AD 677528

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO NO. AI-69

September 13, 1968

PROJECT TECHNICAL REPORT

John McCarthy,	Professor of Computer Science Principal Investigator
Edward Feigenbaum,	Associate Professor of Computer Science Associate Investigator
Arthur Samuel,	Research Associate in Computer Science Associate Investigator

D D C
RECEIVED
NOV 19 1968
B

Sponsored by

Advanced Research Projects Agency

ARPA Order No. 457

This document has been approved for public release and sale; its distribution is unlimited.

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151

BEST
AVAILABLE COPY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT September 13, 1968
MEMO NO. AI-69

PROJECT TECHNICAL REPORT

ABSTRACT: Recent work of the Stanford Artificial Intelligence Project is summarized in several areas:

Scientific Hypothesis Formation
Symbolic Computation
Hand-Eye Systems
Computer Recognition of Speech
Board Games
Other Projects

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

1. Introduction

Research interests in the Stanford A.I. Project cover a number of areas in machine perception, heuristic programming, symbolic computation, and the mathematical theory of computation. This report summarizes recent results, drawn largely from material that has been prepared for publication in forthcoming conference proceedings.

Some of this work is finding immediate application. Our work on hypothesis formation in organic chemistry has recently caught the interest of research chemists and a rewarding interaction is developing. The work on symbolic computation has already contributed to research in theoretical physics and to the solution of complex engineering problems.

Hand-eye and speech recognition research is not so advanced in relation to its objectives. Nevertheless, we have demonstrated successful solutions to several important subproblems and are actively expanding the domain of soluble problems.

Work on board games such as chess and checkers continues to be an important part of artificial intelligence research, both as a source of new ideas on the mechanization of problem solving and as a measure of performance that allows rather direct comparison with human abilities.

The following sections discuss results in several areas. Appendix A lists some recent publications of project participants. Appendix B lists Stanford A.I. Memos to date.

2. Scientific Hypothesis Formation

Part I, below, is an excerpt from "Artificial Intelligence: Themes in the Second Decade" by Edward A. Feigenbaum. This was an invited paper at the 1968 IFIP Congress in Edinburgh.

BLANK PAGE

Part I.

Our primary goal was to study processes of hypothesis formation in a complex task of a scientific nature involving the analysis of empirical data. The task environment chosen as the medium for this study was the analysis of the mass spectra of organic molecules: the generation of a hypothesis to best explain given mass spectral data. This is a relatively new area of organic chemistry of great interest to physical chemists. In this sense, the problem is not a "toy" problem; and a program that solves problems of this type is a useful application of A.I. research to a problem of importance to science.

We have written a program to infer structural hypotheses from mass spectral data. The program is called Heuristic DENDRAL. It was developed at the Stanford University Artificial Intelligence Project by a small group including Professor Joshua Lederberg of the Stanford Genetics Department, Dr. Bruce Buchanan, Mrs. Georgia Sutherland, and me, with the assistance of chemists and mass spectrometrists of the Stanford Chemistry Department. It is an 80,000 word program written in LISP for the PDP-6 computer, and was developed (and is run) interactively under the time-sharing monitor (1, 4, 5).

Heuristic DENDRAL will perform the following two classes of tasks:

1. Given a mass spectrum of an organic molecular sample and the chemical formula of the molecule, the program will produce a short list of molecular "graphs" as hypotheses to explain the given data in the light of the program's models of mass spectrometric processes and stability of organic molecules. The list is rank-ordered from the most satisfactory explanation to the least satisfactory.
2. If no mass spectrum is given, but only a formula, the program will produce a list of all the chemically plausible isomers of the molecule in the light of its model of chemical stability of organic molecules.

The flow diagram of the system is a closed loop consisting of phases of data inspection, hypothesis generation, prediction, and test, corresponding closely to a simple "scientific method" loop.

At the heart of the program is a systematic hypothesis generator. It is based on an algorithm developed by Lederberg called DENDRAL which is capable of generating all of the topologically possible isomers of a chemical formula. The generator is essentially a topologist, knowing nothing about chemistry except for the valences of atoms; but the generating algorithm serves as the guarantor of the completeness of the hypothesis space, in a fashion analogous to the legal move generator in a chess program. Since the generating process is a combinatorial procedure, it produces for all but the simplest molecules a very large set of structures, almost all of which are chemically implausible though topologically possible. Implicit in its activity is a tree of possible hypothesis candidates. At the top node of the tree all the atoms are found but no structures. At the terminal nodes, only complete structures are found, but no unallocated atoms. Each intermediate node specifies a partially built structure and a residual set of atoms yet to be allocated.

This tree is the implicit problem space for Heuristic DENDRAL. Various heuristic rules and chemical models are employed to control the generation of paths through this space, as follows:

1. A model of the chemical stability of organic molecules based on the presence of certain denied and preferred subgraphs of the chemical graph. It is called the a priori model since it is independent of processes of mass spectrometry.
2. A very crude but efficient theory of the behavior of molecules in a mass spectrometer, called the Zero-order Theory of Mass Spectrometry, used to make a rough initial discarding of whole classes of structures because they are not valid in the light of the data, even to a crude approximation.
3. A set of pattern recognition heuristic rules which allow a preliminary interpretation of the data in terms of the presence of key functional groups, absence of other functional groups, weights of radicals attached to key functional groups, etc. It is called the Preliminary Inference Maker. Its activity allows the Hypothesis Generator to proceed directly to the most plausible subtrees of the space.

The output of the Preliminary Inference and Hypothesis Generation processes is a list of molecular structures that are candidate hypotheses for an explanation of the mass spectrum. They are all chemically plausible under the a priori theory and valid explanations of the data under our zero-order theory of mass spectrometry. Typically the list contains a few candidates (but not dozens or hundreds).

Next a confrontation is made between this list of "most likely" hypotheses and the data. For each candidate hypothesis, a detailed prediction is made of its mass spectrum. This is done with a subprogram called the Predictor, a complex theory of mass spectrometry in computer simulation form. The Predictor is not a heuristic program. It is an elaborate but straightforward procedure for deducing consequences of a theory of mass spectrometry extracted by us from chemists and their literature. The spectral prediction for each candidate is matched with the empirical input data by a process called the Evaluation Function. This is a heuristic, hierarchial, non-linear scoring procedure. Some hypothesis candidates are immediately discarded because their predicted spectra fail certain critical confrontations. The remainder are scored, ranked, and printed out in rank order from most to least satisfactory.

For the class of non-ringed organic structures with which we have been working up to the present time, the program's behavior approaches or exceeds the performance of post-doctoral laboratory workers in mass spectrometry for certain classes of organic molecules. These include amino acids, with which for tangential reasons we have done much of our work, and a large variety of simple organic groups that, however, turn out to be considerably more complicated than amino acids from the point of view of mass spectrometry.

Heuristic programming provided only the skelton for the problem solving processes of Heuristic DENDRAL and the computer techniques to handle the implementation. The heuristics of chemical plausibility of structures; of preliminary inference; of evaluation of the predictions; and also the zero-order and complex theories of mass spectrometry--these were all extracted from our chemist colleagues by man-machine interaction, with the process carefully guided by one of our research team. The

success of this mixed discipline for pulling out of the heads of practicing professionals the problem solving heuristics they are using has worked far better than we had any right to expect, and we are now considering further mechanization of this process.

Part II. Additional Remarks

Because the theory of mass spectrometry is incomplete and piecemeal we must rely heavily on chemist's intuitions about the processes affecting chemical molecules within a mass spectrometer. This is significant for two reasons: we must invest much time questioning chemists about the processes (as opposed to looking into a published, unified account of the theory), and we must be prepared to change the computer program as often as chemists change their minds about any aspect of any of these processes.

Much of our work is now concentrated on reducing the time needed to extract new information from chemists. The dialog between chemist and computer now frequently requires a human intermediary as translator between the two. We would like to mechanize the dialog so that the computer can elicit information directly from the expert. Still more desirable is an extension of the dialog program which will direct the questioning by detecting gaps in its own model and will challenge the chemist with possible counterexamples to new rules he has suggested.

Another time-saving scheme is to use the computer's complex theory of mass spectrometry as the source of new pattern recognition rules. This compresses two interactive sessions into one; and, more important, results in a guarantee of consistency between prediction rules and pattern recognition rules.

A very flexible computer program is required for modifying the program as frequently as chemists revise specific details of their accounts of mass spectrometry. We are working on new ways of organizing the program to make the model building process easier, less expensive in time, and more elegant. One step is to separate the chemical theory from the parts of the program that manipulate the theory. Another is to organize the theory so that all parts of the program can reference it and chemists can easily change it. Still another is to keep the manipulation in a simple form that can be easily conceptualized and easily changed.

We are fortunate to have the cooperation of several members of the Stanford Chemistry Department. For the last nine months two or more of

them have met twice weekly with our staff to improve the computer program or to use the program for their own work. As a result of this interdisciplinary research, we have collaborated on two articles for publication in chemistry journals (3, 4). At the same time, we have brought the program up to near human standards of performance in the analysis of spectra of some types of molecules.

REFERENCES

1. B. Buchanan and G. Sutherland, "Heuristic Dendral: A Program for Generating Explanatory Hypotheses in Organic Chemistry", Artificial Intelligence Working Paper No. 62 (Computer Science Department, Stanford University), to be published in D. Michie et. al. (eds.), Machine Intelligence 4 (in preparation).
2. B. Buchanan, C. Djerassi, A. Duffield, E. Feigenbaum, J. Lederberg, A. Robertson, and G. Sutherland, "Applications of Artificial Intelligence for Interpretation of Mass Spectra. I. The Number of Possible Organic Compounds: Acyclic Structures Containing C, H, O and N." Forthcoming.
3. ---, "Applications of Artificial Intelligence for Interpretation of Mass Spectra. II. Low Resolution Mass Spectra of Ketones". Forthcoming.
4. E. A. Feigenbaum, J. Lederberg and B. Buchanan, "Heuristic Dendral", in Proceedings of the Hawaii International Conference on System Sciences, University of Hawaii and IEEE (University of Hawaii Press, 1968).
5. J. Lederberg and E. Feigenbaum, "Mechanization of Inductive Inference in Organic Chemistry", in: B. Kleinmuntz (ed.) Formal Representation of Human Judgment (New York, John Wiley, 1968).

3. Symbolic Computation

The following paper, titled "The Problem of Substitution", was presented by Anthony C. Hearn at the IBM Summer Institute on Symbolic Mathematics by Computer, held in Boston, July 1968.

Professor Hearn is associated with the Institute of Theoretical Physics as well as the Artificial Intelligence Project at Stanford and is an Alfred P. Sloan Foundation Fellow. In addition to the ARPA support cited above, this research was sponsored by the Air Force Office of Scientific Research, Office of Aerospace Research, U. S. Air Force, under AFOSR Contract No. F44620-68-C-0075.

I. INTRODUCTION

One of the most significant features of programs designed for non-numeric calculation is that the size of expressions manipulated, and hence the amount of storage necessary, changes continually during the execution of the program. It is therefore usually not possible for the user to know ahead of time just how much output his program will produce, or whether the calculation will in fact fail because of lack of available computer memory. The key to keeping both the size of intermediate expressions and output under control often lies in the manner in which substitutions for variables and expressions declared by the programmer are implemented by the system. In this paper, we discuss various methods which have been developed to perform these substitutions in the author's own system REDUCE.[1,2] REDUCE, like FORMAC, is a program designed for general algebraic computations of interest to mathematicians, physicists and engineers. However, in spite of the many common capabilities of the two systems, there are marked differences in the design of each. REDUCE began as a system designed to handle the special problems of non-commutative and tensor algebra encountered in calculations in elementary particle physics scattering theory.[3] However, it was found that the techniques employed were capable of extension so that many problems involving manipulation of large algebraic expressions by known algorithmic methods could be handled.

One major difference between REDUCE and FORMAC is that whereas the latter is largely a machine-coded system, REDUCE has been programmed entirely in LISP 1.5.[4] The big advantage of using such a language is that it is possible to develop a system which may be easily modified and which is also relatively machine independent. Thus the <u>same</u> program is operating at Stanford on two entirely different machines, an IBM 360/67 and a Digital Equipment Corporation PDP-6.

The plan of the paper is as follows. In Sec. 2, the simplest type of substitution problem is used to introduce the REDUCE system and discuss the general characteristics which permit the efficient coding of many types of substitutions. In Sec. 3, the general problem of substitution is discussed in terms of the matching of expressions.

Finally, in Sec. 4, the problem of substitution as a means of reducing the complexity of program output is discussed.

II. SIMPLE SUBSTITUTIONS

An assignment statement of the form

```
A = 2*B*C - 3*D**2 + COS(-X)*COS(Y)                    (2.1)
```

has an entirely different interpretation in a non-numeric calculation than in a PL/I or FORTRAN program. In the latter case, the right-hand side evaluates to a number which is then stored in a machine location reserved for A. In a non-numeric system, the 'evaluation' of such an expression is not so unambigous. Here evaluation is usually referred to as 'simplification' in the sense that the expression is reduced to a canonical form by various rules built into the program or provided by the user. There are as many philosophies on what simplification of expressions means as there are systems. FORMAC, for example, makes substitutions for variables with assigned values and performs several unambigous reductions of the expression. In Eq. (2.1), for example, COS(-X) would be replaced by COS(X). However, the basic form of the expression would remain the same, apart from conversion from infix notation to an internal Polish prefix representation. In REDUCE, however, reduction of an expression to canonical form is more complicated. For one thing, this reduction always involves expansion of expressions, an operation under user control in FORMAC. This flexibility is not present in REDUCE because many of the operations associated with high-energy physics calculations require the expression to be in a fully expanded form. It has also been found that there is often a considerable gain in speed of calculation and storage requirements if such expansions are made at an early stage. To describe the canonical form used, it is easier to begin by restricting ourselves to rational functions of polynomials in several variables. The simplification operation reduces such expressions to a canonical form consisting of a pair of <u>standard forms</u> which represent the numerator and denominator of the expression respectively. The standard form used is similar to that described by Collins in Ref. 5. In this, an expression in n variables $f(x_1, x_2 \ldots x_n)$

is written as a power series in one variable whose coefficients are themselves functions of n-1 variables. Thus

$$f(x_1, x_2 \ldots x_n) = \sum_{i=0}^{m_1} f_i(x_2 \ldots x_n) x_1^i \qquad (2.2)$$

The polynomial coefficients are themselves expanded in a similar manner, and the representation continued until only integers remain. In Backus normal form, using the LISP dotted pair notation the REDUCE standard form is:

<standard form> :: = <integer> | (<standard term> · <standard form>) (2.3)

<standard term> :: = (<standard power> . <standard form>) (2.4)

<standard power> :: = (<variable> · <non-zero positive integer>) (2.5)

Thus a standard term represents one term in the power series Eq. (2.2) and a standard power represents a variable raised to a positive integer power. Comparison with Eq. (2.2) also shows that the dotted pair represents an implicit addition in Eq. (2.3), multiplication in Eq. (2.4) and exponentiation in Eq. (2.5).

In large expressions, the same fixed power of a given variable appears many times in the expanded form of the expression. Thus a considerable saving in storage is made by storing all standard powers uniquely on a single ordered list. In some cases, other sub-expressions may occur several times, but no explicit attempt is made to store these uniquely, although this often occurs automatically.

An ordering convention based on the machine location of the variables in core is used to decide the position of a variable in a standard form. Thus two equal polynomials will have the same standard form.

If fractional powers of variables or expressions are encountered during reduction, then a new variable is created to represent that power, and the user informed, ensuring that no fractional powers remain in the standard form. Likewise, any real numbers encountered are usually converted to the ratio of two integers, unless the user specifies floating point arithmetic.

An extension of the basic polynomial representation to include other operators is made in a straightforward manner. Each operator in the system has a simplification function associated with it, and this function may transform its arguments in one of two ways. First, it may convert the expression completely into other operators in the system, leaving no functions of the particular operator for further manipulation. This is in a sense true of the simplification functions associated with the operators +,* and / , for example, because the standard form does not include these operators explicitly. It is also true of an operator such as the determinant operator DET; the relevant simplification function calculates the appropriate determinant, and the operator DET no longer appears. On the other hand, the simplification process may leave some residual functions of the relevant operator. For example, a residual expression COS(Y) will remain after simplifying Eq. (2.1) unless a rule for the reduction of cosines into exponentials is introduced. These residual functions of an operator are termed <u>kernels</u> and are stored uniquely like variables. Subsequently, the kernel is carried through the calculation as a variable unless transformations are introduced for the operator at a later stage. To include kernels in our standard form representation, we simply replace Eq. (2.5) by

<standard power> :: = (<kernel> ' <non zero positive integer>) (2.6)

and add:

<kernel> :: = <variable> | (<operator> ' <simplified list of arguments>) (2.7)

Often an assignment statement such as Eq. (2.1) is intended in the sense of a 'side relation' in that a substitution for A should be made if it occurs in expressions encountered later in the calculation. As the initial reduction of an expression to canonical form often involves considerable computation, it is obviously desirable to simplify it only when necessary, and then only once during a calculation. In such circumstances, no evaluation of the expression is necessary at the time of definition of the substitution, and it may therefore be stored in <u>quoted</u> form rather than evaluated form. To indicate this, Eq. (2.1) might best be written

```
A = '2*B*C - 3*D**2 + COS(-X)*COS(Y)'                    (2.8)
```

In REDUCE, a quoted assignment is introduced by the instruction LET, as in

```
LET A = 2*B*C - 3*D**2 + COS(-X)*COS(Y);                 (2.9)
```

whereas an intended simplification is written

```
SIMPLIFY A = 2*B*C - 3*D**2 + COS(-X)*COS(Y);            (2.10)
```

When an expression to be simplified contains variables which have been previously assigned either quoted or evaluated values, the speed of the calculation, and more important, the amount of storage used often depends crucially on just <u>when</u> the substitution for the relevant variables is made. There are of course many ways to make such substitutions. One way is to substitute for variables as they are met during reduction to canonical form, recognizing, as AUTSIM does in FORMAC[5], those variables whose substitutions have themselves already been reduced to canonical form to avoid repetitious calculation. Secondly, substitutions can be made after reduction of the whole expression to canonical form. Two simple examples will illustrate that neither method is better in all circumstances. With the substitution (2.1) already defined, consider the following assignments:

```
A1 = (A - 2*B*C + 3*D**2 - COS(-X)*COS(Y))**1000         (2.11)
B1 = A**1000 - A**1000                                   (2.12)
```

Both A1 and B1 evaluate to zero, but in the former case, it is obviously better to substitute for A before raising the expression to the thousandth power and simplifying, whereas in the second case, the opposite is true. These are extreme cases, of course, but illustrate just what can happen if you are not careful. Both substitution mechanisms are implemented in REDUCE and the decision as to whether the substitution of variables is made during or after reduction to standard forms is to a limited extent under user control. However, it has been found in practice that the system can often make a better decision than the average user in this regard.

We note in passing that a simplication of

```
C1 = (A - 2*B*C + 3*D**2 - COS(-X)*COS(Y))**1000
   + A**1000 - A**1000                                   (2.13)
```

would involve catastrophic term growth regardless of which of the above methods of substitution is used. The user could of course avoid this by simplifying A1 and B1 as described above, and then adding them to form C1. However, a simplification step such as Eq. (2.13) may occur in the middle of an extensive calculation without the user's knowledge. To ask the system to make such a simplification in a single step, however, would require sophisticated heuristics to decide the optimal substitution method for each variable encountered far beyond the scope of present simplification systems.

As an extension of the simple variable substitutions discussed so far, REDUCE allows the user to define substitutions for powers of variables, and expressions which reduce to kernels or powers of kernels. Again, these substitutions may be made either during reduction to canonical form or after, and because of the organization of the system, are as efficient to implement as substitutions for variables. We illustrate this by discussing in detail the mechanism for substituting for kernels or kernel powers after reduction of an expression to canonical form. There are two ways by which this may be done. The first is to scan the expression, and check whether each kernel has a substitution defined for it. If it does, then at the <u>first</u> occurrence of this kernel its substitution in canonical form must itself be checked for replacements by the same routine and then stored in this new form. After this expression has been substituted for the kernel, the search continues until the end of the expression is reached. The reconversion of the expression to canonical form can be made concurrent with the search procedure. This method is somewhat inefficient, however, as the same kernel often occurs many times in the same expression. The second method recognizes this, and performs the substitutions in three passes. In the first, the lists of kernels which are kept in the system are searched, and a change of list structure made if a substitution is required. Since kernels and kernel powers are stored uniquely, this change in list structure means that every occurrence of the substitution expression has been changed in all expressions, and so the current expression being simplified can then be reconverted to standard forms by a second pass. The last pass,

which is quite trivial, restores the original list structure of all substituted kernels without affecting the reconverted canonical form.

In actual practice, REDUCE uses a combination of these methods; it is assumed that the expression is large and the substitutions all relatively small. So in checking whether a substitution expression itself contains terms which themselves have substitutions, the first method is used, whereas the second method is used on the current expression being simplified.

In substituting for powers of variables, or kernels, there are occasions when a distinction has to be made between substituting for that explicit power, and a general substitution for that power whenever it occurs. For example, LET I**2 = -1, implies that I**3 = -I, I**4 = 1 and so on. However, in integrating an expression by explicit substitution, a substitution

X**2 = Y**3/3

is not intended to apply to higher powers of X.

This latter type of substitution is really a matching operation and is treated as such by the REDUCE system. Thus a user would say

MATCH X**2 = Y**3/3; (2.14)

to affect such a replacement. The general matching operation requires an altogether different programming technique than we have used so far, and is discussed in the next section.

III. MATCHING OF EXPRESSIONS

The substitutions considered so far have been rather limited in scope, as they involve only substitutions for variables and kernels. As we have seen, these are very efficient to implement because variables and kernels are stored uniquely in REDUCE. However, a more general type of substitution is often needed which requires extensive pattern matching within a given expression. Such substitutions cannot be as efficiently implemented as our earlier examples, since much more searching is involved in their application.

The ideal system would allow for the replacement of any given expression f(a,b..x,y..) by another expression g(a,b..x,y) where a,b..

stand for fixed sub-expressions and x,y.. for arbitrary expressions. For example, in Eq. (2.1), it might be convenient to replace COS(X)*COS(Y) by (COS(X+Y) - COS(X-Y))/2 . Presumably this type of replacement should apply whenever an arbitrary product of cosines is encountered, so that X and Y in the replacement rule should stand for any expression at all. Thus X and Y are <u>free</u> variables as far as the substitution rule is concerned. Similarly, if X is free, then the rule

SIN(X)**2 + COS(X)**2 = 1 (3.1)

should imply that $\sin^2(\cos(\log 2)+3) + \cos^2(\cos(\log 2)+3)$ is to be replaced by 1.

This general matching problem, which we mentioned in an earlier publication[2] remains unsolved in a manner efficient enough to be used in large scale calculations, and most systems, including REDUCE, compromise at some point in the types of substitutions allowed. There is also a basic ambiguity associated with any substitution rule involving addition, such as Eq. (3.1). For example, given this rule, should $2\cos^2(v) + \sin^2(v)$ be replaced by $1 + 2\cos^2(v)$, $2 - \sin^2(v)$ or be left unchanged? The choice made can often influence the compactness or symmetry and hence the intelligibility of the result, as we shall see in Sec. 4.

Even though REDUCE does not implement a general pattern matching algorithm, it does provide for substitutions for products of kernel forms or expressions which reduce to this form by means of the instruction MATCH.

The argument of MATCH is a list of equivalence expressions of the form

<kernel form> * <kernel form> ... * <kernel form> = <expression> (3.2)

where a kernel form is an expression which reduces to a kernel on simplification. Examples of the use of MATCH are

```
MATCH A**2*B = 3*C,
      COS(X)*COS(Y) = (SIN(X+Y)-SIN(X-Y))/2;
```
(3.3)

In the second example, the fact that X and Y may stand for any expression is signified by the prior declaration

```
FREE X,Y;
```
(3.4)

The implementation of this algorithm involves setting up a list of declared 'matches' and then scanning the relevant expression in canonical form for kernels which appear in these matches. If such a kernel is found, then all matches containing that kernel are added to the list of matches with that kernel moved from the left to the right hand side of the substitution. This process then continues until one of two things occurs; either a particular match is successful, in that all products in a matching rule have been encountered, or the end of a standard term is reached. If a match is successful, a second pass makes the required substitutions in the relevant term.

If a match was made during the complete scan of the relevant expression, the whole process must then be repeated in case another valid match developed during the previous pass. Thus the expression must be scanned at least twice to implement such matches.

In spite of the limited nature of the types of substitutions allowed in REDUCE, it is surprising how useful a matching operation of the form defined in Eq. (4.2) can be. This is especially true of problems involving analytic integration of multivariable expressions by table lookup which occur quite frequently in elementary particle physics.

IV. SUBSTITUTIONS IN OUTPUT

Almost as catastrophic as the growth of expressions during a calculation can be the growth of output to the astonished user. This is not a trivial problem, because several physicists and engineers to the author's knowledge have given up calculations when confronted with fifty pages of output from a relatively simple problem in matrix manipulation. It is obvious that if any real progress is to be made in handling algebraic problems too tedious and complicated to be done by hand, then a lot of research must be devoted to presenting output in a compact, intelligible form. One way, of course, is to pick out the leading terms in the expression by order of magnitude arguments, but this method often conceals symmetries in the answer which can only be seen in the complete expression. Another method, developed by

Baker[7], is to recognize common sub-expressions within an expression and replace them by a single variable, thus displaying the underlying fundamental, or 'skeletal' structure of the expression. In many cases however such underlying structure, if it exists, is hidden because of the existence of various relations between the variables occurring or functional identities such as Eq. (3.1). The example shown in Fig. 1 illustrates this point very well. Figure 1a shows a shows a 'raw' expression produced by the computer. However, as in many problems in physics and engineering, not all the variables appearing in the expression are independent, and certain combinations have a more relevant physical interpretation than others. The relations between the variables used is given in Fig. 1b, and it can be seen that of the thirteen variables occurring, only six are independent. About five man hours spent sitting in front of a CRT display modifying expressions and checking in the computer that no errors were introduced by the hand modifications resulted in the expression in Fig. 1c. Considerable reduction in the size of the expression has been made by appropriate substitutions for the variables appearing in the answer. The explicit skeletal structure of this result could also be displayed by replacing the common sub-expressions PROP1+PROP3 and PR*PROP1+RS*PROP3 by simple variables. The goal of 'simplification' in this context is surely reduction of the size and/or symmetrization of the expression. There is something of an art involved in guessing the right substitutions, but it is obvious that the computer could be programmed to do a lot of this automatically. The author's progress in this area during the past year has not been outstanding. However, some success has been achieved by successive substitution for each relevant variable at each place that it occurs in an expression, and then checking if the substitution was successful in that the number of terms in the expression decreased. However, this method is painfully slow, so that a human and computer interactive combination remain economically more attractive at the moment. This type of problem is analogous in many ways to theorem proving on a computer, and it is probable that similar heuristics will have to be developed here before a successful solution can be found. There may be other

algorithmic methods which could be used also. Engeli[8] for example, has suggested dividing the expression by any substitution equivalent to zero and thus keeping only the remainder for further manipulation, but the author has found that this provides little reduction in expressions involving low powers of many variables such as the exampl in Fig. 1.

The problem of substitution then is one of the key problems to be considered in designing and using a simplification system for large expressions. Expressions must be kept small, both during a calculation and at the end if anything really new is to be learned from doing non-numerical mathematics on a computer.

```
(        M**4 *

(2 * PROP1 * PR * RS    -    2 * PROP1 * PR * RT    -    4 * PR**2 * RS
 -    4 * PR**2 * RT    -    4 * PR * RS**2    +    14 * PR * RS * RT    +
  2 * PR * RS * PROP2    -    4 * PR * RS * PS    -    4 * PR * RS * PT    -
    4 * PR * RS * QT    -    4 * PR * RS * QS    -    4 * PR * RS * QR    -
   10 * PR * RT**2    -    2 * PR * RT * PROP2    +    4 * PR * RT * PS    +
    4 * PR * RT * PT    +    4 * PR * RT * QT    +    4 * PR * RT * QS    -
   4 * PR * RT * QR    -    6 * RS**2 * RT    -    4 * RS * RT**2    -    6 *
  RS * RT * QR    -    6 * RT**3    +    6 * RT**2 * QR)


   +    M**2 *

(    -    PROP1 * PR * RS * RT    +    PROP1 * PR * RT**2    +    PROP1 * P
R * RT * PROP2    +    PROP1 * RS**2 * RT    +    2 * PROP1 * RS * RT**2
 -    2 * PROP1 * RS * RT * PT    +    PROP1 * RT**3    +    2 * PROP1 * RT
**2 * PS    +    6 * PR**2 * RT * QT    -    2 * PR**2 * RT * QS    +    4 *
  PR**2 * RT * QR    -    4 * PR * RS * RT * PROP2    +    4 * PR * RS * RT
  * PS    +    8 * PR * RS * RT * PT    +    4 * PR * RS * RT * QT    +    2
* PR * RS * RT * QS    -    4 * PR * RS * RT * QR    +    8 * PR * RS * PS
  * QT    +    8 * PR * RS * PS * QR    -    4 * PR * RT**3    +    2 * PR *
RT**2 * PROP2    +    4 * PR * RT**2 * PT    +    6 * PR * RT**2 * QT    -
   4 * PR * RT**2 * QS    +    PR * RT * PROP2**2    -    2 * PR * RT * PRO
P2 * PS    -    2 * PR * RT * PROP2 * PT    -    2 * PR * RT * PROP2 * QT
   -    2 * PR * RT * PROP2 * QS    -    8 * PR * RT * PS * QT    -    2 * P
R * RT * PS * QR    -    2 * PR * RT * PT * QR    +    2 * RS**2 * RT * QT
   +    4 * RS**2 * RT * QR    -    4 * RS * RT**3    -    2 * RS * RT**2 *
PROP2    +    4 * RS * RT**2 * PS    -    4 * RS * RT**2 * PT    +    6 * R
S * RT**2 * QT    -    2 * RS * RT**2 * QS    -    2 * RS * RT * PROP2 * P
T    +    RS * RT * PROP2 * QR    +    4 * RS * RT * PS * PT    +    2 * RS
* RT * PS * QR    +    4 * RS * RT * PT**2    +    4 * RS * RT * PT * QT
  +    4 * RS * RT * PT * QS    +    4 * RT**3 * PS    -    2 * RT**3 * QS
  +    4 * RT**3 * QR    +    2 * RT**2 * PROP2 * PS    -    RT**2 * PROP2 *
 QR    -    4 * RT**2 * PS**2    -    4 * RT**2 * PS * PT    -    4 * RT**2
* PS * QT    -    4 * RT**2 * PS * QS    +    4 * RT**2 * PS * QR    +    2
* RT**2 * PT * QR)


   -    2 * PROP1 * RS * RT**2 * PS    -    2 * PR**2 * RT * PROP2 * QT
  +    8 * PR * RS * RT**2 * QT    +    2 * PR * RS * RT * PROP2 * QT    -
    8 * PR * RS * RT * PS * QT    -    8 * PR * RS * RT * PS * QR    -    4
  * PR * RS * RT * PT * QT    -    4 * PR * RT**2 * PS * QT    +    4 * PR
* RT**2 * PS * QS    +    4 * PR * RT * PROP2 * PS * QT    +    2 * PR * R
T * PROP2 * PS * QR    +    4 * RS**2 * RT * PT * QT    -    4 * RS * RT**
2 * PS * QT    -    8 * RS * RT * PS * PT * QT    -    4 * RS * RT * PS *
PT * QR    +    8 * RT**2 * PS**2 * QT) /


(    -    4 * PROP1 * PR * RS * RT**2 * PROP3)
```

a) Expression initially produced by computer.

Figure 1

```
PQ = M**2 - PROP1/2,

PR = QR + RT - RS,

PS = QS + RT - PROP1/2,

PT = QS - PR + RT,

QS = M**2 - PROP3/2,

QT = PS - QR - RT,

PROP2 = PROP1 - 2*RT + 2*RS
```

b) Relations between variables.

```
(4*M**4 - (PROP1+PROP3)**2)*(- 2*M**2*QR - 4*QR*RT
        + 2*RT**2 - RT*(PROP1+PROP3)+(PR*PROP1+RS*PROP3)
        + 2*M**2*PR*RS/RT)
+ 4*M**2*QR*(PR + RS)*(2*M**2 + RT + (PROP1+PROP3))
+ 2*M**2*PR*RS*(2*QR - 6*RT - 3*(PROP1+PROP3))
+ 2*(QR - RT)*((PR*PROP1+RS*PROP3)*(M**2 - (PROP1+PROP3))
        + 2*QR*RT*(PROP1+PROP3))
+ 2*(QR**2 + RT**2)*(2*QR*RT - (PR*PROP1+RS*PROP3)
        + RT*(PROP1+PROP3)) + 6*M**2*RT**2*(PROP1+PROP3))
( 4*PROP1*PROP3*RT*PR*RS)
```

c) Final result produced by man and machine.

Figure 1

REFERENCES

1. A. C. Hearn, "REDUCE User's Manual", Institute of Theoretical Physics Stanford ITP-202, Stanford Artificial Intelligence Memo No. 50 (revised) April 1968.

2. A. C. Hearn, "REDUCE, A User-Oriented Interactive System for Algebraic Simplification", Proceedings of the ACM Symposium on Interactive Systems for Experimental Applied Mathematics, held in Washington D.C., August 1967 (to be published).

3. A. C. Hearn, "Computation of Algebraic Properties of Elementary Particle Reactions Using a Digital Computer", Communications of the ACM 9, 575, (1966).

4. J. McCarthy, et. al., LISP 1.5 Programmer's Manual, Computation Center and Research Lab of Electronics, M.I.T. Press, Cambridge Massachusetts, 1965.

5. G. E. Collins, Communications of the ACM 9, 578, (1966).

6. Tobey, R. G., Bobrow, R. J., and Zilles, S. N., "Automatic Simplification in FORMAC", Proceedings of the AFIP 1965 Fall Joint Computer Conference, Part 1, November 1965, p. 37.

7. R. G. Tobey, "Experience with FORMAC Algorithm Design", Communications of the ACM 9, 589, (1966).

8. Max Engeli, private communication.

4. Hand-Eye Systems

The following paper, titled "Computer Control of a Mechanical Arm through Visual Input", was presented at the 1968 IFIP Congress by Karl K. Pingle, Jonathan A. Singer, and William M. Wichman. Mr. Pingle and Mr. Singer are research staff members of the Artificial Intelligence Project. Mr. Wichman is a former graduate student, now with Sanders Associates.

Hand-eye research at Stanford was initiated by Professor John McCarthy and has recently come under the leadership of Professor Jerome Feldman.

BLANK PAGE

INTRODUCTION

A growing body of research within the field of Artificial Intelligence has been concerned with giving computers the ability to interact directly with their environment. Much of this effort has been concerned with visual perception, while a smaller amount has been directed toward such areas as computer control of mechanical manipulators.

Ernst [1] used a computer-controlled arm equipped with tactile and position sensors to find and pick up blocks scattered on a table, while pioneering work by Roberts [4] produced a computer program that could recognize certain solid objects and their photographs. Recently work has begun on systems that combine perception and manipulation in a coherent way. Impetus for the development of such systems stems from a number of sources. Even those systems with very modest capabilities find immediate application in various assembly-line tasks. Other potential application areas include planetary probes, where communication delays necessitate some degree of autonomy in the device.

Hand-eye work in the Stanford Artificial Intelligence Project is aimed at bringing together the perception and manipulation processes to perform interesting tasks. Related work is being done at M.I.T. At Stanford, we have assembled a rudimentary system employing a vidicon television camera and an electrically powered arm controlled by a program running under a time sharing system on a PDP-6 computer. This initial system performs simple sorting and stacking operations on cubical blocks.

Our main goal is to develop computer-hand-eye systems that are better for some purposes than human systems. For example, they may be faster, stronger, more economical, or more expendable. The problems of perceiving three-dimensional objects in complex scenes and of manipulating them without accidently striking other objects are quite challenging.

The following sections discuss both the equipment and programs employed in the current system and give some quantitative results.

System Description

1. Task Descriptions

The two tasks which the system is currently able to perform are the sorting of cubes by size into separate stacks, and the alignment of one cube on top of another.

In the first task, the eye program locates a cube in the television camera's field of view and passes the coordinates of the television camera's vertices to the arm program. The size of the block is computed and compared with block sizes in any stack already begun. If it matches one of them within a set tolerance, the arm picks up the cube and places it in that stack. Otherwise the cube is placed on the table as the beginning of a new stack.

The second task begins with two cubes in full view of the vidicon. The eye program locates one block, and the arm picks it up and removes it from view. The second block is then located and the arm places the first block on top of it. The arm is moved away and the eye program measures the alignment error between the two blocks. The arm then repositions the top block so as to eliminate the error. This process is repeated until the error is within a fixed tolerance.

Depth information in the foregoing tasks is inferred by a technique known as the support hypothesis [4]. Simply stated, it says that from a single viewing position the location of any point in a known plane can be determined. Mathematically this is simply a mapping between two fixed planes, which within the Hand-eye system are the table top or support plane, and the image plane of the camera.

2. Hardware

The current Hand-eye system consists of a vidicon television camera and an electrically powered arm, both connected to a Digital Equipment PDP-6 computer.

The video signal is digitized and stored in the computer memory. Any rectangular portion of the image up to 250 by 333 points in size may be read under program control, with the light intensity at each point encoded as 4 bits (16 levels).

The arm currently in use was designed and built by a group at Rancho Los Amigos Hospital near Los Angeles as a device to be strapped to a paralyzed human arm. Consequently, it is quite anthropomorphic in size and form. It is powered by small permanent magnet gear-head motors mounted on the arm, giving joint velocities of 4 to 6 r.p.m.

without load. Position feedback is provided by potentiometers, one of which is mounted at each of the six joints. The hand is a two finger parallel grip device approximately the size of a human hand, and has a maximum finger opening of three and a half inches. The maximum reach of the arm is about 27 inches, and its weight is about 15 pounds. Power to the actuator motors is provided by an external D.C. supply controlled by instructions from the computer. This power takes the form of 16 volt pulses of constant width, whose repetition rate is determined by the controlling program.

Other peripheral equipment used includes a point plotting CRT display, a standard TV monitor, and a teletype console. The arm and camera are mounted on top of a large table. The other equipment is distributed around the table at convenient locations.

3. Software

All of the programs of the Hand-Eye system are run within the time sharing system of the PDP-6. The present set of programs consists of: an "eye" section which is capable of reading the camera and locating cubes on the table; an arm section which, given the position of a cube, can pick it up and move it to a desired position; a control program which sequences the other programs; and calibration routines which relate the camera and arm coordinate systems to the workspace.

This collection of programs occupies approximately 24,000 words of storage including data areas. The eye programs and the arm control routine are written in machine language, while the arm solution programs and coordinate transformation are in Fortran.

INFORMATION FLOW

Figure 1 shows the logical flow of the program. Control routines to be discussed later route the information from one section of the program to another, modifying the flow slightly to perform various tasks.

Before running the program a calibration routine relates four points on the table to their positions in the TV image by tracing around four triangles permanently placed on the table and locating their right angles, whose position on the table is known to the routine. The program uses

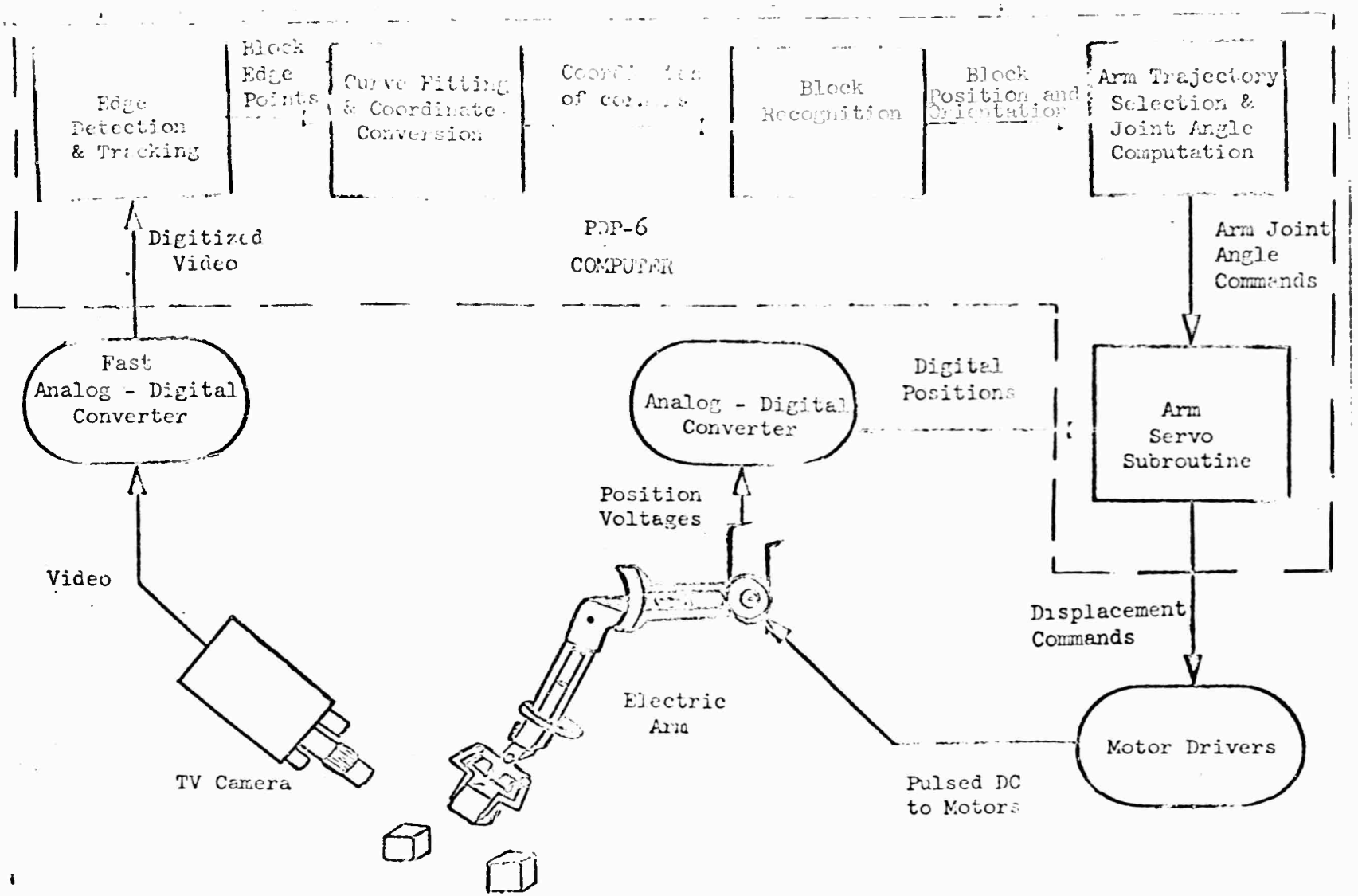
Edge Detection & Tracking
Block Edge Points
Curve Fitting & Coordinate Conversion
Block Recognition
Block Position and Orientation
Arm Trajectory Selection & Joint Angle Computation
Digitized Video
PDP-6 COMPUTER
Arm Joint Angle Commands
Fast Analog - Digital Converter
Analog - Digital Converter
Digital Positions
Arm Servo Subroutine
Position Voltages
Video
Displacement Commands
Electric Arm
Motor Drivers
TV Camera
Pulsed DC to Motors

Figure 1

this information to compute a matrix for transforming a point in the camera coordinate system into the arm coordinates of its projection onto the table [2]. Since the transformation is two-dimensional, only points lying on the table are transformed correctly.

When an object is to be located, an edge following routine is called [3]. It starts at the bottom of the image and scans across and up looking for an edge and computing the average intensity of the background. When a change in intensity is detected, the program moves around the outer edges of the object it has found making a list of the coordinates of the points it finds on the edges. After each point is found, the program moves to a point a small distance in the direction the edge is running, and applies a gradient operator, using the nine points surrounding the new point. If the magnitude of the vector produced by the operator is over a set threshold, the program is assumed to still be on the edge. Since the edges are usually several resolution elements wide, the program uses a hill climbing technique to find the largest magnitude, which is taken to correspond to the center of the edge. If the magnitude is below the threshold, the program is assumed to be off the edge. It then compares the intensity at that point with the average intensity of the background to determine whether it is inside or outside the object and moves in the appropriate direction to find the edge again. Having found the center of the edge, the program uses the direction of the vector to determine the direction of the edge. Only the outline of the object is seen since the hardware does not yet provide the capability to trace interior edges. Figure 2 shows the three types of outlines the program finds. Only the solid lines are traced.

After the outline of the object has been traced, a curve fitting routine converts the list of points into the coordinates of the corners of the object in two passes. In the first pass, the routine goes through the list of points, combining them into short line segments. The second pass combines each adjacent pair of lines into a single line if they are nearly parallel. Otherwise, the first of them is taken to be a complete side of the object and the process is continued with the second line.

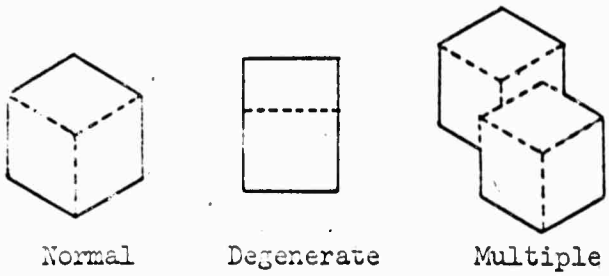

Figure 2

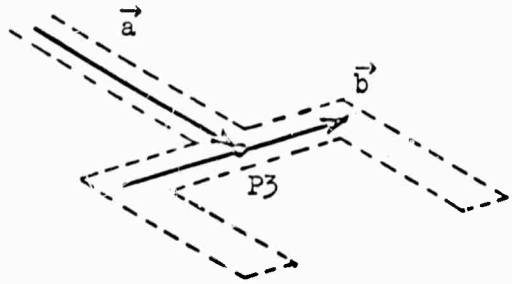

Figure 3

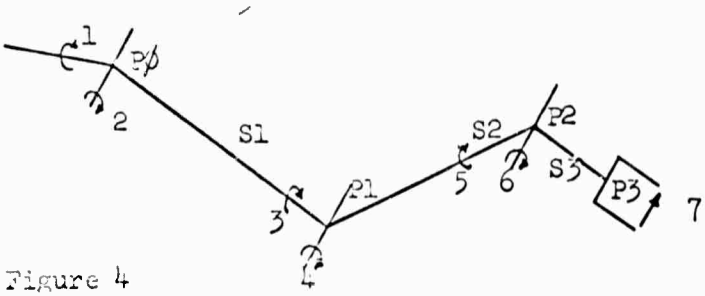

Figure 4

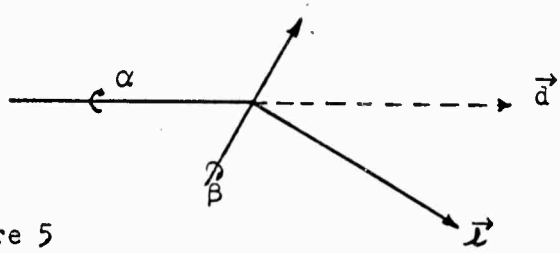

Figure 5

The output of the edge follower and curve fitting routine is a data structure which represents the object in a way that simplifies further processing. The data structure is generated by means of a list processing language designed for this program. The data structure, which resembles that of CORAL, is created by MACRO statements in the PDP-6 assembly language which set up calling sequences and then call subroutines in the language's processor. The edge follower can trace around objects of any shape. The curve fitting routine, however, can only fit straight lines to the points.

Before exiting the curve fitting routine, the coordinates of the corners are transformed into the arm's coordinate system. The difference between the actual location of the corners and the computed location is generally less than .04 inches and repeatable to about .03 inches. The repeatability error is due to the resolution limitation of the TV, as well as to slight jitter, quantization error, and noise. Much of the location error is probably a result of errors in the positions of the points used to calculate the transformation matrix.

After the objects have been located, the program searches for the cube nearest the camera. The object outline is tested in various ways to determine whether or not it is a cube. These tests include checking the number of points seen along the outline of the object and rejecting the object if the number is too small, comparing the lengths of the edges meeting at the lowest corner for equality, and testing the angle between the edges for perpendicularity in table space. Special tests are used when the block is seen head on, giving a degenerate outline with only two visible corners on the table. Since only the outline is traced and several blocks may be close enough together so that their images overlap, the object may be the outline of several blocks. The coordinates of the lowest corner in the image and of the two adjacent corners are given to the arm routine. In the degenerate case, the invisible corner is given calculated coordinates. The coordinate transformation is correct only when the points are on the table, and these are the only points which are sure to be on the table. The arm program must calculate any other information needed from these three points and from the knowledge that the object is a cube.

In order for a manipulator to grasp objects with arbitrary positions and orientations, it must have at least seven degrees of freedom: three for position, three for orientation, and one for grasping.

In order to control such a device, we must calculate for the actuators of the mechanism, the deflections that will locate it at a desired point in the six-space of position and orientation. In the case of a parallel jaw hand, the position to be reached can be described in terms of a point and two orthogonal vectors, as shown in Figure 3, where P3 is the point the hand must reach, $\vec{a}$ is the vector indicating the direction the hand is pointing (the approach direction), and $\vec{b}$ is the orientation of the plane of the hand about its approach direction.

The geometric configuration of the arm is shown in Figure 4. 1, 2,...,6,7 are the actuations, S1, S2, S3 are links in the arm, and PØ, P1, P2 are the shoulder, elbow, and wrist joints respectively.

Calculating the defection angles required to achieve a desired position and orientation is done as follows: the arm is imagined to be in the desired location. Using the shoulder position and the constraints on the hand, the position of the wrist and elbow can be calculated. This in effect determines the direction of each link in the arm, whence the actuation angles can easily be calculated.

In general, there are multiple solutions to any given positioning problem. There are two locations for P1 (the elbow) which yield identical wrist positions. In addition, there is a pair of solutions at each joint, since each one is symmetric. That is, the same link direction $\vec{\gamma}$ (Figure 5) can be achieved by rotating α 180 degrees and deflecting $\vec{\beta}$ by the same angle θ to the other side of line $\vec{d}$. Since there are three joints, there are eight possible combinations for each of the two elbow positions, or a total of 16 solutions. Not all of these are realizable because the arm has mechanical stops limiting maximum deflections.

The existing control system used with the arm consists of an analog-to-digital converter for reading potentiometers on joints, an output register for motor pulsing, and a servo program. Since this program must operate in real time within a time sharing system, it is treated as a

special case by the system, and is given control every 16.7 ms (60 Hz). The program is a simple proportional servo which calculates the pulse rate for each motor. Pulse width for each motor is manually adjustable, and is set to a maximum consistent with the requirement that a single pulse give as small a motion as is necessary for fine movements.

The delay between pulses to each motor is calculated by dividing by a constant the position error and using the result (limited by a minimum and maximum value) to determine the period of time before the next pulse. Velocity damping is unnecessary since the joints have a great deal of internal friction.

CONTROL

The eye and arm routines are called by a control program coded to perform a specific task. Two tasks are currently being performed: sorting cubes by size and building a stack of blocks of each size, and stacking one cube on top of another using the eye to attain more precise alignment. Since both tasks are similar in their control, the sorting program will be described in detail and the other one outlined.

The control program calls the edge follower, requesting a cube position. After curve fitting and testing, if the control program decides that the object returned is not a cube, or has not been traced well enough, it calls the edge follower for another cube. If the object is all right, the arm routine is called and given the block position.

The arm program checks the size of the cube to see if it has seen a cube this size before. If it has, the arm picks up the cube and puts it on the stack for that size. Otherwise, it computes a position a small distance from the last stack and starts a new one. If the outline given it contained a number of cubes, the arm will attempt to pick up the lowest one on the TV image. With the bottom cube removed, the next pass of the eye program will give the outline of the group without that cube, and, eventually, the arm will have removed all the cubes in the group. Thus we avoid the problem of analyzing the composite outline.

To simplify processing, visual input is not used while moving and stacking the cubes. When a cube has been picked up, the control program

can tell from the position of the fingers if the cube has been dropped. If so it reverts to the search phase. After moving the cube to the proper position and releasing it, the fingers are closed again to determine if the cube is still there. If not, the stack has fallen over and the control routine asks the operator to tell how many cubes are left on the stack. The program continues to run through this loop until the stacks are so high the arm cannot reach the top, or it runs out of cubes.

For the other task [2], the transformation calculation is done twice, first using points on the plane and a second time using points at a height above the table equal to that of a two-block stack. The program finds two cubes and sets the second one it finds on top of the first. It then moves the arm out of view of the camera and traces around the stack. It puts the three lowest corners through the transformation computed on the table and the three highest through the other transformation. If the cubes are aligned perfectly, the top four corners of the cubes will have the same coordinates, in two dimensions, as the bottom corners directly below them. Otherwise, the program computes the center of the cubes and their orientation, uses this information to compute the rotation and translation necessary to align them, and moves the upper cube accordingly. This process repeats until the blocks are aligned to within a set tolerance.

CONCLUSION

The present system performs the various tasks at a speed considerably slower than a person would. The visual portion is fairly fast; it takes under five seconds to trace around an object and about 1/10 second to do the curve fitting. The arm routines are limited by the speed of the arm which must run rather slowly to maintain sufficient positioning accuracy. The arm requires about 30 seconds to pick up and stack a block. In the case where visual feedback is used, the program generally requires two or three trials to position the block correctly. If many more attempts are made, the accumulation of arm and position errors usually prevents the program from ever succeeding; the top cube will eventually be knocked off the stack and the process will start over.

Several additions and modifications to the program and hardware are planned to improve the performance of the system. A new eye, employing an image dissector, is being tested as a possible alternative to the vidicon. So far tests have shown that the device has more resolution and less noise than the vidicon although it is considerably slower. We hope that with this device it will be possible to find interior edges quite reliably. This should enable us to use visual feedback without having to release the block being held, since the information needed can then be determined from inspecting the edges where the cubes meet.

A hydraulic arm has recently been completed which appears to be faster, smoother and more accurate. This should permit more accurate stacking and reduce the chance of stacks being knocked over by shaky arm movements.

The transformation matrix calibration will be altered to produce three dimensional coordinates, thus providing correct coordinates for corners off the table and facilitating work with objects other than cubes.

Finally, work is starting on a higher level language for describing tasks and operating on the data structure so that the operator can input a task in fairly general terms and the program will be able to carry it out. Using this language we plan to develop a more powerful system capable of a large number of manipulative tasks.

ACKNOWLEDGEMENTS

The authors wish to thank Professor John McCarthy for his suggestions and support which were instrumental in the initiating of this project.

REFERENCES

[1] Ernst, Heinrich A. MH-1, A Computer-Operated Mechanical Hand. Doctoral Thesis M.I.T. December 1961.

[2] Wichman, William M. Use of Optical Feedback in the Computer Control of an Arm. Engineering Thesis, Stanford University. August 1967.

[3] Pingle, Karl K. A Proposal for a Visual Input Routine. Stanford A.I. Memo No. 42. June 1966.

[4] Roberts, L. G. "Machine Perception of Three-Dimensional Solids" in Optical and Electro-Optical Processing of Information. p. 159. M.I.T. Press. Cambridge, Massachusetts. 1965.

5. Speech Recognition

The following paper by John McCarthy, Lester Earnest, D. Raj Reddy, and Pierre Vicens is entitled "A Computer with Hands, Eyes, and Ears". It will be presented at the 1968 Fall Joint Computer Conference.

This paper overlaps the hand-eye paper to some extent, but focuses on recent speech recognition work done under the leadership of Professor Reddy.

BLANK PAGE

I. INTRODUCTION

The anthropomorphic terms of the title may suggest an interest in machines that look or act like men. To this extent it is misleading. Our interest is in extending the range of tasks to which machines can be applied to include those that, when performed by a human, require coordination between perceptual and motor processes. We attempt to suppress the egocentric idea that man performs these tasks in the best of all possible ways.

In place of "eyes, ears, and hands" we could refer to "cameras, microphones, and manipulators", but find latter terms less suggestive of the functions that we wish to emulate. We leave the term "robot" and the ideas that go with it to the science fiction writers who have made them so entertaining.

Shannon, Minsky, McCarthy, and others had considered the possibility of a computer with hands, eyes and ears at one period of another during the latter part of the last decade. The main obstacles to the realization of the idea were the unavailability of suitable computers and I-O devices, and the prohibitive cost of such a system. Ernst[1] and Roberts[2] were among the first few who used a computer to realize these objectives. Glaser, McCarthy, and Minsky[3] proposed that the first major attempt at the biological exploration of Mars should be made by a computer controlled automatic laboratory, containing a wide variety of visual input devices and mechanical manipulators which can under computer control perform many of the tasks of bio-chemical laboratory, requiring only a limited supervision by the experimenter on earth.

The work reported here and a related project at M.I.T. were undertaken several years ago to combine and improve techniques for machine perception and manipulation. Progress has been slower than we hoped because there have been many previously unrecognized problems and few simple solutions. Nevertheless, we have a system that does such things as recognize spoken messages that are combinations of terms previously "learned", "see" blocks scattered on a table, and manipulate them in accordance with instructions.

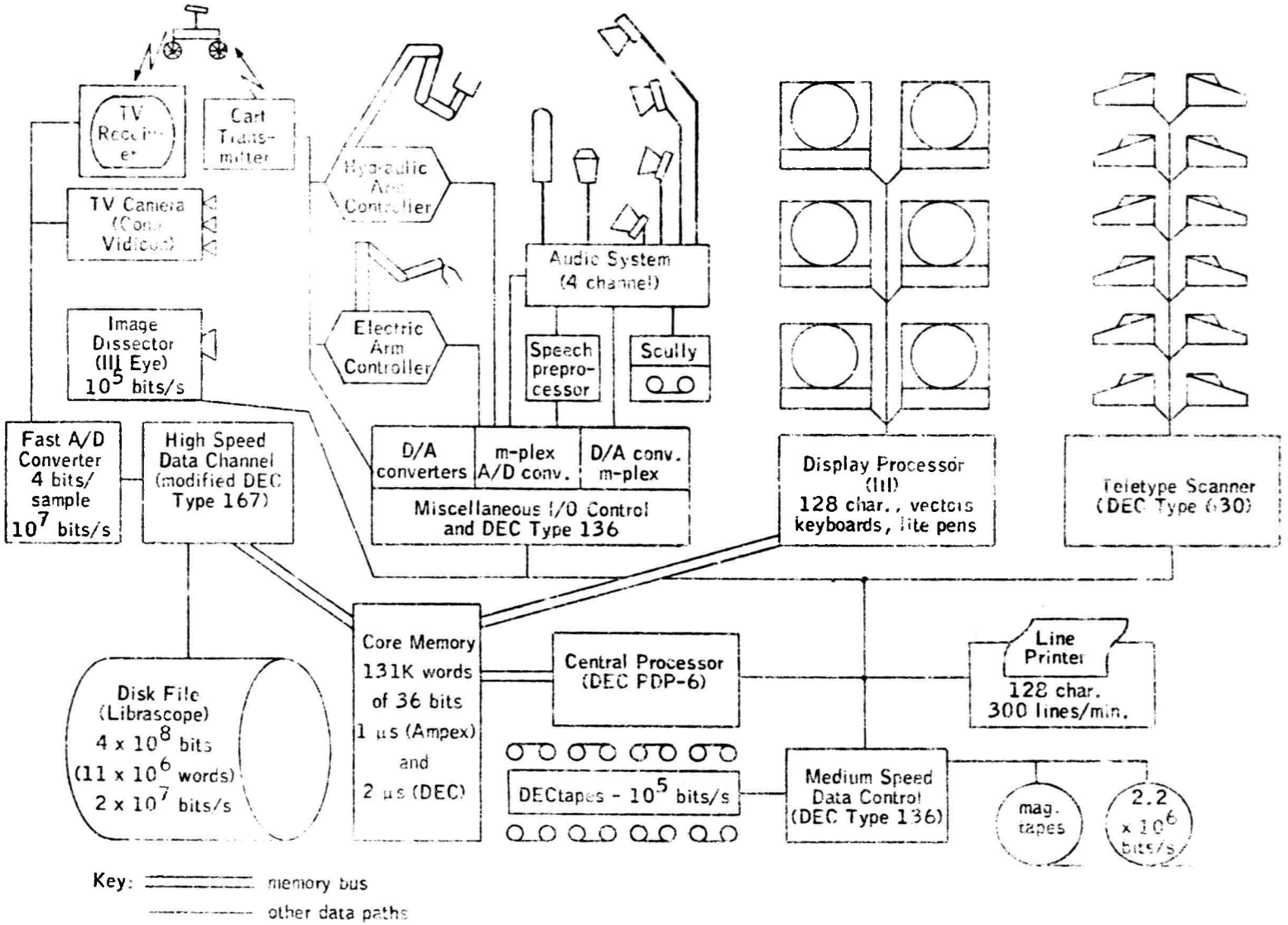

Figure 1. Stanford A.I. Computer System

The work is just beginning. Our existing system exhibits more problems than solutions, but that was its purpose. The following sections discuss considerations leading to the choice of equipment, techniques used to convert the huge masses of television camera and microphone data into useful information, and the control of arms.

II. SYSTEM CONFIGURATION

Major considerations in the choice of system components have been off-the-shelf availability and ease of interfacing. This approach has both advantages and disadvantages. The main advantage, of course, is a relatively quick start. Figure 1 shows major components of the existing system. Figure 2 is a photograph of the hand-eye system.

At the center of the system is a time-shared PDP-6 computer with 131K words of core memory of 36 bits each and an 11 million word fixed-head disk file. The PDP-6 was chosen for having a working time-sharing system, ease of adding special I-O devices, and unrestricted data transfer rates of up to 30 million bits per second between memory and external devices. The Librascope disk file has a 24 million bit per second transfer rate and provides both swapping and permanent file storage.

There are a number of local and remote teletypes, CRT displays, a line printer, plotter, and tape units for general user services.

Our research objectives imposed several special requirements on the time-sharing system. When a person begins to speak, the system must ensure that the audio system is listening to him and must not swap out the program just because its time is up. Similar considerations apply to arms that are in motion. The system must also provide for communication between programs. The DEC time-sharing system was modified locally to meet these requirements.

Visual input to the system is provided by a vidicon television camera operating in accordance with EIA standards. The video signal is digitized to four bits (16 levels of light intensity) and sampled at an instantaneous rate of 6.5 million samples per second. Making use of interleaving, any rectangular portion of the image, up to 666 x 500 points for the full field of view, may be read into memory under program

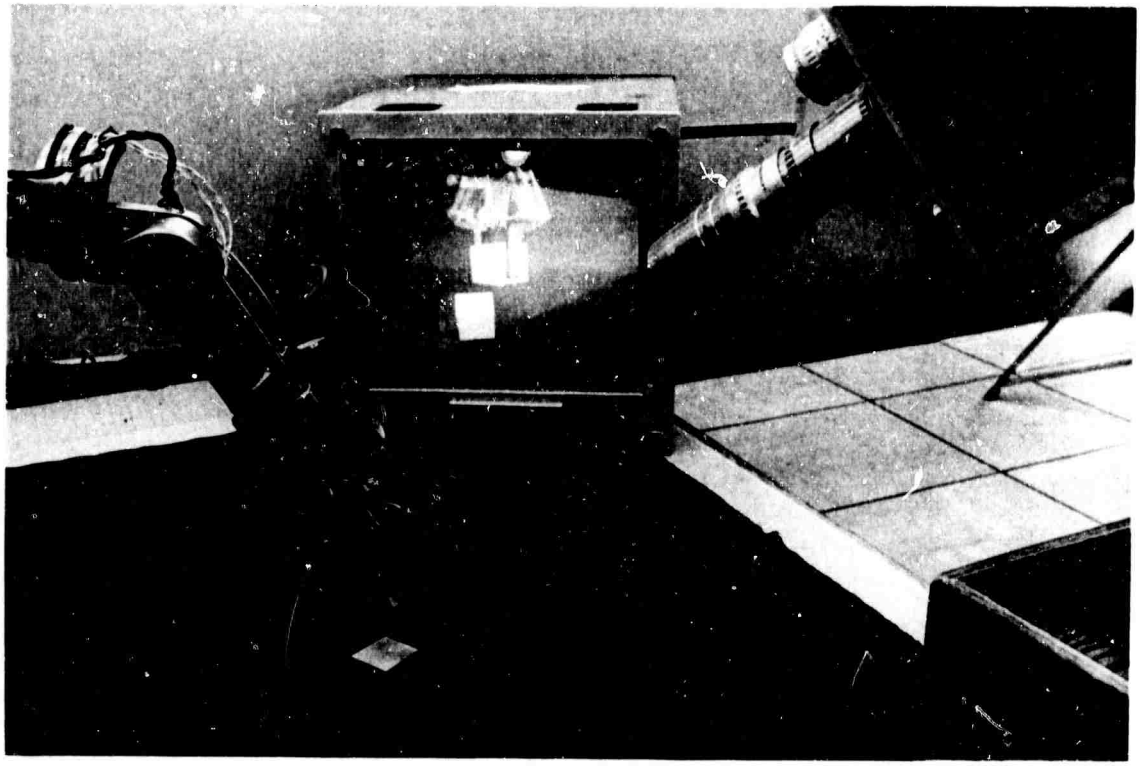

Figure 2. A Picture of the Hand-Eye Equipment

control in two frame times (1/15 second). For static scenes, finer gray-scale resolution can be obtained by averaging measurements over multiple frames.

The laboratory also has an image dissector camera which is capable of measuring the brightness of image points in arbitrary order. It is capable of directly resolving better than 1000 x 1000 points with a gray-scale resolution of 6 bits or more. It is relatively slow if a large number of points are to be read and suffers from settling-time problems when large deflections are used.

A comparative study of optical sensors for computers, including the possibility of a laser eye with direct depth determination, has been made by Earnest[4].

Audio input to the system is provided through an A-D converter connected to the PDP-6. Two different audio devices are currently attached. In one, composed of a condenser microphone and a high quality amplifier, the speech signal is sampled at a rate of 20,000 samples per second and digitized to 9 bits. In the other, shown on Figure 3, the output of a crystal microphone is amplified and filtered into three frequency bands. In each band, the maximum amplitude of the signal and the number of zero crossings are measured by analog circuitry. Every 10 milliseconds, each hold circuit is read by the A-D converter to 12 bits and then reset for the next 10 milliseconds.

Input of the raw speech waveform without any preprocessing hardware, such as a filter bank, has the disadvantage of requiring more processing by the computer and more storage. But on the other hand, it provides the user with a very flexible means of analysis and permits all kinds of processes to be simulated. In fact, we believe that no solution should be implemented in hardware until it has been proven to be one of the best possible solutions by computer simulation. Reddy[5] states that prosodic parameters of speech, requiring the use of segmentation and pitch detection are more easily determined from the direct speech signal than from the output of a bank of filters.

The second audio-device arises directly from the preprocessing program of Reddy and Vicens[6]. They use two smoothing functions which

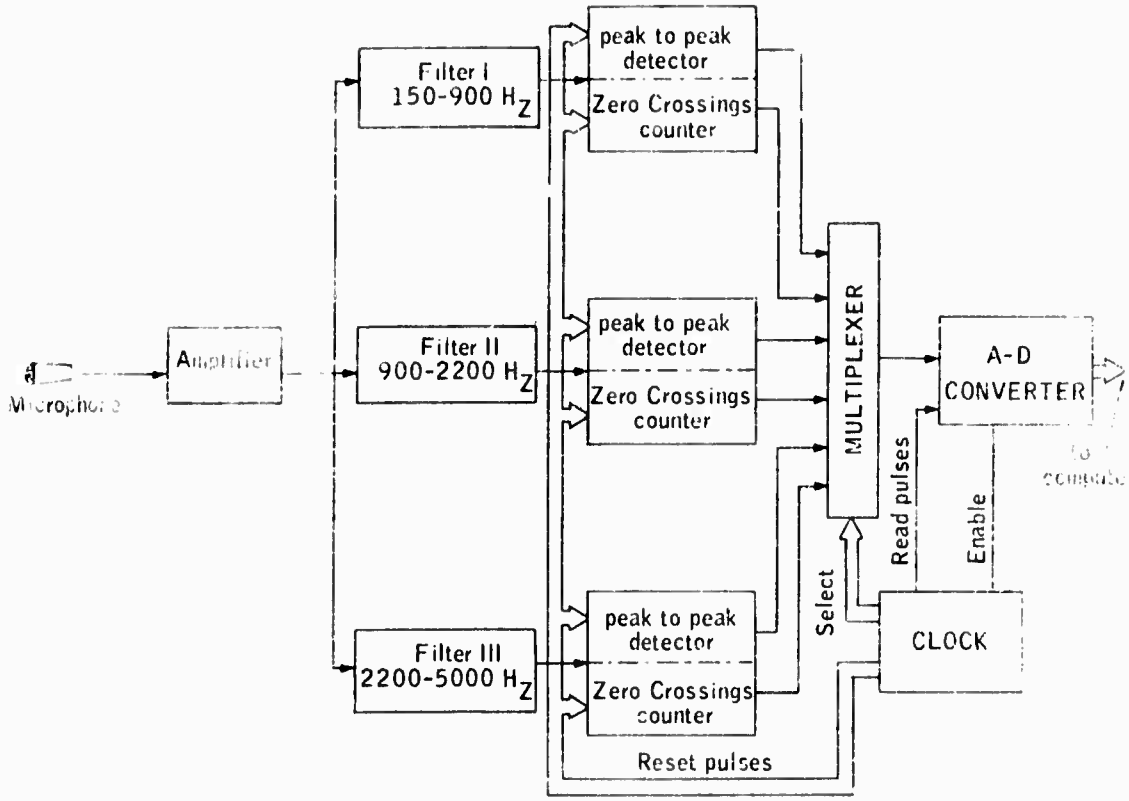

Figure 3. SPEECH PREPROCESSOR

produce approximations of the same parameters (maximum amplitude and zero-crossings). This kind of system is required when one wishes to analyze long utterances, because a direct analysis of the speech wave would consume large amounts of core storage and preprocessing time.

The example of speech recognition discussed below uses the preprocessing hardware. The sampling rate of this device, being very slow (3600 bits/sec.) allows the central processor to process the input as it comes, which is an important consideration if real time speech recognition is desired.

The electric arm was originally designed as a device to be strapped to a paralyzed human arm. It has six degrees of freedom which permits it to place the hand in arbitrary positions and orientations within reach, plus a finger-closing motion. It is powered by small permanent magnet gear-head motors mounted on the arm, giving joint velocities of 4 to 6 r.p.m. with small loads. Position feedback is provided by potentiometers mounted at each of the six joints. The hand is a two finger parallel grip device approximately the size of a human hand and has a maximum finger opening of 6.4 centimeters (3.5 inches). The maximum reach is about 68 centimeters (27 inches), and its weight is about 7 kilograms (15 pounds). Power to the motors takes the form of 16 volt pulses whose width and repetition rate are controlled by the computer program.

In its original form, this arm had a number of maladies such as severe mechanical play in the joints and imprecision in the potentiometer readings due to unstable mountings. Despite several improvements, the arm is still rather slow, shakey, and inaccurate. The position of the hand may differ from computed values by as much as a centimeter.

A hydraulic arm, recently completed, is faster, smoother, and more accurate in its motions. It is also capable of dealing a fatal blow to the experimenter, exhibits other antisocial properties such as leaking hydraulic fluid, and tends to destroy itself periodically.

III. SCENE ANALYSIS AND DESCRIPTION

If we digitize the light intensity at every point in the whole field of view of the television camera, the computer will receive

666 x 500 or 333,000 samples, or 1,332,000 bits of data per frame. The problem of scene description is the formulation of a program which will abstract meaningful descriptions of objects of interest in the scene and their positions.

The problem of scene description must be distinguished from the problem of classifying pictures into categories with which much of the published theory deals. The first working description program was reported by Roberts[2]. Narasimhan[7] suggests that richly structured pictures such as bubble chamber pictures, line drawings, and others are best studied in the form of picture analysis and description and proposes the use of a linguistic model. Recent work by Miller and Shaw[8], and Shaw[9] illustrates the power of the linguistic models in the analysis and generation of pictures.

The linguistic models have the advantage that they can be used for both analysis and generation of pictures, and that many of the powerful tools developed for syntax-directed analysis of languages can be directly utilized for the analysis of pictures. The weaknesses of the present linguistic models, at least as far as the analysis of images of 3-D scenes is concerned, are the following:

- Attempts at describing the connectivity of 3-dimensional objects, using a data structure primarily developed for the description of strings often results in unwieldy and awkward descriptions leading one to doubt whether such descriptions really facilitate analysis. Extension of the models to use list structures instead of strings should remedy this weakness.

- The present linguistic models also suffer from many of the problems of error recovery of syntax directed compilers. This is especially critical when dealing with analysis of pictures; as a result of noise and jitter in the input device spurious lines and edges may appear all over the pictures, and often some of the expected edges may be missing.

- One of the main problems with images of 3-D scenes is not so much how to describe what is visible but rather how to describe what is only partially visible. Heuristics for handling degenerate views of objects cannot be conveniently incorporated into presently proposed linguistic models.

In view of the above consideration it appears that generalizations of linguistic models will be needed before they can be used effectively for the analysis of images of 3-D scenes.

Our existing scene analysis program is related to the one used by Roberts[2] and has recently been described by Pingle, Singer, and Wichman[10]. Instead of repeating that description, we shall note some shortcomings of the existing system and some possible solutions.

The existing eye program locates cubical blocks of various sizes scattered at random on a contrasting background. Its "world model" has room for just one block at a time and those that are partly obscured by others may be perceived only after the intervening blocks have been removed by the hand. Depth determination depends on the assumption that all objects rest on a known planar surface.

The edge tracing program in use does not reliably detect subtle differences in brightness between adjacent surfaces of the same or similar objects. The block stacking tasks that have been undertaken to date do not require this information.

A more general world model, in the form of a multiply-linked data structures, is being devised that will accommodate at least multiple objects bounded by combinations of planar surfaces in arbitrary positions. More powerful edge tracing procedures are being tested, and we plan to employ some of the contiguity recognition techniques of Guzman[11] together with three-dimensional plausibility tests of postulated objects.

In related work, the problems of combining information from several views and viewpoints into a single model is being attacked. We expect these combined efforts to produce a much more complete description of the work environment.

IV. SPEECH ANALYSIS AND DESCRIPTION

If we plot the changes in air pressure produced by a speech utterance as a function of time we will see a waveform such as the one given in Figure 4. This signal, as reflected by the changes in voltage generated by the microphone, is digitized in our system to 9 bit accuracy every 50 μs, resulting in a data rate of about 180,000 bits per second of

speech. In normal speech, every second of speech contains about 5 to 10 different sounds which usually require less than 50 bits to represent in the written form. The problem of speech description, then, is the development of a set of procedures which will reduce the 180,000 bits of information to about 50 bits. Of course, human speech carries other information such as speaker identity, emotional state, his location, age, sex, health and other such features. But what is of primary interest to us here is the message uttered by the speaker.

First attempts at speech recognition by a computer were restricted to the recognition of simple sounds like vowels and digits, just as preliminary attempts at picture recognition were restricted to the recognition of characters. The approaches developed for the recognition of simple sounds, such as the use of a metric in a multidimensional space partitioned by hyperplanes, are not easily extendable for the analysis of a complex sequence of sounds which may be part of a spoken message. The structure of the message and the interrelationships among the sounds of the message are the important factors.

Speech is, perhaps, the most extensively investigated of all the human perceptual and motor processes. And yet a large body of this research is not directly relevant for machine recognition of speech. Even the relevant literature on the acoustic characteristics of speech is more qualitative than quantitative and is meant for use by men rather than by machines. Stevens[12] has recently summarized much of the known data on acoustic properties of speech sounds in a form amendable for machine processing. Recent attempts at computer speech recognition by Bobrow and Klatt[13] and Reddy[5] provide models which can be used for the recognition of phrases, sentences, and connected speech. The latter forms the basis of present work. The model currently being used consists of four stages: segmentation, sound description, phrase boundary determination, and phrase recognition.

Segmentation

If you consider the sound in Figure 4 you will see that it is not clear where one word ends and another begins or where a particular sound within a word ends and the next begins. This is because the shape of

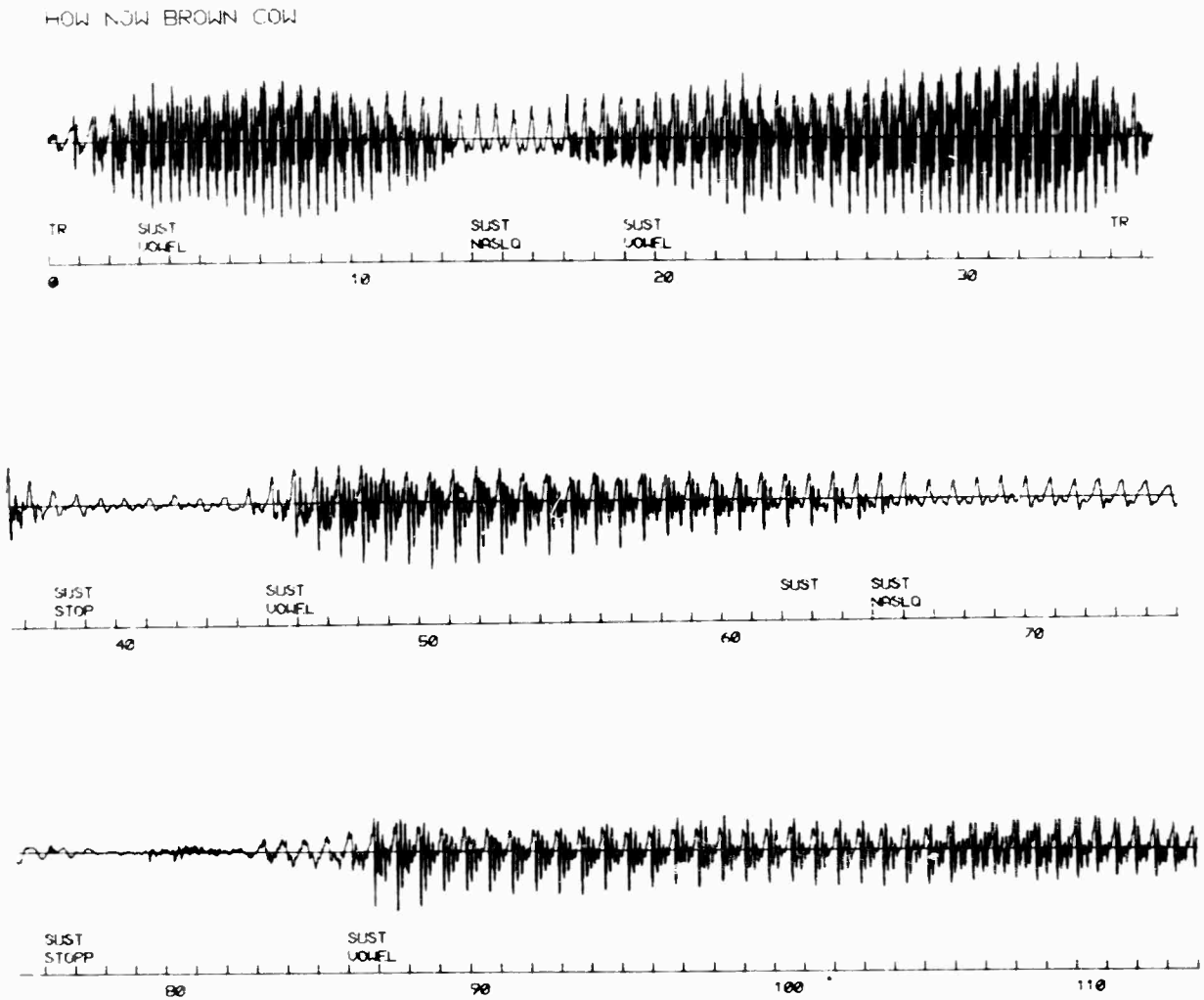

Figure 4. Analysis of the Sound 'How Now Brown Cow'.

the vocal tract is continuously changing and there is no clear cut point in time where we stop saying one sound and start another. To be able to associate discrete symbols with the continuous speech wave, a machine must be able to segment a connected speech utterance into discrete parts.

To be able to segment speech we need to answer questions such as "what is a sound?" and "How do you distinguish one sound from another?" One can define a sound on purely acoustical basis: a sound is a part of the speech wave in which the acoustic characteristics are similar (a sustained segment) or one in which the characteristics vary with time (a transitional segment). To distinguish one sound from another according to the above definition we need a measure of similarity or closeness between two adjacent units of speech.

Conventional metrics such as the Euclidean distance fail to be satisfactory. To be usable the closeness function should be based on the following heuristics:

- Since some parameters are more variable than others the closeness function should provide for appropriate weighting of parameters.
- Although most of the parameters may be similar a drastic change in one parameter should result in a 'not-similar' indication.
- If the difference between two corresponding parameters is less than a minimum, then the two parameters should be considered as identical.
- The greater the parameter value, the greater should be the difference we are willing to accept, suggesting the use of a relative error function such as dy/y.
- When the parameters are close to zero the relative error function dy/y can take abnormally large values, suggesting the use of a function such as $k \cdot dy/\sqrt{y}$.

A closeness function which satisfies the above heuristics and a detailed description of segmentation are given by Reddy and Vicens[6]. The segmentation process can be summarized as follows. A preprocessing procedure divides the speech wave into 10 ms minimal segments and calculates estimators of four characteristic parameters: the amplitude and zero crossings of the dominant frequency under 1000 cps and the

amplitude and zero crossings of the dominant frequency under 1000 cps. An alternative for this preprocessing task is to use special hardware[14]; it divides the speech wave into 10 ms minimal segments and accumulates six parameters: amplitude and zero crossings of the signal in three frequency bands: 150-900 Hz, 900-2200 Hz, 2200-5000 Hz.

A primary segmentation procedure calculates closeness values and groups together adjacent minimal segments that may be regarded as similar. A secondary segmentation procedure divides these primary segments into smaller segments if the within-the-segment variation of parameters is too high. The closeness values are recomputed between the secondary segments, giving greater weight to the frequency components than the amplitude components. If two secondary segments are sufficiently close and are not local maxima or minima, they are combined to form larger segments.

Sound Description

The purpose of generating a sound description is to abstract, from the wide range of possible values of parameters, a label which would adequately describe the nature of a sound segment. The higher the level at which a sound is described, the easier it is for the pattern matching and recognition routines to determine what was said. At one extreme the description might consist of the average parameters of the segments and the other a single label for the whole utterance. In between, descriptions can be attempted at the level of phoneme groups, phonemes, diphones, or syllables. The nature of our segmentation is such that it is more appropriate for us to attempt description in terms of phonemes or phoneme groups.

Phoneticians have classified the sounds we produce according to the shape of our vocal tract. There are about 40 such different sounds (phonemes) in English. One natural description of a speech utterance is in terms of its phonemic transcription. For example the word picks could be described as consisting of four sounds, P, I, K, and S in that sequence. However, various allophones of the same phoneme exhibit widely varying acoustic characteristics depending on the context in which they occur. This often results in a substantial overlap of characteristics

among similar phonemes. Thus it is not uncommon for the word bits to have similar acoustic parameters to picks. This possibility of error preludes the use of simple pattern matching routines. At present, we generate descriptions in terms of phoneme groups: vowel, nasal, fricative, stop, burst and so on. We will see later how such a description is useful in reducing the search space.

The procedure used for the classification of segments into phoneme groups is an extension of the one given by Reddy[5]. If a segment is noiselike, then it is labeled FRICS. Otherwise if the segment-amplitude is a local maximum, then the segment is labeled VOWEL. Otherwise the segment is labeled STOP, NASAL, CONSONANT depending on segment parameters. For example, the description generated for the word picks might be as follows: "The sound consists of a stop, followed by a transition, followed by a vowel, followed by a stop, followed by a fricative each with the following parameters ...".

Word and Phrase Boundary Determination

Like many other aspects of English, the problem of determination of word boundaries in connected speech is ambiguous. For example the sound description /AISCREEM/ could have resulted from the words 'I scream' or 'ice cream'. The problem of word or phrase boundary determination can be completely by-passed if the problem at hand requires only the recognition of a limited set of words, phrases or sentences. Then the sound description of the whole utterance can be stored in the lexicon for future matching with a similar description. However, as the lexicon gets larger and larger it becomes necessary to consider breaking up a connected speech utterance into words and phrases which can then be recognized by a phrase recognition program.

One obvious solution to this problem is to require the speaker to pause for a few milliseconds between words or phrases. But this gets to be annoying after a while. At present we are considering connected speech utterances of the form

<command> ::= <function name> <argument list>

<argument list> ::= <argument> | <argument list> <preposition> <argument>

<function name> ::= PICK UP | STACK | ASSIGN | ADD | SUBTRACT | ...
<argument> ::= THE LARGE BLOCK | FAR LEFT | ALPHA | BRAVO | ...
<preposition> ::= AT | OF | TO | FROM | ...

By carefully choosing the function names, the possible arguments, and the associated prepositions it is possible to determine the word or phrase boundaries. Certain keywords that we can call phonemic context free words play an important part in this determination. A good example of such a word is BLOCK, starting with a silence (B) and ending with a silence (K), the only vowel is never altered by the adjacent phonemes. As a result such heuristics as 'scan until you find BLOCK' may be done with a low percentage of error. Use of restricted special purpose command languages for communication to the computer such as the one above is not unreasonable in view of the fact that we have had to make a similar compromise for programming languages. How interesting the spoken languages can get will depend on how reliably and precisely we can generate sound descriptions.

Word and Phrase Recognition

Given a sound description of a word or phrase, we need a description matching algorithm to determine what was said. If we can guarantee that the description is an error-free phonemic transcription then all that is needed is a simple classification net which grows as new sounds are uttered. Since such a guarantee will not be forthcoming in the near future, if ever, we need a recognition procedure which will cater to the possible errors in the sound description. Two utterances of the same phrase by a single speaker can exhibit different descriptions even under the same environmental conditions. Due to minor changes in the emotional state he may say it faster, slower, or with slightly different stress and intonation resulting in a loss of segment, insertion of a segment, or assignment of a wrong label, e.g., NASAL instead of STOP. This possibility of error in the description poses the well known lack of synchronization problem, i.e., if two descriptions of the same phrase differ by one or more segments, the problem of determining which one it is that is missing.

Given two descriptions which are to be compared, a mapping procedure determines the possible correspondences between segmental descriptions. It uses the heuristic that VOWEL and FRICS segments can be reliably detected and first maps VOWEL for VOWEL and FRICS for FRICS. The remaining unmapped segments are then mapped on the basis of similarity of parameters.

Given the correspondence between segment descriptions, an evaluation procedure compares the parameters of the mapped segments to determine if they could possibly be two different utterances of the same phrase. Similarity of the parameters is given based on the heuristics given for the closeness function in the section on segmentation. Of course, any unmapped segments have a detrimental effect on the similarity measure. If the similarity value is over a certain threshold then the two descriptions are considered the same.

If no candidate was found during the first try, the program, assuming that it knows what was said, supposes that the FRICS were not well determined and were classified as a high frequency stop (burst) or vice versa. If this is the case, a second search is attempted mapping only VOWEL for VOWEL. All the remaining unmapped segments are then mapped on the basis of similarity of parameters as in the first try (but with FRICS included).

If after this second try no satisfactory candidate was found, two different actions may take place: in learning mode, the new description is entered into the lexicon along with the print name; in recognition mode, it is rejected.

Unless the candidates for pattern matching with an incoming description are chosen carefully from the lexicon, it could take a long time to determine what was said. To minimize the search space, the lexicon is ordered on the basis of the number of vowel segments and the number of FRICS segments, furthermore the vowels are classified in nine subclasses according to the values of their zero crossing parameters. The direct matching of these subclasses easily eliminates some entries in the initial list of the possible candidates. Each

entry in the lexicon consists of a packed version of the description and parameters generated by the sound description procedure. A detailed description and evaluation of the phrase recognition procedure is given by Reddy and Vicens[15].

Remarks

The preceding subsections attempt to give an overview of the state of accomplishment in speech recognition at our project. Segmentation and phrase recognition procedures give correct results about 95 percent of the time. This can be improved slightly by using more sophisticated preprocessing routines. Work has barely begun on the determination of classes of words and/or phrases for which boundaries can be determined unambiguously in connected speech. A great deal of work remains to be done in the generation of reliable sound description.

If a reliable description can be obtained in the form of a phonemic transcription (or some such unit) we can reduce the search space of the word and phrase recognition routines considerably, and we will be able to unambiguously determine word boundaries for a larger class of words. Only then can we hope to recognize words from a lexicon of, say, 20,000 words in close to real time. We have already mentioned that the main difficulty in obtaining a reliable phonemic transcription is the wide variability of acoustic characteristics of a phoneme depending on context. Theoretically every phoneme can occur in 1600 or so different contexts. Many of them do not occur in natural speech and the remaining can perhaps be grouped together into 10-20 contextual categories for each phoneme. The hugh task that remains to be done is the investigation and methodical cataloguing of the modifications to the features of a phoneme, and the development of rules for transformations on phonemic features based on context. It will perhaps be many years before such a study is complete but a great deal can be done in computer speech recognition even with incomplete results using the model proposed here.

V. CONTROL OF A MECHANICAL ARM

In order to carry out complex manipulation tasks, it is necessary to do planning for and control of the arm at several levels. At the

top level there is a goal-seeking process which integrates the activities of the various sensory, perceptual, model-building, and manipulation processes. Next there must be planning of subtasks. For example, if we are given a description of an object to be assembled and a description of available components, we must plan which components will go in which locations and the order in which they are to be placed.

Each subtask generates a sequence of motions (e.g., move hand H to point P and open fingers). At this point, the model of the environment should be checked for space occupancy conflicts (i.e., the arm shouldn't bump into things accidentally). In case of conflicts, we must replan the arm motions and, possibly, the order in which components are put in place.

Given that the hand is to reach a certain position, we must calculate how each of the arm joints is to be positioned. For arms with certain geometric properties, this can be a very quick and reliable calculation. For others, it may involve a slow and uncertain iterative process.

Finally, there is a process that servos the arm from place to place, possibly with constraints on the velocity or force to be employed.

Our existing system exhibits each of these levels of planning and control in some form, but without much generality. In most cases, an ad hoc sequence of subroutine calls takes the place of a flexible planning function. As one consequence, the arm readily runs into objects in its vicinity. The calculation of joint positions required to reach a given point is relatively straightforward for the arms we have and has been described by Pingle, Singer and Wichman[10].

An obstacle avoidance technique has been devised by Pieper[16]. It attempts to make the point of closest approach greater than a specified value between all parts of the arm, modelled as a series of cylinders, and an environment containing planes, spheres, and cylinders.

There is much to be done in the area of planning assembly tasks. Many of the things that we do instinctively, such as building things from the bottom up or from the inside out, need to be formalized and translated into programs.

VI. AN EXAMPLE

As an illustration of existing capabilities, we describe a system that obeys the experimenter's voice commands to find blocks visually and stack them as ordered. The grammer chosen for this example is as follows.

Syntax

< command > ::= < command1 > | < command2 >

< command1 > ::= < order1 > | < order1 > EMPTY

< order1 > ::= RESCAN | STOP

< command2 > ::= PICK UP < argument list>

< argument list> ::= < size indicator > EMPTY < position indicator >

< size indicator > ::= EMPTY | < size > BLOCK

< size > ::= SMALL | MEDIUM | LARGE | EMPTY

< position indicator > ::= EMPTY | < position1 > | < position2 >

< position1 > ::= < position > SIDE

< position2 > ::= < position′ > < position″> < position word >

| < position″ > < position′ > < position word >

< position word > ::= ANGLE | CORNER

< position > ::= < position′ > | < position″ >

< position′> ::= EMPTY | LEFT | RIGHT

< position″ > ::= EMPTY | UPPER | LOWER

Semantics

The meaning of some of the terminal symbols is obvious, but some others, like RESCAN and EMPTY need explanation.

The command 'rescan' is used to indicate that the scene might be disturbed and that the vision program should generate a new scene description.

The terminal symbol EMPTY means no speech utterance at all or sounds not recognized by the word recognizer. If any of the non-terminal symbols is finally reduced to EMPTY the middle value is assumed. For example if < size indicator > = EMPTY , a block of medium size will be assumed.

Sentences like 'pick up the small block standing on the upper right corner', 'rescan the scane', 'pick up any block' are correct according to the grammar.

After the preliminaries such as training the phrase recognition system and calibration of the arm and eye coordinate systems, the picture recognition program looks at the image and generates a scene description of all the cubes present in the field of view. The description for each block consists of the location, size and orientation of the block.

Given a command, the speech analysis program segments the speech and generates a sound description. This description is then used by the scanner-recognizer which decodes it and passes the result of its analysis to the main program.

The scanner-recognizer requires a good word recognizer utility program. The recognition is done by scanning the speech utterance description forward and backward using feedback from the grammar. The decoding of a sentence like 'pick up the small block standing on the right side' will be done as follows:

Recognize PICK UP

Then scan until BLOCK

If BLOCK backtrace to find size attribute

Backtrace from the end to find SIDE

Backtrace to find position attributes

At any step, feedback is used so that the only candidates considered are those that are syntactically correct. For example, when the program is trying to reduce the non terminal symbol < size > the only available candidates for the matching process are the descriptions of LARGE, SMALL and MEDIUM.

Based on the command, the arm is directed to pick up or stack a block. If it is to pick up, the location and orientation of the block are given. If it is to stack, the location of the stack is given. The movie, to be shown, illustrates the response of the arm to various commands, and presents the details of various analysis and description generation processes displayed on a CRT.

VII. CONCLUSIONS

Many of the problems discussed at the end of the preceeding sections can and will be solved in the next few years. However, it will probably be a long time before a computer can equal the perception and dexterity of a human being. This will require not only advances in the areas of computer architecture and the quality of the external devices, but also a better understanding of perceptual and motor processes.

Even the limited progress achieved so far can result in computer-hand-eye-ear systems that are better suited for some purposes than human beings. For example, they may see things and hear sounds that a person cannot, and they may be faster, stronger, more economical, or more expendable.

The fact that a computer may not be able to see all the things we can see or carry on fluent conversation should not be a cause for extra concern. Consider the case of programming languages. Although we have not been able to communicate with computers in our natural language, we have managed to achieve a great deal using contrived and ad hoc languages like Fortran. There is no reason to suspect that the same will not be the case with visual and voice input to the computers or with computer control of manipulators.

We forsee several practical applications that can profitably use the techniques described in this paper. One that is most often mentioned is the possible bandwidth reduction in picture and speech transmission systems. We believe that computer controlled carts which can navigate themselves, and automated factories, where computer controlled manipulators with visual feedback can handle many situations which cannot be presently handled by fixed sequence manipulators, are also within the range of the present state of the art.

ACKNOWLEDGEMENTS

The success of any system of this magnitude depends on the team effort of many people. It is a pleasure to acknowledge the contributions of Karl Pingle, who did most of the eye programming, and Jeff Singer, who developed most of the arm and hand programs. Excellent hardware and software system support were provided by Stephen Russell, Gerald Gleason, David Poole, John Sauter and William Weiher.

PEFERENCES

1. Ernst, H.A., 'MH-1, a Computer Operated Mechanical Hand', Doctoral Thesis, M.I.T., Cambridge, Massachusetts (1961).
2. Roberts, L.G., 'Machine Perception of Three-Dimensional Solids', Optical and Electro-Optical Processing of Information, MIT Press, Cambridge, Massachusetts (1963).
3. Glaser, D., McCarthy, J. and Minsky, M., 'The Automated Biological Laboratory', Report of the subgroup on ABL, summer study in exobiology sponsored by the Space Science Board of the National Academy of Sciences (1964).
4. Earnest, L.D., 'On Choosing an Eye for a Computer', AI memo No. 51, Computer Science Department, Stanford University (1967).
5. Reddy, D.R., 'Computer Recognition of Connected Speech', J. Acoust. Soc. Am., 42,329-347 (1967).
6. Reddy, D.R. and Vicens, P.J., 'A Procedure for Segmentation of Connected Speech', to be published in J. Audio Engr. Soc, 16,4 (1968).
7. Narasimhan, R., 'Syntax-directed Interpretation of Classes of Pictures', CACM 9,3,166-173 (1966).
8. Miller, W. and Shaw, A., 'A Picture Calculus', GSG Memo 40, Computation Group, Stanford Linear Accelerator Center, Stanford, California (1967).
9. Shaw, A., 'The Formal Description and Parsing of Pictures', Ph.D. Thesis, CS Report No. 94, Computer Science Department, Stanford University, Stanford (1968).
10. Pingle, K.K., Singer, J.A. and Wichman, W.M., 'Computer Control of a Mechanical Arm Through Visual Input', To be published in the proceedings of the IFIP '68 (1968).
11. Guzman, A., 'Some Aspects of Pattern Recognition by Computer', MAC-TR-37, Project MAC, MIT, Cambridge, Massachusetts (1967).
12. Stevens, K.N., Unpublished paper (1968).
13. Bobrow, D.G. and Klatt, D., 'A Limited Speech Recognition System', To be published in the proceedings of FJCC '68 (1968).

14. Vicens, P.J., 'A Speech Preprocessing Device', Stanford Artificial Intelligence Memo, to be published (1968).

15. Reddy, D.R. and Vicens, P.J., 'Spoken Message Recognition by A Computer', to be published (1968).

16. Pieper, D., 'Kinematics of Manipulators under Computer Control', Ph.D. Thesis, Stanford University, to be published (1968).

6. Board Games

Research into game-playing programs is represented by the following introduction to Barbara Huberman's Ph.D. thesis, "A Program to Play Chess End Games". The full text is given in Memo AI-65.

Another line of research in this area is represented by Dr. Arthur Samuel's "Studies in Machine Learning Using the Game of Checkers", reported in Memo AI-52 and subsequently published in the IBM Journal (November 1967).

INTRODUCTION

This research is concerned with the process of translating book descriptions of problem solving methods into program heuristics. Many books have been written for the purpose of teaching how to perform some task. The task under discussion may be almost any kind of activity, including intellectual activities such as proving theorems in geometry or solving differential equations. People are able to learn from these books although the difficulty in learning varies from task to task. Therefore we can consider the information in the books as sufficient for people. It would be convenient if the book information could be used by computer programs. We are interested in whether the information is sufficient for computers, and if not, then we want to know what kind of additional information is needed.

The fact that book information is sufficient for people does not mean that it can be used directly. If the book describes an algorithm, then sometimes only memorization is required of the reader; for example, the method of finding truth values of sentences in propositional calculus by means of truth tables can be learned by memorization. Many tasks, however, require substantial learning before the student can understand the book. The task of playing chess end games by computer provides a simple but not trivial area for this research. By chess end games we mean those games where the number of pieces on the board is small, but the number of moves to checkmate large: for example, Two Bishops and King against King, or the various Pawn endings. Chess books give rules for these end games which are not algorithms but are supposed to be simple and complete enough that beginners at chess can learn to play the end games fairly easily. A certain amount of intelligence is required of the student, but still we expect to need only a minimal amount of additional information. In this study the programmer will do the translation. Since this translation from the chess books to the program is not direct, as it would be in the case of truth tables, we expect to learn something from the translation process.

Methods and Models

Computer researchers are well aware by now of the fact that any task requiring intelligence can be profitably approached by distinguishing between models and methods. The model, which is a representation of the structure of the problem [Minsky, 1961], determines the overall logic of the program. The methods are the heuristics which the program uses within this structure. For example, in the Logic Theory Machine [Newell, Shaw, and Simon, 1957], the model is a backwards tree and is represented by that part of the program called the "Executive Routine". Within this framework substitution, detachment and chaining methods are used; these are encodings of the way people apply the rules of inference in propositional calculus.

Generally books are concerned only with teaching the methods which should be used to solve problems in the task area. The methods must be applied within a structure which is assumed in the book but not generally defined explicitly. It is necessary to build a model of this structure in the computer before information about methods can be taken from the book.

We expect that different models are required for different tasks. Very often the model is a backwards tree; the General Problem Solver [Newell and Simon, 1961] is based upon this fact. However there are problems which would require a different model: for example, bidding in bridge. The closer the model used in the program is to the way that the author of the book thinks about the problem, the easier it will be to translate the methods of the book into heuristics for the program. Chess end games could be handled by the General Problem Solver; however in this research a model is used which is much closer to the abstract model assumed in the chess books. In this way we hope to eliminate making changes in the methods to account for a difference between the program's model and the abstract model assumed in the book. This means that any difficulty experience in translating the book methods into program heuristics can only be due to inadequacy in the method description.

Model and Methods for Chess End Games

The model used for chess end games is a forcing tree. The program is supplied with two functions better and worse (containing the methods)

which compare positions. From a given starting position p , in which the program has the move, it uses tree search to find positions q which are better than p . It will search until such a position q is found for every sequence of moves by the opposition. An example of such a tree is given in Figure 1.1. The program will then make the moves dictated by the tree until it reaches a q at the end of a branch in the tree; then it recalculates the tree to force positions better than q . This process continues until checkmate is reached. worse is used by the program to cut off branches of the tree which lead to disaster (stalemate, etc), and also to prune the tree. This model is described in detail in Chapter 2.

The forcing tree model will be used for all the different end games. However each end game is played by different methods which will result in different definitions of better and worse. This enables us to examine the problems of translation from methods to program heuristics several times and for games of varying degrees of difficulty.

better and worse are built up out of pattern recognition functions of positions which can be defined in a natural manner from information given in the chess books. The methods, or rules, of play are defined in two ways in the books. First of all, written statements are made. For example, in the description of the Rook and King against King game in Capablanca [1935] we find: "The principle is to drive the opposing King to the last line on any side of the board" and then the student should "Keep his King as much as possible on the same rank, or...file, as the opposing King". The play of other games (and in other books) is described by similar rules. It is not difficult to convert a principle into a pattern recognition function of positions because the pattern is inherent in the principle. For example, to express the first principle quoted above we define

$f(x) \equiv$ the opposition king is confined to an edge of the board in x , for x a position. Then we might decide a position q was better than position p if

$$f(q) \wedge \neg f(p)$$

because the principle is satisfied by making the moves leading from p to q .

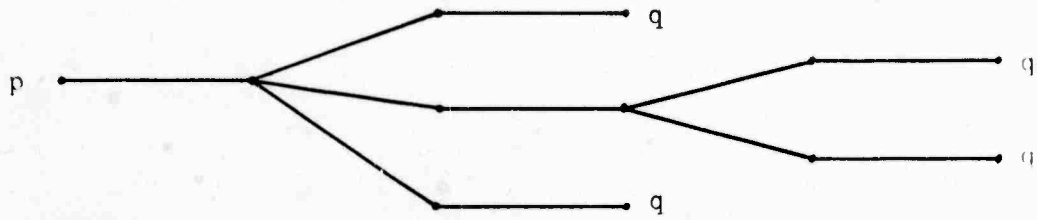

Figure 1.1. Example of a Forcing tree. The program has the move in p; it must make a move leading to a position q judged <u>better</u> than p for every sequence of moves by the opposition. Each iteration of the program will produce a tree like this; several iterations will be required to reach checkmate.

The chess books supplement the principles with examples of program play. The principles generally cover the gross features of the game and form a framework for viewing the play of the game. The majority of moves are only partly derived from the principles; they are more directly derived from the examples of program play. The examples contain more or less complete information about methods of play; the difficulty comes in deciding what pattern features of the positions are important. Obviously, induction is required to make this decision. Each example is considered representative of a large class of positions and a general rule must be defined for that class. If the example is accompanied by principles, this simplifies the induction by providing clues to important features (see Figure 3.1). The induction leads automatically to the kind of pattern recognition functions used in better and worse.

Goals of the Research

The primary goal of the research is to study the translation process. We begin by stating two criteria which will help us achieve this goal. First we would like to see if our model is a good one for chess end games. Our first hypothesis is: the model used in the program is a good representation of the abstract model assumed by chess books. We can support this hypothesis by successfully running the program on different end games. Furthermore, conditions can be given on better and worse which permit us to prove informally that the program works correctly. The proof depends heavily on the model and could not be given for a different model (for example the General Problem Solver model).

Our second hypothesis is: the information in the chess books is sufficient for the definitions of better and worse. The chess books information will suffice for worse for all disastrous positions are described. For better much more information is needed; the books must give rules for recognizing progress frequently enough that the tree search between positions is reasonable. For example it is not enough to have rules recognizing only checkmate positions.

Finally we turn our attention to the primary goal of studying the translation process. We assume that the two criteria are satisfied.

First we consider how closely the definitions of better and worse correspond to the chess book methods, measuring the correspondence by comparing program play with the book examples. Also we consider the difficulty encountered in defining better and worse.

Outline of the Thesis

In Chapter 2, the overall organization will be described. A detailed definition of the uses of better, worse, and tree search will be given; this constitutes the model which we use for chess end games.

In Chapter 3 the form of the contents of functions better and worse will be discussed. These functions are different for each end game, since different methods are used for each game. However, the form given for better and worse is used in all end games. Some rules are given for better and worse which will enable us to prove that the program is correct in the sense of being able to achieve checkmate from a given starting position.

Chapters 4, 5, and 6 each describe the definitions of better and worse for a different end game. Rook and King against King is discussed in Chapter 4, two Bishops and King against King in Chapter 5, and Bishop, Knight and King against King in Chapter 6. These games are presented in order of difficulty. The rook end game is quite a simple one; two Bishops is a game of moderate difficulty, while the Bishop-Knight end game is very difficult. The process of translating from the book information into pattern recognition functions will be described, and reasons will be given for the programming decision. Examples of program play will be included for each game.

Chapter 7 contains an informal proof of program correctness. This proof is given after the various end games are described because it depends on the heuristics used for each game.

Chapter 8 will contain an evaluation of the better, worse format in terms of the two primary goals. Subjects covered will include program efficiency, a description of a way to have the program do some of the inductive learning, and extensions to other task areas.

In the following chapters, ordinary chess notations will be used [Capablanca, 1935]. The program is written in LISP [McCarthy, Abrams,

Edwards, Hart and Levin, 1965], and the reader is expected to have some knowledge of this language. Function definitions are given using notation and basic functions which are defined in Appendix A. They are built up of the connectives $\equiv$ (equivalence), $\supset$ (implication), $\wedge$ (conjuction), $\vee$ (disjunction), and $\neg$ (negation). These are used in the same way LISP (not ALGOL) uses them; i.e., if in $p \wedge q$, p is evaluated and found to be false, then q is not evaluated.

REFERENCES

In addition to the chess books referred to in the body of the thesis, several other books are mentioned here which were also found useful.

Baylor, G. W., and Simon, H. A., 1966, A Chess Mating Combinations Program, Proceedings of the AFIPS Spring Joint Computer Conference, Spartan Books, Washington D. C., 28: 431-447.

Capablanca, J. R., 1935, A Primer of Chess, Harcourt, Brace, New York.

Fine, R., 1944, Chess the Easy Way, David McKay, New York.

Foster, A. W., and Kemp, R. E., 1943, Chess: An Easy Game, David McKay, New York.

Greenblatt, R. D., and Crocker, S. D., 1967, The Greenblatt Chess Program, Proceedings of the AFIPS Fall Joint Computer Conference, Thompson, Washington D. C., 31: 801-810.

Horowitz, I. A., 1957, How to Win in the Chess Endings, David McKay, New York.

Mason, James, 1905, The Art of Chess, Howard Cox, London.

McCarthy, J., Abrams, P. W., Edwards, D. J., Hart, T. P., Levin, M. I., 1965, LISP 1.5 Programmers Manual, The M.I.T. Press, Cambridge, Massachusetts.

Minsky, M.L., 1961, Steps toward Artificial Intelligence, Proceedings of the I.R.E., 8-30; reprinted in Computers and Thought, Feigenbaum, E., and Feldman, J. (Ed), McGraw-Hill, New York, 406-450.

Newell, A., Shaw, J. C., and Simon, H. A., 1957, Empirical Explorations with the Logic Theory Machine, Proceedings of the 1957 Western Joint Computer Conference, I.R.E., New York, 15: 218-239.

Newell, A., and Simon, H. A., 1961, GPS - A Program That Simulates Human Though, Lernende Automaten, H. Billing, Munich, 109-124.

Slagle, J. R., 1963, A Heuristic Program that Solves Simple Symbolic Integration Problems in Freshman Calculus, Computers and Thought, Feigenbaum, E., and Feldman, J. (Ed), McGraw-Hill, New York, 191-203.

7. Other Projects

Our research in the mathematical theory of computation has been concerned mainly with the problem of proving properties of algorithms, namely validity (i.e., termination and correctness) and equivalence.

Kaplan has investigated the problem of proving the equivalence of program schemata, mainly by means of automata theory (regular expressions). This work has been reported in Memos AI-59, AI-60, and AI-63, the latter being his Ph.D. thesis.

Professor Manna has investigated the problem of proving the equivalence and validity of programs [Memo AI-64] and of recursively defined functions [AI-68, with A. Pnueli] by means of mathematical logic. This research is now being carried further by McCarthy and Manna.

Research on grammatical inference [Memo AI-55] has been pursued by Professor Feldman and his students. A theoretical paper will be published shortly establishing a number of results on the decidability of the grammatical inference question under various conditions.

The Artificial Intelligence Project continues to interact with certain separately supported projects. Work in this area includes digital image formation from holograms (Reference 9, Appendix A), computer synthesis of polyphonic sound, and computer extraction of human belief systems (References 2, 3, and 21, Appendix A).

BLANK PAGE

APPENDIX A

RECENT PUBLICATIONS OF PROJECT PARTICIPANTS

1) B. Buchanan and G. Sutherland, "Heuristic Dendral: A Program for Generating Explanatory Hypotheses in Organic Chemistry", in D. Michie et. al. (eds.), Machine Intelligence 4 (in preparation).

2) K. M. Colby and H. Enea, "Inductive Inference by Intelligent Machines", Scientia (French), Jan.-Feb. 1968.

3) K. M. Colby and H. Enea, "Machine Utilization of the Natural Language Word 'Good'", Math. Bio., Feb. 1968.

4) E. A. Feigenbaum, J., Lederberg and B. Buchanan, "Heuristic Dendral", Proc. International Conference on System Sciences, University of Hawaii and IEEE (University of Hawaii Press, 1968).

5) E. A. Feigenbaum, "Artificial Intelligence: Themes in the Second Decade", Proc. IFIP Congress 1968 (in press).

6) J. Feldman (with D. Gries), "Translator Writing Systems", Comm. ACM, February 1968.

7) J. Feldman (with P. Rovner), "The Leap Language and Data Structure", Proc. IFIP Congress 1968 (in press).

8) J. Feldman, "Plans for the Stanford Hand-Eye Project", Proc. FJCC, 1968 (in press).

9) J. Goodman, "Digital Image Formation from Electronically Detected Holograms" in Proc. SPIE Seminar on Digital Imaging Techniques, Soc. Photo-Optical Instrumentation Engineering, Redondo Beach, California, 1967.

10) A. C. Hearn, "REDUCE, A User-Oriented Interactive System for Algebraic Simplification, Proc. ACM Symposium on Interactive Systems for Experimental Applied Mathematics, August 1967 (in press).

11) A. C. Hearn, "The Problem of Substitution", Proc. IBM Summer Institute on Symbolic Math. by Computer, July 1968 (in press).

12) D. Kaplan, "Some Completeness Results in the Mathematical Theory of Computation", ACM Journal, January 1968.

13) J. Lederberg and E. Feigenbaum, "Mechanization of Inductive Inference in Organic Chemistry", in: B. Kleinmuntz (ed.), Formal Representation of Human Judgment (New York, John Wiley, 1968).

14) J. McCarthy, L. Earnest, D. R. Reddy, and P. Vicens, "A Computer with Hands, Eyes, and Ears", Proc. FJCC 1968 (in press).

15) K. Pingle, J. Singer, and W. Wichman, "Computer Control of a Mechanical Arm through Visual Input", Proc. IFIP Congress 1968 (in press).

16) D. R. Reddy and Ann Robinson, "Phoneme-to-Grapheme Translation of English", IEEE Trans. Audio and Electroacoustics, June 1968.

17) D. R. Reddy, "Computer Transcription of Phonemic Symbols", J. Acoust. Soc. Am., August 1968.

18) D. R. Reddy and P. J. Vicens, "Procedure for Segmentation of Connected Speech", J. Audio Eng. Soc., October 1968.

19) D. R. Reddy, "Consonental Clustering and Connected Speech Recognition", Proc. Sixth International Congress on Acoustics, Tokyo, 1968 (in press).

20) A. Samuel, "Studies in Machine Learning Using the Game of Checkers", IBM Journal, Nov. 1967.

21) L. Tesler, H. Enea, and K. Colby, "A Directed Graph Representation for Computer Simulation of Belief Systems", Math. Bio. 2, 1968.

APPENDIX B

ABSTRACTS OF

ARTIFICIAL INTELLIGENCE MEMOS

1963

1. J. McCarthy, Predicate Calculus with "undefined" as a Truth-Value, March.
   The use of predicate calculus in the mathematical theory of computation and the problems involved in interpreting their values.

2. J. McCarthy, Situations, Actions, and Causal Laws, July.
   A formal theory is given concerning situations, causality and the possibility and effects of actions is given. The theory is intended to be used by the Advice Taker, a computer program that is to decide what to do by reasoning. Some simple examples are given of descriptions of situations and deductions that certain goals can be achieved.

3. F. Safier, "The Mikado" as an Advice Taker Problem, July.
   The situation of the Second Act of "The Mikado" is analyzed from the point of view of Advice Taker formalism. This indicates defects still present in language.

4. H. Enea, Clock Function for LISP 1.5, August.
   This paper describes a clock function for LISP 1.5.

5. H. Enea and D. Wooldridge, Algebraic Simplification, August.
   Herein described are proposed and effected changes and additions to Steve Russell's Mark IV Simplify.

6. D. Wooldridge, Non-Printing Compiler, August.
   A short program which redefines parts of the LISP 1.5 compiler and suppresses compiler printout (at user's option) is described.

7. J. McCarthy, Programs with Common Sense, September.
   Interesting work is being done in programming computers to solve problems which require a high degree of intelligence in humans. However, certain elementary verbal reasoning processes so simple that they can be carried out by any non-feeble-minded human have yet to be simulated by machine programs.
   This paper will discuss programs to manipulate in a suitable formal language (most likely a part of the predicate calculus) common instrumental statements. The basic program will draw immediate conclusions from a list of premises. These conclusions will be either declarative or imperative sentences. When an imperative sentence is deduced the program takes a corresponding action. These actions may include printing sentences,

1963 (cont.)

moving sentences on lists, and reinitiating the basic deduction process on these lists.
Facilities will be provided for communication with humans in the system via manual intervention and display devices connected to the computer.

8. J. McCarthy, Storage Conventions in LISP 2., September.
Storage conventions and a basic set of functions for LISP 2 are proposed. Since the memo was written, a way of supplementing the features of this system with the unique storage of list structure using a hash rule for computing the address in a separate free storage area for lists has been found.

9. C. M. Williams, Computing Estimates for the Number of Bisections of an N x N Checkerboard for N Even, December.
This memo gives empirical justification for the assumption that the number of bisections of an N x N (N even) checkerboard is approximately given by the binomical coefficient

$$\binom{A}{\frac{A}{2}}$$

where 2A is the length of the average bisecting cut.

10. S. R. Russell, Improvements in LISP Debugging, December.
Experience with writing large LISP programs and helping students learning LISP suggests that spectacular improvements can be made in this area. These improvements are partly an elimination of sloppy coding in LISP 1.5, but mostly an elaboration of DEFINE, the push down list backtrace, and the current tracing facility. Experience suggests that these improvements would reduce the number of computer runs to debug a program a third to a half.

11. D. Wooldridge, Jr., An Algebraic Simplify Program in LISP, December.
A program which performs "obvious" (non controversial) simplifying transformations on algebraic expressions (written in LISP prefix notation) is described. Cancellation of inverses and consolidation of sums and products are the basic accomplishments of the program; however, if the user desired to do so, he may request the program to perform special tasks, such as collect common factors from products in sums or expand products. Polynomials are handled by routines which take advantage of the special form by polynomials; in particular, division (not cancellation) is always done in terms of polynomials. The program (run on the IBM 7090) is slightly faster than a human; however, the computer does not need to check its work by repeating the simplification. Although the program is usable - no bugs are known to exist - it is by no means a finished project. A rewriting of the simplify system is anticipated; this will eliminate much of the existing redundancy and other inefficiency, as well as implement an identity-recognizing scheme.

1964

12. G. Feldman, Documentation of the MacMahon Squares Problem, January.
An exposition of the MacMahon Squares problem together with some "theoretical" results on the nature of its solutions and a short discussion of an ALGOL program which finds all solutions are contained herein.

13. D. Wooldridge, The New LISP System (LISP 1.55), February.
The new LISP system is described. Although differing only slightly it is thought to be improvement on the old system.

14. J. McCarthy, Computer Control of a Machine for Exploring Mars, January.
Landing a 5000 pound package on Mars that would spend a year looking for life and making other measurements has been proposed. We believe that this machine should be a stored program computer with sense and motor organs and that the machine should be mobile. We discuss the following points. 1. Advantages of a computer controlled system. 2. What the computer should be like. 3. What we can feasibly program the machine to do given the present state of work on artificial intelligence. 4. A plan for carrying out research in computer controlled experiments that will make the Mars machine as effective as possible.

15. M. Finkelstein and F. Safier, Axiomatization and Implementation, June.
An example of a typical Advice-Taker axiomatization of a situation is given, and the situation is programmed in LISP as an indication of how the Advice-Taker could be expected to react. The situation chosen is the play of a hand of bridge.

16. J. McCarthy, A Tough Nut for Proof Procedures, July.
It is well known to be impossible to tile with dominoes a checkerboard with two opposite corners deleted. This fact is readily stated in the first order predicate calculus, but the usual proof which involves a parity and counting argument does not readily translate into predicate calculus. We conjecture that this problem will be very difficult for programmed proof procedures.

17. J. McCarthy, Formal Description of the Game of Pang-Ke, July.
The game of Pang-Ke is formulated in a first-order-logic in order to provide grist for the Advice-Taker Mill. The memo does not explain all the terms used.

1964 (cont)

18. J. Hext, An Expression input Routine for LISP, July.
The expression input routine is a LISP function, Mathread [ ] with associated definitions, which reads in expressions such as (A+3 - F(X,Y,Z)). Its result is an equivalent S-expression. The syntax of allowable expressions is given, but (unlike ALGOL's) it does not define the precedence of the operators; nor does the program carry out an explicit syntax analysis. Instead, the program parses the expression according to a set of numerical precedence values, and reports if it finds any symbol out of context.

19. J. Hext, Programming Languages and Translation, August.
A notation is suggested for defining the syntax of a language in abstract form, specifying only its semantic constituents. A simple language is presented in this form and its semantic definition given in terms of these constituents. Methods are then developed for translating this language, first into a LISP format and from there to machine code, and for proving that the translation is correct.

20. R. Reddy, Source Language Optimization of For-Loops, August.
Program execution time can be reduced, by a considerable amount, by optimizing the 'For-loops' of Algol Programs. By judicious use of index-registers and by evaluating all the sub-expressions whose values are not altered within the 'For-loop', such optimization can be achieved.
In this project we develop an algorithm to optimize Algol Programs in List-structure form and generate a new source language program, which contains the "desired contents in the index registers" as a part of the For-clause of the For-statement and additional statements for evaluating the same expressions outside the 'For-loop'. This optimization is performed only for the innermost 'For-loops'.
The program is written entirely in LISP. Arrays may have any number of subscripts. Further array declarations may have variable dimensions. (Dynamic allocation of storage.)
The program does not try to optimize arithmetic expressions. (This has already been extensively investigated.)

21. R. W. Mitchell, LISP 2 Specifications Proposal, August.
Specifications for LISP 2 system are proposed. The source language is basically ALGOL 60 extended to include list processing, input/output and language extension facilities. The system would be implemented with a source language translator and optimizer, the output of which could be processed by either an interpreter or a compiler. The implementation is specified for a single address computer with particular reference to an IBM 7090 where necessary.
Expected efficiency of the system for list processing is

1964 (cont)

significantly greater than the LISP 1.5 interpreter and also somewhat better than the LISP 1.5 compiler. For execution of numeric algorithms the system should be comparable to many current "algebraic" compilers.
Some familiarity with LISP 1.5, ALGOL and the IBM 7090 is assumed.

22. R. Russell, Kalah - The Game and the Program, September.
A description of Kalah and the Kalah program, including subroutine descriptions and operating instructions.

23. R. Russell, Improvements to the Kalah Program, September.
Recent improvements to the Kalah program are listed, and a proposal for speeding up the program by a factor of three is discussed.

24. J. McCarthy, A Formal Description of a Subset of Algol, September.
We describe Microalgol, a trivial subset of Algol, by means of an interpreter. The notions of abstract syntax and of "state of the computation" permit a compact description of both syntax and semantics. We advocate an extension of this technique as a general way of describing programming language.

25. R. Mansfield, A Formal System of Computation, September.
We discuss a tentative axiomatization for a formal system of computation and within this system we prove certain propositions about the convergence of recursive definitions proposed by J. McCarthy.

26. R. Reddy, Experiments on Automatic Speech Recognition by a Digital Computer, October.
Speech sounds have in the past been investigated with the aid of spectrographs, vo-coders and other analog devices. With the availability of digital computers with improved i-o devices such as Cathode Ray tubes and analog to digital converters, it has recently become practicable to employ this powerful tool in the analysis of speech sounds.
Some papers have appeared in the recent literature reporting the use of computers in the determination of the fundamental frequency and for vowel recognition. This paper discusses the details and results of a preliminary investigation conducted at Stanford. It includes various aspects of speech sounds such as waveforms of vowels and consonants; determination of a fundamental of the wave; Fourier (spectral) analysis of the sound waves formant determination, simple vowel recognition algorithm and synthesis of sounds. All were obtained by the use of a digital computer.

1965

27. J. McCarthy, A Proof-Checker for Predicate Calculus, March.
A program that checks proofs in J. A. Robinson's formulation of predicate calculus has been programmed in LISP 1.5. The program is available in CTSS at Project MAC and is also available as a card deck. The program is used for class exercises at Stanford.

28. J. McCarthy, Problems in the Theory of Computation, March.
The purpose of this paper is to identify and discuss a number of theoretical problems whose solutions seem feasible and likely to advance the practical art of computation. The problems that will be discussed include the following:
1. Semantics of programming languages. What do the strings of symbols representing computer programs, statements, declarations, labels, etc., denote? How can the semantics of programming languages be described formally?
2. Data spaces. What are the spaces of data on which computer programs act and how are they built up from simpler spaces?
3. How can time dependent and simultaneous processes be described?
4. Speed of computation. What can be said about how much computation is required to carry out certain processes?
5. Storage of information. How can information be stored so that items identical or similar to a given item can be retrieved?
6. Syntax directed computation. What is the appropriate domain for computations described by productions or other data format recognizers?
7. What are the appropriate formalisms for writing proofs that computer programs are equivalent?
8. In view of Gödel's theorem that tells us that any formal theory of computation must be incomplete, what is a reasonable formal system that will enable us to prove that programs terminate in practical cases?

29. C. M. Williams, Isolation of Important Features of a Multi-toned Picture, January.
A roughly successful attempt is made to reduce a multi-toned picture to a two-toned (line drawing) representation capable of being recognized by a human being.

30. E. Feigenbaum and R. W. Watson, An Initial Problem Statement for a Machine Induction Research Project, April.
A brief description is given of a research project presently getting under way. This project will study induction by machine,

1965 (cont)

using organic chemistry as a task area. Topics for graduate study research related to the problem is listed.

31. J. McCarthy, Plans for the Stanford Artificial Intelligence Project, April.
The following is an excerpt from a proposal to ARPA and gives some of the project plans for the near future.

32. H. Ratchford, The 138 Analog Digital Converter, May.
A discussion of the programming and hardware characteristics of the analog to digital converter on the PDP-1 is given; several sample programs are also presented.

33. B. Huberman, The Advice Taker and GPS, June.
Using the formalism of the Newell-Shaw-Simon General Problem Solver to solve problems expressed in McCarthy's Advice Taker formalism is discussed. Some revisions of the formalism of can and cause described in AI Memo No. 2 are proposed.

34. P. Carah, A Television Camera Interface for the PDP-1, June.
This paper is a discussion of several methods for the connection of a television camera to the PDP-1 computer. Three of these methods are discussed in detail and have in common that only a 36 bit portion of any horizontal scanning line may be read and this information is read directly into the working registers of the computer. The fourth involves a data channel to read information directly into the core memory of the computer, and is mentioned only in passing. The major concepts and some of the details of these methods are due to Marvin Minsky.

35. F. Safier, Simple Simon, June
SIMPLE SIMON is a program which solves the problem of finding an object satisfying a predicate from a list of facts. It operates by backware chaining. The rules of procedure and heuristics are discussed and the structure of the program is outlined.

36. J. Painter, Utilization of a TV Camera on the PDP-1, September.
A description of the programming required to utilize the TV camera connected to the PDP-1 and of the initial collection of programs.

37. K. Korsvold, An On Line Algebraic Simplify Program, November
We describe an on-line program for algebraic simplification. The program is written in LISP 1.5 for the Q-32 computer at System Development Corporation in Santa Monica, California. The program has in its entirety been written and debugged from a teletype station at Stanford University.

1965 (cont)

37. K. Korsvold, Appendix B, to A.I. 37
This appendix contains the program written in m-expressions. The four functions ADDK, TIMESKL, *GSD and *RFD are not included since they are written in LAP.

1966

38. D. Waterman, A Filter for a Machine Induction System, January.
This report contains current ideas about the Machine Induction Research Project, and attempts to more clearly define some of the problems involved. In particular, the on-line data acquisitional problem, the filter, and the inductive inference problem associated with the filter are discussed in detail.

39. K. Pingle, A Program to Find Objects in a Picture, January.
A program is described which traces around objects in a picture, using the picture scanner attached to the PDP-1 computer, and fits curves to the edges.

40. J. McCarthy and J. Painter, Correctness of a Compiler for Arithmetic Expressions, April.
This is a preprint of a paper given at the Symposium on Mathematical Aspects of Computer Science of the American Mathematical Society held April 6 and 7, 1966. It contains a proof of the correctness of a compiler for arithmetic expressions.

41. P. Abrams and D. Rode, A Proposal for a Proof-checker for Certain Axiomatic Systems, May.
A proposed design for a proof-checker to operate on many axiomatic domains is presented. Included are descriptions of the organization and operation of the program to be written for the PDP-6.

42. K. Pingle, A Proposal for a Visual Input Routine, June.
Some comments are made on the characteristics believed desirable in the next eye for the Stanford Artificial Intelligence Project and a proposal is given for a program to input scenes using the eye.

43. R. Reddy, An Approach to Computer Speech Recognition by Direct Analysis of the Speech Wave, September.
A system for obtaining a phonemic transcription from a connected speech sample entered into the computer by a microphone and an analog-to-digital converter is described. A feature-extraction program divides the speech utterance into segments approximately corresponding to phonemes, determines pitch periods of those segments where pitch analysis is appropriate, and computes a list of parameters for each segment. A classification program assigns a phoneme-group label (vowel-like segment, fricative-like segment, etc.) to each segment, determines whether a segment should be classified as a phoneme or whether it represents a phoneme boundary between two phonemes, and then assigns a phoneme label to each segment that is not rejected as being a

1966 (cont)

phoneme boundary. About 30 utterances of one to two seconds duration were analyzed using the above programs on an interconnected IBM 7090 - PDP1 system. Correct identification of many vowel and consonantal phonemes was achieved for a single speaker. The time for analysis of each utterance was about 40 times real time. The results are encouraging and point to a new direction in speech research.

44. J. Painter, Semantic Correctness of a Compiler for an Algol-like Language, Revised March, 1967.
This is a semantic proof of the correctness of a compiler. The abstract syntax and semantic definition are given for the language Mickey, an extension of Micro-algol. The abstract syntax and semantics are given for a hypothetical one-register single-address computer with 14 operations. A compiler, using recursive descent, is defined. Formal definitions are also given for state vector, a and c functions, and correctness of a compiler. Using these definitions, the compiler is proven correct.

45. D. Kaplan, Some Completeness Results in the Mathematics Theory of Computation, October.
A formal theory is described which incorporates the "assignment" function $a(i, k, \xi)$ and the "contents" function $c(i, \xi)$. The axioms of the theory are shown to comprise a complete and consistent set.

46. S. Persson, Some Sequence Extrapolating Programs: A Study of Representation and Modeling in Inquiring Systems, September.
The purpose of this thesis is to investigate the feasibility of designing mechanized inquiring-systems for finding suitable representations of problems, i.e., to perform the "creative" task of finding analogies. Because at present a general solution to this problem does not seem to be within reach, the feasibility of mechanizing a particular representational inquirer is chosen as a reasonable first step towards an increased understanding of the general problem. It is indicated that by actually designing, programming and running a representational inquirer as a program for a digital computer, a severe test of its consistency and potential for future extensions can be performed.

47. B. Buchanan, Logics of Scientific Discovery, December.
The concept of a logic of discovery is discussed from a philosophical point of view. Early chapters discuss the concept of discovery itself, some arguments which have been advanced against logics of discover, notably by N. R. Hanson, and S. E. Toulmin. While a logic of discovery is generally understood to be an algorithm for formulating hypotheses, other concepts have been suggested. Chapters V and VI explore two of these: (A) a set of criteria by which a hypotheses could be judged reasonable, and (B) a set of rational (but not necessarily effective) methods for formulating hypotheses.

1967

48. D. Kaplan, Correctness of a Compiler for Algol-like Programs, July.
A compiling algorithm is given which maps a class of Algol-like programs into a class of machine language programs. The semantics, i.e., the effect of execution, of each class if specified, and recursion induction used to prove that program semantics is preserved under the mapping defined by the compiling algorithm.

49. G. Sutherland, DENDRAL - A Computer Program for Generating and Filtering Chemical Structures, February.
A computer program has been written which can generate all the structural isomers of a chemical composition. The generated structures are inspected for forbidden substructures in order to eliminate structures which are chemically impossible from the output. In addition, the program contains heuristics for determining the most plausible structures, for utilizing supplementary data, and for interrogating the on-line user as to desired options and procedures. The program incorporates a memory so that past experiences are utilized in later work.

50. A. Hearn, Reduce Users' Manual, February.
REDUCE is a program designed for general algebraic computations of interest to physicists and engineers. Its capabilities include:
1) expansion and ordering of rational functions of polynomials,
2) symbolic differentiation,
3) substitutions in a wide variety of forms,
4) reduction of quotients of polynomials by cancellation of common factors,
5) calculation of symbolic determinants,
6) calculations of interest to high energy physicists including spin 1/2 and spin 1 algebra.

The program is written completely in the language LISP 1.5 and may therefore be run with little modification on any computer possessing a LISP 1.5 compiler or interpreter.

51. L. Earnest, Choosing an Eye for a Computer, April.
In order for a computer to operate efficiently in an unstructured environment, it must have one or more manipulators (e.g., arms and hands) and a spatial sensor analogous to the human eye. Alternative sensor systems are compared here in their performance on certain simple tasks. Techniques for determining color, texture, and depth of surface elements are examine.
Sensing elements considered include the photomultiplier, image dissector, image orthicon, vidicon, and SEC camera tube. Performance measures strongly favor a few (and undemonstrated) configuration that may be termed a laser jumping spot system.

1967 (cont.)

52. A. L. Samuel, Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress, June.
A new signature table technique is described together with an improved book learning procedure which is thought to be much superior to the linear polynomial method described earlier. Full use is made of the so called "alpha-beta" pruning and several forms of forward pruning to restrict the spread of the move tree and to permit the program to look ahead to a much greater depth than it otherwise could do. While still unable to outplay checker masters, the program's playing ability has been greatly improved. Some of these newer techniques should be applicable to problems of economic importance.

53. B. Weiher, The PDP-6 Proof Checker, June.
A description is given for the use of a proof checker for propositional calculus. An example of its use as well as the M and S expression for the proof checker are also included.

54. J. Lederberg and E. A. Feigenbaum, Mechanization of Inductive Inference in Organic Chemistry, August.
A computer program for formulating hypotheses in the area of organic chemistry is described from two standpoints: artificial intelligence and organic chemistry. The Dendral Algorithm for uniquely representing and ordering chemical structures defines the hypothesis-space; but heuristic search through the space is necessary becuase of its size. Both the algorithm and the heuristics are described explicitly but without reference to the LISP code in which these mechanisms are programmed. Within the program some use has been made of man-machine interaction, pattern recognition, learning, and tree-pruning heuristics as well as chemical heuristics which allow the program to focus its attention on a subproblem and to rank the hypotheses in order of plausibility. The current performance of the program is illustrated with selected examples of actual output showing both its algorithmic and heuristic aspects. In addition some of the more important planned modifications are discussed.

55. J. Feldman, First Thoughts on Grammatical Inference, August.
A number of issues relating to the problem of inferring a grammar are discussed. A strategy for grammatical inference is presented and its weaknesses and possible improvements are discussed. This is a working paper and should not be reproduced, quoted or believed without the author's permission.

56. W. Wichman, Use of Optical Feedback in the Computer Control of an Arm, August.
This paper reports an experimental investigation of the application of visual feedback to a simple computer-controller block-stacking task. The system uses a vidicon camera to examine a table top containing two cubical blocks, generating a data structure which is analyzed to determine the position of one block. An electric arm picks up the block and removes it from

1967 (cont.)

the scene, then after the program locates the second block, places the first on top of the second. Finally, the alignment of the stack is improved by analysis of the relative position error as seen by the camera. Positions are determined throughout by perspective transformation of edges detected from a single viewpoint, using a support hypothesis to supply sufficient information on depth. The Appendices document a portion of the hardware used in the project.

57. A. C. Hearn, Reduce, A User-Oriented Interactive System for Algebraic Simplification, October.

This paper describes in outline the structure and use of REDUCE, a program designed for large-scale algebraic computations of interest to applied mathematicians, physicists and engineers. The capabilities of the system include:

1) expansion, ordering and reduction of rational functions of polynomials,
2) symbolic differentiation,
3) substitutions for variables and expressions appearing in other expressions,
4) simplification of symbolic determinants and matrix expressions,
5) tensor and non-commutative algebraic calculations of interest to high energy physicists.

In addition to the operations of addition, subtraction, multiplication, division, numerical exponentiation and differentiation, it is possible for the user to add new operators and define rules for their simplification. Derivations of these operators may also be defined.

The program is written completely in the language of LISP 1.5 and is organized so as to minimize the effort required in transferring from one LISP system to another.

Some particular problems which have arisen is using REDUCE in a time-sharing environment are also discussed.

58. M. D. Callero, An Adaptive Command and Control System Utilizing Heuristic Learning Processes, December.

The objectives of the research reported here are to develop an automated decision process for real time allocation of defense missiles to attacking ballistic missiles in general war and to demonstrate the effectiveness of applying heuristic learning to seek optimality in the process. The approach is to model and simulate a missile defense environment and generate a decision procedure featuring a self-modifying, heuristic decision function which improves its performance with experience. The goal of the decision process that chooses between the feasible allocations is to minimize the total effect of the attack, measured in cumulative loss of target value. The goal is pursued indirectly by considering the more general problem of maintaining a strong defense posture, the ability of the defense system to protect the targets from both current and future loss.

1967 (cont.)

Using simulation and analysis, a set of calculable features are determined which effectively reflect the marginal deterioration of defense posture for each allocation in a time interval. A decision function, a linear polynomial of the features, is evaluated for each feasible allocation and the allocation having the smallest value is selected. A heuristic learning process is incorporated in the model to evaluate the performance of the decision process and adjust the decision function coefficients to encourage correct comparison of alternative allocations. Simulated attacks presenting typical defense situations were cycled against the decision procedure with the result that the decision function coefficients converged under the learning process and the decision process became increasingly effective.

1968

59. D. M. Kaplan, A Formal Theory Concerning the Equivalence of Algorithms, May.
Axioms and rules of inference are given for the derivation of equivalence for algorithms. The theory is shown to be complete for certain subclasses of algorithms, and several applications of the theory are illustrated. This paper was originally presented at the Mathematical Theory of Computation Conference, IBM Yorktown Heights, November 27-30, 1967.

60. D. M. Kaplan, The Formal Theoretic Analysis of Strong Equivalence for Elemental Programs, June.
The syntax and semantics is given for elemental programs, and the strong equivalence of these simple ALGOL-like flowcharts is shown to be undecidable. A formal theory is introduced for deriving statements of strong equivalence, and the completeness of this theory is obtained for various sub-cases. Several applications of the theory are discussed. Using a regular expression representation for elemental programs and an unorthodox semantics for these expressions, several strong equivalence detecting procedures are developed. This work was completed in essentially its present form March, 1968.

61. T. Ito, Notes on Theory of Computation and Pattern Recognition, May.
This is a collection of some of the author's raw working notes during the period Dec. 1965-Oct. 1967 besides the introduction. They have been privately or internally distributed for some time. Portions of this work have been accepted for publication; others are being developed for submission to journals. Some aspects and ideas have been referred to and used, sometimes without explicit references, and others are developed by other researchers and the author. Hence we have decided to publish this material as Computer Science Technical Report, although the author is planning to submit all of these works to some journals, adding several new results (not mentioned in this report), improving notations, definitions and style of presentation in some parts and reformulating completely in other parts.
The author appreciates it very much if the researchers who use or refer to the results and ideas of this report communicate with him. The publication of this report was encouraged by Prof. George E. Forsythe and Prof. John McCarthy.

62. B. Buchanan and G. Sutherland, HEURISTIC DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry, July.
A computer program has been written which can formulate hypotheses from a given set of scientific data. The data consist of the mass

1968 (cont.)

spectrum and the empirical formula of an organic chemical compound. The hypotheses which are produced describe molecular structures which are plausible explanations of the data. The hypotheses are generated systematically within the program's theory of chemical stability and within limiting constraints which are inferred from the data by heuristic rules. The program excludes hypotheses inconsistent with the data and lists its candidate explanatory hypotheses in order of decreasing plausibility. The computer program is heuristic in that it searches for plausible hypotheses in a small subset of the total hypothesis space according to heuristic rules learned from chemists.

63. D. M. Kaplan, Regular Expressions and the Equivalence of Programs, July.
The strong equivalence of ALGOL-like programs is, in general, an undecidable property. Several mechanical procedures are discussed which nevertheless are useful in the detection of strong equivalence. These methods depend on a regular expression representation of programs. An unorthodox semantics for these expressions is introduced which appreciably adds to the ability to detect strong equivalence. Several other methods of extending this ability are also discussed.

64. Z. Manna, Formalization of Properties of Programs, July.
Given a program, an algorithm will be described for constructing an expression, such that the program is valid (i.e., terminates and yields the right answer) if and only if the expression is inconsistent. Similar result for the equivalence problem of programs is given. These results suggest a new approach for proving the validity and the equivalence of programs.

65. B. Huberman, A Program to Play Chess End Games, August.
A program to play chess end games is described. The model used in the program is very close to the model assumed in chess books. Embedded in the model are two predicates, better and worse, which contain the heuristics of play, different for each end game. The definitions of better and worse were obtained by programmer translation from the chess books.
The program model is shown to be a good one for chess end games by the success achieved for three end games. Also the model enables us to prove that the program can reach checkmate from any starting position. Insights about translation from book problem solving methods into computer program heuristics are discussed; they are obtained by comparing the chess book methods with the definitions of better and worse, and by considering the difficulty encountered by the programmer when doing the translation.

1968 (cont.)

66. J. Feldman and P. Rovner, An Algol-Based Associative Language, August.

A high-level programming language for large complex relational structures has been designed and implemented. The underlying relational data structure has been implemented using a hash-coding technique. The discussion includes a comparison with other work and examples of applications of the language. A version of this paper will appear in the communications of the ACM.

67. E. Feigenbaum, Artificial Intelligence: Themes in the Second Decade, August.

In this survey of artificial intelligence research, the substantive focus is heuristic programming, problem solving, and closely associated learning models. The focus in time is the period 1963-1968. Brief tours are made over a variety of topics: generality, integrated robots, game playing, theorem proving, semantic information processing, etc.

One program, which employs the heuristic search paradigm to generate explanatory hypotheses in the analysis of mass spectra of organic molecules, is described in some detail. The problem of representation for problem solving systems is discussed. Various centers of excellence in the artificial intelligence research area are mentioned. A bibliography of 76 references is given.

68. Z. Manna and A. Pnueli, The Validity Problem of the 91-Function, August.

Several methods for proving the weak and strong validity of algorithms are presented.

For proving the weak validity (i.e., correctness) we use satisfiability methods, while for proving the strong validity (i.e., termination and correctness) we use unsatisfiability methods.

Two types of algorithms are discussed: recursively defined functions and programs.

Among the methods we include known methods due to Floyd, Manna, and McCarthy. All the methods will be introduced quite informally by means of an example (the 91-function).

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)* Artificial Intelligence Project Computer Science Department Stanford University	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

Project Technical Report

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

A.I. Memo

5. AUTHOR(S) *(First name, middle initial, last name)*

John McCarthy, Edward Feigenbaum, Arthur Samuel

6. REPORT DATE September 13, 1968	7a. TOTAL NO. OF PAGES 91	7b. NO. OF REFS 45
8a. CONTRACT OR GRANT NO. SD-183 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) AI-69 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

Statement No. 1 - Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

Recent work of the Stanford Artificial Intelligence Project is summarized in several areas:

Scientific Hypothesis Formation
Symbolic Computation
Hand-Eye Systems
Computer Recognition of Speech
Board Games
Other Projects

DD FORM 1 NOV 65 1473

Security Classification