-word names" like "registration fee" and "travel budget". This special function exists because of wanting to do these "word" events before any other ones.

e.g. (DOEVTS) or (DOEVT 3 4 7)

10. Printing a control data structure.

[P <data structure>]

P is an NLAMBDA which takes as its argument the name of a data structure, which it then prints in a form easy to read and understand. Currently, the following data structure names are acceptable arguments to P:

LATTICE - prints out the word lattice
EVENTS - prints out the eventqueue in an easily readable, though sketchy, way
EVENT <n> - printsout event <n> in full detail
PROPOSALS, PSLS - prints out the extant proposals
THEORIES - prints out the theories
THEORY <n> - prints out theory n
MATCHES <word> - prints out all wordmatches for <word>
WLATMON <bdry> - prints out the word lattice monitors either starting or ending at <bdry>
CFT <n> - prints caseframe token <n>
WORDSTARTS <n> - prints the list of wordmatches whose left boundary is <n>
WORDENDS <n> - prints the lst of wordmatches whose right boundary is <n>

e.g. (P THEORY 2) or (P LATTICE)
or (P MATCHES TRIP)

11. Scanning a region.

[SCANREGION <direction> <boundary or context>]

SCANREGION allows one to search a specific region of the utterance, for example, the beginning of the utterance or a region where no nice words have been found. <direction>, as in the PSL functions, can be either LEFT/OF, RIGHT/OF or BETWEEN. <boundary or context> has the same form as that used in the proposal functions (See 6.). If <direction> is BETWEEN and the second argument is a dotted pair of boundaries, the scan is done sliding. Otherwise the scan is anchored at the appropriate side of the wordmatches or the appropriate boundary. Like SCAN, SCANREGION currently returns the 15 best matches in the region. Those matches which are not above SCANTHRESHOLD (currently set to 100) are put on a list of REJECTS, which the user is shown and ask to dispose of. Each one may be

put into the wordlattice or forgotten about.

e.g. (SCANREGION BETWEEN (7 . 10))
(SCANREGION LEFT/OF (2 9))
(SCANREGION RIGHT/OF (WM 6 14))

12. Ascertaining better wordmatches.

[BETTERMATCH? <n>]

A long word which matches well may be found to match better if the original constraints on its left and/or right boundary are lifted. BETTERMATCH? takes a wordmatch index and looks freely around the utterance for overlapping matches of that word better than the given one.

13. Creating a fuzzy wordmatch.

[FUZZ? <n>]

FUZZ? takes a wordmatch index <n> as argument and looks in the word lattice for other matches of the same word which are fuzzily close to the given match. If there are, it combines them into a fuzzy wordmatch in descending order of quality. This is returned as the value of FUZZ?. If no close matches exist, the wordmatch corresponding to <n> is returned. (Note a fuzzy wordmatch is represented as a list of simple wordmatches.)

e.g. (FUZZ? 3)

14. Accessing forks directly.

[MATCHCONTROL]
[SYNCONTROL]

There are debugging situations in which one wants to root around in one of the lower forks to find the cause of an error. MATCHCONTROL will put the user in direct contact with the lexical retrieval fork, and SYNCONTROL, to the lower LISP fork housing Syntax. To exit from the former, the user should type Q<cr>; from the latter, OK<cr>.

Incremental Simulation - an example:

The control strategy we are simulating in the incremental simulation, excerpts from which follow this introduction, relies initially on the best matching words the lexical retrieval component can find on an anchored left-to-right scan over some region of the utterance, starting from the initial one. After each left-to-right scan, the best matching word is given to semantics, who notes the contexts in which that word could occur. (If several words are tied in word-match for best match, the strategy is to consider the likelihood of occurrence of the matched pronunciation of the given word.) Processing the proposals made by a higher-level component and notices of detected word coincidences takes precedence over doing the next left-to-right scan, starting from the right end of the last best matching word. During this process, multiple theories may be created either because of noted semantic associations between the word matches or just excellence of match quality. Whenever a theory is spawned which has two or more adjacent word matches, it is sent to Syntax for evaluation, perhaps resulting in further proposals or events whose processing, as before, takes precedence over left-to-right scan.

A final element of this control strategy, another deviation from strict left-to-rightness, is meant to get

around a problem caused by anchored scans. Requiring a word to match starting and/or ending at some pre-specified position may result in some contorting of the match to meet that constraint and, therefore, in a lower score for the match. Giving the Matcher freedom to choose both ends will allow it to make the best possible match. Therefore, in this strategy, if a long word (longer than six phonemes) matches well anchored, a sliding scan is made for it to see if it would match better with slightly different boundaries.

To make reading this extract easier, note that a word match is printed across the line as:

<wordmatch index> <word> <left boundary>
<right boundary> <match quality>
<a priori likelihood of the particular
pronunciation used in the match>
<inflection, or -- if uninflected>

Lines typed by the user are preceded by a line number followed by an underline.

```
35_SENTENCE!(JJW110 C)
T
36_(SCANREGION RIGHT/OF 0)
1 WHAT-R 0 3 193 -31 --
2 WHAT 0 3 193 0 --
3 ONE 0 3 191 0 --
4 WHEN 0 3 104 0 --
5 ON 0 3 95 -23 --
6 WAS-R 0 3 83 -31 --
7 WAS 0 3 83 0 --
8 ALL 0 3 79 0 --
9 WOULD-R 0 3 66 -31 --
10 WENT 0 3 46 0 --
11 ALL 0 2 38 0 --
12 OH 0 2 29 0 --
13 L.A. 0 4 27 0 --
```

```
14 ONTO 0 4 25 -93 --
15 ON-R 0 3 22 -31 --
NIL
37_(MKTHRY (FUZZ? 2))
THEORY#1
NIL
38_(P THEORY)
193  THEORY#1
  2 WHAT 0 3 193 0 --
  NIL
  NIL
  (NIL NIL NIL NIL NIL)
  NIL
  NIL
  ((193 . 3)
   0 0 0 0)
  (NIL NIL NIL)
NIL
(* semantic evaluation requested)
39_SEMVAL]
SEMANTICS PROCESSING THEORY#1
  2 WHAT 0 3 193 0 --
     THEORY#1  WHAT
AS [ WHAT 686]
     PUTTING A CEM ON [ BE 684]
     PUTTING ' CEM ON [ CONCEPT OF GIVEABLES 720]
T
(* placing case event monitors on concepts in the
  semantic network)
40_(P THEORY)
193  THEORY#1
  2 WHAT 0 3 193 0 --
  NIL
  NIL
  (686 NIL (CFT#1)
       ((2 CFT#1))
       (686))
  NIL
  NIL
  ((193 . 3)
   0 0 0 0)
  (NIL NIL NIL)
NIL
41_(P CFT 1)
(* print caseframe token)
CASEFRAME FOR CONCEPT [WHAT BE X QUESTIONS 685]
(((REALIZES . CLAUSE)
  (CONCEPT . 685))
 (HEAD (EQU . (BE))
          NIL OBL)
 (QWORD (WHAT . (WHAT))
          NIL OBL)
 (PATIENT (MEM . [CONCEPT OF GIVEABLES 720])
```

```
          NIL OBL))
NIL
42_(P PROPOSALS)
NIL
43_(P EVENTS)
NIL
44_(* SO CONTINUE LEFT-TO-RIGHT)
SO
45_(SCANREGION RIGHT/OF (2))
16 IS-R 3 5 72 -31 --
17 IS 3 5 72 0 --
18 A 3 4 35 0 --
19 EIGHTH 3 5 1 0 --
20 AI 3 5 1 0 --
THE FOLLOWING MATCHES WERE REJECTED DUE TO THEIR LOW SCORE:
21 US 3 5 -14 0 --
22 A-R 3 4 -25 -16 --
23 IF-R 3 5 -66 -31 --
24 IF 3 5 -66 0 --
25 I 3 4 -77 0 --
26 AM-R 3 4 -77 -16 --
27 EIGHTY 3 6 -78 0 --
28 IS-R 3 4 -79 -31 --
29 IS 3 4 -79 0 --
30 ON-R 3 4 -84 -16 --
DO YOU WANT TO PUT ANY OF THEM IN THE LATTICE? TYPE EITHER N
 OR A LIST
OF THOSE YOU WISH TO ENTER
N
NIL
46_(MKTHRY (FUZZ? 17))
THEORY#2
NIL
47_(P THEORY)
72  THEORY#2
  17 IS 3 5 72 0 --
  NIL
  NIL
  (NIL NIL NIL NIL NIL)
  NIL
  NIL
  (72 . 2)
   0 0 0 0)
  (NIL NIL NIL)
NIL
48_SEMVAL]
SEMANTICS PROCESSING THEORY#2
  17 IS 3 5 72 0 --
     THEORY#2  IS
AS [ BE 684]
     PUTTING A CEM ON [ WHAT 686]
     PUTTING A CEM ON [ CONCEPT OF GIVEABLES 720]
AS [ WHAT PE X QUESTIONS 685]
```

```
   NOTICING EVENT LINKING THEORY#1 TO THEORY#2
     SCORE = 315
T
49_(P EVENTS)
1 315  SEM
     THEORY#1  WHAT
     THEORY#2  IS
NIL
50_(P PROPOSALS)
NIL
51_(* SO DO THE EVENTS)
SO
52_DOEVTS]
SEMANTICS PROCESSING EVENT JOINING THEORY#1
  2 WHAT 0 3 193 0 --
 TO THEORY#2
  17 IS 3 5 72 0 --
CREATING THEORY#3
     PUTTING A CEM ON [ CONCEPT OF GIVEABLES 720]
AS [ WHAT BE X QUESTIONS 685]
NIL
53_(P THEORY 3)
270  THEORY#3
  2 WHAT 0 3 193 0 --
  17 IS 3 5 72 0 --
  (THEORY#1 THEORY#2)
  NIL
  (NIL NIL (CFT#3)
       ((2 CFT#3)
                                    (17 CFT#3))
       (685))
  NIL
  NIL
  ((265 . 5)
   10 0 0 0)
  (NIL NIL NIL)
NIL
54_(P CFT 3)
CASEFRAME FOR CONCEPT [WHAT BE X QUESTIONS 685]
(((CFTISA 685)
  (SONOF CFT#1)
  (REALIZES . CLAUSE)
  (CONCEPT . 685))
 (HEAD (IS . (BE))
          NIL OBL)
 (QWORD (WHAT . (WHAT))
          NIL OBL)
 (PATIENT (MEM . [CONCEPT OF GIVEABLES 720])
          NIL OBL))
```

* * * * *

and so on...