AD-A011 120

SPEECH UNDERSTANDING SYSTEMS

William A. Woods, et al

Bolt Beranek and Newman, Incorporated

Prepared for:

Office of Naval Research
Advanced Research Projects Agency

May 1975

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BBN Report No. 3080	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD-A011120
4. TITLE (and Subtitle) SPEECH UNDERSTANDING SYSTEMS Quarterly Technical Progress Report No. 2 1 February 1975 to 1 May 1975		5. TYPE OF REPORT & PERIOD COVERED Quarterly Tech. Prog. Rep. 1 Feb. 1975 to 1 May 1975
		6. PERFORMING ORG. REPORT NUMBER BBN Report No. 3080
7. AUTHOR(s) William A. Woods, Richard M. Schwartz, Craig C. Cook, Dennis H. Klatt, Jared J. Wolf, Lyn A. Bates, Bonnie L. 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 May 1975
		13. NUMBER OF PAGES ~~X~~ 63
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 February to 30 April 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)

PRICES SUBJECT TO CHANGE

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 second QTPR contains such technical notes on acoustic-phonetic research, speaker normalization, prosodics, semantic network retrieval, and a new language - PCOMPILER - designed for the phonetic and phonological modules of the lexical retrieval component.

ia

SPEECH UNDERSTANDING SYSTEMS

Quarterly Technical Progress Report No. 2
1 February 1975 to 1 May 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.

ib

Table of Contents

I. PROGRESS OVERVIEWS

A. Acoustic-Phonetics

In the last quarter the general dip detector was improved and generalized so that it could be applied to any energy parameter. Using the boundary information produced by this dip detector on energy measures from three different spectral regions, we developed a program which produces rough segment lattices. This is discussed in more detail in Section II.A.1.

The interface between the Acoustic-Phonetic Recognition (APR) program and the new word matcher developed by Klovstad [1] has also been completed. The structure of the segment lattice now includes all the probability information necessary for operation of the word matcher.

We have also investigated two methods of formant tracking and several methods of formant smoothing, in order to improve our vowel and glide recognition (See Sections II.A.2 & II.A.3).

With respect to general utility packages, we now have programs which allow the user to interactively compute the energy in any spectral band using the preemphasized or unpreemphasized spectrum. These functions can also be smoothed if desired. The acoustic-phonetics statistics package has been expanded to allow the user to specify optional segments within a phonetic context, thus making it more useful for experiments dealing with the acoustics of groups of phonemes.

B. Verification

In the past quarter, work on word verification has been concerned with developing several of its subcomponents as well as with creating a language in which these components can be expressed. To review first, in word matching, when the phonetic transcription of an utterance is particularly ambiguous, it is often useful to have a component which performs a detailed word match at the parametric level. It is the purpose of the word verification component to perform this match when and where called upon by the control component. (See [2] for a discussion of word verification in a speech understanding system.) The word verification component consists of subcomponents which include the following:

- Control module
- Phonological component
- Phonetic component
- Speech synthesizer (for debugging)
- Spectrum generator
- Time normalization strategy
- Spectrum matching comparator

The phonological component and the phonetic component are being written in a Fortran-like language that has been designed to combine the power and convenience of a Bobrow-Fraser notation [3] for expressing phonological conditions of rule application with the number-crunching capabilities of Fortran. The normal Fortran syntax has been augmented to permit a more readable code format. The new rule language called PCOMPILER is converted into standard Fortran by a preprocessor which has been written in Interlisp by Bill Woods and Craig Cook. The characteristics of PCOMPILER are described in Section II.C.

The phonetic component has been written in the above-mentioned language and is currently being debugged. The strategy is based on a program written earlier in Fortran by Dennis Klatt. With the new rule language, rules can be incorporated concerning phonetic details which were very difficult to express in previous versions of the program due to the lack of a flexible programming language.

The M.I.T. synthesis-by-rule program had only a primitive phonological component, because Fortran does not support symbol manipulation well. Work is now beginning on a more sophisticated phonological component, made possible by the extensive symbol manipulation capabilities of PCOMPILER. This work is expected to progress rapidly, since most of the rules are already written in linguistic notation, which can be easily expressed in PCOMPILER.

A speech waveform synthesizer program has been brought over from M.I.T. The synthesizer configuration has been improved recently through experience with speech perception experiments at M.I.T. The latest version includes (a) amplitude controls on the parallel formants that are used to produce better approximations to frication spectra, (b) a pole pair and zero pair in cascade with the cascade formants of the sonorant generator in order to better approximate nasals and the nasalization of vowels, and (c) a new voicing source that is based on the detailed analysis of the voicing source characteristics of one of the authors (DHK). A single period of glottal volume velocity waveform during normal voicing was measured by speaking into a 4 cm diameter tube having an anechoic termination so that no sound was reflected back into the oral cavity. The voicing source in our speech synthesizer reproduces this waveform; fundamental frequency is varied by changing the duration of the closed phased.

C. Prosodics

We have recently implemented Wayne Lea's "syntactic boundary detection" algorithm using a Fortran program (BOUND3) obtained from the UNIVAC speech understanding project. Results from testing it on 16 sentences (by 3 speakers) from our on-line data base show its performance (using threshold values suggested by Lea) to be roughly comparable to the results reported by Lea. There is some question as to whether this level of performance is adequate for use in a speech understanding system. These results are discussed in more detail in Section IID.

D. Syntax

During the past quarter we have begun to use the syntax component with the scoring mechanism that was developed the previous quarter. To test it, a set of 25 one-word theories was formed by choosing one word from each of our 21 syntactic categories and adding a few words with features which make a significant syntactic difference (e.g., "he" and "him"). This set of theories was parsed in two ways: following all possible paths and following only those with the best scores. The number of configurations, transitions, monitors and proposals constructed by the parser was reduced by 25% in the latter case, showing that the scoring mechanism does significantly reduce the number of alternatives which the parser considers.

The format of PUSH arcs in the grammar has been changed to allow three tests in the test component instead of the original two which checked the register settings and the constituent. The new test is a look-ahead test which is performed on the next word of input (if a next word exists) to decide whether to establish a process to look for that constituent. The value of the look-ahead test is NIL if the test fails and a small integer otherwise. The integer is a rough indication of how likely it is that the next word begins a constituent of the desired type, and is added to the score of the initial configuration which is set up to look for that constituent. Thus when pushing for a noun phrase, it is recognized that an adjective or quantifier is more likely to be the first word than a verb, even though constructions like "remaining trips" are allowed.

This look-ahead test can also be used to establish monitors at the end of an island instead of trying to begin to parse the constituent with no information at all and subsequently generating many specific monitors for each type of constituent which could occur. For one typical theory, this method reduced the number of predictions made by Syntax from 103 to 60.

The grammar has not been appreciably changed this quarter.

E. Dictionary

During the past quarter, the dictionary has been expanded from 350 words to 502. Also, during this quarter, we have made some changes to the base form pronunciations of the words to be consistent with a new set of phonological rules and an expanded set of phonemes which is more specific in pnonetic detail than those used previously. For example, we have added phonemes for dipthongs and affricates and have created specialized phonemes for syllabic nasals, unreleased plosives, and unvoiced vowels.

We have made a variety of extensions to the Bobrow-Fraser phonological rule tester programs that we received from SCRL, and have adapted it to perform the expa ion of the base form pronunciations to surface pronunciations for the dictionary. Extensions to the rule tester programs include addition of facilities for applying all combinations of optional rules from an ordered list of rules, for conditional application of rules depending on the success or failure of previous rules, for applying rules successively to words read from a dictionary file and for constructing regularly inflected forms of regular nouns and verbs automatically. A variety of audit trail functions for debugging rule sets have also been added. These latter include keeping records of which rules applied (and how many times and to which words) and, for each word, a record of which rules were applied to it. Details of the extensions to the Bobrow-Fraser package and their use will be described in a subsequent report. Our phonological rule set is currently being debugged using this facility.

Syllable boundaries have also been added to the base form. We will continue collecting words during the next quarter to extend the dictionary to 1000 entries.

F. Semantics

During the past quarter, we worked on extending the scope of the semantic associations used for noticing and proposing semantically related words and concepts. In particular, we implemented the general semantic notion of "property" (i.e., A is a property of B) as another means of identifying sets of word matches which could meaningfully co-occur in an utterance. Using this one notion, we are now able to associate properties and the thing they are properties of, e.g., "location" (the property) and "conference" (e.g., "the location of the conference", "the conference's location") or particular instances of a propert and the thing it is a property of (e.g., "the Pittsburgh conference", "the conference in Pittsburgh").

G. Pragmatics

Work on the Pragmatics component has been proceeding in two major areas: evaluation and execution. ˉ aluation includes completion of an utterance interpretation, scoring of the completed structure, and suggestions to Syntax and Semantics about changes or gaps in the interpretation.

The evaluation portion of Pragmatics is being written in the Augmented Transition Network (ATN) formalism to express the modes of interaction which we are using to model discourse. A node in the ATN represents a temporal location in the discourse and an arc represents a possible action, i.e., an utterance with its associated intention, which takes the discourse to a new state. Transitions are entirely determined by conditions on the arcs. A condition has three parts: syntactic, discourse level, and presuppositional. The discourse part is computed by functions which give the probability of a transition based on the current state and configuration in the ATN. These functions are essentially a reformulation of the function MODE-STATUS [1]. The ATN formalism allows us to isolate much of the bookkeeping of discourse position which formerly had to be done by MODE-STATUS. The presuppositional and the syntactic parts of an arc's condition are computed by special functions for each intention. This replaces the less flexible function INSTANCE-MAP mentioned in [1].

Currently 8 intentions have been at least partially encoded as such special functions. We have identified 12 other intentions which need to be encoded.

Work on the execute portion of Pragmatics has centered on extending the SEMNET package [1] to accommodate various desirable features in network retrieval. These include using variables for items, having a more efficient BOOLFIND (for Boolean retrieval operations), and having incompletely specified retrieval requests satisfied. These features were not important in our previous use of the SEMNET functions because of different task characteristics. We are currently working on developing more general retrieval functions and studying the relationship between retrieval tasks and the types of functions needed. For Pragmatics a special set of retrieval functions has been written. These are discussed in technical note II.E.

Special purpose functions are also being written for factual retrieval in the travel budget management domain. These include functions to estimate the cost of a trip, to calculate its duration, whether or not it is explicitly specified, and to add and retrieve trips and fares. A more complete description of these functions will appear in succeeding QPR's.

H. Control

During the past quarter, we made several improvements to the interfork communication interface between the two LISP forks, the one housing the semantics and control components, the other, the syntactic component. As a result, both the number of required interactions and the time spent in each interaction was cut down drastically.

In addition, we constructed and debugged a preliminary interface with the new lexical retrieval component. Though many improvements are still planned for it, we will now be able to concentrate on developing control strategies with lexical retrieval, syntax and semantics all operating together.

References

[1] Woods et al. (1975)
Speech Understanding Systems. Quarterly Technical Progress Report No. 1, BBN Report No. 3018, Bolt Beranek and Newman Inc., Cambridge, Massachusetts 02138.

[2] Klatt, Dennis H. (1975)
"Word Verification in a Speech Understanding System", BBN Report No. 3082, Bolt Beranek and Newman Inc., Cambridge, Massachusetts 02138.

[3] Bobrow, D.G. and Fraser, J.B. (1968)
"A Phonological Rule Tester", CACM Vol. 11, No. 11, pp. 766-772.

II. TECHNICAL NOTES

A. Acoustic-Phonetic Research

Richard Schwartz

1. Dip Detection and Segmentation

We have written a subroutine which searches a time function for different kinds of dips. This dip detection routine can produce a list of weighted boundaries of different types, for any time-varying energy function. A preliminary segmentation program combines the lists of boundaries (corresponding to energy measures of different regions of the spectrum), along with knowledge of durational constraints, to form a very rough segment lattice. We are using, in particular, three regions of the spectrum for this initial phase of the segmentation. The energy between 120-440 Hz is used to separate an utterance into sonorant sequences and obstruent sequences. Within sonorant sequences, the energy in the mid frequencies (roughly 500-2700 Hz) is used to separate vowels from nasals and glides. Within obstruent sequences, energy in the high frequencies (3400-5000 Hz) is used to separate strident fricatives from silences and weak fricatives. Using an energy threshold, silences are separated from weak fricatives. Some pairs of silences and frication periods are combined into unvoiced fricatives using durational constraints. Also some are identified as retroflexed unvoiced plosives. Flaps and Schwa are also identified based on duration.

This rough segmentation program currently distinguishes 12 classes of sounds. These are: Vowels and glides, Schwa, Sonorants (nasals), Intervocalic sonorants (nasals and glides), Intervocalic obstruents (V,DH,HH,DX and sometimes unvoiced plosives), Flaps, Unclassified obstruents, Fricatives, Strident Fricatives, Plosives, Unvoiced Plosives, and Retroflexed unvoiced plosives. These classes are clearly overlapping and are defined acoustically rather than phonetically. For example, many segments identified as "VOWEL" contain several vowels and semivowels. This is only the initial segmentation, however, and these will later be separated using formant motion and targets. This initial phase of the segmentation will be used to guide the rest of the APR program to make finer distinctions.

In this initial phase of segmentation, there are very few optional paths in the segment lattice, since most of the acoustic cues used to make decisions are robust. We have examined rough segment lattices for 39 utterances. In one out of every 2 utterances there is one optional segment. Only 3% (or less than one per utterance) of the non-optional boundaries are in error.

2. Formant Tracking Methods

Since our previous formant tracker was inadequate in that it made several tracking errors during vowels and glides, we have been investigating alternate methods. In the past quarter, we have looked at two. The first was a modification of the algorithm used by Stephanie McCandless of Lincoln Laboratories [1] in which, instead of "enhancing" to find extra formants, we solve for the roots of the equation, to derive all the poles at once. This is faster when a signal processor is not available, and also yields all the necessary information at once. In the second method, the poles of the preemphasized spectrum are examined, and the three poles with the narrowest bandwidths are picked as formants. There are two advantages to this method: first, it is simpler in that it does not really use continuity constraints to the same degree. Second, it does not require preliminary segmentation (as does the first method), and hence, will not make errors due to segmentation errors. Of course, the formants computed during obstruents are not reliable, but they would not be used anyway. Each of the two methods makes about one error per 3 second utterance, so it is not clear at the present which one is preferable. We will be using the second until there is more time to investigate the matter in more detail.

3. Formant Smoothing Algorithm

There are two reasons for smoothing the formants after they have been computed: first, intelligent smoothing can correct errors in the original tracking. Second, small irregularities in the tracks increase the complexity of algorithms which examine them and can usually be eliminated with no loss in information. To accomplish the smoothing, we are using a 3-point median smoothing procedure [2,3]. Rather than using a 3-point Hanning window with coefficients 1/4-1/2-1/4 which we felt destroyed too much of the information in the transitions, we are processing the median-smoothed formants with coefficients of 1/8-3/4-1/8. Since only small irregularities remain after this median smoothing, and the transitions remain intact, we are satisfied with this method.

References

[1] McCandless, Stephanie (1974)
"An Algorithm for Automatic Formant Extraction Using Linear Prediction Spectra", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-22, no. 2, April 1974, pp. 135-141.

[2] Tukey, J. W. (1974)
"Nonlinear (Nonsuperposable) Methods for Smoothing data," 1974 EASCON Record, p. 673.

[3] Rabiner, L. R., Sambur, M.R., and Schmidt, C.E. (1975)
"Applications of Nonlinear Smoothing to Speech Processing," 89th Meeting: Acoustical Soc. of America, April, 1975.

B. Speaker Normalization

John Makhoul

Thus far in our project we have employed one speaker normalization scheme developed by Richard Schwartz [1], that of using the average fundamental frequency to normalize for vowel formant frequency. This normalization was developed using the Peterson-Barney vowel data, which consisted of data from men, women and children. When tested against this whole data corpus, the normalization scheme achieved 91% correct recognition on first choice, and 98.5% on first or second choice [1]. When this technique was applied to vowels in continuous speech of male speakers (in the Nov. 1973 system), we achieved 50% correct recognition of 15 vowels and glides on first choice, and 90% on first and second choice.

In order to enrich our repertoire of tools for speaker normalization, we have implemented an algorithm that estimates the instantaneous vocal tract length of the speaker for each vowel frame. The algorithm is based on the works of Wakita [2] and Paige and Zue [3]. Briefly, the procedure is as follows:

1. Compute the frequencies and bandwidths for the first three formants.
2. By assuming some sampling frequency F_s (greater than twice the third formant), compute the corresponding poles in the z plane.

3. Multiply out the pole factors to form the predictor polynomial.

4. From the predictor coefficients compute the reflection coefficients and then the log area function.

5. Find the variance of the log area function.

6. Change F_s and repeat steps 2-5 until the log area variance is minimum.

7. The vocal tract length is then obtained from the relation $L=CN/F_{opt}$, where C is the velocity of sound in air, N is the number of formants (N=3 in our case), and F_{opt} is the sampling frequency F_s that minimizes the log area variance.

Our experience has been that the plot of log area variance vs. F_s has a broad minimum, which makes the determination of F_{opt} (and hence L) a sensitive procedure. For example, the minimum seems to be very sensitive to changes in formant bandwidths. We shall experiment further to determine the usefulness of this procedure in the task of speaker normalization.

References

[1] Schwartz, Richard (1971)
"Automatic Normalization for Recognition of Vowels of All Speakers", S.B. Thesis, MIT, May 1971.

[2] Wakita, Hisashi (1975)
"An approach to vowel normalization," J. Acoust. Soc. Am., Vol. 57, Supplement No. 1, S3, Spring 1975.

[3] Zue, Victor, and A. Paige (1970)
"Computation of vocal tract area functions," IEEE Trans. Audio Electroacoust., Vol. AU-18, pp. 7-18, March 1970.

C. PCOMPILER -- A Language for Stating Phonological and Phonetic Rules

Dennis Klatt
Craig Cook
William Woods

During the past quarter, we have developed and implemented a language and a compiler called PCOMPILER for expressing acoustic-phonetic rules in the synthesis phase of the word verification component. The compiler is written in INTERLISP, and translates PCOMPILER source code into efficiently operating FORTRAN code.

The language and its advantages are best described through the use of several examples. The examples indicate minor extensions to conventional Fortran (Examples 1-4), a new method for defining data array values (Example 5), the format for defining legal input symbols, categorization features and input symbol definitions in terms of these categorization features (Example 6), and the syntax of phonetic and phonological rule statements (Examples 7-8). The statement of conditions under which phonetic and phonological rules apply is nearly identical to the Bobrow-Fraser (1968) notation. However, any resultant changes that are to be made to the input string or to aspects of the control parameter information that forms the output of the phonetic component must be written in Fortran.

1) Any ordinary Fortran statement is recognized as such and is left unchanged by PCOMPILER.

2) More that one Fortran statement may appear on a single line through the use of the ";" separator. E.g.:
```
          M=N; X=Y
```
would be transformed into the Fortran statements:
```
               M=N
               X=Y
```

3) The indentation of Fortran DO and IF statements implies the scope of the statement if no statement number is provided. Thus, the statements
```
          DO N=1,3
               M=N
          X=Y
```
would be transformed into the Fortran statements
```
               DO 10001 N=1,3
               M=N
      10001    CONTINUE
               X=Y
```
and the statements
```
          IF (M.EQ.N)
               M=K
               N=L
          X=Y
```
would be transformed into
```
               IF (.NOT.M.EQ.N) GO TO 10002
               M=K
               N=L
      10002    CONTINUE
               X=Y
```

4) Embedding by additional levels of indentation is permitted up to the length of a single line.

5) The user program is divided into blocks, each of which commences with a ":" in Column 1. A program normally begins with a :DATA command which indicates that a block of Fortran DATA and DIMENSION statements are to be created. DATA and DIMENSION statements may use variable names as arguments as long as these names have been given values via a previous statement. A DATA statement has a simplified syntax. The format improves readability of array values, and simplifies changing array sizes and values. E.g.
```
      :DATA
          PHSIZE=5
          DIMENSION F1TAR(PHSIZE)
          IY=4
          F1TAR(IY)=320
```

```
    :END
```

would be transformed into the Fortran statements:

```
        DATA PHSIZE/5/
        DIMENSION F1TAR(5)
        DATA IY/4/
        DATA F1TAR(4)/320/
```

The end of the dimension and data statements is indicated by a :END statement.

6) All input symbols to the phonological and phonetic components (phonemes, syntactic markers, semantic markers and phonetic segments) are defined in terms of binary features. PCOMPILER provides a convenient notation for defining input symbols and features:

```
    :PHONEMES=(IY,IH,WBOUND,STR1)
    :FEATURES=(SEG,VOWEL,FRONT,HIGH,LAX,STRESS,SYNTAX)
    [IY]=(SEG,VOWEL,FRONT,HIGH)
    [IH]=(SEG,VOWEL,FRONT,HIGH,LAX)
    [WBOUND]=(SYNTAX)
    [STR1]=(STRESS)
    :END
```

would be transformed into the Fortran statements:

```
        DATA IY/1/
        DATA IH/2/
        DATA WBOUND/3/
        DATA STR1/4/
        DATA SEG/1/
        DATA VOWEL/2/
        DATA FRONT/4/
        DATA HIGH/8/
        DATA LAX/16/
        DATA STRESS/32/
        DATA SYNTAX/64/
        DIMENSION FMTRX1(4)
        DATA FMTRX1(1)/15/
        DATA FMTRX1(2)/31/
        DATA FMTRX1(3)/64/
        DATA FMTRX1(4)/32/
```

(where FMTRX1 is a matrix which is created to hold the feature assignments for the phonemes.) PCOMPILER assigns unique sequential numbers to the phonemes (segments and markers). The binary features which define a given phoneme are stored bitwise in the feature matrix entry pointed to by this unique number. (PCOMPILER assigns a specific bit position in the feature matrix entry for each binary feature.) Taking [IY] as an example, its unique number is 1, which means its feature matrix entry is FMTRX1(1). PCOMPILER has also assigned unique powers of two (i.e., bit positions) to each of the binary features, and these values are summed for the features which are true of a given phoneme to give its feature matrix entry. For [IY] as above, this entry is equal to 15 which is 1+2+4+8 or SEG+VOWEL+FRONT+HIGH. For a PDP-10, there can be up to 36 features per entry. If more than that number are required, PCOMPILER will generate additional feature matrices (FMTRX2, FMTRX3, etc.) as needed and keep track of relevant bookkeeping.

7) The general format for the specification of phonetic and phonological rules in PCOMPILER takes the form of a left-hand side and context specification (specifying the conditions of applicability of the rule) followed by an indented sequence of FORTRAN statements which are to be executed if the rule conditions are satisfied. The general format of the rule conditions is:

```
      X / Y ...Z
```

where X, Y, and Z are either bracketed input symbols, parenthesized feature lists, or logical combinations of symbols and feature lists, X represents a condition on the current phoneme. (The rules are applied within a global loop which steps the current phoneme across the utterance from left to right.) Y and Z are conditions on the left and right contexts, specifying the environment in which the current phoneme must be located in order for the rule to apply. The position of the current phoneme with respect to the left and right contexts is designated by "...". Any two of X, Y and Z can be deleted in a conditional statement. E.g.:

```
         [IY]
             M=N
         K=L
```

would be transformed into the statements:

```
             IF (PHOCUR.NE.IY) GO TO 10000
             M=N
     10000   CONTINUE
             K=L
```

where PHOCUR is a reserved variable which designates the current input symbol being processed. This is simply a test for the presence of IY at the current location in the input string.

Other reserved variables include PHONEX and PHOLAS, which designate the input symbols to the immediate right and immediate left of the current input symbol (PHOCUR). where the user refers to input symbols beyond these two, PCOMPILER reserves the variables PHOCP2, PHOCP3, ... for input symbols located two, three, ... positions to the right and PHOCM2, PHOCM3, ... for input symbols successively located to the left.

If features are used, the following conventions apply. Two features separated by a space imply the logical AND of both conditions. Logical OR must be stated explicitly. Unfilled parentheses indicate that any input symbol satisfies the constraint on that place in the input string. Examples follow:

```
          /...(+FRONT -LAX)
              M=N
          K=L
```

would be transformed into the statements:

```
              IF ((LAND(FMTRXi(PHOCP1),FRONT).NE.0).OR.
             1(LAND(FMTRXj(PHOCP1),LAX).EQ.0)) GO TO 10000
              M=N
     10000    CONTINUE
              K=L
```

where the suffixes "i" and "j" would be set to the appropriate integers. The machine-language function "LAND" returns zero if the logical "and" of the bits in its two arguments is zero, and it returns true otherwise. It is up to the programmer to see that the current input symbol, PHOCUR, and the next input symbol, PHOCP1, are set to the appropriate values.

Another example:

```
          /((-VOICED) OR (+STOP))()...
              M=N
          K=L
```

This would be transformed into the Fortran statements:

```
              IF (LAND(FMTRXi(PHOCM2),VOICED).NE.0) GO TO 10001
              IF (LAND(FMTRXj(PHOCM2),STOP).EQ.0) GO TO 10001
              M=N
     10001    CONTINUE
              K=L
```

Again, the programmer must see that PHOCM2 is set to the appropriate value.

8) In the phonological component, where the input string appears in the array INPUT(NPHON), and NPHON ranges from 1 to NPMAX (maximum length of input string expressed in number of segments), it is possible to write a more-complex phonological

```
          /...(# 0 9 (-SYL))(+WBOUND)
              M=N
          K=L
```

This condition says "look for a right context, having the feature +WBOUND (word boundary). Quit (i.e. do not apply the rule(s)) if you encounter a symbol marked +SYL(syllabic) or if you process ten symbols beforehand". The Fortran code that would be generated by PCOMPILER is:

```
               NPX1=NPHON+1+0
               IF (NPX1.GT.NPMAX) GO TO 10003
               NPX2=NPHON+1+9
               IF (NPX2.GT.NPMAX) NPX2=NPMAX
               DO 10001 NPX=NPX1,NPX2
               IF (LAND (FMTRXi(INPUT(NPX)),WBOUND).NE.0)
              1GO TO 10002
               IF (LAND(FMTRXj(INPUT(NPX)),SYL).NE.0)
              1GO TO 10003
       10001   CONTINUE
               GO TO 10003
       10002   CONTINUE
               M=N
       10003   CONTINUE
               K=L
```

where NPX1 and NPX2 define the left and right-hand limits of that portion of the input string to be examined. As can be seen, the Fortran code takes care of the tests for the physical end of the input string automatically.

Conclusion: The current status of the PCOMPILER program is evolving, and new features are still being added. It is being evolved simultaneously with the construction of the synthesis program for the word verifier so that the features which are provided are tuned to particular real needs. The above examples should illustrate the degree to which the expressions in the PCOMPILER notation are more readable and convenient to work with than the corresponding expressions in the target FORTRAN.

D. Prosodics

Lyn Bates
Jerry Wolf

One source of knowledge available to speech understanding systems is the interpretation of the suprasegmental information contained in the fundamental frequency contour of a sentence. Lea's earlier research [1,2,3] showed that a decrease in fundamental frequency usually occurs at the end of each major syntactic constituent, with an increase usually near the beginning of the next one. He proposed an algorithm for "detecting" syntactic boundaries by recognizing this fall-rise pattern in F0. Phonetic effects (especially unvoiced consonants) can also cause such a fall-rise pattern, but the effect is generally somewhat smaller, so they can be screened out by requiring that an F0 decrease exceed a "fall threshold" and that an F0 increase exceed a "rise threshold" in order for a boundary to be recognized.

Lea's algorithm marks the detected syntactic boundary at the end of the fall in F0. This does not in general place the boundary precisely at the end of the constituent. When the following constituent begins "weakly" (with unstressed or reduced syllables), the F0 valley bottom may occur within that weak beginning. Also, when a previous constituent exhibits a "Tune II" intonation contour (having a small rise at the end), the F0 valley bottom may occur before the end of the constituent. For this reason, Lea describes his algorithm as "boundary detection", not "boundary location". This uncertainty in the boundary position is a potential drawback for using these boundaries in speech understanding systems.

Boundaries that have prosodic acoustic correlates do not correspond precisely to those boundaries that a linguist would pick on purely syntactic grounds. Lea defines syntactic boundaries to be of two types: major and minor. Major syntactic boundaries correspond to some of the generally accepted linguistic constituents; they occur (1) before a prepositional phrase (PP); (2) before a noun phrase (NP) unless the NP follows a preposition or conjunction, or is a single pronoun; (3) either before or after a conjunction; or (4) before a complement construction (e.g., I want _to_ _go_ _away_).

Minor syntactic boundaries occur within sequences which form constituents: (1) between a NP and a main verb (not an auxiliary); (2) between nouns in a noun-noun modifier (e.g., "summer trip", "Pittsburgh conference"); (3) between two adjectives which modify the same noun (e.g., "the recent expensive trips"); (4) after a quantifier (e.g., each, every, some, most, all); (5) between a participle and the noun it modifies (e.g., "oxidizing agent", "estimated cost").

The most direct reason for attempting to locate major syntactic boundaries in a speech understanding system like SPEECHLIS is to help the parser decide if constituent boundaries occur at certain positions in the theory it is considering (or more accurately, to modify the ordering of parse paths using this acoustic information). Another use stems from the fact that the UNIVAC group's stressed-syllable detection program [3,4] (as yet untried here at BBN) requires as one of its inputs the boundary positions found by the boundary detector program. Syllable stress should be useful throughout the front end of SPEECHLIS for locating the parts of the utterance where the segmental information is most reliable, and for providing stress information which the word matcher can compare with the stress markings of dictionary pronunciations.

The conversion of the UNIVAC source program BOUND3 to use in our system was straightforward, primarily converting its top level "main program," which read input data from cards, to a subroutine which receives its inputs from arguments and which (optionally) writes its results in a file format consistent with the rest of the system and in a "debug listing" file.

The initial result of running BOUND3 on F0 data from some existing sentences was to drive home the fact that our current fundamental frequency extraction routine makes too many errors for its output to be usable by BOUND3. (The F0 extraction routine uses a center-clipped autocorrelation method [5], which we adopted after learning of the UNIVAC group's satisfaction with the method [2]. Most of the errors consisted of being too liberal in accepting a frame as voiced, so some unvoiced frames following voiced intervals were called voiced, and the murmur in the stopgap of voiced stops sometimes gave bad values of F0. Rather than fight the battle of "tuning" the F0 extractor at this point, we elected to hand-edit some F0 data, based on examination of the speech waveform for testing purposes. This was done for 16 sentences by 3 speakers. The resulting output of the boundary detection routine was much more in keeping with results reported by Lea.

Some results of the program are shown in Figures 1 and 2. These results were obtained using fall and rise thresholds of 5 eighth-tones (about 7.5% change), as recommended by Lea. The following information is shown in these figures, starting at the bottom.

1. Time scale.
2. The manual transcription. (Unfortunately, some of the segment labels are illegible, due to the compression of the time axis necessary to plot the data on the page. The word labels above the line should suffice.) Major syntactic boundaries are marked below the transcription with solid lines; minor ones are marked with broken lines.
3. "bnd", the boundary detector output. Confidence numbers are given for the boundaries found by the

program, which we will refer to as "Lea-boundaries," to distinguish them from the surface syntactic boundaries of the sentence.

4. "FOXT", the hand-edited F0 data, converted to eighth-tones, used as the input to the boundary detection program.

5. "RC", the energy.

6. "FOX", a superposition of F0, the original fundamental frequency data (in Hz) and FOX, the edited data. This is shown merely to display the changes due to the hand-editing.

In Figure 1 (sentence DWD115), we see that all five of the major syntactic boundaries have Lea-boundaries marked by the program. In each case, the location of the Lea-boundary is slightly after the start of the actual constituent, falling in the interval between the start of the constituent and its first stressed syllable, as described above. The Lea-boundary at t=2.84 seconds does not correspond to an actual syntactic boundary. The F0 data in the region of the last syllable shows considerable irregularity, indicating vocal fry during the last syllable as the speaker lets his vocal effort die away. We notice this in our data frequently, and to avoid bad boundaries we have adopted the rule to ignore any boundary found in the final syllable of the utterance. (This rule, or an approximation to it, can be implemented as an algorithm to filter out such boundaries.)

Note also that the boundaries marked by the program carry confidence numbers. The confidence number is a function of the extent of the F0 rise after the boundary, the duration of the rise, and, if an unvoiced interval intervenes, whether F0 rises or falls on the other side. Lea has suggested that boundaries having a confidence number of less than 30 be rejected. This rule would also reject the spurious boundary at t=2.84.

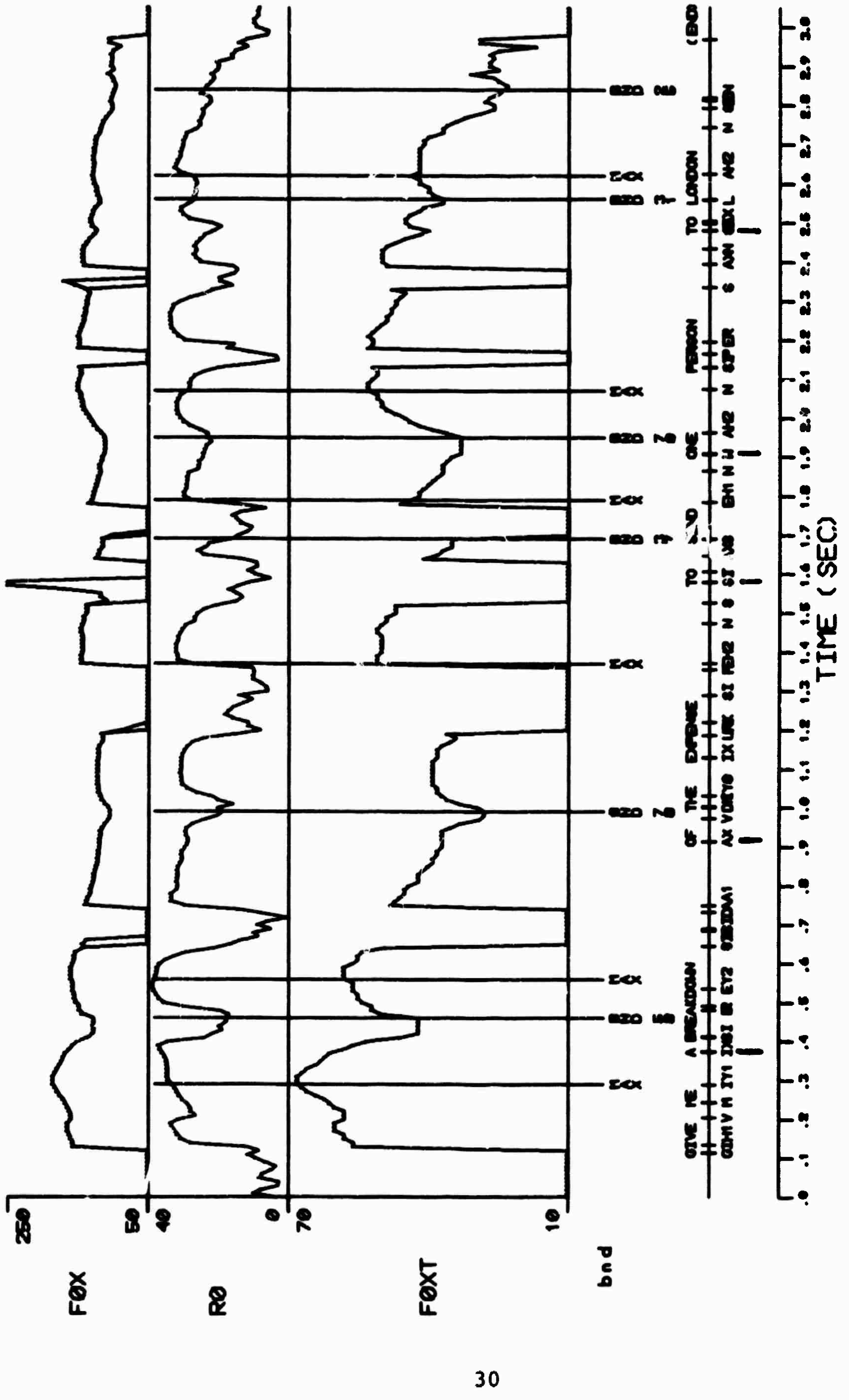

Figure 1. Sentence DWD115, "Give me a breakdown of the expense to send one person to London," showing (from the bottom) the manual transcription, the BOUND3 boundaries, the edited F0 contour (in eighth-tones), the energy, and the unedited F0 contour.

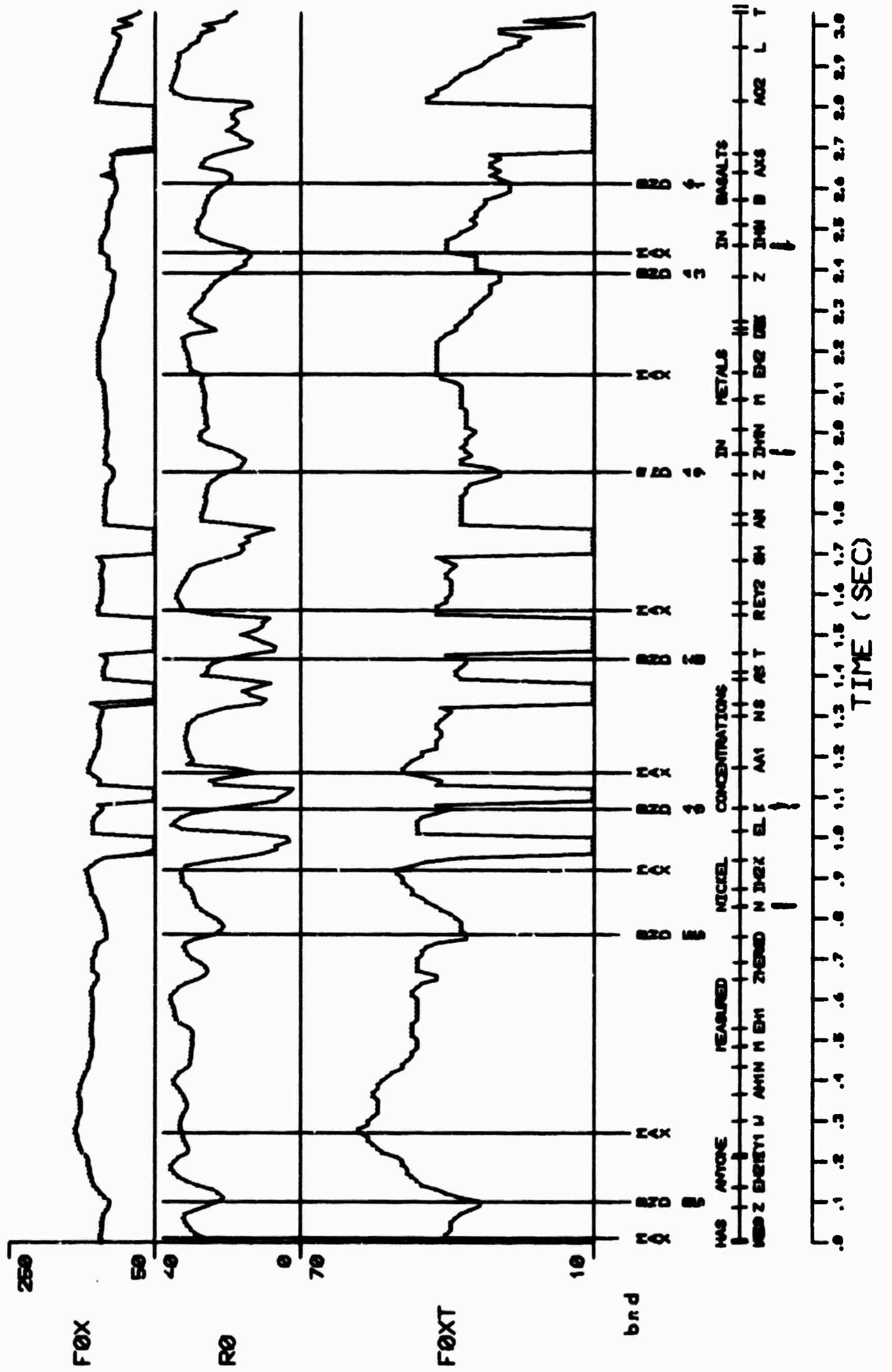

Figure 2. Sentence DWD30, "Has anyone measured nickel concentrations in metals in basalts?"

Figure 2 (sentence DWD30) was chosen from our set of sixteen to illustrate a case where the Lea-boundaries are more difficult to interpret.

Time	Comment
0.10 sec.	Has a high confidence, yet appears to correspond to no syntactic boundary. (Although "anyone" is a noun phrase, Lea does not expect pronoun noun phrases to be detected. However, "anyone" is somewhat longer (and more noun-like) than most pronouns.) More probably, the F0 dip may be a phonetic effect of the /z/.
0.76	Probably due to the boundary between "measured" and "nickel". Occurs <u>before</u> the actual boundary, probably due to the phonetic effect of the /d/ pulling F0 down.
1.07	Perhaps due to the minor syntactic boundary between "nickel" and "concentrations".
1.44	Confidence less than 30, so rejectable.
1.90	Due to the boundary before "in metals". Presumably the dip in F0 just there is due to the preceding /z/.
2.39 2.61	There is certainly a syntactic boundary between "in metals" and "in basalts". Whichever way you decide to assign the two boundaries found by the program, one of them is spurious.

In evaluating the results of this experiment from a syntactic point of view, two problems emerge. The first is how to measure the success of the BOUND3 algorithm, the second is how to determine the usefulness of the results.

For each of the 16 sentences tested, the ideal boundaries were identified by hand in the surface string. A

Lea-boundary was counted as detecting an actual boundary if it occurred within the range specified in the second paragraph of this section. The table below summarizes the results (a) for all the Lea-boundaries and (b) excluding Lea-boundaries that were below the recommended confidence level of 30 or that were in the last syllable of the utterance. There were 51 major syntactic boundaries and 14 minor ones in the 16 sentences.

	Lea Boundaries	Major Found	Minor Found	False
before cutoff	86	47(92%)	9(64%)	30(35%)
after cutoff	62	42(82%)	5(36%)	15(23%)

Initially, these results seem very pleasing, but they must be evaluated in terms of the help they could give to a speech understanding system. It is one thing to know the ideal boundaries (and their types) in an utterance and then observe the ability of the BOUND3 algorithm to detect them; it is quite another thing to be faced with an unknown utterance and a set of Lea-boundaries and then endeavor to use them for guidance.

Let us take a hypothetical example and see what information we can gain from assumed Lea-boundaries. Suppose that for some portion of an utterance the word "summer" has been suggested by the lexical retrieval and match routines and that a Lea-boundary with score 50 occurs a few segments later. The score given to it is not a reliable indicator of whether it is a major or a minor boundary, since the major boundaries that were detected had scores ranging from 16 to 88 with a mean of 49.5 (using the cutoff criterion described above the range was 31 to 88 with a mean of 52.7) and the minor boundaries had scores from 25 to 61 with a mean of 37.5 (after cutoff, 40-61, mean = 49.6).

Thus, although we have an indication that there's some sort of boundary after the word "summer", there is no indication as to whether it is a noun-noun modifier boundary ("summer trip"), a prepositional phrase boundary, a noun-verb boundary ("summer means warm weather"), a conjunction boundary ("summer and winter"), or a clause boundary ("I want summer to come"). In fact, using the current BBN SPEECHGRAMMAR the only things which could occur after the word "summer" which should not cause a Lea-boundary are auxiliary verbs ("summer doesn't come early") and participial modifiers ("summer fishing trips" -- although it is possible that the noun and participle would act enough like adjectives to cause a boundary between them, and adverbs ("summer quickly faded"). The detection of a boundary here does not help much in reducing the syntactic alternatives within the parser.

Similarly, if the Lea-boundary were near the beginning of the word "summer", it could indicate that the word is preceded by a preposition, a quantifier ("every summer"), a participle ("A swinging summer"), a noun ("It was a watermelon summer"), a conjunction, a verb ("I love summer"), an article, or virtually anything which would precede a noun phrase. Again, this eliminates only a few of the syntactic possibilities (such as an adjective or question word) for a predecessor.

A particular piece of syntactic information that would be very useful in reducing the number of ambiguous parsings produced by the syntactic component is whether a prepositional phrase modifies the noun preceding it or whether it modifies some previous element. ("I shot a bird in the wing" vs. "I shot a bird in the tree"). Unfortunately there does not seem to be any prosodic difference that is detectable by the BOUND3 algorithm, for although it detected 22 out of 23 prepositional phrases in the 16 utterances, there was no difference in the scores that could be used to help determine the correct placement of the modifier.

However, the fact that BOUND3 reliably detects prepositional phrase boundaries can make it useful to syntax, particularly in verification mode. Since prepositions are frequently hard to recognize acoustically and since they can precede almost any noun phrase, it is very easy to create numerous prepositional phrases; the absence of a Lea-boundary would be a reliable cue to indicate that a preposition is spurious.

We have run BOUND3 on this same set of 16 utterances, independently varying the rise and fall thresholds among 5, 6, and 7 eighth-tones (changes of 7.5%, 9.1%, and 10.6% respectively) to see if we could cut down the number of false boundaries without losing too many genuine ones. The results have not been tabulated in time to be included in this report.

In the near future, we will be investigating the shortcomings of our F0 extraction routine and improving it so that BOUND3 can be run directly on the output data.

It appears to us that one of the principal problems with boundaries found by the BOUND3 algorithm is the lack of definite guidelines for delineating the region of the utterance in which the constituent boundary "detected" by the program actually lies. That is, given a Lea-boundary found by the algorithm, what additional segmental and supersegmental information must be brought to bear to delineate the region around the Lea-boundary in which the constituent boundary must lie, and how narrow can such a region be? Some kind of answer to this question is needed before the detection of Lea-boundaries can be incorporated into a parsing strategy. We plan to pursue this question with the help of the UNIVAC group.

References

[1] Lea, W.A. (1972) "Intonational Cues to the Constituent Structure and Phonemics of Spoken English," Ph.D. dissertation, Purdue University, Lafayette, Indiana.

[2] Lea, W.A. (1973) "An Approach to Syntactic Recognition without Phonemics," IEEE Trans. Audio Electroacoustics AU-21, pages 249-258.

[3] Lea, W.A., Medress, M.F. and Skinner, T.E. (1973) "Prosodic Aids to Speech Recognition: II. Syntactic Segmentation and Stressed Syllable Location," Report No. PX10232, Sperry Univac, St. Paul, Minn.

[4] Lea, W.A., Medress, M.F. and Skinner, T.E. (1973) "Prosodic Aids to Speech Recognition: II. Relationships between Stress and Phonemic Recognition," Report No. PX10430, Sperry Univac, St. Paul, Minn.

[5] Sondhi, M.M. (1968) "New Methods of Pitch Extraction," IEEE Trans. Audio , AU-16, pp. 262-266.

E. Retrieving Information From The Net

Bertram C. Bruce

1. Introduction

The retrieval problem for a semantic network can be stated as follows: Take an intensional characterization of a class of items in the network and retrieve a list of those items in an efficient manner. The characterization need not be minimal, nor exhaustive. In fact we would expect various, more or less detailed, characterizations to produce identical sets of items.

In the speech system we are using the network formalism, SEMNET [1] in both the Semantics and the Pragmatics components. Generally speaking, the basic functions of SEMNET can be used in both components. However, most of SEMNET's current development has been directed towards building networks rather than towards performing complex types of retrievals. For example, in the Semantics component, retrieval usually means following predetermined paths through the net to determine possible relationships between words. But in the Pragmatics component the typical task is to retrieve factual information, such as elements of a travel report. Rather than following paths, the task is to match patterns which may be over- or under-specified. This different orientation towards retrieval has necessitated modifications and additions to the existing retrieval functions.

Work is currently underway to extend SEMNET with a general set of retrieval functions which can be used for any purpose. Until such a package exists we will be using an outgrowth of the BOOLFIND and IFIND functions of SEMNET for factual retrieval operations. The modified retrieval function is called PFIND. This technical note is a description of PFIND and associated functions. While the set of functions is not complete, they do facilitate most of the basic retrieval operations.

A retrieval program should allow the description it is given to be as unconstrained as is possible. In the Pragmatics component the primary retrieval function (PFIND) allows descriptions equivalent to the following requests expressed in English:

> All items with a HAS/AS/PART link to SKIN and an ISA link to BODYPART.
>
> All items with either an AGENT link to JOHN or an OBJECT link to MARY (except those which also have a DATIVE link to SUSAN)
>
> All items with an ISA link to TRIP and a COST property whose value is greater than $500.
>
> All items with a CONNECTED/TO link to any item with an ISA link to COASTAL/NATION and a POPULATION property whose value is less than 1,000,000.
>
> The set of items which match on at least half of a list of descriptors.

Essentially PFIND is a function which produces a list of items which match a description. The description may be simple, such as "has a PART/OF link to TREE" or quite complex, involving lists of items, Boolean operators, property checking, and evaluating for the best partial match.

PFIND is designed to be general enough for most retrieval operations, whether they be on relations or properties, or on one or more items. Simple retrieval requests can be stated in an obvious, straightforward way. More complex requests are handled fairly efficiently, taking advantage of arrays and ordered storage of items.

In the next section the PFIND program is examined at three levels: first, in its basic form in which it will find all items which have stated relation/item links; second, at an intermediate level, where (among other things) embedded calls to PFIND are allowed; and third, in its full form which encompasses Boolean operators, "fuzzy" calculations, and property checking. This modular description is given both to clarify the presentation and to isolate various aspects of PFIND which are subject to change. Section 3 is a discussion of various functions for use with PFIND which allow the Boolean operations and property checking. Section 4 discusses the FUZZYFIND program, which is a first approximation to a general procedure to do "fuzzy" calculations. Section 5 discusses some implementation issues and use of PFIND, specifically interpretation of literals and keywords. Section 6 is a list of some needed extensions to PFIND.

2. PFIND

a. Basic PFIND

The simplest way to think of PFIND is to consider it as a function which takes a list of relation/item pairs and returns a list of items which have all the specified links. For example, to find all Chinese restaurants which serve brunch one might say,

```
(PFIND (ISA CHINESE/RESTAURANT)
       (SERVES BRUNCH))
```

This will return a list of all items which have an ISA link to the item whose PNAME is CHINESE/RESTAURANT and also a SERVES link to the item whose PNAME is BRUNCH.

Note that PFIND is an NLAMBDA no-spread function. Thus it takes an indefinite number of non-evaluated arguments. The item can be specified by either its PNAME, as in the example above, or by its item number (array index). For example, the call,

```
(PFIND (ISA  7)
       (SERVES BRUNCH)
       (LOCATION  34))
```

would retrieve all items with an ISA link to item 7, a SERVES link to the item whose PNAME is BRUNCH, and a LOCATION link to item 34.

We can summarize basic PFIND as follows: PFIND takes as arguments one or more descriptors. Each descriptor is a relation/item pair, where relations are indicated by their PNAMES and items are indicated by either their PNAMES or their numbers. Thus,

```
<call-to-PFIND> ::: (PFIND <descriptor>+)
<descriptor>    ::: ((<relation-PNAME> <item-spec>)
<item-spec>     :::  <item-PNAME> ¦ <item-number>
```

PFIND first finds the list of items for each descriptor by following the inverse relation from the item specified in the descriptor. Then it does an intersection of the lists. The intersection is made efficient by the fact that all the lists are ordered, i.e. the basic SEMNET storage algorithms maintain sorted lists of items.

b. PFIND with Item Lists

An important generalization to basic PFIND is to allow lists of items in the descriptors, or, more precisely, to allow (non-atomic) forms which evaluate to (ordered) lists of item numbers. Suppose, for instance, that one wanted to find all items linked by KINDS to any item in the list generated by (FOO X). This can be done by:

```
(PFIND (KINDS (FOO X))
```

When a non-atom appears as the second element of a descriptor, PFIND evaluates it, then finds all items which are linked via the relation to any element of the item list, and finally, performs a union of the items found.

Any non-atomic LISP form is allowed as the second element of a descriptor. In particular, there can be a call to PFIND. For example, suppose one wanted a list of all small cities located near a major river. This might appear in a call to PFIND as,

```
(PFIND  (ISA CITY)
        (SIZE SMALL)
        (NEAR/TO (PFIND (ISA RIVER)
                        (IMPORTANCE MAJOR))))
```

Calls to PFIND (or other functions) can be embedded to any level. To find all small cities located near major rivers which empty into warm oceans infested with pirates, one could say,

```
(PFIND (ISA CITY)(SIZE SMALL)
    (NEAR/TO
      (PFIND (ISA RIVER)(IMPORTANCE MAJOR)
        (EMPTY/INTO
          (PFIND (ISA OCEAN)(TEMP WARM)
            (INFESTED/WITH PIRATES))))))
```

c. Generalized PFIND

A second generalization of PFIND makes possible the inclusion of Boolean operators and other functions in calls to PFIND. Arguments to PFIND can be either descriptors as discussed above, or (non-atomic) forms which evaluate to (ordered) lists of items. As described above, PFIND simply performs an intersection on its lists of items. However these lists may arise either from descriptors or from arbitrary forms.

There is a set of functions associated with PFIND which operate on lists of lists of items and return lists of items (see Section 3). In a sense these merely provide an assortment of useful retrieval functions. But they can also be viewed as fundamental extensions to PFIND. By embedding such functions in a call to PFIND one can specify a retrieval operation by an arbitrary Boolean combination of descriptors. Basic PFIND, on the other hand, has only implicit conjunction.

The generalized description of a call to PFIND is as follows:

```
<call-to-PFIND> ::: (PFIND <arg>+)
          <arg> :::   <form> ¦ <descriptor>
   <descriptor> ::: (<relation-PNAME> <item-spec>)
    <item-spec> :::     <item-PNAME>¦<item-number>
                            ¦<form>
```

where <form> is a non-atomic LISP expression which evaluates to an ordered list of item-numbers. The next section describes some current functions which return ordered lists of item numbers and are thus appropriate to appear in <form>.

3. Functions for Use With PFIND

a. Boolean Operators

There are three functions which can be used with PFIND to allow arbitrary Boolean combinations of descriptors. These are IAND, IOR, and ISDIFF for conjunction, disjunction, and set difference, respectively. Together they provide a complete set of Boolean operators.

For example, suppose one wanted a list of all cars which are either coupes or station wagons, and, are either red or old, but not black. Depending on how the English is parsed one might make the call to PFIND as,

```
(PFIND (IAND (ISA CAR)
             (IOR (ISA COUPE)
                  (ISA STATION/WAGON))
             (ISDIFF (IOR (COLOR RED)
                          (AGE OLD))
                     (COLOR BLACK))))
```

(Actually, the call to PFIND is unnecessary; IAND can be called directly. The reason why one might still make the call as shown above is that conceptually IAND, IOR and ISDIFF are just Boolean operators. The current PFIND happens to have these operators implemented as LISP functions. However, we have and are still considering alternate implementations.)

There is no INOT function, because of the gross inefficiencies inherent in its use, e.g.,

```
(PFIND(INOT (ISA COUPE)))
```

might return a list of every item in the net. In virtually every case there should be an encoding of "not" in terms of ISDIFF. A pure INOT would have to be simulated by

```
(PFIND (ISDIFF(EVERYITEM) (ISA COUPE)))
```

where EVERYITEM is a function which returns the entire list of items in the net.

As should be evident from these examples, the arguments to IAND,IOR, and ISDIFF are of the same type as arguments to PFIND, that is, each <arg> may be either a relation/item-spec pair or a form which evaluates to a list of items. Like PFIND, the Boolean functions take efficient advantage of the sorted item lists specified by their arguments, and return sorted item lists.

b. Property Checking

Another thing a user might want to have along with PFIND is the ability to select items with specific values for given properties, e.g. to find all senators between 5′8" and 5′11". One way to do this is to write a special purpose function, MIDDLE-HEIGHT, which screens a list of items, returning those which satisfy the property. One could then call:

```
(MIDDLE-HEIGHT (PFIND (ISA SENATOR))
```

However, there has been added to the SEMNET package a general purpose function, IPROPCHECK, which makes it easier, in many cases, to do property checking. IPROPCHECK takes a descriptor or a form which evaluates to a list of items, a property name, and a predicate. It then applies the predicate to the value of the given property for each item and returns the sublist for which the predicate evaluates to T. It is often easier to use IPROPCHECK for property checking rather than calling PFIND, then following property links, and then applying a test predicate. For example,

```
(IPROPCHECK (ISA SENATOR)
                    HEIGHT
                   (LAMBDA(X)(AND(GREATERP X 67)
                                  (LESSP X 72))))
```

might be an equivalent formulation of the above retrieval request. (IPROPCHECK is an NLAMBDA which obviates the need to enclose its second and third arguments in QUOTE or FUNCTION.)

As with the other PFIND functions, the first argument to IPROPCHECK can be a relation/item-spec pair or a form which evaluates to a list of items. To find all DRONs which have numbers as their value under property PIFFLE and lists as their value under property WIFFLE, one could write

```
(PFIND (IPROPCHECK (IPROPCHECK (ISA DRON)
                                PIFFLE
                                NUMBERP)
                    WIFFLE
                    LISTP))
```

Note that this expression is functionally, but not computationally equivalent to:

```
    (PFIND (IAND (IPROPCHECK (ISA DRON) PIFFLE NUMBERP)
                 (IPROPCHECK (ISA DRON) WIFFLE LISTP)))
```

While the latter expression should return the same list of items, it is not as efficient since the LISTP predicate is applied to the entire list of DRONs.

4. FUZZYFIND

In many cases one does not want only items which match a description perfectly, but also those items which match on most of a description (where "most" is defined to mean matching above a given threshold), or perhaps the set of items which match more descriptors than other items. There is a function, FUZZYFIND, which returns a list of the best match to a list of descriptors (or forms). The goodness of a match is defined to be the fraction of the descriptors which are matched. (Currently exact matches are required.) The form

```
    (FUZZYFIND (ISA TREE)(SEED/POD CONE)(WOOD/TYPE SOFT))
```

will return a list of those items which have the most matches of the three links specified. PINE should match on 3 descriptors; OAK on 1; and GORILLA on none. Currently FUZZYFIND would not be able to recognize a tree whose WOOD/TYPE is MEDIUM as matching better than one whose

WOOD/TYPE is HARD.

There are three important global variables used by FUZZYFIND. FUZZYTHRESHOLD (which is initially set to 0) can be used to reject partial matches below a given fraction. (Currently all descriptors have equal weight.) If FUZZYTHRESHOLD were .5 in the example above, then OAK would be rejected, even if it were the best match found. FUZZYSEQUENT (which is initially set to T) is used by FUZZYFIND to remember the less than best matches. If FUZZYFIND is called with FUZZYSEQUENT not equal to T then it simply returns the next best matching list. Finally, FUZZYVALUE records the value of the best match.

For example, suppose OAK, CEDAR, PINE, MAPLE, BALSA, and GORILLA exist as items in the net with item numbers 1.2,3,4,5, and 6 respectively. The call to FUZZYFIND as shown above would return (2 3) as value, with FUZZYVALUE set to 1.0 and FUZZYSEQUENT set to ((5) . .67)((1) . .33)). A subsequent call to FUZZYFIND (with or without arguments) e.g. (FUZZYFIND) will return (5) as value, with FUZZYVALUE set to .67 and FUZZYSEQUENT set ((1 4). .33). The next call to FUZZYFIND returns (1 4). Thereafter the value of FUZZYFIND is NIL.

FUZZYFIND can thus be used either as a generating function which produces lists of less and less good matches, or as an operator within PFIND. (Note that the particular way FUZZYFIND has been implemented as a generating function takes advantage of the ordered union and intersection and array storage facilities of SEMNET. A different storage format might well result in a different implementation of FUZZYFIND, one which, say, finds elements one at a time.) In either mode, FUZZYSEQUENT should be set to T before the first call to FUZZYFIND.

5. Implementation

a. Alternate Method of Evaluation

The definition of "descriptor" as given in section 2.c,

```
<descriptor> ::=  (<relation-PNAME><item-spec>)
<item-spec> ::=  <item-PNAME>|<item-number>|<form>
```

is not always the most convenient. Frequently, one would like to use a variable which evaluates to an item number as the item-spec. This is especially true when PFIND is called within other functions rather than at the top-level of LISP.

In order to make this alternate mode of evaluation possible, without altering the basic way of calling PFIND, there is a set of companions to the basic PFIND functions. Each of these are distinguished by names which end in V, e.g., IANDV, IPROPCHECKV, PFINDV, and FUZZYFINDV. The functions are identical to their corresponding versions without the V except that they (locally) reset the descriptor evaluation function. It is even possible to intermix the two types of functions:

```
(PFIND (IOR  (IAND(ISA SOLDIER)(AGE OLD))
                  (ISDIFF (ISA X)(ISA Y))))
```

Here, X and Y are assumed to be variables whose values are item numbers.

b. Keywords

Because of certain global variables and the method of evaluation used by PFIND functions, there are certain cautions a user should observe.

First, since both descriptors and forms are allowed as PFIND arguments there can be an ambiguity if a function name is also a relation name, e.g., if ISA is both a relation and a function, the form

```
(PFIND (ISA BIRD))
```

is ambiguous. The assumption made in such a case is that the relation name is meant, consistent with the early version of PFIND. As a result if the user wants to have relations named PFIND, ISDIFF, IPROPCHECK, etc. he will be giving up the use of these as functions in calls to PFIND. If a function is used other than in PFIND then there is no harm in also making it a relation name.

Second, there are global variables used by PFIND functions, which should not be reset. These are FUZZYTHRESHOLD, FUZZYALUE, FUZZYSEQUENT, and EVALFLAG. The first three are used only by FUZZYFIND. EVALFLAG is used by the function (ARGEVAL) which evaluates descriptors, to distinguish between the two modes of evaluation.

6. Inadequacies in the Current PFIND

There are several areas in which the retrieval functions were either incomplete or awkward. Specifically, functions are needed to

1) allow variable specification of relations as well as items

2) follow paths as well as check for one link connections, e.g.:

```
(PFIND (IOR(ISA ANIMAL)
            (ISA (PFIND (ISA ANIMAL))
            (ISA (PFIND (ISA (PFIND (ISA ANIMAL)))))))
```

3) (for FUZZYFIND)
   (a) allow weighting of descriptors
   (b) use probabilities associated with arcs.

The first of these to be addressed will be the FUZZYFIND extensions, making use of agumented arcs to store probabilities and some modificaton of the calling format to introduce weighting of descriptors.

References

[1] Woods, et al. (1975)
Speech Understanding Systems, Quarterly Technical Progress Report No. 1, Report No. 3018, Bolt Beranek and Newman Inc., Cambridge, MA.

APPENDIX A

(ARGEVAL ARG)

ARG is a descriptor of the form (<relation-PNAME> <item-spec>) ARGEVAL turns the item-spec into a list of item numbers. If EVALFLAG is NIL then if the item-spec is a literal atom it is assumed to be the PNAME of an item. Otherwise it is assumed to be a variable whose value is an item number. If the item-spec is a non-atom then it should evaluate to a list of item numbers.) [LAMBDA]

(FUZZYFIND ARGS)

Returns a list of "best" matches to a list of descriptors (or forms) The score of the best match is recorded in FUZZYVALUE. The list of next best matches is kept in FUZZYSEQUENT. Matches are returned only if the match value is greater than or equal to FUZZYTHRESHOLD. If FUZZYFIND is called with FUZZYSEQUENT equal to T then it does a retrieval. If FUZZYSEQUENT is NIL then FUZZYFIND returns NIL. Otherwise, the value of FUZZYFIND is the next best match from the previous retrieval by FUZZYFIND.) [NLAMBDA]

(FUZZYFINDV ARGS)

Version of FUZZYFIND for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see ARGEVAL)) [NLAMBDA]

(IAND ARGS)

Boolean operator (conjunction) used in calls to PFIND. Takes indefinitie number of descriptors (or forms) as arguments) [NLAMBDA]

(IANDV ARGS)

Version of IAND for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see ARGEVAL)) [NLAMBDA]

(PFIND1 ARGS)

Finds an ordered set of items pointed to via *REL from at least one of the elements of ITEMS) [LAMBDA]

(PFIND2 ARGS)

ARG is either a descriptor of the form (<relation-PNAME> <item-spec>) or a form which evaluates to a list of items. Calls PFIND1 to produce ordered set of items matching the descriptor)

(PFIND ARGS)

General purpose, top-level retrieval function. ARGS is a list of descriptors and forms which evaluate to lists of items (see PFIND2) The forms are usually made up of calls to IAND, IOR, ISDIFF, IPROPCHECK, FUZZYFIND, and PFIND itself) [NLAMBDA]

(PFINDV ARGS)

Version of PFIND for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see ARGEVAL)) [NLAMBDA]

(IOR ARGS)

Boolean operator (disjunction) used in calls to PFIND. Takes indefinite number of descriptors (or forms) as arguments) [NLAMBDA]

(IORV ARGS)

Version of IOR for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see ARGEVAL)) [NLAMBDA]

(IPROPCHECK ARGS)

IPROPCHECK selects those elements from the list returned by (APPLY* (FUNCTION PFIND2) FORM) for which the value of PROP satisfies the single argument function FN. Used primarily in calls to PFIND. [NLAMBDA]

(IPROPCHECKV ARGS)

Version of IPROPCHECK for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see ARGEVAL)) [NLAMBDA]

(ISDIFF ARGS)

Boolean operator (set difference) used in calls to PFIND. Takes indefinite number of descriptors (or forms) as arguments) [NLAMBDA]

(ISDIFFV ARGS)

Version of ISDIFF for which literal atoms in descriptors are assumed to be variables which evaluate to item numbers (see AREVAL)) [NLAMBDA]

III. PUBLICATIONS

The following publications were produced during this quarter.

Bruce, Bertram (January 1975)
"Belief Systems and Language Understanding," BBN Report No. 2973, Bolt Beranek and Newman Inc., Cambridge, MA.

Bruce, Bertram (April 1975)
"Case Systems for Natural Language," BBN Report No. 3010, Bolt Beranek and Newman Inc., Cambridge, MA.

Makhoul, John (April 1975)
"Linear Prediction: A Tutorial Review," Proc. of the IEEE, Vol. 63, No. 4, pp. 561-580.

Wolf, Jared (April 1975)
"Speech Recognition and Understanding," in K.-S. Fu (ed.), _Pattern Recognition_, New York: Springer-Verlag (in press).

Woods, William (April 1975)
"Syntax, Semantics, and Speech," in Reddy, D.R. (ed.) _Speech Recognition: invited papers presented at the IEEE Symposium_, New York: Academic Press (in press). Also as BBN Report No. 3067, Bolt Beranek and Newman Inc., Cambridge, MA.

Woods, William (April 1975)
"What's in a Link: Foundations for Semantic Networks," in Bobrow, D. and Collins, A. (eds.) _Representation and Understanding: Studies in Cognitive Science_, New York: Academic Press (in press).