DOCUMENT RESUME

ED 046 222 PM 008 640

AUTHOR	Van Campen, Joseph A.
TITLE	Towards the Automatic Generation of Programmed Foreign-Language Instructional Materials.
INSTITUTION	Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
SPONS AGENCY	Office of Naval Research, Washington, D.C. Psychological Sciences Div.
REPORT NO	TR-163
PUB DATE	11 Jan 71
NOTE	66p.; Psychology Series
EDRS PRICE	EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS	*Computer Assisted Instruction, Computer Based Laboratories, *Language Instruction, Language Patterns, Program Descriptions, Programed Instruction, Programing, *Programing Languages, Semantics, Sentence Structure, Syntax, Vocabulary
IDENTIFIERS	*Elementary Verbal Communicator, EVC

ABSTRACT

The purpose of this report is to describe a set of programs which either perform certain tasks useful in the generation of programed foreign-language instructional material or facilitate the writing of such task-oriented programs by other researchers. The programs described are these: (1) a PDP-10 assembly language program for the selection from a coded vocabulary list of individual words to be used in generating a number of concrete drill sentences, (2) a coding system designed to allow the concise statement of a large set of semantic-syntactic patterns, (3) a program which utilizes material encoded according to the coding system described above, (4) a program for the automatic listing of individually coded vocabulary items under their semantic classes, (5) a new string-manipulation language for the PDP-10 computer, and (6) an Elementary Verbal Communicator program. (Author/MF)

ED046222

TOWARDS THE AUTOMATIC GENERATION OF PROGRAMMED FOREIGN-LANGUAGE INSTRUCTIONAL MATERIALS

BY

JOSEPH A. VAN CAMPEN

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

TECHNICAL REPORT NO. 163

JANUARY 11, 1971

PSYCHOLOGY SERIES

Reproduction in whole or in part is permitted for any purpose of the United States Government. Distribution of this document is unlimited.

This research was sponsored by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-67-A-0112-0042, Contract Authority Identification Number, NR No. 154-318.

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

EM008040

TECHNICAL REPORTS

PSYCHOLOGY SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

(Place of publication shown in parentheses; if published title is different from title of Technical Report, this is also shown in parentheses.)

(For reports no. 1 - 44, see Technical Report no. 125.)

50 R. C. Atkinson and R. C. Calfee. Mathematical learning theory. January 2, 1963. (In B. B. Wolman (Ed.), Scientific Psychology, New York: Basic Books, Inc., 1965. Pp. 254-275)

51 P. Suppes, E. Crothers, and R. Weir. Application of mathematical learning theory and linguistic analysis to vowel phoneme matching in Russian words. December 28, 1962.

52 R. C. Atkinson, R. Calfee, G. Sommer, W. Jeffrey and R. Shoemaker. A test of three models for stimulus compounding with children. January 29, 1963. (J. exp. Psychol., 1964, 67, 52-58)

53 E. Crothers. General Markov models for learning with inter-trial forgetting. April 8, 1963.

54 J. L. Myers and R. C. Atkinson. Choice behavior and reward structure. May 24, 1963. (Journal math. Psychol., 1964, 1, 170-203)

55 R. E. Robinson. A set-theoretical approach to empirical meaningfulness of measurement statements. June 10, 1963.

56 E. Crothers, R. Weir and P. Palmer. The role of transcription in the learning of the orthographic representations of Russian sounds. June 17, 1963.

57 P. Suppes. Problems of optimization in learning a list of simple items. July 22, 1963. (In Maynard W. Shelly, II and Glenn L. Bryan (Eds.), Human Judgments and Optimality, New York: Wiley, 1964. Pp. 116-126)

58 R. C. Atkinson and E. J. Crothers. Theoretical note: all-or-none learning and intertrial forgetting. July 24, 1963.

59 R. C. Calfee. Long-term behavior of rats under probabilistic reinforcement schedules. October 1, 1963.

60 R. C. Atkinson and E. J. Crothers. Tests of acquisition and retention, axioms for paired-associate learning. October 25, 1963. (A comparison of paired-associate learning models having different acquisition and retention axioms, J. math. Psychol., 1964, 1, 285-315)

61 W. J. McGill and J. Gibbon. The general-gamma distribution and reaction times. November 20, 1963. (J. math. Psychol., 1965, 2, 1-18)

62 M. F. Norman. Incremental learning on random trials. December 9, 1963. (J. math. Psychol., 1964, 1, 336-351)

63 P. Suppes. The development of mathematical concepts in children. February 25, 1964. (On the behavioral foundations of mathematical concepts. Monographs of the Society for Research in Child Development, 1965, 30, 60-96)

64 P. Suppes. Mathematical concept formation in children. April 10, 1964. (Amer. Psychologist, 1966, 21, 139-150)

65 R. C. Calfee, R. C. Atkinson, and T. Shelton, Jr. Mathematical models for verbal learning. August 21, 1964. (In N. Wiener and J. P. Schoda (Eds.), Cybernetics of the Nervous System: Progress in Brain Research, Amsterdam, The Netherlands: Elsevier Publishing Co., 1965. Pp. 333-349)

66 L. Keller, M. Cole, C. J. Burke, and W. K. Estes. Paired associate learning with differential rewards. August 20, 1964. (Reward and information values of trial outcomes in paired associate learning. (Psychol. Monogr., 1965, 79, 1-21)

67 M. F. Norman. A probabilistic model for free-responding. December 14, 1964.

68 W. K. Estes and H. A. Taylor. Visual detection in relation to display size and redundancy of critical elements. January 25, 1965, Revised 7-1-65. (Perception and Psychophysics, 1966, 1, 9-16)

69 P. Suppes and J. Donio. Foundations of stimulus-sampling theory for continuous-time processes. February 9, 1965. (J. math. Psychol., 1967, 4, 202-225)

70 R. C. Atkinson and R. A. Kinchla. A learning model for forced-choice detection experiments. February 10, 1965. (Br. J. math. & st. Psychol., 1965, 18, 184-206)

71 E. J. Crothers. Presentation orders for items from different categories. March 10, 1965.

72 P. Suppes, G. Groen, and M. Schlag-Rey. Some models for response latency in paired-associates learning. May 5, 1965. (J. math. Psychol., 1966, 3, 99-128)

73 M. V. Levine. The generalization function in the probability learning experiment. June 3, 1965.

74 D. Hansen and T. S. Rodgers. An exploration of psycholinguistic units in initial reading. July 6, 1965.

75 B. C. Arnold. A correlated urn-scheme for a continuum of responses. July 20, 1965.

76 C. Izawa and W. K. Estes. Reinforcement-test sequences in paired-associate learning. August 1, 1965. (Psychol. Reports, 1966, 18, 879-919)

77 S. L. Blehart. Pattern discrimination learning with Rhesus monkeys. September 1, 1965. (Psychol. Reports, 1966, 19, 311-321)

78 J. L. Phillips and R. C. Atkinson. The effects of display size on short-term memory. August 31, 1965.

79 R. C. Atkinson and R. M. Shiffrin. Mathematical models for memory and learning. September 20, 1965.

80 P. Suppes. The psychological foundations of mathematics. October 25, 1965. (Colloques Internationaux du Centre National de la Recherche Scientifique. Editions du Centre National de la Recherche Scientifique. Paris: 1967. Pp. 213-242)

81 P. Suppes. Computer-assisted instruction in the schools: potentialities, problems, prospects. October 29, 1965.

82 R. A. Kinchla, J. Townsend, J. Yellott, Jr., and R. C. Atkinson. Influence of correlated visual cues on auditory signal detection. November 2, 1965. (Perception and Psychophysics, 1966, 1, 67-73)

83 P. Suppes, M. Jerman, and G. Groen. Arithmetic drills and review on a computer-based teletype. November 5, 1965. (Arithmetic Teacher, April 1966, 303-309.

84 P. Suppes and L. Hyman. Concept learning with non-verbal geometrical stimuli. November 15, 1965.

85 P. Holland. A variation on the minimum chi-square test. (J. math. Psychol., 1967, 3, 377-413).

86 P. Suppes. Accelerated program in elementary-school mathematics -- the second year. November 22, 1965. (Psychology in the Schools, 1966, 3, 294-307)

87 P. Lorenzen and F. Binford. Logic as a dialogical game. November 29, 1965.

88 L. Keller, W. J. Thomsen, J. R. Tweedy, and R. C. Atkinson. The effects of reinforcement interval on the acquisition of paired-associate responses. December 10, 1965. (J. exp. Psychol., 1967, 73, 268-277)

89 J. I. Yellott, Jr. Some effects on noncontingent success in human probability learning. December 15, 1965.

90 P. Suppes and G. Groen. Some counting models for first-grade performance data on simple addition facts. January 14, 1966. (In J. M. Scandura (Ed.), Research in Mathematics Education. Washington, D. C.: NCTM, 1967. Pp. 35-43.

91 P. Suppes. Information processing and choice behavior. January 31, 1966.

92 G. Groen and R. C. Atkinson. Models for optimizing the learning process. February 11, 1966. (Psychol. Bulletin, 1966, 66, 309-320)

93 R. C. Atkinson and D. Hansen. Computer-assisted instruction in initial reading: Stanford project. March 17, 1966. (Reading Research Quarterly, 1966, 2, 5-25)

94 P. Suppes. Probabilistic inference and the concept of total evidence. March 23, 1966. (In J. Hintikka and P. Suppes (Eds.), Aspects of Inductive Logic. Amsterdam: North-Holland Publishing Co., 1966. Pp. 49-65.

95 P. Suppes. The axiomatic method in high-school mathematics. April 12, 1966. (The Role of Axiomatics and Problem Solving in Mathematics. The Conference Board of the Mathematical Sciences, Washington, D. C.: Ginn and Co., 1966. Pp. 69-76.

(Continued on inside back cover)

ED046222

TOWARDS THE AUTOMATIC GENERATION OF PROGRAMMED FOREIGN-LANGUAGE INSTRUCTIONAL MATERIALS

BY

JOSEPH A. VAN CAMPEN

TECHNICAL REPORT NO. 163

JANUARY 11, 1971

PSYCHOLOGY SERIES

Reproduction in whole or in part is permitted for any purpose of the United States Government. Distribution of this document is unlimited.

This research was sponsored by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-67-A-0112-0042, Contract Authority Identification Number, NR No. 154-318.

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

ERIC

Table of Contents

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Stanford University
Institute for Mathematical Studies
in the Social Sciences

2a. REPORT SECURITY CLASSIFICATION
Unlimited

2b. GROUP

3. REPORT TITLE
Towards the Automatic Generation of Programmed Foreign-Language Instructional Materials

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*
Joseph A. Van Campen

6. REPORT DATE
January 11, 1971

7a. TOTAL NO. OF PAGES
60

7b. NO. OF REFS
0

8a. CONTRACT OR GRANT NO.
N00014-67-A-0112-0042

b. PROJECT NO.
NR 154-318

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Technical Report No. 163

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

The report includes descriptions of the following: 1) A PDP-10 assembly language program for the selection from a coded vocabulary list of individual words to be used in generating a number of concrete drill sentences corresponding to a single abstract sentence pattern, 2) a coding system designed to allow the concise statement of a large set of semantic-syntactic patterns in which a given vocabulary item can be employed, 3) a program which utilizes material encoded according to the system described under 2) above to produce the total set of individual phrase and sentence patterns available for the teaching of the given vocabulary item, 4) a program for the automatic listing of individually coded vocabulary items under the semantic classes to which they belong, 5) a new string-manipulation language for the PDP-10 computer, designed to facilitate the creation of programs dealing with language material without excessive utilization of machine storage, and 6) an Elementary Verbal Communicator program for the conversion of a limited set of English language statements concerning a base and/or target language to a) an operation-code string which can be used to locate appropriate frame-generation routines, and b) a set of instructional variables to be utilized by such frame-generation routines.

FORM 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
computer-based instruction						
computer-based generation of learning materials						
language instruction						
programmed instruction						
programming languages						
sentence generation						
string manipulation						
utterance patterns						

TOWARDS THE AUTOMATIC GENERATION OF PROGRAMMED FOREIGN-LANGUAGE INSTRUCTIONAL MATERIALS*

Joseph A. Van Campen
Institute for Mathematical Studies
in the Social Sciences
Stanford University
Stanford, California 94305

Introduction.

An attempt to provide a thorough-going answer to the question of the extent to which the utilization of a computational system can facilitate the generation of programmed materials for the teaching of foreign languages would involve a consideration of so many elusive variables as to necessitate a prolonged research effort by a number of specialists in such diverse areas as computer hardware, systems programming, compilers and interpreters, peripheral devices, foreign language teaching, and programmed instruction. The purpose of this report is a much more modest one--to describe a set of programs, written for and successfully implemented on the PDP-10 computing system of the Institute for Mathematical Studies in the Social Sciences at Stanford University, which either perform certain tasks useful in the generation of programmed foreign-language instructional material or facilitate the writing of such task-oriented programs by other researchers.

Since no further Federal funds have been requested by the principal investigator for research in this area, each program dealt with below will be presented insofar as possible, on its own terms, without overly great emphasis on the advantages which might flow from its integration with programs not as yet implemented. (The ties between two or more existing programs are, of course, pointed out in the introduction to the sections dealing with each of the individual programs.)

1. Program for the Cycled Selection of Individual Vocabulary Items Belonging to Certain Semantic Classes.

1.0 Purpose and Justification.

It would appear that a certain economy in the generation of foreign-language instructional material could be achieved by taking advantage of the fact that in the case of vast numbers of concrete utterances one or more of the vocabulary items which make up the utterance can be replaced by any one of a (in the case of elementary language courses almost always fairly limited) set of other vocabulary items without destroying the

*This research was supported by ONR Contract Number N00014-67-A0112-0042 and Project Number ONR Code 458.

acceptability of the utterance. Thus, for instance, in the English utterance the doctor is writing a book, the word doctor can be replaced by any member of a set including such words as lawyer, man, teacher, author, and girl. The word book could be replaced by any member of a set which includes the words poem, story, novel, letter, and speech. It would appear that if we labelled the first set 'non-infant humans' and the second 'written object' we could rewrite the utterance as the (non-infant human) is writing a (written object), in which the set labels are set off from concrete words by parentheses. Assuming that the words listed above exhaust the membership of the sets in question (a situation quite possible in the early stages of a first-year language course), we see that the rewritten utterance could itself be rewritten in twenty-five different ways by replacing both of the set labels by different members of the sets in question.

The above example is, of course, an extremely crude one. On the one hand it takes no account of the additional acceptable utterances which might be produced by allowing some variation in the grammatical categories expressed in the original utterance (e.g. the doctor has written a book, the doctors are writing a book, etc.). On the other hand, it ignores troublesome cases of the type the doctor is writing the prescription (in which the set of items which can replaced doctor is very small) and such complications as the difference in set membership needed to account for the doctor is reading (not writing) a newspaper.

However, there is as yet no reason to believe that, within the framework of a given language course, the use of set labels would not result in considerable economies in a large number of cases. The fact that such economies might not stand in any direct relationship to those attainable on the basis of a thorough analysis of the total semantic structure of a given language need not concern us here.

Given the desirability of the replacement of some or all of the constituents of utterance types by set labels, there can be no doubt that it would be useful to develop a program capable of replacing any given set label by an individual member of the set in question. Furthermore, it would also be desirable to have 1) some record of the relative frequency of usage of the various members of the set, and 2) a means of insuring that this frequency would remain relatively uniform for all members of the set.

1.1 Documentation

1.11 Function of the Program

For the above mentioned ends the principal investigator wrote and implemented a PDP-10 assembly language program which examines a string containing one or more set labels and replaces those labels it encounters with words belonging to the sets in question. In addition, this program examines the usage index accompanying each member of the given set and selects the member to be used in a given case from the subset with the lowest index. Finally, the usage index of the member selected is increased by one, unless

this would result in an index greater than the upper limit (decimal 9). In the latter case, the index of the member selected is set at one and those of the other set members are reset to zero.

1.12 File Format.

1.121 Input Files.

The input for the program consists of 1) a disk file (channel 1) containing one or more strings including set labels to be replaced by set members, and 2) a disk file (channel 2) containing an alphabetized list of set labels each accompanied by the members of the given set.

1.1211 String Format.

Each string on channel 1 must include 1) one or more set labels, each of which must be enclosed in parentheses, and 2) a slash, not enclosed in parentheses, indicating the end of the given string. The string may include other items not enclosed in parentheses, e.g., punctuation marks and concrete language material. The only restriction on such items is that they not include any of the following: an opening or closing parenthesis, a slash, a plus mark (see 1.12111), a comma (see 2.2322), a divide sign (see 2.2412), or either of the symbols $\leq$ and $\geq$ (see 1.1222). Except for the terminating slash, which is omitted from output strings, material not enclosed in parentheses is simply transferred from the input to the output file without affecting the flow of the program in any other way.

1.12111 Format of Set Labels in Strings.

As was pointed out in the preceding paragraph, set labels within input strings are regularly enclosed in parentheses. However, it may often prove useful for other purposes to include with a given set label additional information concerning the item in question, such as its syntactic role in the given string or possible restrictions on the set of grammatical categories (e.g. number, tense) it may exhibit. In order to permit the convenient notation of such additional information it was decided to utilize a plus sign placed immediately after the final character of a given set label in place of the closing parenthesis. The latter is used to terminate the information accompanying the label. Thus, material accompanying a set label may include any characters other than an opening or closing parenthesis, a slash or a plus sign.

1.12112 Examples of String Input.

1.121121 String Containing only Set Labels Unaccompanied by Additional Information.

(person) (emotion verb) (person)/

1.121122 Same String as 1.121121 with Punctuation Mark (Period).

(person) (emotion verb) (person)./

1.121123 Same String as 1.21122 with Concrete Language Items.

(person) does not (emotion verb) (person)./

1.121124 Same String as 1.21123 with Additional Syntactic Information.

(person + subject) does not (emotion verb) (person + direct object)./

1.121125 Same String as 1.21124 with Additional Information on Category Restrictions.

(person + subject singular) does not (emotion verb)
(person + direct object plural)./

Note that only the items in 1.121121 are significant for the program in question. The other examples merely illustrate the manner in which additional material, presumably of use in connection with other programs, can be included along with the significant items.

1.1212 Set-List Format.

Each of the alphabetically ordered set labels on channel 2 is immediately followed by the members of the set in question.

1.12121 Set-Label Format.

Each set label is enclosed in parentheses. Labels may include any character except a parenthesis, slash, or plus mark. In contrast to 1.2111, no additional information of any kind is permitted within the parentheses enclosing the label.

1.12122 Set-Member Format.

1.121221 Set-Member Delimiters.

Each member of a set is set off by 1) a preceding less-than-or-equal sign ($\leq$), and 2) a following greater-than-or-equal sign ($\geq$).

Aside from the usage index discussed in 1.121222 below, the characters occurring between these two delimiters are never examined by the program and may include, in addition to the basic form of a given vocabulary item, additional coded information on the item in question, such as its inflectional or syntactic peculiarities. While it is obvious that certain conventions would have to be established in order to insure the proper interpretation of such information by programs concerned with generating inflected forms or establishing the appropriate categories to be assigned to words governed by or modifying the word in question, such conventions are irrelevant to the operation of the program under consideration.

1.121222 Usage Index.

Immediately after the initial $\leq$ there is a single decimal digit indicating the relative frequency of usage for the word in question. This digit, which can range from $\emptyset$ to 8, is increased by one each time the given word is selected for use in a concrete sentence. The program rejects for

use in any given case a set member with a usage index higher than that of any other member of the set, thus insuring a relatively uniform frequency of use for all members of the set. (When the usage index of each member of the set reaches eight, all of the indices in question are reset to zero.)

1.12123 Examples of Set-List Format.

1.121231 Set Labels.

(person)
(square object)
(emotion verb)

1.21232 Set Members (with zero usage index).

≤ ∅ man ≥
≤ ∅ cube ≥
≤ ∅ like ≥

1.21233 Set Labels followed by Set Members (with zero usage index).

(person)
≤ ∅ man ≥
≤ ∅ woman ≥
≤ ∅ boy ≥

(square object)
≤ ∅ cube ≥
≤ ∅ square ≥

(emotion verb)
≤ ∅ like ≥
≤ ∅ hate ≥
≤ ∅ despise ≥
≤ ∅ adore ≥

1.122 Output Files.

The output of the program consts of 1) a disk file (channel 3) consisting of one or more strings containing members of the sets specified in the input strings discussed under 1.1211 above, and 2) a disk file (channel 4) containing the updated version (i.e., the version with usage indices reflecting the utilization of set members employed in the output strings) of the set list described under 1.1212 above.

1.221 Output String Format.

Each set label of the input string has been replaced by a member of the set in question. Where the set label of the input string was accompanied by additional information of the type described in 1.12111, such additional information is placed immediately before the closing delimiter of the set member. The usage index of the set member is omitted in the output string.

1.2211 Examples of String Output with Corresponding Input.

1.22111 Input Set Labels Lacking Additional Information.

Input: (person) (emotion verb) (person) ./
Output: ≤ man ≥ ≤ like ≥ ≤ woman ≥.

1.22112 Input Set Labels Including Additional Information.

Input: (person + definite) (emotion verb + 3. s. pres.)
(person + definite)./
Output: ≤ man + definite ≥ ≤ like + 3. s. pres. ≥
≤ woman + definite ≥ .

1.222 Updated-Set-List Format.

The format of the updated set list is identical with that of the set-list input file described in 1.1212. Only the usage indices of set members utilized in the output strings described under 1.221 differ from the corresponding items on the input file.

1.2221 Example of Updated Set List Corresponding to Input Set List Given Under 1.21233 after Generation of Output String Given Under 1.22111.

(person)
≤ 1 man ≥
≤ 1 woman ≥
≤ ∅ boy ≥

(square object)
≤ ∅ cube ≥
≤ ∅ square ≥

(emotion verb)
≤ 1 like ≥
≤ ∅ hate ≥
≤ ∅ despise ≥
≤ ∅ adore ≥

1.2 Evaluation and Prospects for Future Development.

The program in question performs a useful, if somewhat trivial, function. It could be made more valuable by the addition of two features: reiterative string generation and optional non-incrementation of the usage index of a given set member. The latter feature would allow the repeated utilization of one and the same set member within a single output string (as in, for example, this man likes Mary and that man hates her). The former would allow the generation of more than one output string for a single specified input string, persumably by means of a decimal digit placed after the final slash of the string. Thus, using the sets given under 1.21233, the input string

(person) (emotion verb) (person) ./3 would give

< man >	< like >	< woman >
< boy >	< hate >	< man >
< woman >	< despise >	< boy >

Whether or not reiterative string generation is implemented, it would be extremely desirable to have some means of avoiding undesirable limitations on the make-up of output strings resulted from random coincidences in the number of members constituting individual sets (e.g., the limitation of a group of ten output strings from sets A, B, and C, each with three members to the types A^1 B^1 C^1, A^2 B^2 C^2, and A^3 B^3 C^3). Where the output strings are all generated from a single input specification this problem can be solved simply by comparing each new output string with those previously generated and replacing one or more members until either a new string is formed or all possible combinations are exhausted. In other cases one might insure that no two sets would have the same number of members by adding dummy members with an "illegal" usage index at the end of some of the sets. (Whenever such a dummy was encountered the first member of the set could be utilized, but only the illegal index usage of the dummy member would be raised. By keeping the illegal indices in a fixed relation to those of the real members of the set it would be possible to include more than one dummy member in a given set.)

2. Coding System for Concise Formulation of a Relatively Large Number of Semantic-Syntactic Patterns.

2.0 Justification and Purpose.

As was pointed out in 1.1 above, the utilization of input strings containing one or more set labels corresponding to groups of words sharing certain semantic features may enable us to generate a large number of concrete sentences from a single specified input string. Such input strings will henceforth be referred to as semantic-syntactic patterns, since, as we have seen in 1.12111, they may include as much information as necessary concerning the syntactic roles played by the members of the string.

The question arises whether it might be possible to attain certain economies in the specification of the semantic-syntactic patterns themselves by developing a coding system which would permit two or more semantic-syntactic patterns including one or more common set labels to be included within a single 'master pattern.' From the point of view of an individual generating the input for a language course, such a formulation would presumably be more economical than the separate specification of each of the semantic-syntactic patterns in question. It would, of course, necessitate the development of some procedure for the retrieval of individual patterns from a master pattern, a procedure discussed in section 3 below.

The exact extent to which the use of master patterns would prove more economical than the specification of all individual patterns is difficult to predict. However, it would appear that in certain cases considerable

gains could be made. One of these cases is that of a noun which can occur with only one of a set of mutually exclusive modifiers, e.g. the noun house, which can be modified by the definite article, the indefinite article, a demonstrative adjective or a possessive adjective (e.g. the house, a house, this house, my house), but not, at least in normal usage, by two or more of these at the same time (e.g. *the a house, *a my house, *this the house, etc.)

Another case occurs when a word dependent on another word may, but need not, itself be modified by yet another word, e.g., good books, very good books, write letters, write short letters, write very short letters.

While it may be that a more sophisticated approach to input coding will account for such cases by the use of general algorithms based on considerations of semantic and/or syntactic compatibility, it would not be unreasonable to provide a more immediate means of reducing the redundancies which might arise from an approach limited to the specification of individual patterns.

2.1 General Characteristics of a Desirable Coding System.

A coding system for master patterns should make a clear distinction between items which must be present in any and all the individual patterns generatable from the master pattern and those which are absent from one or more of the individual patterns. Within the latter group it must distinguish, on the one hand, between items which can cooccur and items which are mutually exclusive, and, on the other, between items which can occur independently of any other item and items which are dependent on another item, i.e., which can occur in an individual pattern only if another item is present. Finally, the coding system must provide for the above distinctions with a maximum of simplicity so that the specification of master pattern does not involve significantly greater effort and possibility for error than the specification of an individual pattern including approximately the same number of characters.

2.2 Introductory Comments on The Proposed Coding System.

2.21 Relation of the Proposed Coding System to the Format of Individual Semantic-Syntactic Patterns.

Since individual semantic-syntactic patterns must contain at least one or more set labels delimited by parentheses (cf. 1.1211) it would seem reasonable to utilize parentheses as the fundamental delimiter in master patterns as well. On the other hand, since the plus sign can occur within individual set labels (cf. 1.12111), it cannot conveniently be used as a set-label delimiter in a master pattern. The slash used to terminate individual syntactic-semantic patterns can be retained to signal the end of a master pattern.

2.22 Relation of the Proposed Coding System to Boolean Algebra.

The proposed coding system, while utilizing somewhat different symbols, was strongly influenced by Boolean algebra in 1) the utilization of parentheses, and 2) the expression of conjunction and disjunction. Since

it may be reasonably assumed that a large percentage of those reading this report will be familiar with the Boolean notation, it would seem that a discussion of the features of the proposed coding system can best be carried out by contrasting these features with their Boolean counterparts.

2.23 Conjunction.

2.231 Use of Parentheses to Express Subordinative Conjunction.

As was pointed out in 2.21, the utilization of parentheses in a delimitative function allows us to retain in master patterns symbols employed as delimiters in concrete semantic-syntactic patterns. In Boolean algebra the parentheses are utilized for determining the order of operations. In the proposed coding system this function has been essentially retained (for illustrations see 2.52 and 2.53 below). In addition, however, parentheses are employed in noting a subordinative conjunctive relationship between a pair of set labels, i.e., a relationship in which one of the conjoined set labels is grammatically dependent on (e.g. in agreement with, a modifier of, governed by) the other. This is accomplished by placing the dependent (i.e. governed, agreeing, or modifying) set label, together with its enclosing parentheses within the parentheses which enclose the non-dependent member of the pair. Thus, for example, the set-label pair (noun subject (verb predicate)) indicates that the predicate depends on (agrees with) the subject.

This additional function results in a much more extensive utilization of parentheses in our notation than in Boolean algebra, since there are numerous cases which necessitate the noting of subordinative conjunction between members of a set-label pair, even though the order of operations is irrelevant.

Thus, in Boolean algebra (A(B(C))) is merely an extremely uneconomical notation of (ABC) or simply ABC. In our notation, since it includes no indication of non-subordinative conjunction or of disjunction (see 2.232 and 2.24 below), (ABC) represents simply a single set label (recall that no set label may occur without parentheses), while (A(B(C))) represents three conjoined set labels such that C is dependent on (modifies or is governed by) B and B is dependent on A. The sequence (AB(C)) represents two conjoined set labels, (AB) and (C), the latter modifying or being governed by the former, while (A(BC)) shows the same relationships between the set labels (A) and (BC).

2.2311 The Ordering of Dependent Set Labels.

In order to simplify the operation of the program discussed in section 3 below, a dependent set label cannot precede the set label on which it is dependent. There is, for example, no such pattern as ((A)BC) or ((AB)C). The notation for such cases must be (BC(A)) and (C(AB)), respectively.

It follows that our dependency notation may entail significant differences between the order of set labels included in a master pattern and the order of concrete words occurring in sentences generated from individual semantic-syntactic patterns derived from the master pattern. Thus, a string of the type poor men might correspond to a master pattern including the set-label sequence (person (wealth adjective)). Provisions for converting dependency-oriented sequences to the order actually employed in utterance strings are discussed in section 3 below.

2.2312 Examples of the Use of Parentheses to Express Subordinative Conjunction.

2.23121 Master-Pattern Notation Matches Utterance Order.

Master-Pattern Notation: (person(emotion verb(person))./
Utterance: John likes Mary.

2.23122 Master-Pattern Notation Differs from Utterance Order.

Master-Pattern Notation: (writing verb(writing object(duration adjective(adjectival intensifier)))).
Utterance: Write very short stories.
(The master-pattern order (A(B(C(D)))) corresponds to the utterance order ADCB.)

2.232 Use of the Comma and Parentheses to Express Non-Subordinative Conjunction.

2.2321 Definition of Non-Subordinative Conjunction.

It is possible that two set labels which are themselves not members of the same dependency pair (i.e., do not have a subordinative conjunctive relationship with one another) may nevertheless play identical (dependent or, less frequently, non-dependent) roles in separate subordinative-conjunctive relationships with one and the same set label. From the point of view of grammatical analysis, such set-label pairs fall into two categories: those which are clearly cases of coordinative conjunction (e.g., big, black clouds) and those in which one of two dependent set labels can be viewed as dependent not on a single non-dependent set label but on the dependency pair formed by the non-dependent set label with the other dependent set label. Thus, for instance, in the phrase this old book the demonstrative adjective can be viewed as modifying the phrase old book. Again, in he rarely writes letters the adverb may be viewed as dependent on the phrase writes letters.

However, it would appear that little or nothing is gained by providing separate notations for these two types of sequences. In all the cases which have come to the attention of the principal investigator thus far, there is no evidence that items which could be viewed as dependent on a dependency pair as a whole behave differently from items dependent only on the non-dependent member of such a pair. Thus, the agreement of the demonstrative in such phrases as this old book follows

the same rules as in phrases of the type this book. Again the adverb in he rarely writes letters is subject to the same rules as the one in he rarely writes.

For this reason it would seem in the interests of notational economy to view both of the cases discussed above as examples of a single phenomenon--non-subordinative conjunction, i.e. a conjunctive relationship between two set labels not entailing the dependency of one of them on the other. Accordingly, phrases of the type this old book will receive the same notational treatment as those of the type old, tattered books.

2.2322 Notational Devices.

2.23221 Non-subordinative Conjunction of Two Set Labels Dependent on Third Set Label.

Since by definition set labels which stand in a non-subordinative conjunctive relationship to one another also play identical roles in separate subordinative conjunctive relationships to a third set label, it follows that if the non-subordinatively conjoined set labels are dependent on the third set label it would be possible simply to include both of the former within the parentheses enclosing the latter. For example, (A(B)(C)) would represent the non-subordinatively conjoined pair of set labels (B) and (C), each of which is dependent on the third set label (A). However, it would appear that not only notational economy, but also graphic clarity would be served by introducing into our system a new delimitational symbol which would permit us to include non-subordinatively conjoined set labels within a single set of parentheses. Since the dot and the multiplication sign, commonly employed to denote conjunction in Boolean algebra, are easily confused with the period and the letter x, respectively, it was decided to utilize the comma for this purpose. Thus, in place of (A(B)(C)) we may write (A(B,C)).

2.23222 Non-Subordinative Conjunction of Two Set Labels with Dependent Third Set Label.

It might seem at first glance that the notation of cases of this type (e.g. expensive hats and shoes, in which the adjective applies to both nouns) could follow the pattern established in 2.23221, i.e., that the non-subordinately conjoined set labels could be separated by a comma and the dependent third set label included within the parentheses surrounding the non-dependent pair. Thus, (A,B(C)) would represent the non-subordinatively conjoined set labels (A) and (B) and the third set label (C) which depends on both (A) and (B).

Unfortunately, the utilization of this notation for cases of this type is rendered less than desirable by the need for a convenient representation of such phrases as this very old book, in which, from the point of view of our notational system, the adjective old participates in three relationships: 1) non-subordinative conjunctive with this, 2) subordinative conjunctive with the non-dependent item book, and 3) subordinative conjunctive with the dependent item very. In accordance

with 2.23221, the master pattern for phrases of this type could include (noun(demonstrative adjective, age adjective)). However, it would appear that the most convenient and graphically clear notation of the set label for the adjectival intensifier modifying old would be its inclusion between the set label corresponding to that adjective and the first closing parentheses, i.e., (noun(demonstrative adjective, age adjective(adjectival intensifier))).

It would therefore appear that, unless we wish to modify the notation adopted in 2.23221, it would be best to devise another notation for phrases of the type expensive hats and shoes. It would seem that this could be accomplished with maximal clarity by 1) placing the dependent third set label outside the parentheses surrounding the non-subordinatively conjoined pair and 2) indicating its dependence on the preceding pair by enclosing it within an extra set of parentheses. Thus, instead of (A,B(C))--a notation reserved for cases in which (C) is dependent only on (B)--we would write (A,B)((C)).

2.23223 Non-Subordinative Conjunction of More Than Two Set Labels.

It is quite possible that the non-subordinative conjunctive relationships discussed in 2.23221 and 2.23222 may hold among more than two set labels. Thus, instead of this old book or old, tattered books we might have this old, tattered book, with three non-subordinatively conjoined items dependent on book. On the other hand, in addition to phrases such as expensive hats and shoes we can expect expensive hats, shoes and gloves, in which expensive is dependent on three conjoined nouns.

The notation of such cases can be adequately handled simply by extending the techniques discussed in 2.23221 and 2.23222 to provide for the insertion of a delimiting comma after every non-subordinatively conjoined set label except the last. Thus, the first series cited in the preceding paragraph would correspond to the notation (A(B,C,D)), while the second would be written as (A,B,C)((D)).

2.2323 Examples of Non-Subordinative Conjunction.

2.23231 Two Non-Subordinatively Conjoined Set Labels.

2.232311 Non-Dependent Third Set Label.

Master-Pattern Notation: (noun(demonstrative pronoun, age adjective))/
Utterance: this old book
Master-Pattern Notation: (noun(age adjective, condition adjective))/
Utterance: old, tattered books

2.232312 Dependent Third Set Label.

Master-Pattern Notation: (purchase verb, sale verb)((object noun))/
Utterance: buy and sell books
Master-Pattern Notation: (male adult, female adult)((wealth adjective))/
Utterance: rich men and women

2.23232 More than Two Non-Subordinatively Conjoined Set Labels.

2.232321 Non-Dependent Subordinatively Conjoined Set Label.

Master Pattern: (noun(demonstrative pronoun, age adjective, condition adjective))/
Utterance: this old, tattered book

2.232322 Dependent Subordinatively Conjoined Set Label.

Master Pattern: (purchase verb, sale verb, exchange verb)((object noun))/
Utterance: buy, sell, and trade books

2.24 Disjunction.

2.241 Primary Role of Exclusive Disjunction.

Since, as was pointed out in 2.0 above, one of the strongest justifications for the creation of a master-pattern notation lies in the fact that certain words can occur with any one of a set of mutually exclusive modifiers, it is not unreasonable to attend first to the notation of the "exclusive or" relationship.

2.2411 Definition of Exclusive Disjunction.

For our purposes an exclusive disjunctive relationship can be said to occur between two or more set labels in a given master pattern when 1) for any concrete semantic-syntactic pattern derivable from the given master pattern only one of the set labels can be present, and 2) any one of the set labels, if it is indeed present in a concrete semantic-syntactic pattern, will play one and the same role in a dependency pair with one and the same subordinatively conjoined set label. Thus, for instance phrases of the type my book and this book could be derived from a single master-pattern formulation in which the set labels corresponding to my and this are in exclusive disjunction.

2.2412 Notation.

2.24121 Exclusive Disjunction between Two Set Labels.

Since we have already employed the plus sign within set labels (cf. 1.1211) it cannot conveniently be employed to indicate a disjunctive relationship between two set labels. Since, however, it seems desirable to utilize a symbol not normally employed in the strings constituting such labels, and because of the associative ties between disjunction, separation and division, it was decided to employ the divide sign (÷) to separate the members of an exclusive disjunctive pair. The subordinative conjunctive relationship of each member of the pair to a third set label in the master pattern is indicated by the devices discussed in 2.23221 and 2.23222 above. Thus, (A ÷ B) indicates that the set labels (A) and (B) are in exclusive disjunction; (C(A ÷ B)) indicates that whichever one of them is present in a given string will depend on (C); while (A ÷ B)((C)) indicates that (C) will depend on whichever one of (A) or (B) occurs in a given string.

2.24122 Exclusive Disjunction between More Than Two Set Labels.

As in the case of non-subordinative conjunction, we simply modify the notation developed for pairs of set labels by writing a delimiter after each set label except the last. Thus, with a non-dependent subordinatively conjoined set label we write (A(B + C + D)), while with a dependent one we write (A + B + C)((D)).

2.2413 Example of Exclusive Disjunction.

Master Pattern: (noun(demonstrative adjective + possessive adjective + definite article))

2.24131 Concrete Patterns.

2.241311 (noun(demonstrative adjective))

2.241312 (noun(possessive adjective))

2.241313 (noun(definite article))

2.24132 Utterances.

2.241321 Corresponding to 2.241311.

this book

2.241322 Corresponding to 2.241312.

my book

2.241323 Corresponding to 2.241313.

the book

2.242 Inclusive Disjunction.

2.2421 Definition of Inclusive Disjunction.

For our purposes an inclusive disjunctive relationship can be said to occur between two or more set labels in a given master pattern when 1) any combination of the set labels in question can occur within a concrete semantic-syntactic pattern derived from the given master pattern and, 2) each of the set labels which does occur in a concrete semantic-syntactic pattern will play one and the same role in a dependency pair with one and the same subordinatively conjoined set label. Thus, if within a single master pattern, in addition to phrases of the type, old, tattered books, which were cited in conjunction with non-subordinative conjunction (cf. 2.23221), we wish to make provision for phrases such as old books and tattered books, we need simply change the relationship between the set labels (age adjective) and (condition adjective) to one of inclusive disjunction.

2.422 Notation.

There would appear to be no strong reasons for introducing a new symbol to represent the inclusive disjunctive relationship. First of all, as is pointed out in 2.631232, this type of relationship is not likely to occur with any great frequency in the material with which we are concerned. Secondly, it is adequately covered by the notational devices utilized to indicate restrictions on the omission of optional dependent set labels (cf. 2.631232 and 2.632232).

2.5 The Ordering of Operations.

2.51 Relationship of Ordering to Subordinative Conjunction.

In order to permit the derivation of an optimally large number of concrete semantic-syntactic patterns from a single master pattern, it is necessary to allow for a wide variety of disjunctive and conjunctive relationship "networks" involving a large number of set labels. This in turn necessitates a set of rules governing the order in which we will perform the selection or grouping of set labels in deriving concrete patterns. Thus, if we are faced with a master pattern of the type (A ↑ B(C ↑ D)), we must be able to decide whether one of the selective operations indicated by the two disjunction symbols depends on the results of the other. A similar question must be asked about the selection and grouping operations indicated by the divide sign and the comma in (A ↑ B(C,D)).

As was pointed out in 2.231, a subordinative conjunctive relationship between two set labels indicates that one of them is grammatically dependent on the other. In addition, if the non-dependent label or labels with which a given dependent label is subordinatively conjoined is (are) not non-subordinatively conjoined with a dummy label (cf. 2.63 and 2.7 below), the dependent label can occur only in those concrete semantic-syntactic patterns which include (one of) the non-dependent label(s).

It follows that subordinative conjunction can serve as a guide to the ordering of operations, since operations affecting the dependent set label of a given dependency pair need be performed only after the completion of those operations which determine the presence or absence in a given concrete pattern of the set label(s) with which the dependent label must be conjoined.

2.52 Parentheses Depth as a Guide to the Ordering of Operations

Since in noting subordinative conjunction we place the dependent set label(s) either within the parentheses enclosing the non-dependent label(s) or (cf. 2.23222) within an extra set of parentheses, it follows that we can order our operations in the manner described in 2.51 simply by performing first those operations enclosed by the smallest number of sets of parentheses.

Operations entailing a greater number of sets of parentheses would be performed only if they involved set labels which were subordinatively conjoined with one or more set labels selected from the master pattern as a result of the performance of preceding operations.

Thus, for example in (A ↑ B(C)), we would first choose between (A) and (B). Only if (B) were chosen would we proceed to the selection of (C), since (cf. 2.23222) this label is not subordinatively conjoined with (A). Similar considerations would apply to (A ↑ B(C ↑ D)) or (A ↑ B(C,D)). On the other hand, in (A ↑ B)((C)), (C) would always be selected, since it depends on either (A) or (B).

2.521 The Utilization of Additional Sets of Parentheses.

The ordering of operations in accordance with the hierarchy of subordinative conjunction does not permit us to impose different orders of grouping and selection on a number of set labels all of which play one and the same role in subordinative conjunction with one and the same set label. Thus, we have so far no way of deciding whether (A(B,C↑D,E)) will result in the concrete patterns (A(B,D,E)) and (A(B,C,E)) or in (A(BC)) and (A(DE)).

It would appear that the simplest solution to this problem is the introduction into our notation of additional sets of parentheses not needed for the representation of subordinative conjunction. Thus, if we wish to insure the derivation of the first pair of concrete patterns mentioned in the preceding paragraph, we need only write (A(B,(C↑D),E)), while the derivation of the second pair is assured by (A((B,C) ↑ (D,E))).

Note that the additional parentheses do not replace any of the other operational symbols. This feature, which leads to the use of what may appear to be superfluous sets of parentheses in certain cases (cf. the second notation above), was introduced to simplify the program discussed in section 3 below.

2.53 Examples of the Derivation of Concrete Semantic-Syntactic Patterns from Master Patterns in which the Ordering of Operations is Significant.

2.531 Additional Parentheses Not Required.

Master Pattern: (proper name ↑ kinship noun(proper name ↑ possessive adjective))((oral noise verb))/

2.5311 Concrete Pattern Resulting from Selection of Set Label (Proper Name).

(proper name(oral noise verb))/

2.5312 Concrete Patterns Resulting from Selection of Set Label (Kinship Noun)/

2.53121 (kinship noun(proper name, oral noise verb))/

2.53122 (kinship noun(possessive adjective, oral noise verb))

2.5313 Utterance Corresponding to 2.5311.

John is singing.

2.53141 Utterance Corresponding to 2.53121.

Mary's brother is talking.

2.53142 Utterance Corresponding to 2.53122.

Their nephew is crying.

2.532 Additional Parentheses Required.

Master Pattern: (noun((possessive adjective + demonstrative adjective), age adjective))/

2.5321 Concrete Patterns.

2.53211 (noun(possessive adjective, age adjective))/

2.53212 (noun(demonstrative adjective, age adjective))/

2.5322 Utterances.

2.53221 Corresponding to 2.53211.

my old house

2.53222 Corresponding to 2.53212.

this new table

2.6 Optional Relationships.

2.61 Definition of Optional Relationships and Optional Set Labels.

As was indicated in 2.0, there are numerous cases in which a set label may, but need not, be subordinatively conjoined with a dependent set label. In such cases the subordinative conjunctive relationship can be referred to as an optional relationship and the dependent set label as an optional set label. Thus, in a master pattern notation (noun(quality adjective(adjectival intensifier))), corresponding to phrases of the type

very good books, there is one optional relationship--(quality adjective (adjectival intensifier))--and one optional set label--(adjectival intensifier).

2.62 The Need for a Notational Device.

It might at first appear that there is little or no need to provide a separate notation for optional relationships and/or set labels. Indeed, in the case just cited and in many others it would be possible to account for such concrete patterns lacking the optional items--in this case the pattern (noun(quality adjective))--by a general algorithm based on the rule for the ordering of operations. Thus, the dependent set label of a subordinatively conjoined pair might be regarded as an optional set label and the subordinative conjunction viewed as an optional relationship whenever the non-dependent set label also occurred as the dependent member in a subordinative conjunctive relationship with a third set label.

There are, however, a number of cases in which such an approach would fail. Thus, in representing prepositional phrases modifying a verb we employ the pattern (verb(preposition(noun))). In many cases the omission of the set label (noun) will result in a concrete pattern leading to the generation of unacceptable utterances. Compare for instance he reads in bed and he reads in. Again, certain transitive verbs (e.g. pulverize, compress) hardly ever occur without a direct object in normal speech, while others (e.g. read, write) do so quite frequently. It would, therefore, appear that the development of a separate notation for optional relationships and/or labels is both necessary and useful.

2.63 The Utilization of the Dummy Label to Indicate Optional Relationships.

2.631 Rationale.

Since, on the one hand, it would seem desirable to keep the set of symbols not permitted within set labels as small as possible, and since, on the other, it would seem less than desirable to develop new notational devices for optional variants of each of the three types of relationships discussed above, it was decided to signal the presence of an optional relationship by placing the optional set label in exclusive disjunction with a "dummy" or "zero" set label consisting of the single digit $\emptyset$. Note that the utilization of the dummy label ($\emptyset$) does not force us to exclude the character $\emptyset$ from other set labels. The only restriction it entails is the exclusion of set labels consisting solely of the character $\emptyset$.

2.6311 Optional Relationship Affecting Only One Dependent Set Label.

This type is extremely simple, requiring only the insertion of the divide sign and the dummy label after the optional label. Thus, while (A(B(C))) includes no optional relationship, in (A(B(C + $\emptyset$))) the relationship (B(C)) and the set label (C) are optional. Again, in

(A ↔ B)((C)) the concrete patterns are limited to (A(C)) and (B(C)), while in (A ↔ B)((C ↔ ∅)) they include (A(C)), (B(C)), (A), and (B).

2.6312 Optional Relationship Affecting Two or More Dependent Set Labels.

2.63121 Dependent Set Labels in Exclusive Disjunction.

This type is also quite simple: since only one of the dependent set labels participating in such a relationship can be present in any concrete semantic-syntactic pattern, we can make the relationship optional by the same means as in 2.6311. Thus, non-optional (A(B ↔ C)), which allows only the concrete patterns (A(B)) and (A(C)), corresponds to optional (A(B ↔ C ↔ ∅)), which permits (A) as well.

2.63122 Dependent Set Labels in Subordinative Conjunction.

This case too is taken care of simply by inserting the exclusive disjunction symbol and the dummy label. Thus, non-optional (A(B(C))), which generates only one concrete pattern, corresponds to optional (A(B(C) ↔ ∅)), which permits both (A(B(C))) and (A).

2.63123 Dependent Set Labels in Non-Subordinative Conjunction.

This case is somewhat more complicated than the preceding ones, since it requires the utilization of additional sets of parentheses to define the order of operations (cf. 2.521).

2.631231 Unrestricted Omission of Optional Labels.

This type entails no complications other than the use of additional sets of parentheses. Thus, if the non-optional notation is (A(B,C,D)), which generates only one concrete pattern, the notation (A(B,C,(D ↔ ∅))) permits both (A(B,C,D)) and (A(B,C)); (A(B,(C ↔ ∅),D)) allows (A(B,C,D)) and (A(B,D)); (A(B,(C ↔ ∅),(D ↔ ∅))) gives all of the above plus (A(B)); (A((B ↔ ∅),C,D)) generates (A(B,C,D)) or (A(C,D)); etc., etc. The maximum number of concrete patterns is generated by (A((B ↔ ∅),(C ↔ ∅), (D ↔ ∅)))) which permits any combination of dependent labels with the label (A), as well as a concrete pattern consisting of (A) alone.

2.631232 Restricted Omission of Optional Labels.

It may prove desirable in some cases to insure that all the concrete patterns derived from a given master will include at least one (or, less probably, more than one) member of a group of non-subordinatively conjoined optional set labels. Thus, one might wish to permit phrases of the type he speaks Russian well, he speaks well, and he speaks Russian, but to exclude phrases such as he speaks. While there is some doubt as to whether restrictions of this type are likely to be utilized very often, they can be adequately conveyed by a combination of optional non-subordinative conjunction and non-optional exclusive disjunction. Thus, if we wish to generate the concrete patterns (A(B,C)),(A(B)) and (A(C)), but to exclude (A), we need write not (A((B ↔ ∅),(C ↔ ∅))), but (A((B,(C ↔ ∅))↔C)).

Insofar as only one of the optional dependent set labels need be present in any concrete pattern, the restricted omission of optional set labels is equivalent to an inclusive disjunctive relationship among them. As was pointed out in 2.422, this fact, combined with the relative infrequency of this relationship, obviates the necessity for a special notational device representing inclusive disjunction as such.

2.632 Examples of the Derivation of Concrete Semantic-Syntactic Patterns from Master Patterns including Optional Relationships.

2.6321 Only One Dependent Set Label.

Master Pattern: (person(reading verb(reading object + ∅))./

2.63211 Concrete Pattern including Optional Relationship.

(person(reading verb(reading object)))./

2.63212 Concrete Pattern without Optional Relationship.

(person(reading verb))./

2.63213 Utterance Corresponding to 2.63211.

John is reading a book.

2.63214 Utterance Corresponding to 2.63212.

John is reading.

2.6322 More than One Dependent Set Label.

2.63221 Dependent Set Labels in Exclusive Disjunction.

Master Pattern: (noun(demonstrative adjective + possessive adjective + ∅))/

2.632211 Concrete Patterns with Optional Relationship.

2.6322111 (noun(demonstrative adjective))/

2.6322112 (noun(possessive adjective))/

2.632212 Concrete Pattern without Optional Relationship.

(noun)/

2.632213 Utterances.

2.6322131 Corresponding to 2.6322111.

this hat

2.6322132 Corresponding to 2.6322112.

my hat

2.6322133 Corresponding to 2.632212.

hat

2.63222 Dependent Set Labels in Subordinative Conjunction.

Master Pattern: (noun(age adjective(adjectival intensifier) + ∅))/

2.632221 Concrete Patterns.

2.6322211 (noun(age adjective(adjectival intensifier)))/

2.6322212 (noun)/

2.632222 Utterances.

2.6322221 Corresponding to 2.6322211.

very old hats

2.6322222 Corresponding to 2.6322212.

hats

2.63223 Dependent Set Labels in Non-Subordinative Conjunction.

2.632231 Unrestricted Omission of Dependent Set Labels.

Master Pattern: (person(read verb((read object + ∅),(speed adverb + ∅)))).

2.6322311 Concrete Patterns.

2.63223111 (person(read verb(read object, speed adverb))).

2.63223112 (person(read verb(read object))).

2.63223113 (person(read verb(speed adverb))).

2.63223114 (person(read verb)).

2.6322312 Utterances.

2.63223121 Corresponding to 2.63223111.

John reads books quickly.

2.63223122 Corresponding to 2.63223112.

John reads books.

2.63223123 Corresponding to 2.63223113.

John reads quickly.

2.632223124 Corresponding to 2.632223114.

John reads.

2.632232 Restricted Omission of Dependent Set Labels (Inclusive Disjunction).

Master Pattern: (person(read verb((read object,(speed adverb + ∅)) + speed adverb))./

The concrete patterns and utterances are identical with those in 2.63223111 through 2.63223113 and 2.63223121 through 2.63223123, respectively.

2.7 Evaluation and Prospects for Future Development.

Apart from the large number of sets of parentheses needed to insure the proper ordering of operations (cf. 2.521) and the complexity of the notational devices for inclusive disjunction (cf. 2.631232), our notation makes no provision for the optional notation of the non-dependent set label in a subordinatively conjoined pair. Since in patterns almost all labels which play a non-dependent role in one subordinative conjunctive relationship also play a dependent role in another such relationship, only such set labels as those playing the role of the subject or, in the case of subjectless imperative sentences, the verb, are not covered by our notation.

There is, however, no reason why we could not cover these cases as well, simply by placing the non-dependent set label in exclusive disjunction with the dummy label. Thus, if we wish to allow for both the concrete pattern (A(B)) and (B), we could write (A + ∅)((B)). Such a modification would require a corresponding modification of the program discussed in the following section of this report.

3. Program for the Generation of Concrete Semantic-Syntactic Patterns from Master Patterns.

3.0 Purpose and Justification.

The creation of a program of this type is necessary in order to realize the economies made possible by the coding system discussed in section 2. The potential value of the latter is discussed in 2.0 above.

3.1 Documentation.

3.11 Functions of the Program.

3.111 The Derivation of Concrete Patterns.

The primary function of the program is to produce the complete set of concrete semantic-syntactic patterns derivable from a master pattern (cf. 2.0). The concrete patterns could then presumably be used as input for a program of the type described in section 1.

3.112 The Representation of Conjunction in Concrete Patterns.

The program discussed in section 1 operates on set labels each of which is enclosed by a single set of parentheses. It follows that it cannot operate on concrete patterns which utilize the devices discussed under 2.231 and 2.232. On the other hand, the information on the subordinative and non-subordinative conjunction of set labels which must be included in master patterns cannot be simply omitted in generating concrete patterns, since it provides a major part of the information on government, agreement and modification needed to generate the correct forms of individual words selected by the program discussed in section 1. Since that program permits the inclusion of additional information relevant to a given set label after a plus sign following the string of characters constituting the label itself and preceding the closing parentheses for that label, it would appear that a second function of the program under discussion must be the conversion of the master-pattern representation of conjunctive relationships to a form compatible with the input format specified in 1.1211.

3.12 File Format.

3.121 Input Files.

The input for the program consists of a disk file (channel 1) containing one or more master patterns with the format specified in section 2. For examples of master pattern input see 2.2312, 2.2323, 2.2413, 2.53, etc. (Note that there is no provision in the existing master pattern notation for the inclusion of literals, i.e. concrete words. At present the only way to include a literal in a master pattern notation is to represent it as a set label corresponding to a set with a single member.)

3.122 Output Files.

3.1221 Equivalence of Concrete Patterns and Input Strings Discussed in 1.1211.

The output of the program consists of a disk file (channel 3) identical in format to that of the input file on channel 1 mentioned in 1.121. Each concrete pattern of our output is identical with one of the input strings discussed under 1.1211.

3.1222 The Representation of Subordinative and Non-Subordinative Conjunction.

As was indicated in 3.112, information on conjunctive relationships between set labels is coded as additional information accompanying individual set labels in accordance with 1.12111. This information takes the form of two alphabetic characters immediately following the plus sign and themselves followed by a space. The space, which serves only to set off the two characters from any other additional information which may precede the closing parenthesis for the given set label, may be omitted if there is no additional information. The first of the two characters

serves to identify the set label in question, while the second identifies a non-dependent set label with which the set label is subordinatively conjoined. Thus, the coding (x + ba) indicates that set label (x) is represented by the character b and is a dependent set label subordinatively conjoined with the non-dependent set label represented by the character a. Again (y + ca) states that set label (y) is represented by the character c and also depends on the set label represented by the character a.

Whether or not (x) and (y) are coordinatively conjoined can be decided only after a consideration of such factors as their semantic compatibility and, in some cases, additional coding. However, it is clear that the fact of their non-subordinative coordination--the only information conveyed by our master-pattern notation--can be established simply by ascertaining that they both depend on the set label represented by the character a.

If a set label does not depend on any other set label, the second character after the plus is identical with the first. Thus, the sequence (z + aa) indicates that the set label (z) is represented by the character a and is not a dependent set label in any subordinative conjunctive relationship. It would appear that, subject to certain limitations connected with clause boundaries, representations such as (x + aa) and (y + bb) would indicate that (x) and (y) are non-subordinatively conjoined.

3.1223 Examples of Output with Corresponding Input and Sample Utterances.

3.12231 Input with No Optional Relationships.

Input: (person(read verb + write verb(write object))).

3.122311 Output.

3.1223111 (person + aa)(read verb + ba).

3.1223112 (person + aa)(write verb + ca)(write object + dc).

3.122312 Utterances.

3.1223121 Corresponding to 3.1223111.

John is reading.

3.1223122 Corresponding to 3.1223112.

John is writing a letter.

3.12232 Input with Optional Relationships.

Input: (person(read verb((read object + ø),(speed adverb + ø)))).

3.122321 Output.

3.1223211 (person + aa)(read verb + ba)(read object + cb)(speed adverb + db)./

3.1223212 (person + aa)(read verb + ba)(read object + cb)./

3.1223213 (person + aa)(read verb + ba)(speed adverb + db)./

3.1223214 (person + aa)(read verb + ba)./

3.122322 Utterances.

3.1223221 Corresponding to 3.1223211.

John reads books quickly.

3.1223222 Corresponding to 3.1223212.

John reads books.

3.1223223 Corresponding to 3.1223213.

John reads quickly.

3.1223224 Corresponding to 3.1223214.

John reads.

3.2 Evaluation and Prospects for Future Development.

Insofar as the notation discussed in section 2 is a satisfactory one, the program under consideration would appear to operate quite satisfactorily: it converts master patterns to concrete patterns without losing any information on subordinative and non-subordinative coordination. However, as was pointed out in 2.2311, the sequence of set labels in a master pattern and, consequently, in concrete patterns as well, may often differ from the order in which individual words corresponding to such labels would occur in normal utterances. It would appear therefore, that the concrete patterns produced by the program documented under 3.1 should be utilized not as input for the program documented under 1.1, but as input for a "re-ordering" program, which would have as its output concrete patterns identical to those of the input in every respect except the sequencing of set labels. The latter would correspond to the normal word order of utterances to be generated from the queen concrete pattern.

While it is impossible to give here a detailed outline of the operation of such a program, it is clear that it would entail the use of transformations based on 1) the information on subordinative and non-subordinative conjunction placed after the plus sign accompanying each set label in a concrete pattern, and 2) other information on the semantic and/or syntactic properties of members of individual sets. Thus, if in a concrete pattern for English utterances 1) (x) depends on (y), and 2) (x) is an adjectival intensifier while (y) is an adjective, the concrete pattern rotation (y + aa)(x + ba) would be converted to (x + ba)(y + aa) to insure the correct order of such phrases as very good, extremely bad, etc.

4. Program for the Automatic Listing of Coded Vocabulary Items as Members of Semantic Sets.

4.0 Justification and Purpose.

The preparation of a set list of the type discussed in 1.1212 necessitates the association of individual vocabulary items with the set labels referring to the semantic sets of which a given vocabulary item is a member. However, once this association has been established, there is no reason why the actual inclusion of the vocabulary item in the list of set members corresponding to each relevant set label cannot be accomplished by a computer program. It would appear that such a program would significantly reduce the time and effort required to create and update the set list.

4.1 Documentation.

4.11 Function of the Program.

The program operates on 1) an alphabetized list of vocabulary items each of which is followed by the set labels corresponding to the sets of which it is a member, and 2) a set list of the type discussed in 1.1212. Each vocabulary item is listed among the set members corresponding to each of the set labels with which it is associated. If a given set label is not found on the set list, it is entered at the appropriate alphabetic position in the list with the new vocabulary item(s) as its member(s).

4.12 File Format.

4.121 Input Files.

The input for the program consists of 1) a disk file (channel 1) containing a list of new vocabulary items with their associated set labels, and 2) a disk file (channel 2) identical in format with the set list discussed under 1.1212.

4.1211 Vocabulary List Format.

Each vocabulary item is immediately preceded by the symbol ≤ and immediately followed by the symbol ≥. The set labels associated with the vocabulary item follow the symbol ≥. Each set label is enclosed in parentheses. Restrictions on set-label format are the same as those discussed in 1.12121.

4.12111 Example of Vocabulary List Format.

≤ boy ≥ (animate being)(juvenile)(male)(person)
≤ go ≥ (motion verb)
≤ house ≥ (building)(inanimate being)(residence)
≤ red ≥ (color adjective)

4.1212 Set List Format.

See 1.1212.

4.122 Output Files.

The output consists of a disk file (channel 3) of the same format as the input set list. It differs from the latter in that it 1) lists the vocabulary items found on the channel-1 input file under each of the set labels with which they are associated, and 2) includes set labels found in the vocabulary list but not in the channel-2 input file.

4.1221 Example of Output with Corresponding Input.

4.12211 Channel-1 Input.

≤ boy ≥ (animate being)(juvenile)(male)(person)

4.12212 Channel-2 Input.

(animate being)
≤ dog ≥
≤ man ≥
≤ woman ≥

(male)
≤ man ≥

(person)
≤ man ≥
≤ woman ≥

4.12213 Output.

(animate being)
≤ boy ≥
≤ dog ≥
≤ man ≥
≤ woman ≥

(juvenile)
≤ boy ≥

(male)
≤ boy ≥
≤ man ≥

(person)
≤ boy ≥
≤ man ≥
≤ woman ≥

4.3 Evaluation and Prospects for Future Development.

While the program documented under 4.2 is a useful labor saver, it would appear that considerably greater economies could be attained by a program which, by taking account of the fact that membership in one set often implies membership in one or more additional sets, would permit a significant reduction in the number of set labels associated with vocabulary items on the input list. Thus, it would appear that the item ≤ boy ≥, used in 4.12211, could be coded simply as (juvenile)(male), since any item associated with the first of these two set labels would also be associated with (animate being) and (person).

5. The Elementary Verbal Communicator (EVC). A New String-Manipulation Language for the PDP-10 Computer.

5.0 Justification and Purpose.

5.01 The Need for a Higher-Level Language.

Although the programs documented in sections 1, 3, and 4 were written in PDP-10 assembly language, it soon became clear that rapid progress in the development of programs capable of generating instructional material would necessitate the use of a higher-level language. This was due to 1) the large number of programs to be written, and 2) the interdependent character of many of the programs. Thus, the three programs documented so far center on a single problem--the generation of large numbers of utterance patterns from a small amount of input. No attention has been paid to such complex problems as the correlation of utterance patterns in the base language with those of the target language or the generation of inflected forms from the basic forms given in set lists.

The interdependency of many programs can be seen from the fact that, of the three programs discussed so far, the output of those documented under 3.1 and 4.1 serves as input for the one documented under 1.1. This means that a modification in one program may easily entail corresponding modifications in another. Such a chain-like relationship between different programs makes the greater programming speed afforded by a higher-level language particularly desirable.

5.02 Reasons for Developing a New Language.

The higher-level languages available on the PDP-10 system used by the principal investigator included Fortran, Gogol, Sail and Lisp. Of these, the first was not at all suited for string manipulation of the type entailed by our programs. Gogol and Sail were also less than satisfactory for our purposes, and, in addition, were not regularly used by the systems programmer connected with our project. The fourth language, Lisp, while more suited to our needs, was so inadequately documented that its use might have entailed a large number of unforeseen difficulties. It was hoped that the IMSSS would be able to make available another string-manipulation language--SNOBOL III. However, this language, which was ideally suited for our purposes, did not become available during the contract period.

In view of these difficulties it was decided to develop a string-manipulation language specifically adapted to the needs of our own research. This approach proved to have two major advantages. First, it made the principal investigator to a large extent independent of other programmers, since the writing of the programs documented above had given him considerable familiarity with the PDP-10 assembly language--the basic tool in the creation of the new string manipulation language. Secondly, it allowed him to try a large number of approaches to one and the same programming problem--a freedom always curtailed to some extent by the utilization of a pre-existing language.

5.1 Documentation.

Any program written in the string-manipulation language EVC consists of a series of instructions. Each instruction must begin with an operation code. An operation code is defined as a string of alphanumeric characters immediately preceded by a line feed and immediately followed by the tabulation character (henceforth referred to as the tab mark). Operation codes may not begin with a hyphen (cf. 5.311). Most, but not all, instructions include one or more additional constituents, henceforth called operands. Operands are discussed in the sections dealing with the operation codes they accompany.

5.11 Input.

5.111 Disk-file Input.

The current version of EVC permits the concurrent usage of two disk files for input purposes. (It would, of course, be relatively simple to increase this number, but such a modification would entail no fundamental changes in the language--only the creation of additional input instructions.) Since the input files currently allowed must be attached to disk channels 2 and 4, we shall henceforth refer to them as file 2 and file 4, respectively.

5.1111 Input from File 2.

5.11111 General Characteristics.

Input from file 2 is transferred to a core storage area which will henceforth be referred to as the item buffer. (The size of the item buffer currently permitted is 50 PDP-10 words, i.e., 250 seven-bit ASCII characters. It would, however, be a simple matter to develop one or more other versions of EVC with item buffers of different lengths.) The storing of new input always begins at the first word of the item buffer so that newly stored input must be transferred to another storage area (see 5.11132 below) if it is to survive the execution of subsequent input instructions.

5.11112 Input Operation Codes.

All instructions resulting in the transferral of input from file 2 to the item buffer operate on a character-by-character basis, i.e., only one seven-bit ASCII character at a time is read into the PDP-10 accumulator utilized for such input. The operation codes, which all begin with the sequence TXT, fall into two groups--those which commence the storage of input in the item buffer with the next available ASCII character on file 2, and those which store no characters until a given character or character sequence has been encountered. The latter type will henceforth be referred to as initiator-dependent operation codes.

5.111121 Operation Codes with Immediate Commencement of Character Storage.

The storage of characters from file 2 in the item buffer continues until the program encounters a terminator. In addition to the end-of-file mark, the terminator may be a single character, a character sequence, or any one of as many as five different characters. Where only a single character or character sequence is utilized to terminate the transmission, it is possible either to stop the storage of characters with the last character preceding the terminator to store the terminator as well.

5.1111211 The Operation Code TXTUPTO.

This operation code causes the storage in the input buffers of the next available character on file 2 and of all characters following it up to (but not including) a given character or character sequence. The character or sequence which is used as a terminator follows the tab mark and is enclosed by a pair of identical delimiting symbols which may take the shape of any character not included in the terminator itself.

Thus, the instruction TXTUPTO x.x will result in the storage in the item buffer of the next available character on file 2 and of all the following characters up to the first period. The same result can be achieved by TXTUPTO 1.1, TXTUPTO -.-, etc. The instruction TXTUPTO -and- would result in the storage of all characters preceding the first occurrence of the word and, while TXTUPTO zbutz would halt storage with the last character preceding the first occurrence of the string but.

5.11112111 Examples of Operation.

5.111121111 Instruction: TXTUPTO y,y.

Next available file 2 input: Tom, Dick and Harry...

Characters stored in item buffer: Tom

5.111121112 Instruction: TXTUPTO ,and,

(File input as in 5.111121111)

Characters stored: Tom, Dick

5.1111212 The Operation Code TXTHRU.

This differs from TXTUPTO (cf. 5.1111211) only in that it results in the storing of the terminator as well as the characters preceding it. Thus, if we use TXTHRU in 5.111121111 the characters stored would be Tom,. Its use in 5.111121112 would result in the storage of Tom, Dick and.

5.1111213 The Operation Code TXTALT.

This operation code is identical in its effect with TXTHRU, except that the terminator may be any one of a set of five or fewer single characters. The characters to be used as terminators immediately follow the tab mark. The first character of the set is immediately followed by a period, while

any others are immediately followed by a comma, which is itself immediately followed by the next terminator. The instruction TXTALT !, ?. will terminate the transmission of input with the storage of the first exclamation point or question mark encountered. Note that both the comma and the period will be treated as terminators, if they occur in an odd-numbered position. Thus, the instruction TXTALT .,,,!,;,?. would result in the termination of input transmission after the storage of the first period, comma, exclamation point, semicolon, or question mark encountered in the text.

5.11112131 Examples of Operation.

5.111121311 Instruction: TXTALT ,,.,?,!.

5.1111213111 Next available input: Tom, Dick and Harry.

Characters stored: Tom,

5.1111213112 Next available input: Dick and Harry.

Characters stored: Dick and Harry.

5.1111213113 Next available input: Is he home? No, he's not!

Characters stored: Is he home?

5.111122 Initiator-Dependent Operation Codes.

As was indicated above, these operation codes do not result in the storage of input characters in the item buffer until the program encounters an initiator, i.e., a character or string of characters specified in the instruction. Storage may commence with the initiator itself or with the first character following the initiator. Termination follows one of the patterns discussed in 5.1111211 and 5.1111212. Note that in these instructions both initiators and terminators are limited to a single character or character sequence. It would, however, be relatively simple to develop additional instructions permitting the use of alternative initiators and/or terminators (cf. 5.1111213).

5.1111221 The Operation Code TXTINC.

This operation code commences storage of characters in the input buffer with the first character of the initiator itself and terminates storage with the final character of the terminator. The initiator follows the tab mark and is set off in the manner specified for the terminator in 5.1111211. The closing delimiter of the initiator must be immediately followed by the terminator, also set off as in 5.1111211. Thus, for example, the instruction TXTINC -and--but- will result in the storage of the first example of the string and encountered on file 2 plus all the following characters through the first occurrence of the string but.

5.11112211 Examples of Operation.

5.111122111 Instruction: TXTINC /D//y/

Next available input: Tom, Dick and Harry.

Characters stored: Dick and Harry

5.111122112 Instruction: TXTINC -Ha-x.x

Input as in 5.111122111.

Characters stored: Harry.

5.1111222 The Operation Code TXTEXC.

This is identical in format and operation with TXTINC, except that neither the initiator nor the terminator is stored. Thus, its use in 5.111122111 would result in the storage of ick and Harr. In 5.111122112 it would store arry only.

5.1111223 The Operation Code TXTEA1.

This operation code is identical in format and operation with TXTINC and TXTEXC, except that it stores the terminator and omits the initiator. Thus, its use in 5.111122111 would give ick and Harry. In 5.111122112 it would store arry. only.

5.1112 Input from File 4.

This follows exactly the same pattern as that described under 5.1111, even to the use of one and the same item buffer. Instruction format is identical except that operation codes which operate on file 4 must begin with the sequence T4T instead of TXT, e.g., T4TUPTO, T4TALT, T4TINC, etc.

5.1113 The Input of Literals.

5.11131 Transfer of Literals to Item Buffer.

It is possible to place a string of one or more ASCII characters in the item buffer simply by using the operation code IKTLIT. The characters to be stored in the item are set off in the same manner as the terminator used with TXTUPTO (cf. 5.1111211). Thus, IKTLIT -boo- places the three characters boo in the item buffer. The same result is, of course, achieved by IKTLIT xboox or IKTLIT /boo/, etc. To place the phrase Tom, Dick and Harry in the item buffer we write only IKTLIT xTom, Dick and Harryx.

5.11132 Input Followed by Storage of Literals and/or Sets of Literals in Core Area Other than Input Buffer.

Since in the vast majority of cases it is desirable to be able to retain stored literals for future usage, the input of a literal will almost always be followed by its transfer from the item buffer to another storage area. In addition, it is often convenient to be able to retrieve more than one literal at a time. The following instructions result in 1) the storage of a literal or a set of literals in an area of core other

than the item buffer, and 2) the association of the literal or set of literals stored with another literal, also stored in an area of core other than the input buffer. The former will henceforth be referred to as a class, while the latter will be called a class name. If a class includes more than one literal, each individual literal will be called a class member.

5.111321 The Operation Code NWCLS.

This operation code, which can be used only when none of the literals to be processed includes either a comma, a period, or a percentage sign, has the following format: the tab mark is immediately followed by the literal which is to become the class name. The latter is immediately followed by a comma which is itself followed either by a space or by the first character of the first class member. The last class member is immediately followed by a period. All the others are immediately followed by commas, which may be followed by a space preceding the first character of the next class member. Thus, the instruction NWCLS vowels a,e,i,o,u. results in the storage of the literal vowels as a class name associated with a class whose members are the five vowel letters preceding the period. The instruction NWCLS ?,?. results in the storage of the literal ? both as a class name and as the sole member of the associated class.

5.111322 The Operation Code NCLNEM.

This may be used in place of NWCLS when it is necessary to store class names or class members including either a comma or a period or both (no class name or class member may ever include the percentage sign). The first character after the tab mark is the delimiter chosen to set off the class name from the first class member and the individual class members from one another. It must be immediately followed by the first character of the class name and it must immediately follow both the last character of the class name and that of each of the class members. There may, however, be a space between the delimiter and the first character of a class member. The delimiter immediately following the last character of the last class member is itself immediately followed by another delimiter. None of the literals to be stored may include the delimiter.

Thus, the instruction NCLNEM /vowels/a/e/i/o/u// has the same results as NWCLS vowel, a,e,i,o,u. (cf. 5.111321). However, the latter cannot be utilized in place of NCLNEM /averages/1.2/3.4/6.2//, where the class members 1.2, 3.4, 6.2 include the period.

5.12 Manipulation of Strings within Core Storage Areas.

5.121 String Stored in Item Buffer.

5.1211 Deletion of String Characters.

5.12111 Operation Codes Specifying Characters to be Deleted.

5.121111 The Operation Code KILLST.

This operation code results in the deletion of the final character and as many as 28 of the immediately preceding characters from the string currently stored in the item buffer. The tab mark is followed by a decimal number from 1 through 29. Thus, if the item buffer contains the string Harry, the instruction KILLST 1 would reduce the contents to Harr, while the instruction KILLST 2 would reduce Harry to Har.

5.121112 The Operation Code KILFRS.

This is identical with KILLST except that deletion begins with the first character and proceeds with the immediately following ones. Thus, KILFRS 1 changes Harry to arry, etc.

5.12112 Operation Codes Specifying Characters to be Retained.

5.121121 Numerical Specification.

5.1211211 The Operation Code IKTLST.

This code causes the deletion of all but the final character and as many as 28 of the characters immediately preceding it. Format follows that for 5.121111. Thus, IKTLST 3 changes Harry to rry.

5.1211212 The Operation Code IKTFRS.

This is the reverse of IKTLST, deleting all but the first character plus as many as 28 of the immediately following characters. Thus, IKTFRS 3 changes Harry to Har.

5.121122 Delimiter-Dependent Specification.

5.1211221 Specification of Initial Delimiter.

5.12112211 The Operation Code IKBGNG.

This code deletes all characters in the input buffer preceding the first occurrence of a given character or character sequence. The latter is set off as in 5.1111211. Thus, IKBGNG -r- changes Harry to rry.

5.12112212 The Operation code IKFLLW.

This differs from IKBGNG only in that the delimiting character or sequence is also deleted. Thus, IKFLLW /r/ changes Harry to ry.

5.1211222 Specification of Terminal Delimiter.

5.12112221 The Operation Code IKTHRU.

This causes deletion of all characters following the first occurrence of a given character or character sequence. Format as in 5.1111211. Thus, IKTHRU xrx changes Harry to Har.

5.12112222 The Operation Code LKUPTO.

This differs from LKTHRU only in that the delimiting character or sequence is also deleted. Thus, LKUPTO ,r, changes Harry to Ha.

5.12113 Deletion Dependent on Immediately Preceding Instruction.

5.121131 The Operation Code LKTRST.

This operation code, which has no operand, must be used immediately after an instruction containing one of the following operation codes: NMBGNG, NMFLLW, NMFRST, NMLAST, NMTHRU, NMUPTO (cf. 5.12122 below). Its effect is to delete from the item buffer those characters stored by the immediately preceding instruction. Thus, the two instructions NMFRST 1, start (cf. 5.1212212) and LKTRST have the same effect on the item buffer as the single instruction KILFRS 1, i.e. they change Harry to arry. The sole difference is that in the former case the H has been stored as the sole member of the class start, while in the latter it has been irretrievably lost.

5.1212 Transfer of Characters in Item Buffer to Other Core Areas.

Transfer of some or all of the characters currently in the item buffer to another core area follows the pattern for the storage of literals described under 5.11132. The characters to be stored are treated as a single literal, i.e. they become the sole member of a class. Since storage and retrieval entail the association of the characters with a class name, each of the operation codes described below contains either the sequence NM or the sequence NAM.

5.12121 Storage of Entire Contents of Item Buffer.

5.121211 The Operation Code NMITEM.

This operation code stores the entire current contents of the item buffer as the sole member of a class. The class name follows the tab mark and, in contrast to 5.121212, is not subject to modification by the program. Thus, NMITEM label stores the entire current contents of the item buffer as the sole member of a class with the name label.

5.121212 The Operation Code SERNAM.

This code differs from NMITEM only in that the class name following the tab is expanded by one of the decimal digits from Ø through 9. The digit chosen is determined by the number of times the class name occurring after the operation-code tab mark has already been used. Thus, the first occurrence of SERNAM label will associate the current contents of the item buffer with the class name label$\emptyset$. The next occurrence of this instruction associates the item-buffer string with label1, the next with label2, etc.

5.12122 Transfer of Part of Item-Buffer Contents.

5.121221 Numerical Specification of Characters to be Stored.

5.12122..1 The Operation Code NMLAST.

This operation code transfers the final character in the item buffer plus as many as 28 immediately preceding characters to a core area other than the item buffer and associates the characters, which are treated as a single class member, with a new class name. The contents of the item buffer are not affected, but pointers are set to indicate which characters in the buffer were not transferred (cf. 5.121131 and 5.1212231).

The tab mark is immediately followed by a decimal number from 1 through 29, which is itself immediately followed by a comma. The latter is followed by the new class name, with or without an intervening space. Thus, the instruction NMLAST 2,end does not change the contents of the item buffer, but, if the latter contains the string Harry, stores the sequence ry as the sole member of the class end. In addition it sets pointers indicating that the characters Har were not stored by this instruction.

5.1212212 The Operation Code NMFRST.

This operation code differs from NMLAST only in that it transfers the first character and as many as 28 immediately following characters. Thus, NMFRST 2,begin leaves Harry unchanged in the item buffer, stores Ha as the sole member of the class begin, and sets pointers to indicate that the characters rry were not transferred by this instruction.

5.121222 Delimiter-Dependent Specification of Characters to be Stored.

5.1212221 The Operation Code NMFLLW.

This operation code transfers all characters in the item buffer which follow the first occurrence of a given character or character sequence and associates them with a new class name as the sole member of the class in question. The contents of the item buffer are unchanged but pointers are set to indicate the characters not stored as a result of this instruction (cf. 5.121131 and 5.1212231). The tab mark is immediately followed by the delimiting character or character sequence, set off as in 5.1111211. The latter is followed by the new class name, with or without an intervening space. If the item buffer contains Harry, the instruction NMFLLW -ar-end stores the sequence ry as the sole member of the class end, leaving the item buffer unchanged, and indicates that Har has not been moved.

5.1212222 The Operation Code NMHGNG.

This parallels NMFLLW, except that the delimiting character or sequence is also stored. Thus, its use in the example for 5.1212221 would result in the storage not of ry, but of arry.

5.1212223 The Operation Code NMUPTO.

This code parallels NMFLLW except that it stores the characters preceding the first occurrence of the delimiting character or sequence. Its use in the example for 5.1212221 would result in the storage of H.

5.1212224 The Operation Code NMTHRU.

This parallels NMUPTO, except that the delimiting character or sequence is also stored. Its use in the example for 5.121222' results in the storage of Har.

5.121223 Transfer Dependent on Immediately Preceding Instruction.

5.1212231 The Operation Code NMREST.

This code, which can be used only immediately following an instruction with one of the operation codes NMBGNG, NMGLLW, NMFRST, NMLAST, NMTHRU or NMUPTO, transfers all the characters in the item buffer not moved as a result of the preceding instruction and associates them with a new class name as the sole member of the class in question. The contents of the item-buffer are unchanged. The tab mark is immediately followed by the new class name. Thus, the instruction sequence NMFRST 1,start and NMREST end leaves an item buffer containing Harry unchanged, but stores the character H as the sole member of the class start and the string arry as the sole member of the class end.

5.122 Manipulation of Strings Stored in Core Areas Other than Item Buffer.

5.1221 Inclusion of Two or More Classes in Another Class.

It may often be useful to be able to manipulate the members of two or more classes at a single stroke. This can be accomplished by including the membership of each of the classes in question among the members of a single larger class.

5.12211 The Operation Code COMPOS.

This operation code establishes a new class the members of which include all the members of two or more other classes. A new class resulting from the use of this operation code will be called a composed class, in contrast to a non-composed class, i.e., a class resulting from the use of any other operation code.

The format of instructions with this operation code exactly parallels that of those used with the operation code NWCLS (cf. 5.111321). The name of the new composed class stands between the tab mark and the first comma, followed by the classes whose members will be included in the membership of the new composed class. The new composed class may include not only the members of non-composed classes but also those of other composed classes. Note that none of the class names or class members may include a period, comma, or percent sign (cf. 5.111321).

If we assume the existence of the non-composed classes start and end, generated as in 5.1212231, we can gain the ability to manipulate the entire name Harry by using the instruction COMPOS name,start,end. The members of the new non-composed class name would be H and arry (Note that the sequencing of the members of a composed class reflects the order in which its component classes are named in the COMPOS instruction.)

As a more meaningful example we may cite the instructional sequence

NWCLS	vowels, a,e,i,o,u.
NWCLS	liquids, r, l.
NWCLS	nasals, n, m.
COMPOS	vocalics, vowels, liquids.
COMPOS	sonorants, vocalics, nasals.

The composed class vocalics includes the members a, e, i, o, u, r, and l. The composed class sonorants includes all of these plus n and m.

5.1222 Deletion of Classes.

The following instructions permit the deletion of strings stored in core areas other than the item buffer. Since such strings are always associated with a non-composed class, their deletion also entails the deletion of the corresponding class name. Note that there is as yet no provision in EVC either for the deletion of composed classes or for their modification to reflect the deletion of non-composed classes which they include. It follows that the deletion instructions must be used with extreme care. On the other hand, the deletion feature permits the repeated use of one and the same class name with different memberships, i.e., it makes possible the storage of both variables and constants in the same manner.

5.12221 The Operation Code DELETE.

This operation code results in the deletion from core of an entire non-composed class--both the class members and the class name. The tab mark is immediately followed by the name of the class to be deleted. Thus, assuming that the instructions cited in 5.1212231 had been executed, the instruction DELETE end would result in the removal from core of both the class name end and the class member arry.

5.12222 The Operation Code DELSER.

This code parallels DELETE except that it deletes the names and memberships of all the non-composed classes constituting a series formed in accordance with 5.121212 above. Thus, if, after executing two of the SERNAM instructions discussed in 5.121212, we write DELSER label, both the class names label0 and label1 and their members would be deleted.

5.1223 Transference of Strings to Item Buffer.

5.12231 Transfer of Individual Class Members to the Item Buffer.

These instructions make possible the examination and manipulation of individual class members. They are applicable only to non-composed classes.

5.122311 The Operation Code LOOKAT.

This operation code transfers the first member of a given non-composed class to the item buffer. In addition, it sets a class-member pointer to the first character of the second member of the class, if any such member exists. The class name immediately follows the tab mark. Thus, if we assume the previous execution of the instruction NWCLS nouns,hat,dog, face. (cf. 5.111321), the instruction LOOKAT nouns places the characters hat in the item buffer.

5.122312 The Operation Code LKCLNX.

This code transfers to the item buffer that member of a given class which is indicated by the pointer mentioned in 5.122311. In addition it resets the class-member pointer to the first character of the next member of the class, if any such member exists. The format parallels that for 5.122311.

Thus, if we follow the LOOKAT instruction illustrated in 5.122311 with LKCLNX nouns, the string dog will be transferred to the item buffer. If we repeat the LKCLNX instruction the item buffer will contain face.

5.122313 The Operation Code LKCRCM.

This operation code also transfers to the item buffer a single member of the class whose name follows the tab mark. However, the member chosen is the member whose sequential position within the class corresponds to that of the last class member of any non-composed class transferred to the item buffer by either an LOOKAT or an LKCLNX instruction. Thus, if the class verbs has the membership run, go, jump, and if after manipulating the string face, placed in the item buffer by the second LKCLNX instruction mentioned in 5.122312, we write LKCRCM verbs, the member transferred to the item buffer will be jump, since it occupies the same sequential position within the class verbs, which face occupies within the class nouns.

5.122314 The Operation Code LKSRBG.

This operation code places in the item buffer the sole member of the first of a series of one-member classes generated by the use of the instruction SERNAM (cf. 5.121212 above). In addition, a series-member pointer is set to indicate the class, if any, which is the second member of the series in question. The format is the same as for the operation code SERNAM (cf. 5.121212). Thus, the instruction LKSRBG label will put the class member associated with the class name label0 in the item buffer.

5.122315 The Operation Code LKSRNX.

This operation code places in the item buffer the class member associated with that class of a series generated by the use of SERNAM instructions (cf. 5.121212) which is indicated by the series-member pointer mentioned in 5.122314. It also resets the pointer to indicate the next class in the series, if any such exists. Thus, if, after execution of the LKSRBG instruction cited in 5.122314, the program encounters

IKSRNX label, it will transfer to the item buffer the class member associated with the class name label1 and reset the series-member pointer to indicate the class name label2, if such a class exists.

5.122316 The Operation Code IKCRSM.

This code parallels IKCRCM (cf. 5.122313), except that it operates with classes included in series generated by SERNAM instructions (cf. 5.121212). Thus, if after the execution of the IKSRNX instruction cited in 5.122315 the program encounters the instruction IKCRSM word, it will place in the item buffer the class member associated with the class name word1 and set the series-member pointer to indicate the class name word2.

5.12232 Transfer of All Members of a Class to the Item Buffer.

5.122321 The Operation Code DUMP.

This operation code moves the entire membership of a non-composed class, with percentage marks separating the individual members, to the item buffer. The class name follows the tab mark. Thus, assuming the same class membership as that given in 5.122311, the instruction DUMP nouns places in the item buffer the sequence hat%dog%face.

5.12233 Transfer to the Item Buffer of the Membership of All Classes in a Series.

5.122331 The Operation Code JOINSR.

This operation code places in the item buffer, without intervening spaces, the membership of all the classes in a series generated by repeated SERNAM instructions (cf. 5.121212). The sequencing of class members corresponds to the ascending numerical order of the digits in the class names associated with them. Thus, if the class label1 has the member xy and the class label11 has the member a, and there is no class label12, the instruction JOINSR label places xya in the item buffer.

5.122332 The Operation Code SPCSPR.

This differs from JOINSR only in that individual class members are separated by a space. Its use in the example cited for 5.122331 places xy a in the item buffer.

5.13 Control of Program Flow.

5.131 Unconditional Transfer.

5.1311 Transfer to an Instruction Preceded by a Label.

A label in an EXEC program is a string of characters immediately preceded and followed by single hyphens. The initial hyphen must be immediately preceded by a line feed. While the terminating hyphen need not be immediately followed by a line feed, the program will ignore all

the characters which occur between that hyphen and the next line feed. (For examples of labels see 5.13111 and 5.13112.)

5.13111 The Operation Code GOTO.

This operation code causes the program to take as its next instruction the first instruction following a given label. The label, minus the preceding and following hyphens, is placed immediately after the tab mark. Thus, GOTO middle causes the program to take as its next instruction the instruction following the label -middle-.

5.13112 The Operation Code GOTOIT.

This operation code differs from GOTO in that the label is not specified in the instruction but is assumed to be already present in the item buffer. There are no other operands. Thus, if the item buffer contains the sequence middle the instruction GOTOIT will have the same effect as GOTO middle.

5.13113 The Operation Code VISIT.

This operation code differs from GOTO only in that it sets a return pointer to indicate the location of the next instruction following the VISIT instruction. The return pointer may then be utilized in connection with a GOHOME instruction (cf. 5.13121). Thus, the use of VISIT middle has the same immediate effect on program flow as the use of GOTO middle.

5.13114 The Operation Code VISITI.

This differs from VISIT (cf. 5.13113) just as GOTOIT (cf. 5.13112) differs from GOTO (cf. 5.13111). Thus, if the item buffer already contains the string middle, VISITI will have the same effect as VISIT middle.

5.1312 Transfer to an Instruction Preceded by a VISIT or VISITI Instruction.

5.13121 The Operation Code GOHOME.

This operation code causes the program to take as its next instruction the instruction indicated by the return pointer mentioned in 5.13113. It has no operands. Note that the use of GOHOME must be carefully coordinated with that of VISIT and VISITI.

5.1313 Omission of Instructions.

5.13131 The Operation Code SKIP.

This operation code causes the program to ignore a number (from 1 to decimal 29) of lines (instructions and/or labels) in selecting the next instruction to be executed. The number of lines to be ignored is specified by a decimal number following the tab mark. Thus, SKIP 3 causes the program to take as its next instruction the instruction on the fourth line following the SKIP instruction. (If that line is occupied

by a label the program will take its next instruction from the following line.)

5.132 Conditional Transfer.

Since all conditional-transfer instructions have operation codes beginning with the sequence IF, they will henceforth be referred to as IF instructions. All IF instructions cause the program to test for the existence of a given condition. Depending on the results of the test, the program either proceeds to the next instruction following the IF instruction or omits that instruction and transfers to the following instruction, i.e., behaves as if the IF instruction were the instruction SKIP 1 (cf. 5.13131).

5.1321 Transfer Conditional on the Results of a Comparison of the Contents of Item Buffer and a Literal.

5.13211 The Operation Code IFIDEN.

This operation code causes the program to omit the next instruction if the contents of the item buffer are not identical with a given literal. The literal, set off as in 5.1111211, follows the tab mark. Thus, if the item buffer contains the string Harry, the instruction IFIDEN -John- will result in the omission of the next instruction. If the item buffer does contain the string John, the next instruction will be executed.

The next instruction after an IFIDEN instruction is usually an unconditional transfer instruction (cf. 5.131). Thus, the instruction IFIDEN xJohnx might be immediately followed by GOTO middle so that the presence of the string John in the item buffer would result in a transfer to the instruction following the label -middle- (cf. 5.13111).

5.13212 The Operation Code IFINC.

This causes omission of the following instruction if the contents of the item buffer do not include a given literal. Format as in 5.13211. Thus, if the item buffer contains the string Johnny, the instruction IFINC /John/ would not result in an omitted instruction, although IFIDEN /John/ would.

5.132131 The Operation Code IFEND.

This causes omission of a line if sequence of characters in the item buffer including the final character are not identical with a given literal. Format as in 5.13211. Thus, IFEND -y- will not result in an omitted instruction whether the item buffer contains Harry or Johnny. However, IFEND -ry- will omit an instruction if the latter string is encountered.

5.13214 The Operation Code IFBE.

This differs from IFEND only in that the string in the item buffer must include the first character. Thus, IFBE /John/ will not omit an instruction if either John or Johnny is in the item buffer, but IFBE -John- will omit an instruction if it encounters the former.

5.1322 Transfer Conditional on Number of Characters in Item Buffer.

5.13221 The Operation Code IFLNGRTH.

This causes omission of an instruction if the item buffer does not contain more than a given number of characters. The number (in decimal notation) is given after the tab. Thus, IFLNGRTH 5 will omit an instruction if the item buffer contains John or Harry, but not if it contains Johnny.

5.13222 The Operation Code IFSHTRTH.

This omits an instruction if the item buffer does not contain fewer than a given number of characters. Format as in 5.13221. Thus, IFSHTRTH 5 omits a line if the item buffer contains Harry or Johnny, but not if it contains Jonn.

5.13223 The Operation Code IFEQLGTH.

This omits a line if the item buffer does not contain a given number of characters. Format as in 5.13221. IFEQLGTH 5 omits an instruction if either John or Johnny is encountered, but not if the item buffer contains Harry.

5.133 Transfer Conditional on the Results of a Comparison of the Contents of the Item Buffer with One or More Class Members.

5.1331 The Operation Code IFMEM.

This operation code omits an instruction if the contents of the item buffer are not identical with one of the members of a given class (composed or non-composed). The class name follows the tab mark. Thus, assuming the same class membership as in 5.122311, the instruction IFMEM nouns will omit a line if the item buffer does not contain one (and only one) of the three strings, dog, face, hat.

5.1332 The Operation Code IFMBSR.

This differs from IFMEM only in that the contents of the item buffer are compared with the membership of each of the classes constituting a series generated by the use of SERNAM instructions (cf. 5.121212). Thus, IFMBSR label, assuming the same classes and membership as in 5.122331, will omit an instruction unless the item buffer contains either a or xy.

5.1333 The Operation Code IFMEXTH.

This operation code omits an instruction if the item buffer does not include a larger number of characters than the number found in the first member of a given non-composed class. Format as in 5.1331. Thus, IFMEXTH nouns, assuming the same membership as in 5.122311, will omit an instruction if the item buffer contains less than four characters since hat, the first member of the class in question, contains three.

5.1334 The Operation Code IFLSEXTH.

This differs from IFMREXTH only in that an instruction is omitted if the item buffer does not contain fewer characters than the first member of the non-composed class. Thus, assuming the same membership as in 5.122311, IFLSEXTH nouns will omit an instruction if the item buffer contains more than two characters.

5.1335 The Operation Code IFEQEX.

This differs from IFMREXTH only in that an instruction is omitted if the item buffer and the first member of the non-composed class do not contain the same number of characters. Thus, in the example cited in 5.1334, IFEQEX nouns will omit a line if the input buffer contains either more or less than three characters.

5.1336 The Operation Code IFLSTH.

This operation code omits an instruction if the first ASCII character in the item buffer is not numerically less than the first ASCII character of the first member of a given non-composed class. Thus, assuming the same membership as in 5.122311, IFLSTH nouns will omit a line if the first ASCII character in the item has an octal value of 150 or more, since octal 150 is the value of ASCII h, the first character of hat.

5.1337 The Operation Code IFGRTH.

This differs from IFLSTH only in that an instruction is omitted if the first ASCII character in the input buffer is not numerically greater than the first character of the first member of a given non-composed class. Thus, in the example cited in 5.1336, IFGRTH nouns will omit a line if the first ASCII character in the item buffer has an octal value of 150 or less.

5.134 Transfer Conditional on Existence of a Given Non-Composed Class.

5.1341 The Operation Code IFEXST.

This operation code omits a line if a given non-composed class is not currently stored in core. The class name follows the tab mark. Thus, assuming only the classes mentioned in 5.122311, IFEXST nouns will not omit an instruction, while IFXST adjs will.

5.1342 The Operation Code IFISNO.

This differs from IFEXT only in that the omission of an instruction takes place if a given non-composed class is currently in core. Thus, in the example cited in 5.1341, IFISNO nouns omits an instruction, while IFISNO adjs does not.

5.14 Output.

Output is always onto a disk file attached to channel 3. For this reason we will refer to the output file as file 3.

5.141 Output of Contents of Item Buffer.

5.1411 The Operation Code PUTIT.

This operation code causes the entire current contents of the item buffer to be written on file 3. No operands are used. Thus, if after the instructions LKLIT abc (cf. 5.11131) and KILFRS 1 (cf. 5.121112) the program encounters a PUTIT instruction, the string bc will be written on file 3.

5.142 Output of Class Membership.

5.1421 The Operation Code PUTALL.

This operation code causes the program to write on file 3 the entire membership of any class--composed or non-composed--with a carriage return following each class member. The class name follows the tab mark. Thus, PUTALL nouns, assuming the same membership as in 5.122311, causes the program to write on file 3 the sequence

hat
face
dog

5.1422 The Operation Code PUTSER.

This results in the writing on file 3 of the class members corresponding to all of the classes in a series generated by repeated SERNAM instructions (cf. 5.121212). Each class member is followed by a carriage return. Thus, assuming the same classes and class members as in 5.122331, the instruction PUTSER label writes on file 3

xy
a

5.143 The Output of Literals.

5.1431 The Operation Code PUTLIT.

This operation code causes a string of characters following the tab mark and set off as in 5.1111211, to be written on file 3. Thus, PUTLIT -abc- writes abc on file 3.

5.15 Program Delimiters.

5.151 The Operation Code BEG.

The first instruction of any EVC program must begin the operation code BEG. The tab mark may, but need not, be followed by additional material. This material must not extend beyond the first line feed following BEG.

5.152 The Operation Code FIN.

The last instruction of any EVC program must begin with this operation code. Program operation is normally terminated by a GOTO instruction (cf. 5.1511) which leads to the execution of the FIN instruction. Material following the tab mark of the FIN instruction is ignored.

6. Program for the Conversion of Assertions Concerning a Target Language to a Coded Format.

6.0 Justification and Purpose.

6.01 Archetypal Assertions as a Substitute for a Programming Language.

One of the major obstacles to the utilization of computational techniques by language teachers has been the need to master a programming language. Such languages, no matter how user-oriented, always entail the assimilation of new modes of discourse about a familiar subject, a task which many find difficult or insurmountable. It would, therefore, seem desirable to make it possible for language teachers (and other scholars as well) to interact effectively with a computing system "on their own terms," i.e. without the previous acquisition of a system of discourse which differs markedly from that which they are accustomed to employ.

It would appear that a satisfactory solution to this problem entails an approach to the problem of discourse systems diametrically opposed to that inherent in the programming language approach. Instead of forcing the language teacher to master a new lexicon and grammar, we must analyze the utterances which characterize his own discourse in the hope of deriving therefrom a finite (if, perhaps, large) set of archetypal assertions, capable of conveying all the information which a language teacher normally wishes to communicate. If we are successful in this task we can hope to establish effective communication between the language teacher and the computing system simply by requiring the former to 1) formulate all statements intended for computer consumption within the framework of the archetypal assertions included in our set, and 2) supply the variables required to convert a given archetypal assertion to a statement applicable to the language or languages he is dealing with. (Such a statement will hereafter be called a concrete assertion.)

To illustrate the isolation of an archetypal assertion let us consider the following statements:

1) Russian stressed Y is pronounced like the vowel of English 'toot'.
2) The closest English sound to that of stressed Russian Y is the vowel sound heard in words like 'foot'.
3) To pronounce Russian stressed Y place the lips and tongue in approximately the same position as that for the vowel of English 'boot'.

We submit that each of the above statements is adequately covered by the following assertion: The sound represented by Russian stressed Y is very similar to the sound represented by the oo of English 'toot'.

This statement may in turn be converted to the archetypal assertion: The sound represented by TARGET LANGUAGE VARIABLE is very similar to that represented by BASE LANGUAGE VARIABLE.

It would appear that a language teacher wishing to communicate to a computational system the information conveyed by any of our three original utterances would 1) easily recognize the equivalence of the original utterance and our archetypal assertion, and 2) find no difficulty in supplying the variables needed for the formulation of a concrete assertion.

6.02 The Value of a Converter Program.

If a concrete assertion is to serve as a useful tool in the generation of programmed instructional material it is necessary to have some convenient means of 1) distinguishing the archetypal portion of the assertion from the variables accompanying it, and 2) retrieving the individual variables whenever they may be needed in the generation of instructional frames. It follows that it will be useful to convert a concrete assertion from its original formulation to 1) a string of characters corresponding to the archetypal portion of the instruction and constituting an operation code to be utilized in calling computer routines used to generate the appropriate instructional frames, and 2) a set of literals corresponding to the variables of the concrete assertion and each associated with a single label utilized in the frame-generation routines.

6.1 Documentation.

6.11 Function of the Program.

(See 6.01 and 6.02.)

6.12 File Format.

6.121 Input File.

The input consists of a disk file (channel 2) containing one or more concrete assertions.

6.1211 Use of the Tab Mark plus a Delimitative Character to Indicate the Beginning of a Concrete Assertion.

Each concrete assertion must begin with a tab mark. The tab mark must be immediately followed by a delimitative character. This may be any ASCII character other than the tab mark which does not occur within the assertion in a non-delimitative function. However, it is generally more convenient to use a character such as a slash or a hyphen than to employ an alphanumeric character.

6.1212 The Delimination of Variables within a Concrete Assertion.

Variables within a concrete assertion are set off by a preceding and a following single occurrence of the delimitative character specified at the beginning of the assertion. Spaces immediately preceding or following the delimitative character are ignored.

6.1213 Use of the Delimitative Character to Terminate a Concrete Assertion.

The last non-delimitative character of a concrete assertion is immediately followed by an uninterrupted sequence of two delimitative characters. If the terminal non-delimitative character is part of a variable, the delimitative character which signals the end of the variable also serves as the first of the two characters used to terminate the assertion. A period preceding the terminating characters is ignored.

6.1214 Example of Concrete Construction with Variables.

6.12141 Hyphen Used as Delimitative Character.

-The sound represented by -Russian stressed Y- is very similar to that represented by -the English vowel in 'toot'.--

(In this assertion the variables are Russian stressed Y and the English vowel in 'toot'.)

6.12142 Slash Used as Delimitative Character.

/The sound represented by /Russian stressed Y/ is very similar to that represented by /the English vowel in 'toot'.//

6.122 Output File.

The output consists of a disk file (channel 3) containing the initial portion of an EVO program including one NCHNM instruction (cf. 5.111322) for each of the concrete assertions of the input file.

6.1221 Delimitative Character.

The delimitative character for the NCHNM instruction (cf. 5.111322) is identical with that for the concrete assertion.

6.1222 Class Name.

The class name for each NCHNM instruction consists of the initial letters of the words which make up the non-variable portion of the concrete assertion corresponding to the instruction. Thus, for the example cited under 6.1214 the class name is rsrbivstvrb.

6.1223 Class Members.

The individual class members for each NCHNM instruction consist of the individual variables included in the corresponding concrete assertion. Thus, for the example cited under 6.1214 the class members are the strings Russian stressed Y and the English vowel in toot.

6.1224 Example of Output Corresponding to the Input Illustrated under 6.1214.

6.12241 With Delimitative Character as in 6.12141.

NCLNIM -tsrbivsttrb-Russian stressed Y-the English vowel in 'toot'--

6.12242 With Delimitative Character as in 6.12142.

NCLNIM /tsrbivsttrb/Russian stressed Y/the English vowel in 'toot'//

6.2 Evaluation and Prospects for Future Development.

The program documented under 6.1 provides material which could be used as input for more sophisticated EVC programs designed to generate instructional frames. Thus, the example cited under 6.1224 could be converted to a presentation frame, consisting of the archetypal assertion itself, and two fill-in frames, one calling for the student to fill in the first variable, the other calling for the second variable.

It would appear that a carefully organized set of archetypal assertions would make possible a much more sophisticated set of frame generation routines. Thus, if we changed the example cited under 6.1214 to -The sound represented by -Russian- stressed -Y- is very similar to that represented by the -oo- in the -English- word -toot--, the more discrete nature of the variables would permit a much wider variety of reinforcement frames.

7. Conclusions.

The work carried out during the contract period resulted in the development of a number of research tools which, it is to be hoped, will prove useful in future work on the automatic generation of materials for programmed language instruction. Thus, the programs and notations discussed in sections 1 through 3 should prove helpful in the automatic generation of a wide variety of phrases and/or utterances utilizing a given vocabulary item. The program discussed in section 4 facilitates the input of information necessary for the successful application of such generative techniques to a new vocabulary. The EVC language discussed in section 5 will, it is hoped, greatly increase the speed with which new programs can be written and debugged. Finally, the EVC program discussed in section 6 allows language teachers to prepare input for computer processing without learning a new mode of discourse.

While it is obvious that many problems must be overcome in developing automatically generated programmed language-instructional materials, it would seem that we can now proceed to the investigation of such problems in a much more efficient manner than was previously possible.

Distribution List

OFFICE OF NAVAL RESEARCH

PERSONNEL AND TRAINING RESEARCH PROGRAMS (CODE 458)

DISTRIBUTION LIST

CONTRACT NO. N00014-67-A-0112-0042 CONTRACTOR Dr. Joseph A. Van Campen

NY

Chief of Naval Research
Code 458
Department of the Navy
Washington, D. C. 20360
*(All)

Director
ONR Branch Office
1030 East Green Street
Pasadena, California 91101
(All)

Director, Naval Research Laboratory
Washington, D. C. 20390
ATTN: Library, Code 2029 (ONRL)
(All)

Office of Naval Research
Area Office
1076 Mission Street
San Francisco, California 94103
(All)

Director
Naval Research Laboratory
Washington, D. C. 20390
ATTN: Technical Information Division
(All)

Defense Documentation Center
Cameron Station, Building 5
5010 Duke Street
Alexandria, Virginia 22314
(All)

Commanding Officer
Service School Command
U. S. Naval Training Center
San Diego, California 92133
(All)

3 Commanding Officer
Naval Personnel and Training
Research Laboratory
San Diego, California 92152
(All)

1 Commanding Officer
Naval Medical Neuropsychiatric
Research Unit
San Diego, California 92152
(12346)

1 Commanding Officer
Naval Air Technical Training Center
Jacksonville, Florida 32213
(2356)

1 Dr. James J. Regan, Code 55
Naval Training Device Center
Orlando, Florida 32813
(All)

1 Technical Library
U. S. Naval Weapons Laboratory
Dahlgren, Virginia 22448
(All)

1 Research Director, Code 06
Research and Evaluation Department
U. S. Naval Examining Center
Building 2711 - Green Bay Area
Great Lakes, Illinois 60088
ATTN: C. S. Winiewicz
(All)

1 Chairman
Behavioral Science Department
Naval Command and Management Division
U. S. Naval Academy, Luce Hall
Annapolis, Maryland 21402
(All)

*In reproducing the addresses in this list, delete the information in parentheses that follows the address. This information is for CNR use only.

1 Dr. A. L. Slafkosky
Scientific Advisor (Code AX)
Commandant of the Marine Corps
Washington, D. C. 20380
(All)

1 Director
Aerospace Crew Equipment Department
Naval Air Development Center
Johnsville
Warminster, Pennsylvania 18974
(1346)

1 Chief
Naval Air Technical Training
Naval Air Station
Memphis, Tennessee 38115
(All)

1 Director
Education and Training Sciences Dept.
Naval Medical Research Institute
National Naval Medical Center
Building 142
Bethesda, Maryland 20014
(All)

1 Commander
Submarine Development Group TWO
Fleet Post Office
New York, New York 09501
(16)

1 LCDR J. C. Meredith, USN (Ret.)
Institute of Library Research
University of California
Berkeley, California 94720
(5)

1 Commander
Operational Test & Evaluation Force
U. S. Naval Base
Norfolk, Virginia 23511
(All)

1 Office of Civilian Manpower Management
Technical Training Branch (Code 024)
Department of the Navy
Washington, D. C. 20390
(1256)

1 Chief of Naval Operations, (Op-07TL)
Department of the Navy
Washington, D. C. 20350
(All)

1 Chief of Naval Material (MAT 031M)
Room 1323, Main Navy Building
Washington, D. C. 20360
(All)

1 Mr. George N. Graine
Naval Ship Systems Command (SHIPS 03H)
Department of the Navy
Washington, D. C. 20360
(All)

1 Chief
Bureau of Medicine and Surgery
Code 513
Washington, D. C. 20390
(All)

6 Technical Library (Pers-11b)
Bureau of Naval Personnel
Department of the Navy
Washington, D. C. 20370
(All)

3 Personnel Research and Development
Laboratory
Washington Navy Yard, Building 200
Washington, D. C. 20390
ATTN: Library, Room 3307
(12345)

1 Commander, Naval Air Systems Command
Navy Department, AIR-4132
Washington, D. C. 20360
(23456)

1 Commandant of the Marine Corps
Headquarters, U. S. Marine Corps
Code AO1B
Washington, D. C. 20380
(All)

1 Technical Library
Naval Ship Systems Command
Main Navy Building, Room 1532
Washington, D. C. 20360
(125)

1 Mr. Philip Rochlin, Head
Technical Library Branch
Naval Ordnance Station
Indian Head, Maryland 20640
(All)

1 Library, Code 0212
Naval Postgraduate School
Monterey, California 93940
(All)

1 Technical Reference Library
Naval Medical Research Institute
National Naval Medical Center
Bethesda, Maryland 20014
(All)

1 Technical Library
Naval Ordnance Station
Louisville, Kentucky 40214
(45)

1 Naval Undersea Research and
Development Center
3202 East Foothill Boulevard
Pasadena, California 91107
ATTN: Code 161
(5)

1 Commanding Officer
U. S. Naval Schools Command
Mare Island
Vallejo, California 94592
(5)

1 Scientific Advisory Team (Code 71)
Staff, COMASWFORLANT
Norfolk, Virginia 23511
(All)

1 Education & Training Developments Staff
Personnel Research & Development Lab.
Washington Navy Yard, Building 200
Washington, D. C. 20390
(All)

1 Dr. Don H. Coombs, Co-Director
ERIC Clearinghouse
Stanford University
Palo Alto, California 94305
(156)

1 ERIC Clearinghouse on
Educational Media and Technology
Stanford University
Stanford, California 94305
(156)

1 ERIC Clearinghouse on Vocational
and Technical Education
The Ohio State University
1900 Kenny Road
Columbus, Ohio 43210
ATTN: Acquisition Specialist
(135)

1 LTCOL F. R. Ratliff
Office of the Assistant Secretary
of Defense (M&RU)
The Pentagon, Room 3D960
Washington, D. C. 20301
(All)

1 Dr. Ralph R. Canter
Military Manpower Research Coordinator
OASD (M&RA) MR&U
The Pentagon, Room 3D960
Washington, D. C. 20301
(All)

1 Deputy Director
Office of Civilian Manpower Management
Department of the Navy
Washington, D. C. 20390
(All)

1 Chief, Naval Air Reserve Training
Naval Air Station
Box 1
Glenview, Illinois 60026
(25)

1 Technical Library
Naval Training Device Center
Orlando, Florida 32813
(All)

ARMY

1 Director
Human Resources Research Organization
300 North Washington Street
Alexandria, Virginia 22314
(All)

1 Human Resources Research Organization
Division #1, Systems Operations
300 North Washington Street
Alexandria, Virginia 22314
(All)

1 Human Resources Research Organization
Division #3, Recruit Training
Post Office Box 5787
Presidio of Monterey, California 93940
ATTN: Library
(All)

1 Human Resources Research Organization
Division #4, Infantry
Post Office Box 2086
Fort Benning, Georgia 31905
(All)

1 Human Resources Research Organization
Division #5, Air Defense
Post Office Box 6021
Fort Bliss, Texas 79916
(All)

1 Human Resources Research Organization
Division #6, Aviation
Post Office Box 428
Fort Rucker, Alabama 36360
(All)

1 Commandant
U. S. Army Adjutant General School
Fort Benjamin Harrison, Indiana 46216
ATTN: ATSAG-EA
(All)

1 Director of Research
U. S. Army Armor Human Research Unit
Fort Knox, Kentucky 40121
ATTN: Library
(All)

1 Armed Forces Staff College
Norfolk, Virginia 23511
ATTN: Library
(235)

1 Director
Behavioral Sciences Laboratory
U. S. Army Research Institute
of Environmental Medicine
Natick, Massachusetts 01760
(All)

1 Chief, Training and Development
Division
Office, Deputy Chief of Staff
for Personnel
Department of the Army
Washington, D. C. 20310
(5)

1 U. S. Army Behavior and Systems
Research Laboratory
Commonwealth Building, Room 239
1320 Wilson Boulevard
Arlington, Virginia 22209
(1246)

1 Division of Neuropsychiatry
Walter Reed Army Institute
of Research
Walter Reed Army Medical Center
Washington, D. C. 20012
(All)

1 Behavioral Sciences Division
Office of Chief of Research and
Development
Department of the Army
Washington, D. C. 20310
(All)

AIR FORCE

1 Director
Air University Library
Maxwell Air Force Base, Alabama 36112
ATTN: AUL-8110
(23456)

1 Headquarters, Electronic Systems Division
ATTN: Dr. Sylvia Mayer / ESMDA
L. G. Hanscom Field
Bedford, Massachusetts 01730
(23456)

1 Commandant
U. S. Air Force School of Aerospace Medicine
ATTN: Aeromedical Library (SMSL-4)
Brooks Air Force Base, Texas 78235
(All)

1 AFHRL (TR/Dr. G. A. Eckstrand)
Wright-Patterson Air Force Base
Ohio 45433
(13456)

1 Personnel Research Division (AFHRL)
Lackland Air Force Base
San Antonio, Texas 78236
(All)

1 AFOSR(SRLB)
1400 Wilson Boulevard
Arlington, Virginia 22209
(All)

1 Headquarters, U. S. Air Force
Chief, Personnel Research and Analysis Division (AFPDPL)
Washington, D. C. 20330
(2345)

1 Headquarters, U. S. Air Force
AFPTRBD
Programs Resources and Technology Div.
Washington, D. C. 20330
(All)

1 AFHRL (HRTT/Dr. Ross L. Morgan)
Wright-Patterson Air Force Base
Ohio 45433
(1456)

1 Dr. Alvin E. Goins, Executive Secretary
Personality and Cognition Research Review Committee
Behavioral Sciences Research Branch
National Institute of Mental Health
5454 Wisconsin Avenue, Room 10A02
Chevy Chase, Maryland 20015
(12456)

1 Office of Computer Information
Center for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234
(2456)

2 Executive Secretariat
Interagency Committee on Manpower Research
1111 Twentieth Street, N. W., Room 251-A
Washington, D. C. 20036
(235)

1 Mr. Joseph J. Cowan, Chief
Psychological Research Branch (P-1)
U. S. Coast Guard Headquarters
400 Seventh Street, S. W.
Washington, D. C. 20226
(All)

1 Executive Officer
American Psychological Association
1200 Seventeenth Street, N. W.
Washington, D. C. 20036
(All)

1 Dr. Bernard M. Bass
University of Rochester
Management Research Center
Rochester, New York 14627
(12345)

1 Mr. Edmund C. Berkeley
Computers and Automation
815 Washington Street
Newtonville, Massachusetts 02160
(5)

1 Dr. Donald L. Bitzer
Computer-Based Education Research Laboratory
University of Illinois
Urbana, Illinois 61801
(1456)

1 Dr. C. Victor Bunderson
Computer Assisted Instruction Laboratory
University of Texas
Austin, Texas 78712
(56)

1 Dr. Lee J. Cronbach
School of Education
Stanford University
Stanford, California 94305
(12456)

1 Dr. F. J. DiVesta
Pennsylvania State University
320 Rackley Building
University Park, Pennsylvania 16802
(56)

1 Dr. Robert Dubin
Graduate School of Administration
University of California
Irvine, California 02650
(2345)

1 Dr. Philip H. DuBois
Department of Psychology
Washington University
Lindell & Skinker Boulevards
St. Louis, Missouri 63130
(15)

1 Dr. Marvin D. Dunnette
University of Minnesota
Department of Psychology
Elliot Hall
Minneapolis, Minnesota 55455
(12345)

1 Mr. Wallace Feurzeig
Bolt, Beranek and Newman, Inc.
50 Moulton Street
Cambridge, Massachusetts 02138
(56)

1 S. Fisher, Research Associate
Computer Facility
Graduate Center
City University of New York
33 West 42nd Street
New York, New York 10036
(156)

1 Dr. John C. Flanagan
American Institutes for Research
Post Office Box 1113
Palo Alto, California 94302
(All)

1 Dr. Robert Glaser
Learning Research and
Development Center
University of Pittsburgh
Pittsburgh, Pennsylvania 15213
(1456)

1 Dr. Albert S. Glickman
American Institutes for Research
8555 Sixteenth Street
Silver Spring, Maryland 20910
(All)

1 Dr. Bert Green
Department of Psychology
Johns Hopkins University
Baltimore, Maryland 21218
(156)

1 Dr. Duncan N. Hansen
Center for Computer Assisted
Instruction
Florida State University
Tallahassee, Florida 32306
(All)

1 Dr. M. D. Havron
Human Sciences Research, Inc.
Westgate Industrial Park
7710 Old Springhouse Road
McLean, Virginia 22101
(145)

1 Dr. Carl E. Helm
Department of Educational Psychology
Graduate Center
City University of New York
33 West 42nd Street
New York, New York 10036
(156)

1 Dr. Albert E. Hickey
Entelek, Incorporated
42 Pleasant Street
Newburyport, Massachusetts 01950
(456)

1 Dr. Robert R. Mackie
Human Factors Research, Inc.
Santa Barbara Research Park
6780 Cortona Drive
Goleta, California 93017
(All)

1 Dr. Richard Myrick, President
Performance Research, Inc.
919 Eighteenth Street, N. W.,
Suite 425
Washington, D. C. 20036
(All)

1 Dr. Gabriel D. Ofiesh
Center for Educational Technology
Catholic University
4001 Harewood Road, N. E.
Washington, D. C. 20017
(156)

1 Mr. Luigi Petrullo
2431 North Edgewood Street
Arlington, Virginia 22207
(All)

1 Dr. Len Rosenbaum
Psychology Department
Montgomery College
Rockville, Maryland 20852
(All)

1 Dr. Arthur I. Siegel
Applied Psychological Services
Science Center
404 East Lancaster Avenue
Wayne, Pennsylvania 19087
(All)

1 Dr. Paul Slovic
Oregon Research Institute
Post Office Box 3196
Eugene, Oregon 97403
(12456)

1 Dr. Arthur W. Staats
Department of Psychology
University of Hawaii
Honolulu, Hawaii 96822
(56)

1 Dr. Benton J. Underwood
Department of Psychology
Northwestern University
Evanston, Illinois 60201
(56)

1 Dr. John Annett
Department of Psychology
Hull University
Hull
Yorkshire, England
(All)

1 Dr. M. C. Shelesnyak
Interdisciplinary Communications
Program
Smithsonian Institution
1025 Fifteenth Street, N. W.,
Suite 700
Washington, D. C. 20005
(1456)

1 Dr. Joseph W. Rigney
Behavioral Technology Laboratories
University of Southern California
University Park
Los Angeles, California 90007
(All)

1 Educational Testing Service
Division of Psychological Studies
Rosedale Road
Princeton, New Jersey 08540
(All)

1 Dr. Harold Gulliksen
Department of Psychology
Princeton University
Princeton, New Jersey 08540
(56)

1 Dr. George E. Rowland
Rowland and Company, Inc.
Post Office Box 61
Haddonfield, New Jersey 08033
(All)

1 Dr. Mats Bjorkman
University of Umea
Department of Psychology
Umea 6, Sweden
(156)

1 Dr. Howard H. Kendler
Department of Psychology
University of California
Santa Barbara, California 93106
(56)

PERSONNEL AND TRAINING RESEARCH PROGRAMS DISTRIBUTION LIST - CHANGE I

ADD to May 1970 Distribution List

1 Technical Services Division
National Library of Medicine
8600 Rockville Pike
Bethesda, Maryland 20014
(126)

1 Head, Aerospace Psychology Department
Naval Aerospace Medical Research
Laboratory
Naval Aerospace Medical Institute
Naval Aerospace Medical Center
Pensacola, Florida 32512
(12456)

1 AFHRL (DOI)
Brooks Air Force Base
Texas 78235
(All)

1 Mr. Emil Jean Caille
Center for Research Studies
and Applied Psychology
Ministry of Army
Marine Arsenal
Toulon, France
(56)

1 Dr. Norman Kerr
Bureau of Naval Personnel (PERS-A321)
Washington, D. C. 20370

(Continued from inside front cover)

96 R. C. Atkinson, J. W. Brelsford, and R. M. Shiffrin. Multi-process models for memory with applications to a continuous presentation task. April 13, 1966. (J. math. Psychol., 1967, 4, 277-300).
97 P. Suppes and E. Crothers. Some remarks on stimulus-response theories of language learning. June 12, 1966.
98 R. Bjork. All-or-none subprocesses in the learning of complex sequences. (J. math. Psychol., 1968, 1, 182-195).
99 E. Gammon. The statistical determination of linguistic units. July 1, 1966.
100 P. Suppes, L. Hyman, and M. Jerman. Linear structural models for response and latency performance in arithmetic. (In J. P. Hill (ed.), Minnesota Symposia on Child Psychology. Minneapolis, Minn.: 1967. Pp. 160-200).
101 J. L. Young. Effects of intervals between reinforcements and test trials in paired-associate learning. August 1, 1966.
102 H. A. Wilson. An investigation of linguistic unit size in memory processes. August 3, 1966.
103 J. T. Townsend. Choice behavior in a cued-recognition task. August 8, 1966.
104 W. H. Batchelder. A mathematical analysis of multi-level verbal learning. August 9, 1966.
105 H. A. Taylor. The observing response in a cued psychophysical task. August 10, 1966.
106 R. A. Bjork. Learning and short-term retention of paired associates in relation to specific sequences of interpresentation intervals. August 11, 1966.
107 R. C. Atkinson and R. M. Shiffrin. Some Two-process models for memory. September 30, 1966.
108 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the third year. January 30, 1967.
109 P. Suppes and I. Rosenthal-Hill. Concept formation by kindergarten children in a card-sorting task. February 27, 1967.
110 R. C. Atkinson and R. M. Shiffrin. Human memory: a proposed system and its control processes. March 21, 1967.
111 Theodore S. Rodgers. Linguistic considerations in the design of the Stanford computer-based curriculum in initial reading. June 1, 1967.
112 Jack M. Knutson. Spelling drills using a computer-assisted instructional system. June 30, 1967.
113 R. C. Atkinson. Instruction in initial reading under computer control: the Stanford Project. July 14, 1967.
114 J. W. Brelsford, Jr. and R. C. Atkinson. Recall of paired-associates as a function of overt and covert rehearsal procedures. July 21, 1967.
115 J. H. Stelzer. Some results concerning subjective probability structures with semiorders. August 1, 1967
116 D. E. Rumelhart. The effects of interpresentation intervals on performance in a continuous paired-associate task. August 11, 1967.
117 E. J. Fishman, L. Keller, and R. E. Atkinson. Massed vs. distributed practice in computerized spelling drills. August 18, 1967.
118 G. J. Groen. An investigation of some counting algorithms for simple addition problems. August 21, 1967.
119 H. A. Wilson and R. C. Atkinson. Computer-based instruction in initial reading: a progress report on the Stanford Project. August 25, 1967.
120 F. S. Roberts and P. Suppes. Some problems in the geometry of visual perception. August 31, 1967. (Synthese, 1967, 17, 173-201)
121 D. Jamison. Bayesian decisions under total and partial ignorance. D. Jamison and J. Kozielecki. Subjective probabilities under total uncertainty. September 4, 1967.
122 R. C. Atkinson. Computerized instruction and the learning process. September 15, 1967.
123 W. K. Estes. Outline of a theory of punishment. October 1, 1967.
124 T. S. Rodgers. Measuring vocabulary difficulty: An analysis of item variables in learning Russian-English and Japanese-English vocabulary pairs. December 18, 1967.
125 W. K. Estes. Reinforcement in human learning. December 20, 1967.
126 G. L. Wolford, D. L. Wessel, W. K. Estes. Further evidence concerning scanning and sampling assumptions of visual detection models. January 31, 1968.
127 R. C. Atkinson and R. M. Shiffrin. Some speculations on storage and retrieval processes in long-term memory. February 2, 1968.
128 John Holmgren. Visual detection with imperfect recognition. March 29, 1968.
129 Lucille B. Mlodnosky. The Frostig and the Bender Gestalt as predictors of reading achievement. April 12, 1968.
130 P. Suppes. Some theoretical models for mathematics learning. April 15, 1968. (Journal of Research and Development in Education, 1967, 1, 5-22)
131 G. M. Olson. Learning and retention in a continuous recognition task. May 15, 1968.
132 Ruth Norene Hartley. An investigation of list types and cues to facilitate initial reading vocabulary acquisition. May 29, 1968.
133 P. Suppes. Stimulus-response theory of finite automata. June 19, 1968.
134 N. Moler and P. Suppes. Quantifier-free axioms for constructive plane geometry. June 20, 1968. (In J. C. H. Gerretsen and F. Oort (Eds.), Compositio Mathematica. Vol. 20. Groningen, The Netherlands: Wolters-Noordhoff, 1968. Pp. 143-152.)
135 W. K. Estes and D. P. Horst. Latency as a function of number of response alternatives in paired-associate learning. July 1, 1968.
136 M. Schlag-Rey and P. Suppes. High-order dimensions in concept identification. July 2, 1968. (Psychon. Sci., 1968, 11, 141-142)
137 R. M. Shiffrin. Search and retrieval processes in long-term memory. August 15, 1968.
138 R. D. Freund, G. R. Loftus, and R.C. Atkinson. Applications of multiprocess models for memory to continuous recognition tasks. December 18, 1968.
139 R. C. Atkinson. Information delay in human learning. December 18, 1968.
140 R. C. Atkinson, J. E. Holmgren, and J. F. Juola. Processing time as influenced by the number of elements in the visual display. March 14, 1969.
141 P. Suppes, E. F. Loftus, and M. Jerman. Problem-solving on a computer-based teletype. March 25, 1969.
142 P. Suppes and Mona Morningstar. Evaluation of three computer-assisted instruction programs. May 2, 1969.
143 P. Suppes. On the problems of using mathematics in the development of the social sciences. May 12, 1969.
144 Z. Domotor. Probabilistic relational structures and their applications. May 14, 1969.
145 R. C. Atkinson and T. D. Wickens. Human memory and the concept of reinforcement. May 20, 1969.
146 R. J. Titiev. Some model-theoretic results in measurement theory. May 22, 1969.
147 P. Suppes. Measurement: Problems of theory and application. June 12, 1969.
148 P. Suppes and C. Ihrke. Accelerated program in elementary-school mathematics--the fourth year. August 7, 1969.
149 D. Rundus and R.C. Atkinson. Rehearsal in free recall: A procedure for direct observation. August 12, 1969.
150 P. Suppes and S. Feldman. Young children's comprehension of logical connectives. October 15, 1969.

(Continued on back cover)

(Continued from inside back cover)

151 Joaquim H. Laubsch. An adaptive teaching system for optimal item allocation. November 14, 1969.

152 Roberta L. Klatzky and Richard C. Atkinson. Memory scans based on alternative test stimulus representations. November 25, 1969.

153 John E. Holmgren. Response latency as an indicant of information processing in visual search tasks. March 16, 1970.

154 Patrick Suppes. Probabilistic grammars for natural languages. May 15, 1970.

155 E. Gammon. A syntactical analysis of some first-grade readers. June 22, 1970.

156 Kenneth N. Wexler. An automaton analysis of the learning of a miniature system of Japanese. July 24, 1970.

157 R. C. Atkinson and J. A. Paulson. An approach to the psychology of instruction. August 14, 1970.

158 R.C. Atkinson, J.D. Fletcher, H.C. Chetin, and C. M. Stauffer. Instruction in initial reading under computer control: the Stanford project. August 13, 1970.

159 Dewey J. Rundus. An analysis of rehearsal processes in free recall. August 21, 1970.

160 R.L. Klatzky, J.F. Juola, and R.C. Atkinson. Test stimulus representation and experimental context effects in memory scanning.

161 William A. Rottmayer. A formal theory of perception. November 13, 1970.

162 Elizabeth Jane Fishman Loftus. An analysis of the structural variables that determine problem-solving difficulty on a computer-based teletype. December 18, 1970.

163 Joseph A. Van Campen. Towards the automatic generation of programmed foreign-language instructional materials. January 11, 1971.