CRT-AIDED SEMI-AUTOMATED MATHEMATICS

Bennett, James H.
Guard, James R.
Haydock, Roger
Oglesby, Francis C.
Paschke, William L.
Settle, Larry G.

APPLIED LOGIC CORPORATION
ONE PALMER SQUARE
PRINCETON, NEW JERSEY

Contract AF F-19628-67-C-0100

FINAL REPORT

Period Covered: 1 October 1966 through 31 May 1968

July 1968

This research was sponsored by the Advanced Research Projects Agency under ARPA Order No. 700

Contract Monitor: Timothy P. Hart

Distribution of this document is unlimited

Prepared

for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

CRT-AIDED SEMI-AUTOMATED MATHEMATICS

Bennett, James H.
Guard, James R.
Haydock, Roger
Oglesby, Francis C.
Paschke, William L.
Settle, Larry G.

APPLIED LOGIC CORPORATION
ONE PALMER SQUARE
PRINCETON, NEW JERSEY

Contract AF F-19628-67-C-0100

FINAL REPORT

Period Covered: 1 October 1966 through 31 May 1968

July 1968

This research was sponsored by the Advanced Research
Projects Agency under ARPA Order No. 700

Contract Monitor: Timothy P. Hart

Distribution of this document is unlimited

Prepared

for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The authors wish to acknowledge the encouragement given during every phase of the work by their technical monitor, Timothy P. Hart, and by Robert Taylor of the Advanced Research Projects Agency.

PERSONNEL

The authors wish to express appreciation for the whole-hearted efforts on the part of the following Applied Logic personnel:

Thomas F. Droege
William B. Easton
John N. Henness
Theodore A. Hess
Thomas H. Mott, Jr.

ABSTRACT

This report describes the status of the sixth in a series of six experiments in semi-automated mathematics. This effort extended from October 1, 1966 through May 31, 1968. These experiments culminated in large complex computer programs which allow a mathematician to prove mathematical theorems on a man-machine basis. SAM VI, the sixth program, uses a cathode ray tube as the principal interface between the mathematician and a high speed digital computer. An elaborate language and logical capability has been implemented in SAM VI. These include I/O languages for expressing mathematical statements in a form suitable for both the mathematician and the machine to recognize and handle with ease and convenience, a language for expressing and handling sorts and range of symbols, and an auto-logic algorithm and matching routine. The latter constitute the capability for handling, automatically, logic with equality. This capability is particularly useful at an intermediate state of the proof when it is desired to have the machine try to verify automatically a given portion of the proof.

SUMMARY

Semi-automated mathematics is an approach to theorem-proving which seeks to combine automatic logic routines with ordinary proof procedures in such a manner that the resulting procedure is both efficient and subject to control and direction by human intervention. Because it renders the mathematician an essential factor in the quest to establish theorems, this approach departs from the usual theorem-proving attempts in which the computer unaided seeks to find proofs. For obvious reasons the term "semi-automated mathematics" is employed to describe this new approach, since it views the basic role of the computer primarily as that of providing as much assistance as possible to the mathematician.

As experimental tools for studying techniques in semi-automated mathematics, a series of six computer programs, called SAM I through SAM VI, have been developed. In this report we describe the status of SAM VI. However, for the reader unacquainted with the background, let us briefly summarize the language and logic capabilities of the preceding SAM programs. A fuller account of these programs can be found in our earlier reports.

The first program, SAM I, implemented the propositional calculus in a framework of natural deduction; the goal of man-machine interaction in SAM I was to obtain proofs of minimal length. SAM II dealt with quantifier-free first-order axiom systems of mathematics. SAM II was adequate to investigate elementary mathematical theories including geometry and elementary set theory. The program left the entire burden of proof generation with the user. SAM II was responsible for checking the validity of steps and generating consequences by the basic rules. SAM III saw the beginning of the development of AUTO-LOGIC, which contained the capability for automatically handling predicate and functional logic containing equality. The capability is particularly useful at an intermediate stage of a proof when it is desired to have the machine attempt to verify a portion of a proof without requiring the user to supply all the elementary steps in the derivation. The years have seen continual increase in the power of AUTO-LOGIC to verify automatically the truth of complex deductions. SAM III initiated development of sophisticated input/output techniques and contained the first general-purpose languages for expressing mathematical statements in suitable form for both mathematician and machine.

The programs, SAM I, II, and III, were implemented on a small scientific computer, the IBM 1620. SAM IV expanded the capability of SAM III in a number of directions and was implemented on an IBM 7040, a medium scale scientific computer. The improvements were primarily in AUTO-LOGIC and in the use of SLIP (a list processing language) as the underlying framework for the program.

SAM V saw advances in AUTO-LOGIC with respect to the semi-automatic handling of equality and the algebraic aspects of mathematical theories. It also included the implementation of a CRT display as the primary interface between man and machine. This is a most convenient and flexible means of interaction and the first allowing truly real-time communication between man and machine at a rate that is efficient for the user.

SAM VI is oriented primarily toward advanced improvements in the AUTO-LOGIC routines of SAM and in experimenting with flexible control and input-output features. These latter include improvements in the CRT routines, experimentation with voice control, and a new "SNOBOL front end" which give the user (in particular, the non-programming user) a fairly natural mode for input and output of formulae and the ability easily to modify and control the AUTO-LOGIC routines. The programs, SAM V and VI, were implemented on a PDP-6, a large-scale computer with a time-sharing system.

This report summarizes and brings up to date the material contained in [2], [3]. Familiarity with our previous final report [1] is assumed.

SECTION I

IMPROVEMENTS IN AUTO-LOGIC

AUTO-LOGIC is our name for the collection of algorithms which enable SAM to generate (hopefully) interesting consequences from a finite set of pseudo-disjunctions (PSD's). It embodies four processes called reduction, expansion, digression, and contradiction, which it applies to the set of PSD's to generate new ones and then eliminate or simplify whichever of these it can. PSD's are allowed to remain in the set only if they can not be reduced by reduction or deleted by contradiction, while expansion and digression serve to generate new PSD's for the set. (For details, see Section II of [1].) Our experience with SAM in exploring various theories has shown us that no single detailed procedure performs optimally in all cases. The basic design of AUTO-LOGIC has been adequate for all of our work, but we have found it convenient to modify certain parts of the algorithms each time our research takes a new tack. In making improvements to AUTO-LOGIC, therefore, the tendency has been to add on options whose strength of application is under the control of the user. The two principal additions to AUTO-LOGIC during the period covered by this report were multiple digression and extended expansion, which we now describe.

Multiple Digression

Digression is an attempt to use the strategy of temporarily complicating a proof in order to gain some later simplification. More specifically, digression as we have implemented it in SAM uses an equality b=c to expand a formula P by replacing an instance of the "simpler" term c in P with the appropriate instance of b. (Recall that AUTO-LOGIC orders equalities in such a way that the "size" of the right-hand side is less than that of the left.) When the result of this digression is brought up from the list of expansions, its progenitors, in particular the equality b=c, are not used in reducing it. If no other PSD's reduce the digression, it is deleted. If some reduction by a PSD other than b=c is possible, the digression is kept and the main algorithm continues as usual.

We were able to improve on this process by introducing a technique which we call multiple digression. PSD's produced in the manner indicated above become "one-step" digressions; for any n, if no PSD on the list of reductions (other than progenitors) can reduce a given n-step digression which has been brought up from the list of expansions, then, using each equality on the list of reductions, the digression procedure above is applied to the n-step digression and n+1-step digressions are generated. The original n-step digression is then deleted. On the other hand, if some reduction by a PSD other than an immediate predecessor is possible, the n-step digression is kept and the main algorithm takes over. In our implementation of it, multiple digression is quite flexible in that the user can specify the maximum number of digression steps AUTO-LOGIC is to use (including none at all). This number must be chosen judiciously; in our experimentation with SAM we have come across many examples of interesting PSD's which in all likelihood would not have been generated without multiple digression, but it is clear that the process can consume a great deal of computer time and storage space if given too much latitude.

Extended Expansion

Most digressions lead nowhere, but some prove very fruitful. In hopes of achieving greater selectivity, we have extended multiple digression to a more general procedure which we call <u>extended expansion</u>. Roughly speaking, extended expansion attempts to apply matching and digression discriminately to certain parts of formulae, leaving other parts unaltered. Given a formula

P, an equality b=c, and a set S of equalities (the "digression set"), extended expansion tries to construct sequences $Q_1, Q_2, \ldots, Q_m$ of formulae satisfying

(i) Q_1 is P.

(ii) Q_{i+1} is obtained from Q_i by applying a (one-step) digression using an equality from S to some term of Q_i.

(iii) Q_{i+1} has a term which is "closer" to matching b than any term of Q_i.

(iv) Q_m contains a term which matches b. If such a sequence is found, Q_m is expanded (in the usual sense) by b=c; otherwise, no PSD is generated. In practice, the user must set an upper limit on m to keep extended expansion from consuming too much time.

Our current definition of "closer" in (iii) involves a concept of "level in b" (where b is the left hand side of our given equality). Occurrences of subterms of b are assigned <u>levels in b</u> as follows:

1. b has a level Ø in b.
2. If $g(t_1, \ldots, t_r)$ has level n in b, then each t_i has level n+1 in b. It t is any term which matches a subterm of b of level n in b, we then say that t <u>matches b at level n.</u> Terms of a formula Q matching b at a level n which

is less than or equal to the level at which any other terms of Q match b are called least-level matches of b in Q, and n is called the b-level of Q. The lower the b-level of a formula, the "closer" it is to containing a matching term for b. In applying rule (ii) above, extended expansion restricts digression to those terms of Q_i which contain a least-level match of b. The net effect of this procedure, if it is successful, is to construct a matching term for b within the given formula P. However implemented, a capability for this sort of manipulation is important for much of the work we have been doing recently, particularly our work with logical circuit design.

Skolemization of Equalities

In addition to experimenting with multiple digression and extended expansion, we have made an improvement in the way in which AUTO-LOGIC handles proof by contradiction. There are two modes of operation for AUTO-LOGIC; in the positive mode AUTO-LOGIC generates new theorems from an initial set of PSD's, whereas in the negative mode the user has a particular formula A in mind which he would like to prove to be a consequence of the original set of PSD's. In this latter mode the original list is augmented by the PSD's representing the logical negation of A and it is hoped that AUTO-LOGIC will obtain FAL as a consequence of this augmented set.

We have noticed that when using the negative mode of operation one is much more likely to be successful if A is not a simple equality. For example, we may have defined a relation R(X,Y) in some algebraic theory and now wish to show that the transitive law

R(X,Y) AND R(Y,Z) IMP R(Y,Z)

holds. The logical negation of this law involves introducing three new constants X2Ø, Y2Ø, and Z2Ø satisfying

1. NOT((R(X2Ø,Y2Ø) AND R(Y2Ø,Z2Ø)) IMP R(X2Ø,Z2Ø))

Since 1. is not in PSD form, SAM Skolemizes it into the logically equivalent

2. R(X2Ø,Y2Ø) AND R(Y2Ø,Z2Ø) AND NOT(R(X2Ø,Z2Ø))

which is added to the list of reductions as three separate PSD's

R(X2Ø,Y2Ø), R(Y2Ø,Z2Ø), and NOT(R(X2Ø,Z2Ø))

The chances of AUTO-LOGIC obtaining FAL in this case are very good (provided that the transitity of R does indeed follow from the original axioms). The reason for this is that AUTO-LOGIC can easily expand each of the separate PSD's with PSD's on the list of reductions, thereby obtaining further consequences involving X2Ø, Y2Ø, and Z2Ø in terms of the relation R and whatever is used in defining it.

On the other hand, we might attempt to show that a certain function S(X,Y) is commutative. If AUTO-LOGIC tried to work with the formula

3. NOT(S(X2Ø,Y2Ø)=S(Y2Ø,X2Ø))

the chance of expansions or digressions involving X2Ø and Y2Ø being generated would be slight, and thus a contradiction would probably not be obtained. Loosely speaking, SAM generally does not have much motivation to work with negated simple equalities like 3. which involve only constants.

Analysis of the earlier example concerning the transitive law suggests a way by which the necessary motivation can be introduced. Suppose NOT(B*=C*) is the logical negation of B=C (where B* is the formula B with all variables changed into constants of the appropriate sort, similarly for C*). If we wish to use the negative mode to prove B=C from the original list of PSD's, we add the three PSD's $B^*=k_1$, $C^*=k_2$, and NOT($k_1=k_2$), where k_1 and k_2 are new constants of the appropriate sort, instead of the single PSD NOT(B*=C*). (Of course, if B* is already a constant, we do not introduce k_1 and similarly for C*.) This procedure gives AUTO-LOGIC a much better chance of obtaining expansions and digressions involving the constants and terms of B* and C*.

Our method of breaking down negated equalities is applicable not only to the case cited, but also to any instance in which the Skolemization of the logical negation of the formula leads to a PSD of the form NOT(B*=C*). For example, the proposition

R(Z,Y) IMP (D(Z,C(Y,X))=C(D(Z,Y),X))

has a logical negation expressed by the PSD's: R(Z2Ø,X2Ø), D(Z2Ø,C(Y2Ø,X2Ø))=D2Ø,C(D(Z2Ø,Y2Ø),X2Ø)=C2Ø, and Not(D2Ø=C2Ø).

This modification in the Skolemization of equalities has been implemented in SAM, and has been found to increase greatly the power and range of application of the negative mode of operation.

SECTION II

CONTROL, INPUT/OUTPUT

The Front End.

Our work with SAM has necessitated the creation of numerous control and debugging routines which now provide the user with an extensive repertory of interactive techniques. There are, for example, routines for creating and manipulating formula libraries, changing weight functions, setting program parameters, and outputting diagnostic information. Until recently, however, these routines were inaccessible to anyone unfamiliar with the inner workings of SAM. The functions which they perform have turned out to be important for the operation of SAM, so for the sake of non-programming users (and our own convenience) we have implemented a comprehensive control package which will, we hope, greatly facilitate their use. This "front end" as we call it, will also permit remote users without a CRT to operate SAM in a fairly natural manner.

SAM's front end is basically an interpreter for a simple command language. Commands are entered from the user's Teletype and have the general format: VERB (SWITCHES) TONAME [CONDITIONS] FROMNAME. At present, VERB can be any of 19 imperatives. Depending on which of these is used, modification of the desired action can be specified by one of 14 available switches. If the verb calls for movement of formulae of formula libraries, origin and destination are specified by FROMNAME and TONAME. The action of a command can be limited to those formulae in a list or library meeting certain conditions by inserting the conditions between square brackets when typing in the command. These conditions may be anything expressible as an arithmetic or Boolean relationship among the nine quantities in the "analysis" of a formula.* Thus, the condition [NUM $\leq$ 200 AND DEP $>$ 5] is intelligible to the interpreter and would indicate that the command in which it appeared was to be applied only to formulae with numbers $\leq$ 200 and at a depth greater than 5 from the axioms. The list of verbs includes several entries which allow the user to initiate and control routine housekeeping procedures. These permit

*Each formula considered by AUTO-LOGIC has attached to it several data words in which the following items are stored: formula number, contradiction bit (set if formula is a consequence of the logical negation of something we are trying to prove), heredity bit (if set in a given formula, will also be set in each of that formula's descendents), heredity depth (depth from a formula in which heredity bit was set originally), formula numbers of major and minor antecedents, type (Ø if reduction, 1 if expansion, n+1 if n-step digression), weight, and depth from axioms. This information constitutes the analysis of a formula.

formulae to be assembled into libraries, input, output, saved on the disc, gotten from the disc, appended to or deleted from other formula lists, and displayed on a CRT. Formerly, most of these things could only be accomplished by painstaking manipulation via the PDP-6 debugging language, DDT.

Another great convenience afforded by the front end is the ease with which program parameters may be set and saved. About 30 of the most important of these have been collected into a single file which is read by the main program; the front end provides easy access to this file. Among the items saved in the parameter file are: upper and lower windows (used for throwing away PSD's with weights outside a desired range), maximum number of steps for multiple digression, instantiation and matching timers, tables of special symbols, lists of associative and commutative functions, and sort structure. The user can thus save his job at the end of a session at the Teletype by saving the parameter file along with his formula lists, sparing himself the necessity of resetting all the parameters when he goes back to work again. We have also found it convenient to regard the weighting function as a set of parameters. Specifically, we now consider the following factors in computing a weight for a formula; length (number of symbols in formula), number of disjuncts, variable density (number of symbols present divided by number of variables), symbol density (number of symbols present divided by number of representatives from a specified set of symbols), and all of the analysis items. The weighting function may be any linear combination (with integer coefficients) of non-negative integer powers of these factors, input from the user's Teletype via the front end in the form

$$\begin{array}{l} coeff_1(name_1{**}exp_1) \\ \quad * \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ \quad + \\ coeff_n(name_n{**}exp_n) \end{array}$$

where each $name_i$ is one of the abovementioned weighting factors. Coefficients and exponents are saved in the parameter file for referencing each time a formula needs to be weighted. (The command REWEIGHT, with appropriate arguments, will cause new weights to be computed for a specified collection of formulae after the weighting function has been changed. This means that the user can have SAM explore a theory stage by stage, giving precedence to different types of formulae at each new step. We have found this flexibility most helpful in our experimentation.)

The front end also embodies a few routines which "pre-massage" formulae for AUTO-LOGIC. Skolemization need no longer be done with paper and pencil but instead is accomplished automatically during an INPUT command. Formulae typed in after this command is given are transformed into PSD's by a routine based on the algorithm in [1], pp.4-6. Use of the switch (NEG) with the INPUT command causes subsequently typed formulae to be negated and Skolemized with all variables replaced by terms of the appropriate sort. This is useful in setting up proofs by contradiction, as explained in Section I of this report. (A positive copy of the original formula is kept for later use should a contradiction be obtained.) These two features can frequently save the user a great deal of bothersome computation, and at the very least help improve the appearance of formulae input to SAM by allowing them to be cast in a form more natural to mathematicians. For this latter purpose, the front end recognizes formula symbols consisting of more than one letter (so a particular group homomorphism in an algebraic theory might be denoted HOM instead of H or H1). There is also a facility whereby certain functions can be declared infix and interpreted as such when formulae are input or output.

Below we give some examples of commands which can be typed to the front end:

OUTPUT (HIS) SAVFIL [NUM $\leq$200 AND DEP $>$ 5]

(creates ASCII disc file, SAVFIL, containing copy of PSD s from list of reductions satisfying the condition; presence of (HIS) switch indicates that command will also be applied to all descendents of these PSD's on either the list of reductions or the list of expansions.)

REMOVE (HIS) [NUM $>$3] LR

(removes from list of reductions and list of expansions all PSD's, along with their descendents, satisfying the condition.)

SAVE FILNAM

(creates three ASCII disc files: FILNAM.LR, FILNAM.LE, and FILNAM.PAM containing copies of current list of reductions, list of expansions, and parameter file information, respectively.)

TYPE PAMSAM SORTS

(outputs current sort formula on Teletype.)

INPUT (NEG) LR [NUM = 3] LIBRARY

(inputs to list of reductions at LOW pointer the PSD's obtained by Skolemizing the logical negation of formula #3 in ASCII disc file LIBRARY.)

Work continues on the implementation of routines which will mediate between the mathematical symbolism of the experimenter and the algorithms of AUTO-LOGIC. It is safe to say that much of this would not be feasible without SNOBOL, the string-processing language in which the front end was wrttten. Because of the flexibility of SNOBOL coding, we will be able to build new features into it with a minimum of bother; the present version of the front end should therefore be considered a preliminary one.

A complete user's manual for SAM, including a detailed description of the front end, will be written in the near future and included in our next report.

Drum Input/Output

One of the main problems encountered in attempting to improve the efficiency of SAM has been that of handling and storing the large numbers of PSD's generated by the expansion process. In the development of SAM V and VI, we have endeavored to solve this problem on two levels. Internally, the use of "windows" and sophisticated formula-weighting techniques helps to weed out unimportant formulae before they can overwhelm SAM's storage facilities. Frequently, though, this is not enough, since many theories tend to generate great quantities of PSD's which should not be discarded right away. In SAM VI procedures to store formulae externally on the million-word drum have been implemented.

The drum I/O routine divides the list of expansions into three parts. The first part is a large drum file (DRMBIG) which contains all PSD's with weights in excess of a computed value. During the generation process, PSD's of this or larger weight are added directly to the end of DRMBIG. The second part is a small drum file (DRMNXT) which contains all the PSD's of weight less than the computed value that are not in core. The third part is the in-core list of expansions (LE). When SAM is operating new expansions are either appended to DRMBIG or melded into LE. If LE becomes empty a number of PSD's are moved from DRMNXT to LE. If DRMNXT is exhausted, DRMBIG is sorted and a new LE and DRMNXT are created. The sorting process is also used to reorder DRMBIG whenever the operator desires to change the weighting function.

Space Allocation

The I/O capabilities have been further extended by allowing the user to make maximum use of the internal I/O routines to do housekeeping and create temporary files at the user's direction. This feature requires a dynamic allocation of I/O buffers. The SNOBOL coding which has been added also requires dynamic storage. To solve these demands for space allocation a general dynamic space allocator has been written which has calls to: get new space, extend existing space, return space (even if not allocated), and clean up space. Internal logic has also been added to SAM to control the size of the core image as a function of the state of the problem that SAM is working on. This is necessary to prevent unstable situations where the core would be extended to the limit.

Auxiliary I/O Capabilities

Work continues on the development of more sophisticated I/O techniques for SAM. During this reporting period we experimented with the hardware and software components of what will eventually become a system for voice control of the CRT display. SAM already has routines which, in effect, permit augmentation of the standard CRT character set by tables of special symbols (Greek letters, mathematical punctuation, etc.); these are being improved upon, as are the routines which enable the user to output formulae with special symbols to the plotter.

SNOBOL

Well before actually writing the front end, we realized that we would need some sort of string processing language for building new I/O capabilities into SAM. SNOBOL, developed by Farber, Griswold, and Poalnsky at Bell Telephone Laboratories, seemed to meet our requirements, so a compiler for it was written and tested. We have continually improved our implementation of SNOBOL since then, and it now provides many features not available in SNOBOL3, the original Bell Labs version. A complete (but slightly outdated) description of our version of SNOBOL may be found in Section III of [3].

SECTION III

APPLIED MATHEMATICS AND SAM

By October 1966, we were for the most part satisfied with SAM's performance in handling theories which admit a simple, natural axiomatization. A group, for example, can be described by means of three short equalities and SAM proved itself capable of generating all interesting consequences of them in the space of a few minutes. Experimentation with modular lattice theory (which culminated in the proof of SAM's Lemma) demonstrated SAM's proficiency with somewhat larger axiom sets consisting mostly of simple equalities. These successes convinced us that our AUTO-LOGIC algorithms were basically sound and efficient, and that we had implemented them properly in SAM. Further major improvements could be motivated only by results obtained from the investigation of more complicated systems. Bearing in mind our original concept of SAM as a tool for anyone who does mathematical work, we decided to experiment with the sort of mathematics that finds widespread application in the physical sciences.

In particular, we wanted to attempt a fairly substantial axiomatic description and investigation of linear algebra. Our immediate goal was to discover a set of axioms which would describe not just one vector space (in a manner analogous to our much-belabored three-axiom treatment of group theory), but an entire universe of vector spaces, all over the same field. For concretness we considered our ground field to be the complex numbers, but since our axiomatization could convey no topological information, it actually described any field of degree 2 over a distinguished subfield. Fortunately, we had already gleaned a good deal of information from previous work with fields, begun late in 1965, and were aware of some of the difficulties we would encounter in exploring linear algebra. Our original axiomatic description of a field had involved a great many pseudo-disjunctions--propositions of the form: (not P_1) or (not P_2) or... or (not P_n) or Q, logically equivalent to: (P_1 and P_2 and ... and P_n) implies Q--which created difficulties previously unnoticed in the investigation of simpler theories. The tendency was for SAM to be swamped by the many trivial results obtained from combinations of the axioms, a problem which persisted despite improvements in the pseudo-disjunct algorithms and the great increase in formula storage space made possibly by our acquistion of a million-word drum in the spring of 1966. Our first foray into linear algebra involved a 36-axiom representation which was basically an extension of the system of 20 axioms we had used to investigate fields. Again, complicated pseudo-disjuncts, arising mostly from the need to identify variables as vectors or scalars, or whatever, predominated over equalities and SAM failed to produce anything of much interest despite the improvements we had made in it. Further development of SAM encouraged us to make another try, this time with a somewhat more ambitious theory employing a cleverer sort structure. Our axiomatic representation, however, was still a "straightforward" one which relied heavily on the use of disjuncts, and the same old problems recurred.

In January of 1967, we tried a different approach to the problem of representation. It was observed that all but one of the disjuncts in our axioms (that one being the disjunct forbidding divisors of zero in the scalar field) served to place variables in the spaces to which they belonged. Disjuncts like these can be eliminated by an elaborate variable sort procedure, but only a rudimentary sort theory was implemented in SAM V, then the current SAM program. Roger Haydock discovered that the same results could be obtained by representing the theory in terms of a powerful but rather non-intuitive logical function having little to do with linear algebra specifically. Details and examples may be found in [2], pp. 13-17, but the basic idea was to define a three-argument function A which applies an operator (first argument) to an element of a space (second argument) to take its value in the space named by the third argument. Thus, if J is an operator on a space, K a vector in the domain of J, and W the range space of J, we may read A(J,K,W) as "the results of applying J to K to obtain a vector in W." This enabled us to sort variables and identify what would normally be functions automatically from context. In our new representation of the theory which we called "elementary generalized graded algebra" (EGGA) only 17 separate axioms were required, a net reduction of 19 over the previous "straightforward" representation. Practically all of the new axioms were equalities and contained no multiple disjuncts--thus making them quite palatable to SAM--but they also tended to be quite long and complicated in appearance. (See [3], pp. 30-33, for a complete list of axioms for EGGA.)

This seemingly artifical construction bypassed all the weaknesses of SAM which had held back development previously. With the advent of the representation which it permitted, the elementary sort capability already incorporated in SAM became genuinely useful. Interesting results appeared almost immediately and a number of unsuspected bugs were eliminated from the coding. In addition to directing us along new avenues in our search for ways to improve SAM, our experience with more sophisticated algebraic systems encouraged us to believe that SAM might one day prove useful in performing the complicated symbolic manipulations required by contemporary physics.

SECTION IV

BOOLEAN MANIPULATION AND CIRCUIT DESIGN

In July of 1967, we began to study the feasibility of applying SAM to some of the computational problems which arise in the design of complex logical switching circuitry. Roughly speaking, every logical circuit has an algebraic representation in terms of Boolean primitives ("and" "or" "not") and truth-valued variables.* Computation of logically equivalent representations in effect "redesigns" one's original circuit by producing others which will do the same job, but in a different way. Formally, a logical circuit with inputs $X_1, X_2, \ldots, X_n$ and outputs $E_1, E_2, \ldots, E_m$ can be represented by the equations

$$
\begin{aligned}
G_1 &= H_1(X_1, \ldots, X_n) \\
G_2 &= H_2(X_1, \ldots, X_n, G_1) \\
&\vdots \\
G_r &= H_r(X_1, \ldots, X_n, G_1, G_2, \ldots, G_{r-1}) \\
E_1 &= F_1(X_1, \ldots, X_n, G_1, \ldots, G_r) \\
&\vdots \\
E_m &= F_m(X_1, \ldots, X_n, G_1, \ldots, G_r)
\end{aligned}
$$

*We did not consider circuits which involve a time delay.

where the F_i's and H_j's are Boolean functions. (The G_j's represent the possibility of an intermediate or final output of a circuit being used in more than one place. The inductive nature of their definition eliminates ambiguities due to feedback.)

These expressions can be written down immediately once we know how the circuit is supposed to behave. Now the problem is to optimize our representation in some sense or another. We might, for instance, insist that our circuit be made up solely of certain predefined logical elements (representable by Boolean functions as "subcircuits" in the same manner as the big circuit), and that it use as few of these as possible. Additional requirements, e. g., that such and such a circuit element be "nested" only so and so many levels deep, may also be imposed in practice. Computations can thus become quite intricate, and numerous attempts have been made to perform them by machine. A precise, efficient, and universally applicable algorithm for optimizing circuit representations with respect to criteria which may change from time to time would be difficult to devise, so a more open-ended approach may be useful, the sort of approach which is embodied in SAM.

Our experience with SAM has always been that it excelled at Boolean algebra and similar theories; in this case, the symbolic calculations associated with circuit design seemed like natural ones for SAM to handle. Here, our goal was different from that of our previous experimentation in that we were not interested in exploring the development of a theory from a collection of axioms. We wished rather to be able to input a system of complex logical formulae to SAM and have it produce a system equivalent in function to the given one but "better" in terms of previously selected design goals. In outline, our procedure was as follows: We first gave SAM a short, simple axiom set for Boolean algebra and allowed it to generate a sizeable list of reductions therefrom. (Since we were not worried about logical independence of the axioms, we felt free to throw in a few useful but hard-to-derive formulae in order to facilitate computation. Our axiom set also included the definitions of whatever circuit elements we wanted to work with.) This list of reductions was then edited by deleting all obviously cumbersome and useless consequences of the axioms, and saved in the usual manner. We next selected the criteria by which the circuit representations were to be manipulated and modified SAM's weighting function accordingly, so that it would assign the lowest weights to those formulae which came closest to meeting the criteria. SAM, with its new weighting function, was saved as a dump file. To do our computations, we could then bring SAM into core, read in the previously saved list of reductions and append to it the formulae to be massaged via the "INSERT" command.

We investigated several different types of manipulative problems this way, with varying degrees of success. Where the goal was merely to simplify the original representation as much as possible, SAM generally performed quite well. Work on the more general problem of changing the original representation into an equivalent representation made up entirely of specified circuit elements and <u>then</u> reducing the total weight of the system (weight being some function of the number of times each component is used) yielded results of a more ambiguous character, but on the whole we were encouraged.

In one experiment, we gave SAM the definitions

X nand Y = not (X and Y)

N^3(X,Y,Z) = not (X and Y and Z)

and allowed it to generate theorems about the functions <u>nand</u> and N^3. We next inserted a representation of an "adder" circuit with inputs X_1,X_2,X_3 and outputs E_1, E_2, namely

$$E_1 = (X_1 \text{ and } X_2) \text{ or } (X_2 \text{ and } X_3) \text{ or } (X_3 \text{ and } X_1)$$

$$E_2 = X_1 \text{ xor } X_2 \text{ xor } X_3$$

(where xor denotes the "exclusive" or: X xor Y = (X or Y) and (not(X and Y))). SAM was given a weighting function which caused it to compute E_1 and E_2 in terms of not, nand, and N^3, and then simplify the resulting expressions as much as possible. SAM's final simplification looked like

$$E_1 = N^3(X_1 \text{ nand } X_2, X_2 \text{ nand } X_3, X_3 \text{ nand } X_1)$$

$$E_2 = ((G_1 \text{ nand } G_2) \text{ nand } (\text{not } X_3)) \text{ nand } N^3(G_1, G_2, X_3)$$

where

$$G_1 = (\text{not } X_1) \text{ nand } X_2$$

$$G_2 = (\text{not } X_2) \text{ nand } X_1$$

A few minutes of paper-and-pencil computation shows that this is not a particularly easy problem. We tried others of an even more difficult nature, but with less success. Whatever the immediate results it produced, all of this experimentation was helpful to us in that it motivated the improvements we made to the expansion and digression procedures in AUTO-LOGIC. We hope that our efforts will one day make SAM into a genuinely useful tool for doing the sort of open-ended symbol manipulation discussed here.

BIBLIOGRAPHY

[1] "CRT-Aided Semi-Automated Mathematics" by J. H. Bennett, W. B. Easton, J. R. Guard, and L. G. Settle. Final Report No.AFCRL-67-0167. January 1967. (Contract No. AF 19(628)-3250)

[2] "CRT-Aided Semi-Automated Mathematics" Semi annual Report covering period: 1 October 1966 through 31 March 1967. (Contract No. AF F19628-67-C-0100)

[3] "CRT-Aided Semi-Automated Mathematics" Semi annual Report covering period: 1 June 1967 through 30 November 1967. (Contract No. AF F19628-67-C-0100-P001)

Unclassified

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) Applied Logic Corporation, One Palmer Square Princeton, New Jersey 08540	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

CRT-Aided Semi-Automated Mathematics

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

Final Scientific Report 1 October 1966 through 31 May 1968

5. AUTHOR(S) (Last name, first name, initial)

Bennett, James H. Haydock, Roger Paschke, William L.
Guard, James R. Oglesby, Francis C. Settle, Larry G.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
July 1968	31	3

8a. CONTRACT OR GRANT NO. AF F19628-67-C-0100 ARPA Order 700	9a. ORIGINATOR'S REPORT NUMBER(S) Final Report
b. PROJECT NO.	
c. TASK	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES

11. SUPPLEMENTARY NOTES Prepared for Hq., AFCRL, OAR(CRB) United States Air Force L. G. Hanscom Field, Bedford, Mass	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency

13. ABSTRACT

This report describes the status of the sixth in a series of six experiments in semi-automated mathematics. This effort extended from 1 October 1966 through 31 May 1968. These experiments culminated in large complex computer programs which allow a mathematician to prove mathematical theorems on a man-machine basis. SAM VI, the sixth program, uses a cathode ray tube as the principal interface between the mathematician and a high speed digital computer. An elaborate language and logical capability has been implemented in SAM VI. These include I/O languages for experessing mathematical statements in a form suitable for both the mathematician and the machine to recognize and handle with ease and convenience, a language for expressing and handling sorts and range of symbols, and auto-logic algorithm and matching routine. The latter constitute the capability for handling, automatically, logic with equality. This capability is particularly useful at an intermediate state of the proof when it is desired to have the machine try to verify automatically a given portion of the proof.

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Man-machine mathematics						
Semi-automated mathematics						
Mathematical displays on CRT						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through ____________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through ____________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through ____________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Unclassified

Security Classification