MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

AN ASSOCIATIVE PROCESSING SYSTEM FOR CONVENTIONAL DIGITAL COMPUTERS

P. D. ROVNER

J. A. FELDMAN

Group 23

TECHNICAL NOTE 1967-19

21 APRIL 1967

LEXINGTON MASSACHUSETTS

PAGES _____

ARE

MISSING

IN

ORIGINAL

DOCUMENT

BSTRACT

A user-oriented system having both algebraic and associative processing capabilities is presented in this report. The algebraic capabilities are essentially those of ALGOL. The associative facilities are:

(1) A language for the expression of associative retrieval requests (the associative language).

(2) A scheme for the internal representation of a store of associations between items of information (an associative information base).

(3) Processing routines for associative retrieval requests.

The associative language is independent of the structure of the associative information base. In the system presented here, the associative information base is implemented via hash-coding techniques. The associative language is implemented by extending an existing ALGOL system.

This report consists of three sections: Sec. I describes the high-level programming language for the overall system; Sec. II outlines the scheme for representing an associative information base; and Sec. III summarizes the processing routines for associative retrieval requests.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office

CONTENTS

AN ASSOCIATIVE PROCESSING SYSTEM FOR CONVENTIONAL DIGITAL COMPUTERS

I. THE ASSOCIATIVE LANGUAGE, LEAP†

A. Introduction

There are three types of constructs in LEAP: algebraic, set-theoretic, and associative. The algebraic operations available are essentially those of ALGOL. In addition, the declarative and sequencing statements of ALGOL are used in LEAP. The set-theoretic constructs include union, containment, etc., as well as certain sequencing rules for operating on the members of sets. The associative operations constitute an extension of AL [Feldman, 1965] to include associations between associations and nesting of statements.

Each identifier in a LEAP program will have a data type, as in ALGOL. To facilitate the expression of set-theoretic and associative operations, we have added four new data types:

Standard ALGOL Data Types	LEAP Additions
REAL	ITEM
BOOLEAN	SET
FRACTION	NAME
INTEGER	LOCAL

Any of the additional data types may be qualified by an ALGOL data type (e.g., BOOLEAN SET, REAL ITEM, INTEGER LOCAL).

B. Variables

The ALGOL notion of "variable" (a location which takes on arithmetic or Boolean values) has been extended in LEAP to include:

(1) ITEMs:- Each variable of type ITEM has a unique internal system name. This "internal name" is used to represent the ITEM in associative and

†LEAP: A Language for the Expression of Associative Procedures.

set-theoretic operations. In addition to an internal name, an ITEM may or may not have an arithmetic or Boolean value. The algebraic operations in the LEAP language manipulate arithmetic and Boolean values.

(2) SETs:– The value of a SET is an unordered set of ITEMs. Internally, a SET element is represented by its internal name.

(3) NAMEs:– NAME variables are used to pass an ITEM as a parameter to a set-theoretic or associative operation. The value of a NAME variable is an ITEM. Within the system, the value of a NAME variable is represented by its internal name.

(4) LOCALs:– LOCALs are used within associative statements to represent the results of an associative retrieval operation. This is the only use of LOCALs.

C. Expressions

In LEAP, an operation of any type which leaves a result will be called an "expression"; one which does not will be called a "statement." There are three kinds of expressions in LEAP:

ALGOL	An ALGOL expression has as its value an arithmetic or Boolean number.
SET	A SET expression has as its value an unordered collection of ITEMs.
NAME	A NAME expression has as its value an ITEM.

D. Language Forms Involving SET Expressions (SX)

A list of basic SET language forms appears in Table I. In addition, there is one sequencing statement† whose form is

FOR <LOCAL> ϵ <SX> DO <STATEMENT>

†NOTE: This statement is a special case of the associative FOR statement, which is discussed in Sec. G below.

TABLE I
LIST OF BASIC SET LANGUAGE FORMS

Form	Result	Name of Form
ITEM ‹ SX	Boolean	Membership
SX ⊂ SX	Boolean	Containment
SX = SX	Boolean	Set equality
SX – SX	SET	Set subtraction
SX ∪ SX	SET	Union
SX ∩ SX	SET	Intersection
|| SX	Integer	Norm
SX → SET variable		Assignment

For each ITEM in the SET expression, the BODY of the FOR statement is executed. On each iteration, the current ITEM is assigned to the LOCAL variable, which behaves like a NAME variable when used within the BODY.

E. Relational Language Forms

The relational operations in LEAP derive from the relation

(1) A · O ≡ V

which is read "Attribute of Object is Value." Typical statements include:

(2) MAKE PART · PICTURE3 ≡ LINE2
ERASE END · LINE3 ≡ POINT2

By leaving none, one, or two of the positions in (1) unspecified (by the use of LOCALs), we indicate a "simple associative form" (SAF).† Some examples are:

(3) PART · PICTURE3 ≡ X
W · LINE3 ≡ Z

†Therefore, there are seven SAFs.

The identifiers involved in (3) are ITEM identifiers (PART, PICTURE3, LINE3) and LOCAL identifiers (X, W, Z). The relational facility of LEAP is based on operations on these simple associative forms.

The simple associative form is used two ways in LEAP. In statements like (2), it is used to specify the parameters of an operator. In constructs like (3), it is used to assign the results of an associative retrieval operation to a LOCAL (or LOCALs).

One way of combining SAFs is by nesting. Nesting can occur in the Object or Value position of a SAF. For example, if we know that

(1) ABOVE the SQUARE is a TRIANGLE
(ABOVE · SQUARE ≡ TRIANGLE)

and

(2) INSIDE the TRIANGLE is a LINE
(INSIDE · TRIANGLE ≡ LINE)

then

(3) INSIDE · (ABOVE · SQUARE) ≡ LINE

expresses the fact that inside the object which is above the square there is a line. In connection with nesting, it is convenient to add the notion of "associative term." There are three basic associative terms:

(1) A · O meaning all Vs such that A · O ≡ V,

(2) A ' V meaning all Os such that A · O ≡ V,

and

(3) A * O meaning all Vs such that A · O ≡ V or A · V ≡ O.

Thus, the construct

INSIDE · (ABOVE * SQUARE)

is a specification of all objects which are inside of objects which are above or below the square. These notations serve to make more associations expressible by a nested SAF.

A much more significant feature is the ability to use an entire SAF as the Object or Value in an associative form. This may be expressed with the aid of parentheses. For example,

$$\text{REASON} \cdot (\text{ABOVE} \cdot \text{SQUARE} \equiv \text{TRIANGLE}) \equiv (\text{BELOW} \cdot \text{TRIANGLE} \equiv \text{SQUARE})$$

states that the reason that the triangle is above the square is that the square is below the triangle. This information would be difficult to express in terms of simple associative forms. The use of this construct also enables one to express some relationships which are not triples, such as "the number of lines in a square is four." One such statement would be

$$\text{NUMBER} \cdot (\text{PART} \cdot \text{SQUARE} \equiv \text{LINE}) \equiv \text{FOUR}$$

where NUMBER is an attribute which applies to all part-whole relationships. The term "compound associative form" (CAF) will be applied to simple associative forms, nested simple associative forms, and associative forms in which the Object or Value is a SAF.

F. Associative Retrieval Descriptions

A CAF in which a LOCAL is used is one form of an associative retrieval description (ARD). The associative retrieval here consists of matching the association implied by the CAF against a store of associations, and extracting information from the store for each match that is found. The information extracted is the ITEM whose position in the matched association is the same as the position of the LOCAL in the search association. For example, in the CAF

$$\text{PART} \cdot \text{PICTURE3} \equiv X$$

X is a LOCAL which represents the set of ITEMs such that each is a PART of PICTURE3.

G. Associative FOR Statement

For a powerful associative language, one needs a language form for combining ARDs. The "associative FOR statement" has been developed for this

purpose. The LOCAL is used as the link between the ARDs in an associative FOR statement – the only statement in which LOCALs may occur. The use of LOCALs corresponds to the use of free and constrained indeterminates in formal mathematics and is discussed by Mendelson [1964]. The first occurrence of a LOCAL in an associative FOR statement is "free" and all subsequent uses "constrained." For example, the following FOR statement creates the "defining point" (DPT) relation for lines in a line drawing, given the start-point and end-point relations. A point is a defining point of a line if it is either the start- or end-point of the line

```
FOR START · X ≡ Y
AND END · X ≡ Z DO
BEGIN
        MAKE DPT · X ≡ Y;
        MAKE DPT · X ≡ Z
END;
```

The part of the system which deals with run-time execution of the associative FOR statement is quite sophisticated and will be described in detail in Sec. III.

The associative FOR statement has the following general form:

FOR <ARD> AND <ARD>...OR <ARD>...DO <STATEMENT>

An ARD may be a CAF in which LOCALs are used, the construct <LOCAL> ∈ <SX> or any LEAP expression involving one or more LOCALs and having a Boolean result. This implies that SET comparisons and ALGOL Boolean expressions, including functions, may be used as ARDs. Appendix C contains examples of the FOR statement used to recognize structural relations in a simple type of mechanical drawing.

H. Conclusion

Thus far, we have discussed LEAP almost as if it were three separate languages. Although the algebraic, set-theoretic, and relational operations

are quite different, the language allows them to be combined in several ways, as shown in the following examples.

Any position in a compound associative form may be occupied by a SET expression; this implies using each element of the corresponding SET in turn. For example, if LINES is a SET identifier, the statement

MAKE PART · PICTURE6 ≡ LINES

will create a PART relation for each ITEM in the SET.

Any position in a CAF may also be occupied by a NAME expression; this implies using the value of the NAME expression (an ITEM). For example, the statement

MAKE PART · PICTURE6 ≡ (n LINES)†

will create an explicit PART relation between PICTURE6 (an ITEM) and an ITEM having as its value LINES (a SET).

The implementation of the LEAP language on the TX-2 computer has been greatly facilitated by VITAL [Feldman, 1964], a time-shared compiler-compiler system. The compiler-compiler accepts as input a formal specification of the syntax and semantics of a language and yields as output a compiler for programs written in that language. Since the specifications of an ALGOL-like language had already been formalized on this system, a relatively small effort was required to imbed the associative language in ALGOL. The Backus Normal Form (BNF) syntax of the associative language is presented in Appendix B.

II. INTERNAL REPRESENTATION FOR A STORE OF ASSOCIATIONS

A. Introduction

Within the computer, an association is composed of three "internal names" (an internal name is a unique number which is used to represent an ITEM). An association is an instance of the basic associative form

† "n" is an operator which yields a NAME expression. In this case, the value of the NAME expression is a special kind of ITEM – one having as its value a SET.

$$A \cdot O \equiv V$$

meaning "Attribute of Object is Value." The three internal names composing an association represent the Attribute, Object, and Value of the association. A store of associations is simply a store of internal name triplets.

By leaving unspecified zero, one, or two positions in the basic associative form, we indicate a simple associative form (SAF). There are seven SAFs. We will call the use of a SAF in LEAP a simple associative retrieval request, and we will consider the processing required by these seven requests as the seven primitive associative tasks. The result of performing a primitive associative task takes one of three forms.

(1) Boolean:– If all three positions in the SAF are specified, the result is Boolean. In this case, the retrieval system must determine whether or not the indicated internal name triple exists in the store.

(2) A Collection† of Internal Names:– If only one position in the SAF is unspecified, the result is a collection of internal names. Here, we desire to find all internal names implied by the context of the unspecified position in the SAF. The retrieval system must find each internal name triple in the store which matches the SAF in its specified positions, and extract from it the internal name corresponding to the unspecified position in the SAF. The collection of internal names so extracted is the result.

(3) A Collection of Internal Name Pairs:– If two positions in the SAF are unspecified, the result is a collection of internal name pairs. Here, the retrieval system must match on the one specified position, and extract internal name pairs corresponding to the two unspecified positions. The collection of internal name pairs so extracted is the result.

B. Representation Scheme

This section outlines a scheme for the internal representation of a store of associations. The scheme is based on hash-coding techniques and multiple

---

†NOTE: The "collection" is distinct from the "set" of Sec. I.

representation of information on secondary storage. Three main criteria led to the design:

(1) The need for a partitioning scheme for very large stores of associations.

(2) The desire to free the user from concern about the structure of his store of associations.

(3) The need for fast performance of any of the seven primitive associative tasks.

The first design criterion states the need for a partitioning scheme. If the representation for a store of associations† is too large to fit into core, it must be divided into pieces (or "pages") and kept on bulk storage (magnetic drum, for example). In a time-sharing environment, it is desirable to partition a representation, even if it fits into core; overall system performance is improved by having only relevant pages in high-speed memory.

If a representation is divided arbitrarily, however, performing a single primitive associative task might require multiple secondary storage accesses. This would cause slow response and large system overhead. Consequently, we designed a representation such that any primitive associative task can be fully performed within only one page. A brief description of that representation follows.

For those simple association retrieval requests having two unspecified positions, the resulting collection of internal name pairs is found in a list in the page determined by the one internal name that is specified and the position in which it occurs in the SAF. Thus, there are three types of pages: one for each position in the basic associative form (A · O ≡ V). Note that for each, several internal names may indicate the same page. For those simple associative retrieval requests having one unspecified position, a page type and a page are indicated by the internal name in one of the specified positions. The scheme for deciding which of the two specified positions is used here depends

† Henceforth termed the "representation."

or which position is unspecified. The two specified internal names are hash-coded, and the result specifies an address within the page at which a collection of internal names begins.

For the simple associative retrieval request in which all positions are specified, any two internal names may be hash-coded, the indicated page set up, and the third internal name used to search for a match in the collection of internal names available at the hash-code address. The result is Boolean.

Since there are three copies of each triple in the representation, the update cost is large. This cost can be cut considerably by keeping track (for each page) of additions and deletions, and by updating the page when it is next brought into core to be queried.

The second design criterion is that the user should be free from concern about the structure of his store of associations. Existing list-processing systems† [McCarthy, 1962] rely heavily on structural connections to represent associations implicitly. Since the statement of association in LEAP is always explicit, there is no need to impart externally apparent structure to a store of associations. Indeed, the entire system is strongly oriented toward dealing with unstructured information.

The system makes use of the apparent lack of structure to organize the internal representation of information for its own convenience. For example, the collection of internal names which may result from evaluating a SET expression is available to the user as an unordered SET. Internally, however, SETs are ordered by the internal names of their elements. This ordering facilitates efficient system routines for manipulating SETs. If the system were unable to recognize such an ordering, these routines would be significantly slower. For example, consider the task of verifying that two SETs are disjoint. If the SETs are unordered, at most, $m \cdot n$ comparisons must be made ("m" and "n" are the cardinalities of the SETs). If the SETs are ordered, however, the task requires, at most, $M + N$ comparisons.

---

† See various memoranda describing the CORAL language and data-structure system at Lincoln Laboratory.

The third design criterion is the fast performance of any of the seven primitive associative tasks. For any of these tasks, a single page access and calculation of an address within the page directly locates the desired information. There is never a need for searching a property list.

There are known problems due to the non-uniqueness of the hash-code scheme [Feldman, 1965; Mendelson, 1964], but these do not seriously affect performance.

III. PROCESSING ASSOCIATIVE RETRIEVAL REQUESTS

As an example of an associative retrieval request, we have the following associative FOR statement in the LEAP language (see Appendix A):

(1) FOR ABOVE · SQUARE ≡ X

(2) AND INSIDE · X ≡ Y

(3) DO DELETE X, Y

In this example, X is a LOCAL which represents the collection of ITEMs that are above the square and contain another ITEM. Y is a LOCAL which represents the collection of ITEMs that are inside the ITEMs above the square.

An associative FOR statement has two parts: a group of associative retrieval descriptions that imply by context which internal names are to be extracted from the associative store, and a statement (the BODY) which describes how these internal names are to be used. These two parts are separated by DO. The LOCAL is used in the associative FOR statement as a placeholder for a collection of ITEMs. Each associative retrieval description is said to "constrain" the LOCALs occurring in it. The first time a particular LOCAL occurs in the upper part of an associative FOR statement, it is termed "free." The corresponding associative retrieval description is processed as if the position in which the LOCAL occurs were unspecified. The resulting collection of ITEMs becomes attached to the LOCAL, which is then termed "constrained" for the remainder of the statement. For brevity, we will call each such collection of internal names a "bound set." Each constrained LOCAL indicates

a unique bound set. Each time the LOCAL occurs again in the upper part of the associative FOR statement, it is said to be further constrained; and the indicated bound set is reduced by the requirement that all ITEMs which remain in the bound set also satisfy the further constraint. See Example 1 in Appendix A. Note that after the associative retrieval description on line (2) of the example is processed, only two ITEMs remain in the bound set for X.

The use of two or more LOCALs in the upper part of a FOR statement implies a relationship between ITEMs in the different bound sets. As an illustration, consider Example 1 in Appendix A once again. After considering each of the two associative retrieval descriptions there, we have written down the bound sets. We will call a relationship between ITEMs in bound sets a "correspondence." There are three correspondences after processing the second associative retrieval description in the example:

X	Y
TRIANGLE	LINE
RECTANGLE	LINE
RECTANGLE	TRIANGLE

Each correspondence is a set of values for the LOCALs in the FOR statement for one iteration of the BODY of the FOR statement.

When a bound set is further constrained, the correspondence tied to each ITEM in the bound set is eliminated whenever the test on the ITEM fails. Thus, there is an implicit constraint on other (not necessarily all) bound sets whenever one bound set is constrained.

The purpose of extracting correspondences, and thus bound sets, is to execute the BODY of the FOR statement for each correspondence which satisfies all constraints. Statements within the BODY may change the store of associations, perform standard ALGOL operations, or perform SET or other LEAP operations.

The principal means of expression of associative procedures in LEAP is the associative FOR statement. The principal technique in the processing of associative procedures is the extraction and use of correspondences. There are two methods of processing:

(1) Extract a correspondence from the associative store, execute the BODY of the FOR statement, then return for the next correspondence, etc.†

(2) Extract all correspondences, then execute the BODY of the FOR statement for each correspondence.

The first method has the advantage that there is no need to keep an intermediate store of information. However, extracting a single entire correspondence requires obtaining one piece of information from each page indicated in the upper part of the FOR statement.‡ In general, this would require either keeping many pages in core, or doing multiple secondary storage accesses for each correspondence extracted.

The second method has the advantage that a page need be brought into core only once in executing an associative FOR statement. At that time, all relevant information from that page may be extracted. This necessitates storing and manipulating intermediate information, but it eliminates the need to do multiple secondary storage accesses for each correspondence, or to hold in core large quantities of uninteresting information. Also, the facility to manipulate stores of correspondences is a straightforward extension to this scheme.

An implementation of the second method has been designed. It uses a LEAP-like scheme for the storage of correspondences. The association between internal name, LOCAL, and correspondence is the basic element of information. A preliminary investigation has indicated that this associative correspondence storage scheme uses less memory space and facilitates easier access to the correspondence information than would a tree structure correspondence storage scheme. Further study in this area is planned.

---

† Feldman's [1965] system used this technique to process associative procedures. However, his system operated outside of time-sharing, and with the entire associative information base always resident in core.

‡ In an associative FOR statement, each associative retrieval description (ARD) may indicate several pages. In our example, the associative retrieval descriptions are SAFs, each of which indicates just one page.

IV. CONCLUSION

The purpose of this report is to present our ideas on how to build a system with a facility for representing and manipulating large, complex associative information bases in a time-sharing environment. Several interesting topics have either been ignored or dealt with briefly in the presentation. We will mention some of these topics here, as an indication of our plans for future research.

(1) Compile-Time Techniques for Optimizing Associative FOR Statement Processing:– The efficiency of extracting correspondences from the associative information base depends heavily on how the ARDs are ordered in the upper part of the FOR statement. Under certain conditions, an associative FOR statement may be reorganized at compile time to increase processing efficiency. Also, compile-time investigations can usually be made to determine which pages will be needed at run time.

(2) Run-Time Data-Type Checking:– If it is found desirable, later versions of the system will have this facility.

(3) Techniques for Representing a Store of Correspondences:– A preliminary search for a suitable representation scheme for a store of correspondences indicated the following: those schemes which were designed to eliminate redundancy, and thus save storage, were so complex that (a) the extra storage used for bookkeeping would be at least as large as the storage saved, and (b) the routines to manipulate the representation would be significantly more complex than would the routines to manipulate a simple representation.

Therefore, a simple representation, based on hash-coding techniques, was designed to be very similar to the scheme for representing an arbitrary associative information base. The relative advantages of hash-coded vs structured representations for stores of correspondences will be studied further.

Anticipated areas of application for the LEAP system include the construction of interactive systems, and automated natural language work. For example, certain parts of a system like Sketchpad [Sutherland, 1963] or Raphael's [1964] question-answering program, could well be written in the LEAP language.

Natural language applications are best illustrated by the work of R. F. Simmons [1966], who postulates the normalization of English text into subject-verb-nominal triples, or "kernels."

APPENDIX A
SIMPLE EXAMPLE OF AN ASSOCIATIVE FOR STATEMENT

Example A-1 – bound sets and correspondences resulting from processing an associative FOR statement.

(1) FOR ABOVE · SQUARE ≡ X

(2) AND INSIDE · X ≡ Y

(3) DO DELETE X

After (1):

Correspondence	Local X
1	HEXAGON
2	TRIANGLE
3	LINE
4	RECTANGLE

After (2):

Correspondence	Local X	Local Y
1	TRIANGLE	LINE
2	RECTANGLE	LINE
3	RECTANGLE	TRIANGLE

3-23-7359

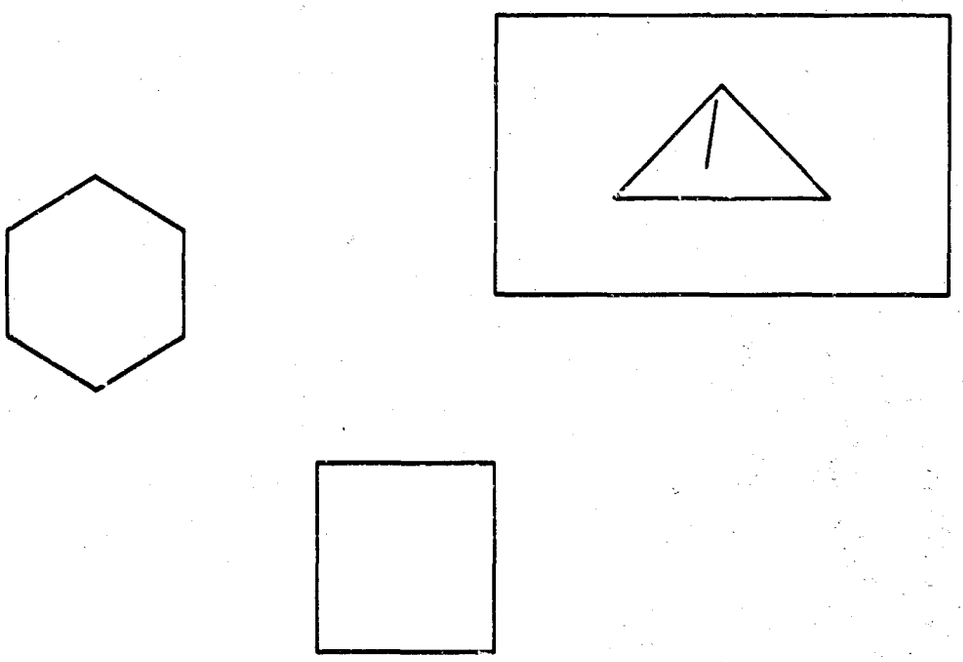


Fig. A-1. Relations between five ITEMs (HEXAGON, SQUARE, TRIANGLE, LINE, RECTANGLE).

APPENDIX B
BNF SYNTAX OF LEAP

<AF>: = <AI> · <AT> ≡ <AT>

<AI>: = <NM>|n<SET>|ANY

<NM>: = <ITEM>|<LOCAL>|<SET>|<NAME>

<AT>: = <AI>|<AT1>|<AE>|<BE>

<AT1>: = <AF>|<AT> <AOP> <AT>

<AOP>: = · | ' | *

<AE>: = ||<SE>|<number>|<AV>|<NM1>|<NM1>|γ<NM1>

−<AE>|<AE> <OP> <AE>

<SE>: = <SET>|{<IL>}|φ|<AT1>|γ<NAME>|<SE> <SOP> <SE>|γ<LOCAL>

<IL>: = <IL1>|<IL1>, <IL>

<IL1>: = <AE>|<NE>|<BE>|<ITEM>|<LOCAL>

<NE>: = n<AE>|n<BE>|n<SE>|n<ITEM>|INTERNAL|n<LOCAL>|<NAME>

<BE>: = <BN>|<NM1>|γ<NM1>|n<BE>|<AE> <RL> <AE>|<BE> <BOP> <BE>|

<SE> <SRL> <SE>|<NE> = <NE>|<NM> ϵ <SE>

<BN>: = <AV>|TRUE|FALSE

<AV>: = (ALGOL variable)

<NM1>: = <ITEM>|<LOCAL>|<NAME>

<RL>: = (arithmetic relations)

<BOP>: = (Boolean operations)

<SRL>: = (set relations)

<SOP>: = ∪| ∩ | ⊖

<OP>: = (arithmetic operators)

<AST>: = <EXP> → <IDENT>|PUT<IL1>IN<NM2>

<EXP>: = <AE>|<BE>|<SE>|<NE>

<IDENT>: = <NM>|<AV>|γ<NAME>

<NM2>: = <LOCAL>|<SET>|γ<NAME>

<FST>: = FOR<COND>DO<BODY>

<COND>: = <BE>|<AF>|<COND> <AOR> <COND>

<AOR>: = AND|OR

<BODY>: = (arbitrary block)

I. TERMS IN BNF DESCRIPTION

AF	associative form	BN	Boolean
AI	associative item	AST	assignment statement
AT	associative term	EXP	expression
AE	arithmetic expression	IDENT	identifier
BE	Boolean expression	FST	associative FOR statement
NE	NAME expression	||	cardinality of (norm of)
SE	SET expression	~	not
AOP	associative operators	ϵ	a member of
IL	item list	ϕ	the empty set

II. NOTES ON BNF DESCRIPTION

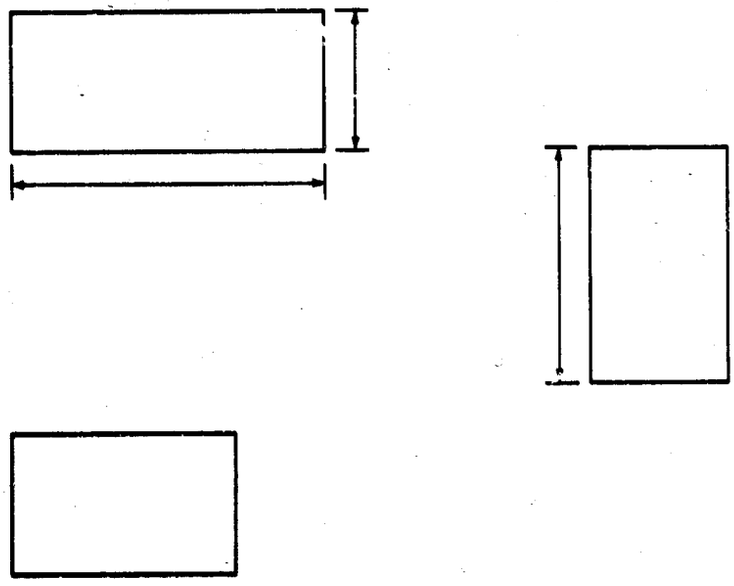
γ operates on ITEMs, LOCALs, and NAMEs. For ITEM, γ yields the value of the ITEM; for LOCAL, γ yields the value of the current ITEM for which the LOCAL stands; and for NAME, γ yields the value of the ITEM whose internal name is the value of the NAME variable.

n an operator which yields the internal name of the operand.

APPENDIX C
EXAMPLE OF A LEAP PROGRAM

As an example of the use of the LEAP language, we present a program segment for recognizing structural relations in a simple type of mechanical drawing. For our example, a drawing is allowed to consist of rectangles, squares, and dimension-line constructs. Each dimension-line construct must conform to certain specifications about the size and position of its parts, and about its relation to the rectangle or square being considered. These specifications are used in the recognition routines to determine whether a construct is to be recognized, marked as erroneous, or simply left unrecognized. There are two phases to the example program: First is the recognition of legal and illegal constructs, using geometrical criteria; second is the application of size and distance constraints to the recognized constructs to further classify them as legal or illegal. For example, the specification that the angle between adjacent lines in a picture construct be 90° is applied to all such constructs during the second phase of the program. The following is an example of a "legal" picture.

Example C-1



When entered in the program, the information about a drawing consists exclusively of associations in the following five forms:

(1) LTYPE · <LINE> ≡ DLINE

(2) START · <LINE> ≡ <POINT>

(3) END · <LINE> ≡ <POINT>

(4) HC · <POINT> ≡ <VALUE>

(5) VC · <POINT> ≡ <VALUE>

The associations of form (1) indicate which lines are of type dimension line (DLINE); those of forms (2) and (3) indicate which points are start or end points of which lines; and those of forms (4) and (5) indicate the horizontal and vertical coordinates of such points.

The program uses this initial information to generate a store of associations about the structure of the drawing, and about the illegal constructs found. This information is put into associations of the following forms:

(6) DPT · <LINE> ≡ <POINT>

(7) DLARW · <DIMENSION LINE> ≡ <ARROW>

(8) APEX · <ARROW> ≡ <POINT>

(9) ARWLINE · <ARROW> ≡ <LINE>

(10) TERMLINE · <ARROW> ≡ <LINE>

(11) SUBPIC · PICTURE ≡ <SUBPIC>

(12) DRAWINGPART · <SUBPIC> ≡ <LINE>

(13) TLPL · <LINE> ≡ <LINE>

(14) DIMENSIONPART · <SUBPIC> ≡ <DIMENSION LINE>

(15) TEST <#> · <CONSTRUCT> ≡ FALSE

The associations of form (6) represent the LINE-DEFINING POINT relationship. Form (7) represents the relationship between an ARROW construct

and a dimension line. Form (10) represents the relationship between an ARROW construct and an ARROW terminal line. Form (13) represents the relationship between an ARROW terminal line and the picture line with which it is collinear.

The program is presented as a collection of routines which generate these associations from the initial associations:

DPT

```
FOR START · X ≡ Y
AND END · X ≡ Z
DO MAKE DPT · X ≡ {Y, Z};
```

DLARW, APEX, ARWLINE

```
FOR LTYPE · X ≡ DLINE
AND DPT · X ≡ Y
DO
  BEGIN
    DPT'Y ⊖ {X} → S1;
    IF || S1 = 2 THEN
      BEGIN
        INTERNAL → NMV;
        MAKE DLARW · X ≡ NMV;
        MAKE ARWLINE · NMV ≡ S1 '
        MAKE APEX · NMV ≡ Y;
      END;
    ELSE MAKE TEST1 · X ≡ FALSE;
END;
```

TERMLINE

```
FOR    START · X ≡ ANY
AND    || DPT ' DPT · X = 1
AND    APEX · W ≡ Y
AND    MIDPOINT {X, Y}
DO     MAKE TERMLINE · W ≡ X;

FOR    APEX · X ≡ ANY
AND    ~ TERMLINE · X ≡ ANY
DO     MAKE TEST2 · X ≡ FALSE;
```

SUBPIC, DRAWINGPART

```
START ' ANY ⊖ LTYPE ' ANY ⊖ TERMLINE · ANY ⊖ ARWLINE · ANY → LINES;
TAG1 ⌐ FOR X ∈ LINES DO
        BEGIN
           DPT ' DPT · DPT ' DPT · X → S1
           LINES ⊖ S1 → LINES;
           IF || S1    4 THEN
                 BEGIN
                    FOR Y ∈ S1 DO
                       IF || DPT ' START · Y ≠ 2 ∨ || DPT ' END · Y ≠ 2
                       THEN GO TO TAG2;
                       INTERNAL → NMV;
                       MAKE SUBPIC · PIC ≡ NMV;
                       MAKE DRAWINGPART · NMV ≡ S1;
```

```
                END
            ELSE
TAG2 ⇐ MAKE TEST3 · S1 ≡ FALSE;
                GO TO TAG1;
        END;
```

TLPL, DIMENSIONPART

```
        FOR   TERMINATE · X ≡ Y
        AND   DRAWINGPART · W ≡ Z
        DO
          BEGIN
            DIST {Y, Z} → TEMP;
            IF (TEMP<TH1) ∧ (TEMP>TH2) ∧ COLIN {Y, Z} THEN
              BEGIN
                MAKE TLPL · Y ≡ Z;
                MAKE DIMENSIONPART · W ≡ DLARW ' X '
              END;
            END;
          FOR TERMLINE · ANY ≡ X
          AND ~ TLPL · X ≡ ANY
          DO MAKE TEST4 · X ≡ FALSE;
```

TEST DIMENSION LINE CONSTRUCTS

```
        FOR   ARWLINE · X ≡ Y
        AND   DLARW · Z ≡ X
```

```
DO    IF || (LNGTH {Y} – ARWL) > TH3 ∨
         ||(||ANG {Z, Y} – ARWANG) > TH4
      THEN MAKE TEST5 · X ≡ FALSE
FOR   TERMLINE · X ≡ Y
DO    IF || (LNGTH {Y} – TLNLNGTH) > TH5
      THEN
         BEGIN
            MAKE TEST6 · X ≡ FALSE;
            MAKE TEST7 · DLARW ' X ≡ FALSE;
         END;
```

TEST FOR 90° ANGLES IN SUBPICTURES

```
      FOR DRAWINGPART · PIC ≡ X
      DO
        BEGIN
            DPT ' DPT · X → S1;
            FOR Y ∈ S1
            AND Z ∈ S1
      DO
            IF ~ (PERP {Y, Z} ∨ PARL {Y, Z})
            THEN MAKE TEST8 · X ≡ FAIL;
      END;
```

The procedures used in the example are

(1) MIDPOINT {X, Y}: Boolean result. Is point Y the midpoint of line X?

(2) DIST {X, Y}: The result is the distance between line X and line Y.

(3) COLIN {X, Y}: Boolean result. Is line X collinear with line Y?

(4) LNGTH {X}: The result is the length of line X.

(5) ANG {X, Y}: The result is the angle between line X and line Y.

(6) PERP {X, Y}: Boolean result. Is line X perpendicular to line Y?

(7) PARL {X, Y}: Boolean result. Is line X parallel to line Y?

As an example, the procedure LNGTH is presented below:

```
REAL PROCEDURE LNGTH (X)
BEGIN
        REAL T1, T2;
        (HC · END · X – HC · START · X) ↑ 2 → T1;
        (VC · END · X – VC · START · X) ↑ 2 → T2;
        RETURN SQRT (T1 + T2);
END.
```

1. IMPLICATIONS OF TEST ASSOCIATIONS

TEST1	This dimension line is missing an arrow.
TEST2	This arrow does not have a terminal line.
TEST3	This line is part of an illegal construct.
TEST4	This terminal line does not correspond to a picture line.
TEST5	This arrow has lines of the wrong length, or has parts which form the wrong angle with the dimension line.
TEST6	The terminal line on the end of this arrow is of the wrong length.
TEST7	This dimension line has an arrow for which TEST6 failed.
TEST8	This subpicture is not rectangular.

BIBLIOGRAPHY

Elliot, R.W., "A Model for a Fact Retrieval System," doctoral thesis, University of Texas, May 1965.

Feldman, J.A., "Aspects of Associative Processing," Technical Note 1965-13, Lincoln Laboratory, M.I.T. (21 April 1965), DDC 614634, H-644.

_____, "A Formal Semantics for Computer Oriented Languages," doctoral thesis, Carnegie Institute of Technology, 1964.

Forman, B., "An Experiment in Semantic Classification," LRC-65-WT-3, Linguistics Research Center, University of Texas (December 1965).

Fuller, R.G., et al., "Study of Associative Processing Techniques," RADC-TR-65-210, Rome Air Development Center, Griffiss Air Force Base, New York (August 1965).

Fuller, R.H., "Content Addressable Memory Systems," doctoral thesis, University of California at Los Angeles, 1963.

Love, H.A., et al., "Associative Processing Techniques Study," RADC-TR-65-32, Rome Air Development Center, Griffiss Air Force Base, New York (1 April 1965).

McCarthy, J., et al., LISP 1-5 Programmer's Manual (M.I.T. Press, Cambridge, Massachusetts, 1962).

Mendelson, E., Introduction to Mathematical Logic (Van Nostrand, Princeton, New Jersey, 1964), p. 300.

Newell, A., "A Note on the Use of Scrambled Addressing for Associative Memories," unpublished paper (December 1962).

Raphael, B., "SIR, A Computer Program for Semantic Information Retrieval," Proc. Fall Joint Computer Conf., San Francisco, California, October 1964.

Rovner, P.D., "An Investigation into Paging a Software-Simulated Associative Memory System," Sc.M degree, University of California at Berkeley, 1966.

_____, an unpublished paper on the Processing of Associative Procedures (August 1965).

Simmons, R.F., Burger, J.F., and Long, R.E., "An Approach Toward Answering English Questions from Text," Proc. Fall Joint Computer Conf., San Francisco, California, 7-10 November 1966.

Sutherland, I.E., "Sketchpad, A Man-Machine Communication System," Proc. Spring Joint Computer Conf., Detroit, Michigan, May 1963.

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)*
Lincoln Laboratory, M.I.T.

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP
None

3. REPORT TITLE
An Associative Processing System for Conventional Digital Computers

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

5. AUTHOR(S) *(Last name, first name, initial)*
Rovner, Paul D. Feldman, Jerome A.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
21 April 1967	36	16

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AF 19(628)-5167	Technical Note 1967-19
b. PROJECT NO. Order 691	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	ESD-TR-67-242

10. AVAILABILITY/LIMITATION NOTICES
Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

A user-oriented system having both algebraic and associative processing capabilities is presented in this report. The algebraic capabilities are essentially those of ALGOL. The associative facilities are:

(1) A language for the expression of associative retrieval requests (the associative language).

(2) A scheme for the internal representation of a store of associations between items of information (an associative information base).

(3) Processing routines for associative retrieval requests.

The associative language is independent of the structure of the associative information base. In the system presented here, the associative information base is implemented via hash-coding techniques. The associative language is implemented by extending an existing ALGOL system.

This report consists of three sections: Sec. I describes the high-level programming language for the overall system; Sec. II outlines the scheme for representing an associative information base; and Sec. III summarizes the processing routines for associative retrieval requests.

14. KEY WORDS

programming languages
digital computers
associative information storage and retrieval