ANL-6447

ANL-6447

MASTER

Argonne National Laboratory

GAMMA I

A GENERAL THEOREM-PROVING PROGRAM FOR THE IBM 704

by

John Alan Robinson

DISCLAIMER

Price $1.00 . *Available from the Office of Technical Services, Department of Commerce, Washington 25, D.C.*

ANL-6447
Mathematics and Computers
(TID-4500, 16th Ed.,
Amended)
AEC Research and
Development Report

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois

GAMMA I

A GENERAL THEOREM-PROVING PROGRAM FOR THE IBM 704

by

John Alan Robinson

Applied Mathematics Division

November 1961

Operated by The University of Chicago
under
Contract W-31-109-eng-38

TABLE OF CONTENTS

GAMMA I
A GENERAL THEOREM-PROVING PROGRAM FOR THE IBM 704

by

John Alan Robinson

INTRODUCTION

GAMMA I is a FORTRAN-compiled program for the IBM 704 Electronic Data-Processing Machine. It embodies a certain general, uniform procedure H of mathematical logic for seeking out a proof of any theorem within any mathematical theory which is given in formal axiomatic form.

The procedure H is theoretically complete. Using it, one will always discover a proof for a theorem if there is one to be discovered. However, as a practical instrument, the procedure H has severe limitations; in most cases of strong mathematical interest it calls for the execution of more steps than can be carried out in any reasonable time by the fastest machines ever likely to be available. The actual capability of GAMMA I is therefore no greater than these practical limitations inherent in the procedure H will allow. Nevertheless, GAMMA I is remarkably effective in a wide class of cases, including, for example, the modern algebraic theory of lattice structures.

Plans are afoot for GAMMA II, a program which will embody other theoretical procedures over and above the procedure H, and which will possess a capability much greater than that of GAMMA I. These plans are discussed in the sequel.

Prior to a detailed description of GAMMA I itself, an extended discussion is provided of the underlying method, and of the necessary background of mathematical logic. No knowledge of this field is presupposed. In the subsequent discussion of the computer program, however, it is assumed that the reader is reasonably well acquainted with IBM 704 programming, and in particular with the FORTRAN symbolic programming system.

GAMMA I was written at the Argonne National Laboratory for the Applied Mathematics Division in the summer of 1961. The work was much facilitated by the active and helpful cooperation of George A. Robinson, Jr., and Herbert L. Gray, both of the Applied Mathematics Division.

CHAPTER I. MATHEMATICAL LOGIC

1. Introduction. The General Role of Modern Logic.

The discourse of mathematicians, when they are giving proofs and stating results, is carried on in one of the natural languages, such as English, liberally supplemented by a terse shorthand notation involving the letters of various alphabets printed in various types of formats, and by many special symbols, such as the equality or identity sign, the summation and product signs, and the sign for an integral.

Within such an enriched natural language the mathematician makes assertions, embodied syntactically in sentences, and furthermore he claims that some of the assertions follow from, or are consequences of, or are deducible from, one or more other assertions. In any particular case, to show that this is indeed so, the mathematician seeks to provide a proof of an assertion T from a set of premises, P_1, P_2, ..., P_n. The burden of the proof is to establish the fact that if the premises P_1, P_2, ..., P_n are true, then so must the conclusion, the assertion T, also be true. The question whether the premises P_1, P_2, ... P_n are indeed true is a separate matter from the question whether, if they are, then the conclusion must be.

The apparatus of modern logical theory provides an exact analysis of the notions lurking behind the words and phrases underlined in the previous paragraph. A major contribution of modern logic has been the construction of a family of artificial (as opposed to natural) languages incorporating the fruits of this exact analysis. These languages are intended as precise counterparts of the enriched natural languages traditionally used by mathematicians, having at least the same expressive power as (and, in some cases, far more expressive power than) their natural cousins. The creation of these artificial languages is a twofold boon: first, a more finely tuned instrument is thereby provided for talking, thinking, and writing mathematically; but second, the language of mathematics is now exhibited as itself a precisely defined structure, capable of being mathematically studied in just the same way as groups, rings, fields, topologies, vector spaces, and other structures are studied in traditional mathematics.

Profound and beautiful results of far-reaching importance have already been obtained in the first few decades of work made possible by the second of these two boons. Several of these results are of importance for our present discussion. The first boon has not yet, however, been exploited systematically. We are just beginning to realize its power, with the relatively recent advent of large, fast, automatic, symbol-manipulation machines.

2. Predicate Calculi of First Order

One subfamily of artificial languages constructed in modern logic consists of the so-called predicate calculi of first order. There are many members of this family, some being but slight variants of others, some being very different indeed from others, but all having in common certain fundamental features which determine the family relationship. As the title given to the family suggests, the central notion underlying the whole family is that of a predicate.

We are used to the idea, in traditional mathematical usage, of formulating such inscriptions as:

$$2 \cdot x + 4 = 10 \tag{1}$$

and

$$12 \cdot y \geq 29 \quad , \tag{2}$$

in which "x" and "y" are variables, thought of as ranging over some set (say, the set N of positive integers), "2," "4," "10," "12," and "29" as constants, "+" "·" as operations, and "=" "≥" as relations. If we systematically replace "x" in (1) by constants denoting particular members of N:

$$2 \cdot 1 + 4 = 10 \quad , \tag{1.1}$$

$$2 \cdot 2 + 4 = 10 \quad , \tag{1.2}$$

$$2 \cdot 3 + 4 = 10 \quad , \tag{1.3}$$

$$2 \cdot 4 + 4 = 10 \quad , \tag{1.4}$$

etc., we obtain a set of specific assertions, some of which are false, others true, about members of N. In our example, (1.3) is true and all the rest false. Similarly, systematically replacing "y" in (2) gives

$$12 \cdot 1 \geq 29 \quad , \tag{2.1}$$

$$12 \cdot 2 \geq 29 \quad , \tag{2.2}$$

$$12 \cdot 3 \geq 29 \quad , \tag{2.3}$$

$$12 \cdot 4 \geq 29 \quad , \tag{2.4}$$

etc., a set of assertions in which (2.1) and (2.2) are false and the rest are true.

If we call the inscriptions (1.1), (1.2), ..., and (2.1), (2.2), ..., sentences, then the inscriptions (1) and (2) are revealed as things which give rise to sentences whenever one replaces the variables which occur within them by constants, or, in other words, by names of individual entities from the set which is the range of the variables. These syntactical things are called predicates. They may have many variables, not simply one, as our examples have, and each variable may occur many times, not just once, as in our examples. Thus, e.g.,

$$3x^2 + 4x + 2y + y^2 + z = w + 3w^3 - z^2 \qquad (3)$$

contains the four variables "x," "y," "z," and "w," each with two occurrences.

So far our examples of predicates have involved just one type of variable (i.e., all ranging over the same set). But we can generalize this feature by introducing other variables of different types, with different sets to range over. Consider, for example:

$$R(\beta(\alpha(2,x), 4), 10) \quad , \qquad (4)$$

which was obtained from (1) first by writing (1) in the form

$$=(+(\cdot(2,x), 4), 10) \quad , \qquad (5)$$

that is, by writing operators (or function signs) prior to their parenthesized arguments, and relation signs prior to their parenthesized arguments; and second by replacing "=," "·", and "+" by variables "R," "α," and "β." We now may think of (4) as being a predicate containing variables of different types: "R" has for its range the set G of all relations on N (a set of which the identity relation is just one member); "α" and "β" have as their ranges the set F of all binary functions over the set N (sum and product being just two particular members of this set); while 'x' as before has as its range simply the set N itself.

The predicate (4) becomes a sentence, embodying a specific assertion, whenever "R" is replaced at each of its occurrences by the name of something in G, 'α' and 'β' are replaced at each of their occurrences by the names of things in F, and 'x' is replaced at each of its occurrences by the name of something in N. Indeed, we may envisage theoretically the results of doing this replacement in all possible ways (of which there are denumerably infinitely many!) to obtain the set I of all the (denumerably many) sentences which are instances of the predicate (4). Then each member of I is either true or false, depending on which specific replacements were used to obtain it from (4).

A further generalization from these illustrations is required. In our example, we have used binary relations and binary functions only: but in general we may work with any (finite, positive) number of arguments for both relations and functions, and not just two as in our example.

These considerations underlie the specifications of that one of the predicate calculi of first order which we shall first consider.

We provide, for this language, an unlimited supply of each of the following categories of symbols:

A. Relational variables.

$P, Q, R, P_1, Q_1, R_1, P_2, \ldots$

B. Functional variables.

$\alpha, \beta, \gamma, \alpha_1, \beta_1, \gamma_1, \alpha_2, \ldots$

C. Individual variables.

$u, v, w, x, y, z, u_1, v_1, w_1, x_1, y_1, z_1, u_2, \ldots$

D. Relational constants.

$=, \geq, \leq, >, <$, (ad libitum)

E. Functional constants.

$+, \times, -, \div$, (ad libitum)

F. Individual constants.

$0, 1, 2, 3, \ldots, \pi, e, \ldots$, (ad libitum)

and the three "grouping" symbols "(", ")", and ",". We then define two different classes of strings of these symbols, terms and predicates:

Terms

(a) An individual constant is a term.

(b) An individual variable is a term.

(c) If $\mathcal{A}$ is a functional variable or a functional constant, and $\mathcal{T}_1$, $\mathcal{T}_2, \ldots, \mathcal{T}_k$ are terms (with $k \geq 1$) then

$\mathcal{A}(\mathcal{T}_1, \ldots, \mathcal{T}_k)$

is also a term (i.e., the string consisting of $\mathcal{A}$, followed by a left parenthesis, followed by $\mathcal{T}_1$, followed by a comma, followed by $\mathcal{T}_2$, etc.).

Predicates

(d) If $\mathcal{R}$ is a relational variable or a relational constant, and $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ (with $k \geq 1$) are terms, then

$$\mathcal{R}(\mathcal{T}_1, \ldots, \mathcal{T}_k)$$

is a predicate.

It will be noted that the exact definition of predicates given above includes, as special cases of predicates, those which contain no variables, and hence are sentences:

Sentences

Any predicate which contains no variables is a sentence.

We now introduce some further ideas, based upon the fact that it is possible and indeed customary and indispensible to make new predicates out of old. There are two ways in which this is done, truth-functional combination and quantification, both of which have intuitive counterparts in the natural language.

Thus we introduce, as a further supply of symbols, the following five:

$$\sim\ ;\ \&\ ;\ \vee\ ;\ \rightarrow\ ;\ \leftrightarrow\ ;$$

and state formally the following addition to the definition of predicate:

(e) If $\mathcal{P}$ is a predicate, then so is $\sim\mathcal{P}$; if $\mathcal{P}$ and $\mathcal{Q}$ are both predicates, then $(\mathcal{P} \& \mathcal{Q})$, $(\mathcal{P} \vee \mathcal{Q})$, $(\mathcal{P} \rightarrow \mathcal{Q})$, and $(\mathcal{P} \leftrightarrow \mathcal{Q})$ are each also predicates.

Informally, the symbol "$\sim$" is intended to correspond to "it is not the case that," and is called the negation sign: "&" is intended to correspond to "and," "$\vee$" to the legalistic barbarism "and/or," and they are called, respectively, the conjunction sign and the disjunction sign; finally "$\rightarrow$" is intended to correspond to "if ... then," and "$\leftrightarrow$" to "if and only if," and they are called, respectively, the implication sign and the equivalence sign.

Our original supply of predicates (those given by part (d) of the definition of predicate) are now called atomic predicates, while those constructed via part (e) of the definition are called compound predicates.

So much, for now, for the first method of constructing further predicates from given ones, by truth-functional combination.

The second method, quantification, is intended as an exact counterpart of the natural language phrases "for all" and "there exists." For instance, harking back to example (1), we may say (falsely):

"For all x, $2x + 4 = 10$" (6)

or we may say (truthfully):

"There exists an x such that $2x + 4 = 10$" . (7)

Two facts are noteworthy about (6) and (7). First, even though they contain variables, they are sentences, that is, they make specific assertions and are hence either true or false. The variables which they contain are not, as in our earlier examples, capable of being replaced by constants to produce sentences; on the contrary, if we replace them by constants we get gibberish, e.g.,

"For all 3, $2 \cdot 3 + 4 = 10$" (8)

"There exists a 17 such that $2 \cdot 17 + 4 = 10$" . (9)

We therefore say that the variables are dummy variables, or that they have been "killed" or "bound" by the phrases "for all ..." and "there exists ... ."

The second noteworthy fact about (6) and (7) is that their truth or falsity depends on that of sentences which are instances of the predicates from which they were obtained. Thus (6) is true just in case all the instances of (1) are true sentences, while (7) is true just in case not all of these instances are false.

We incorporate these ideas into our artificial language by adding the symbols "E" and "A" to our stock of symbols, and by appending a third clause to our definition of predicate:

(f) If $\mathcal{V}$ is an individual variable, and $\mathcal{P}$ is a predicate, then

$(\mathrm{A}\,\mathcal{V})\mathcal{P}$ and $(\mathrm{E}\,\mathcal{V})\mathcal{P}$

are both also predicates.

In order to make precise the notions of dummy variables or bound variables, we must now add the following formal characterization:

(g)(i) Each occurrence of any individual variable appearing in an atomic predicate is a free occurrence of that variable in that predicate.

(ii) Free occurrences of variables in $\mathcal{P}$ and $\mathcal{Q}$ remain so in $(\mathcal{P} \,\&\, \mathcal{Q})$, $(\mathcal{P} \vee \mathcal{Q})$, $(\mathcal{P} \rightarrow \mathcal{Q})$, $(\mathcal{P} \leftrightarrow \mathcal{Q})$, and $\sim\mathcal{P}$, $\sim\mathcal{Q}$.

(iii) Each occurrence of $\mathcal{V}$ which is free in $\mathcal{P}$ is bound in $(A\mathcal{V})\mathcal{P}$ and in $(E\mathcal{V})\mathcal{P}$, and is furthermore bound by that particular occurrence of "A" or "E." (In addition, that occurrence of $\mathcal{V}$ immediately after "A" or "E" is also bound in $(A\mathcal{V})\mathcal{P}$ or $(E\mathcal{V})\mathcal{P}$, and is likewise bound by that particular occurrence of "A" or "E.") All other free occurrences of variables in $\mathcal{P}$ remain free in $(A\mathcal{V})\mathcal{P}$ and $(E\mathcal{V})\mathcal{P}$. $(A\mathcal{V})\mathcal{P}$ is called the scope of that occurrence of "A," likewise for "E." The group of symbols "$(A\mathcal{V})$" is called a universal quantifier: "$(E\mathcal{V})$" is called an existential quantifier.

Informally, in examining a predicate to determine which occurrences of its variables are bound and which free, and, if bound, by what occurrences of "A" or "E," we work from the "inside" of the predicate to the "outside," by starting with those innermost occurrences of "A" or "E" which have no other quantifiers in their scopes, and allotting to them any occurrences of 'their' variables which occur in their scopes. Then we repeat this operation for the other quantifiers, repeatedly taking the innermost quantifier not yet treated, allotting to them all free occurrences of their variables inside their scopes (for now there may be some bound occurrences, owned by quantifiers inside the scope, which have therefore already been allotted).

Thus, every occurrence of every individual variable in every predicate is either free or bound, and, if bound, bound by one and only one quantifier: and we can always determine the unique way in which this must be the case.

This completes the morphology of our language. We comment here that the restriction of quantification to individual variables is what determines our language to be a predicate calculus "of first order"; languages in which quantification over relational variables and functional variables are also studied are designated as being of higher order than the first.

3. Interpretations. Validity, Satisfiability, Consequence.

We have already, in our informal preliminary discussion, touched upon the interpretation of this artificial language. Let us now go into the matter more precisely.

No specific interpretation is provided for the language defined above. Indeed, it is a "general-purpose" language, and there comes with it a set of instructions as to how to make your own interpretation suitable to the job you have in mind in using the language. The instructions are these:

A. Choose a specific set S as the range of each individual variable, and to each member of S allot a unique distinct one of the individual constants to be its name. (You may have a lot of individual constants left over, with nothing in S to name; if so, forget about them.)

B. Now consider successively the set of all singulary functions from S to S, all binary functions from SxS to S, and so on; these sets are to be, respectively, the ranges of any functional variable which occurs in a predicate with one, two, ..., arguments.

C. Allot to each member of each of these sets of functions over S, exactly one of the functional constants to be its name.

D. Do the same as in B and C, for all the singulary, binary, ..., relations over S, assigning thereby ranges to all occurrences of relational variables, and assigning distinct unique relational constants as names for the distinct relations over S.

Notice that the only choice you have in making an interpretation is of the initial set S, and of the names of the various entities thus determined from the stock of general-purpose names provided for you by the language. Once you have done this, the status of all predicates of the language is uniquely determined except those containing unused names, which we ignore in what follows. In particular, the truth or falsity of each sentence is completely fixed by your choice of S and your assignment of names. Two other labels are provided in order to assign a status to predicates which are not sentences. We say that a predicate is satisfiable in an interpretation if not all of the sentences which are its instances are false in the interpretation; a predicate is valid in an interpretation if none of the sentences which are its instances are false in the interpretation.

Now it makes at least as much sense here as anywhere else in mathematics to pass to the notion of all possible sets S which might be chosen as the initial set for an interpretation of our general-purpose language, and thence to the class of all possible interpretations which the language can be given. In terms of this idea, we can assign absolute statuses to the predicates as follows: a predicate is valid (period) just in case it is valid in all interpretations; a predicate is satisfiable (period) just in case there is at least one interpretation, among all the possible interpretations, in which it is satisfiable.

We are now at a point where we can define exactly what is meant by saying that in our language a sentence T "follows from" or " is a consequence of" one or more other sentences P_1, ..., P_n. The definition is this: T is a consequence of P_1, ..., P_n just in case that T is true in every interpretation in which P_1, ..., P_n are each true. (It might also be true in other interpretations besides, but it must at least be true in the ones stated.)

The reader may be thinking that these definitions of "validity," "satisfiability," and "consequence" are highly impractical to use as a down-to-earth means of assigning a status to a predicate, or of determining whether one sentence is a consequence of one or more others. He is right. They are. But they are not intended as practical criteria for determining these properties of predicates, but rather only as analyses of the actual content of these notions in general mathematical (i.e., set-theoretic) terms. For practical purposes, we pass now to another set of concepts and definitions which are intended, in a down-to-earth sense, to be usable by people and machines.

4. Deduction; Proof; Theorems; The Decision Problem.

The criterion for determining whether a sentence T is a consequence of one or more other sentences P_1, P_2, ..., P_n given in the previous section is not, in general, practically applicable, since it involves the totality of possible interpretations. A different kind of criterion, whose application depends only on the immediate syntactical structure of the sentences in question, is therefore provided. In outline, the criterion consists of a small set of structural relationships which one sentence T can have with respect to a set of other sentences, P_1, P_2, ..., P_n. In any particular case, where a sentence T does in fact bear one of these relationships to a set P_1, P_2, ..., P_n, we say that T is immediately deducible from P_1, P_2, ..., P_n by virtue of the relationship in question. Each of the relationships is carefully defined so that the question as to whether it does or does not obtain between a sentence T and a set of sentences P_1, P_2, ..., P_n is always effectively decidable by a mechanical procedure which is uniform and which is a part of the specification of the relationship.

If we have a sequence T_1, T_2, ..., T_m of sentences with the property that each T_i, $1 \leq i \leq m$, in the sequence either (a) is itself one of a given set of sentences P_1, ..., P_n, or (b) is immediately deducible from a set of sentences each of which occurs earlier in the sequence, then we say that the sequence is a deduction of its last member, T_m, from the set P_1, ..., P_n as premises.

The question whether a given sequence of sentences, alleged to be a deduction of its last member from a given set of sentences, is or is not indeed such a deduction, is again mechanically decidable in a uniform way.

What is not, alas, a mechanically decidable question is whether or not for a given sentence T and a given set P_1, P_2, ..., P_n of sentences, there exists a deduction of T from P_1, P_2, ..., P_n as premises. The proof that this question is not mechanically answerable by "Yes" or "No" through application of an algorithm is one of the great results of modern logic. It was first given in 1936 by Alonzo Church of Princeton University. This is not the place to discuss this matter at any more length. Suffice it to say that, as will appear in the sequel, there are mechanical methods which will uniformly determine a deduction of a sentence T from a set P_1, P_2, ..., P_n of sentences, provided that such a deduction exists; but if no deduction exists, these methods will, in general, never terminate in a discovery of this nonexistence. They are, therefore, only "semi-algorithms," capable only of answering "Yes," not capable, in general, of answering "No."

A second great result of importance here was first given by Kurt Gödel in 1931. It is that any sentence T which is valid in the sense of the previous section can be obtained as the last member of a deduction from the empty set of premises. (This property is known as the completeness of the deductive apparatus of the language.) It is also the case that only valid sentences can be so obtained. Thus, although the characterizations are utterly different, the concepts of validity and deducibility from the empty set of premises in fact determine precisely the same class of sentences.

It would require too much space to discuss the details of the relationships governing immediate deducibility. Excellent accounts are available in the literature [see especially W. V. Quine's Methods of Logic, Revised Edition, Holt-Dryden (1959) and P. Suppes' Introduction to Logic, Van Nostrand (1957)], but each differs from the other in various ways which do not affect the completeness and consistency properties. For our purposes, we note just one of the deducibility principles, namely, that if a sentence S is deducible from premises P_1, ..., P_n, then the sentence $(P_1 \rightarrow (P_2 \rightarrow \ldots (P_n \rightarrow S)\ldots))$ is deducible from no premises at all, and is therefore valid (or "logically true") by virtue of Gödel's completeness theorem. Hence, to every deduction there corresponds a valid sentence, and the question whether a sentence T can be deduced from premises P_1, ..., P_n is equivalent to the question whether the sentence $(P_1 \rightarrow (P_2 \rightarrow \ldots (P_n \rightarrow S)\ldots))$ is valid. Since the first question admits of no algorithmic method for its settlement, neither can the second.

Now the question whether a sentence S is valid is equivalent to the question whether its negation, $\sim S$, is satisfiable (to be precise, S is valid just in case $\sim S$ is not satisfiable). And the negation of $(P_1 \rightarrow (P_2 \rightarrow \ldots (Pn \rightarrow S)\ldots))$ is $(P_1 \& P_2 \& \ldots \& P_n \& \sim S)$, where inner parentheses have been omitted for the sake of revealing the pattern. If, therefore, we could show $(P_1 \& P_2 \& \ldots \& P_n \& \sim S)$ to be unsatisfiable, we would have shown that S is deducible from P_1, ..., P_n as premises.

But this is just the general problem of theorem proving in axiomatized mathematical theories. Let the axioms of a theory be written as sentences $P_1, \ldots, P_n$ of the first-order predicate calculus given here, and let the theorem to be proved be written as a sentence T. To prove that T is a theorem of the theory embodied in the axioms $P_1, \ldots, P_n$ is then essentially the same task as that of showing the single sentence $(P_1 \& P_2 \& \ldots \& P_n \& \sim T)$ to be unsatisfiable.

CHAPTER II. THE PROCEDURE H.

1. Prenex Conjunctive Normal Form. The Procedure H.

There is a straightforward technique [described in Quine, op. cit., or in Hilbert and Ackermann's Mathematical Logic, Chelsea (1950)] whereby any sentence of the predicate calculus can be put into a certain standard form called the prenex conjunctive normal form. In this form, all the quantifiers (if any) of the sentence occur at the beginning, and jointly comprise the prefix of the sentence. Furthermore, the predicate part of the sentence (often called the matrix) has the form of a conjunction of disjunctions, each member of which is either an atomic predicate or the negation of an atomic predicate.

In preparing to carry out the procedure H in order to show a sentence $(P_1 \& P_2 \& \ldots \& P_n \& \sim S)$ inconsistent, we first reduce P_1, $P_2, \ldots, P_n$, and $\sim S$ to prenex conjunctive normal form, each separately from the other. This step results in a finite list of sentences each beginning with a finite sequence of quantifiers.

The second step is to drop, from the front of the first sentence which begins with an existential quantifier, that existential quantifier, and to replace each occurrence of the variable thus freed by an individual constant (the same one at each occurrence) which does not occur elsewhere anywhere in the list of sentences. This operation is repeated until each sentence begins either with a universal quantifier or with no quantifiers at all.

(These two steps must be performed prior to inputting the sentences to the program GAMMA I.)

Now we proceed to append successively to the list of sentences further sentences obtained by systematically dropping initial quantifiers from earlier sentences in the list and replacing the variables, thus freed, by systematically chosen individual constants. (In what follows, we employ numerals as the individual constants.) The systematic method is given as follows:

Suppose the list at the n^{th} step of this process consists of the sentences $S_1, S_2, \ldots, S_p$, and that the largest numeral to have been used so far as an individual constant is ϕ. Then:

(i) If S_p begins with an existential quantifier, let S_{p+1} be the sentence obtained from S_p by dropping that existential quantifier, and replacing each of the occurrences of the variable thus freed by the numeral $\phi+1$.

(ii) If S_p does not begin with an existential quantifier, let Ψ be the earliest numeral with which some universally quantified sentence on the list has not yet been instantiated; if $\Psi > \phi$, the process terminates. Otherwise, let S_j be the first universally quantified sentence on the list which has not yet been instantiated with Ψ, and let S_{p+1} be the result of instantiating S_j with Ψ.

Successive sentences added in this way are all consequences of the original starting list. There is a great theorem, first proved by Jacques Herbrand in 1930, that the conjunction of the sentences in the starting list is unsatisfiable just in case, for some integer x, the conjunction of the first x sentences on the generated list is truth functionally unsatisfiable. Furthermore, we can drop from this conjunction any sentences that begin with quantifiers, and consider only the conjunction of the quantifier-free sentences (i.e., those containing no variables).

But we can always test the quantifier-free conjunction of sentences at any point in the generating process and arrive at a decision algorithmically whether or not it is truth-functionally unsatisfiable, and thus, if there is an x at which the conjunction of all the quantifier-free sentences up to and including S_x is unsatisfiable, we shall certainly find it, and thereby have proved that the original list of sentences is inconsistent, and that therefore the associated deduction can be made.

Earlier attempts to mechanize this method were handicapped by a lack of an efficient method for testing for truth-functional unsatisfiability. But recently, Martin Davis and Hilary Putnam (A Computing Procedure for Quantification Theory, Journal of the Association for Computing Machinery, Vol. 7, No. 3, July 1960) gave a remarkably efficient method which is now described here, slightly modified. An elegant and useful additional feature (due to Herbert L. Gray) has been added.

2. The Truth Functional Method of Davis and Putnam

The conjunction of a finite number of sentences, each of which is in conjunctive normal form (i.e., is a conjunction of disjunctions of atomic sentences), is itself a sentence in conjunctive normal form. Let $A_1,\ldots,A_k$ be the distinct atomic sentences occurring anywhere in the conjunction. Each disjunction contains one or more members of the set $[A_1, \ldots, A_k]$, either negated or unnegated (but not both A_i and $\sim A_i$, for any i). The question whether the entire conjunction is satisfiable or not is the question whether or not truth values can be assigned to each of the A_i in such a way that each disjunction in the conjunction is made true. In order to make a disjunction true, one need only make at least one of its members true. If none of the 2^k possible assignments of truth values to the set $[A_1, \ldots, A_k]$ makes the whole conjunction true, then it is unsatisfiable; otherwise it is satisfiable.

The Davis-Putnam procedure consists of eliminating successively each atomic sentence from the conjunction until either all of them are eliminated (in which case the original conjunction was satisfiable) or a stage is reached at which two different disjunctions are obtained, both containing only one member, such that the member of one disjunction is the negation of the member of the other. Specifically, we perform the following process:

1. If the conjunction is now empty, then the original conjunction was satisfiable. The process terminates.

2. Otherwise, if, in the current conjunction, each disjunction contains at least one unnegated atomic sentence, then the original conjunction is satisfiable; likewise if each disjunction contains at least one negated atomic sentence. The process terminates. (This step is due to Herbert L. Gray.)

3. Otherwise, if the current conjunction contains a pair of disjunctions whose only members are, respectively, an atomic sentence and the negation of the same atomic sentence, the original conjunction is unsatisfiable, and the process terminates.

4. Otherwise, if there is at least one disjunction which contains only one sentence (either an atomic sentence or the negation of an atomic sentence), we delete from the conjunction all disjunctions containing that sentence, and delete all individual occurrences of its negation ($\sim S$ is the negation of S, and S the negation of $\sim S$) wherever they occur. Then return to step 1.

5. Otherwise, if any atomic sentence occurs only unnegated throughout the entire conjunction, or occurs only negated, eliminate all disjunctions which contain it. Then return to step 1.

6. Otherwise, we have the situation that every atomic sentence occurs both negated and unnegated in the conjunction, and no disjunction contains less than two members. Write therefore, the conjunction in the form

$$(A \vee D_1) \& \ldots \& (A \vee D_m) \& (\sim A \vee E_1) \& \ldots \& (\sim A \vee E_n) \& G_1 \& \ldots \& G_r$$

and then write

$$(D_1 \vee E_1) \& \ldots \& (D_1 \vee E_n) \& (D_2 \vee E_1) \& \ldots \& (D_2 \vee E_n) \& \ldots \& (D_m \vee E_1) \& \ldots \&$$

$$(D_m \vee E_n) \& G_1 \& \ldots \& G_r,$$

a conjunction of disjunctions in which the atomic sentence A does not occur. The D_i, E_j, and G_h are disjunctions involving atomic sentences other than A. Then return to step 1.

In the above process, each step where a new conjunction is obtained with at least one less atomic sentence occurring in it than in the conjunction from which it was obtained carries with it the assurance that the new sentence is truth-functionally satisfiable if and only if the old one is. Proofs may be found in the paper by Davis and Putnam cited previously.

Each time an iteration of the process is carried out, at least one atomic sentence is removed, and hence the entire process terminates in a decision, as to satisfiability or unsatisfiability of the starting conjunction, in at most k iterations. In practice, far fewer iterations than k are found to be required for most cases that are actually encountered.

CHAPTER III. THE PROGRAM GAMMA I

1. The language used by GAMMA I

The actual artificial language employed by GAMMA I is but a sub-language of that described in Chapter I. GAMMA I's language contains no functional variables and no functional constants. Hence, all of its terms are either individual variables or individual constants. Despite this apparent loss of expressive power, we still in fact have just as expressive an instrument as before; now, however, in order to state certain things, we must resort to a slightly less convenient and familiar technique of formulation. Instead of saying, e.g.,

$$2 \cdot x + 4 = y \quad , \tag{1}$$

we must introduce a relational constant, say E, and write

$$E(x, y) \quad , \tag{2}$$

defining E to be such a relation that (2) is true for just those ordered pairs of constants for which (1) is true.

By this subterfuge we can reformulate any assertion, or predicate, which involves functional variables or functional constants, by introducing relational variables or relational constants.

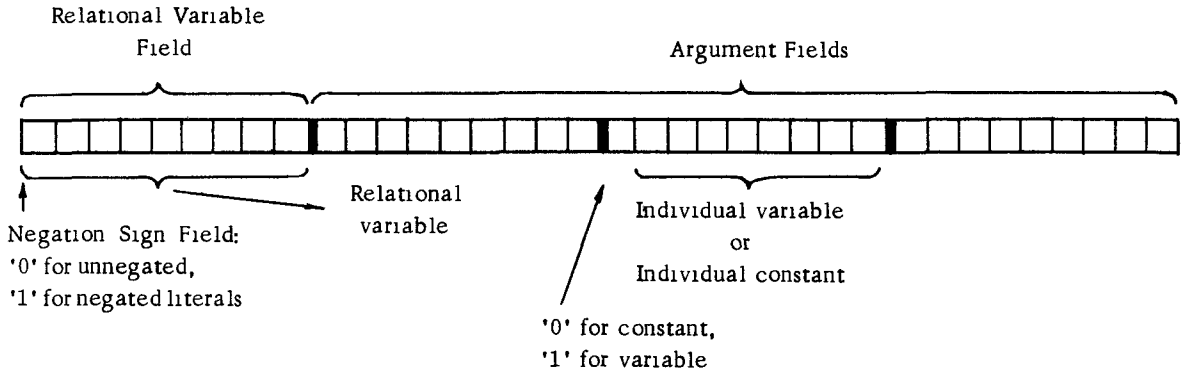
A further restriction on GAMMA I's language is that we may use no more than three arguments in any predicate. This still leaves us with plenty of room to operate; most of the interesting examples require no more than three-term relations. The reason for this restriction is that it rendered the programming problem immeasurably easier. It is planned that in GAMMA II no such restriction will be imposed, and the full apparatus of the language introduced in Chapter I will be the language employed.

In addition to these quite major restrictions, several minor ones should be pointed out. By confining ourselves to fixed-format data fields in GAMMA I's design, we created the restriction that no more than 255 different relational variables could be employed, no more than 255 different individual variables, and no more than 255 different individual constants. These are minor simply because problems which would not fit within them would be already absurdly infeasible problems for GAMMA I, on other grounds entirely. We shall be discussing these other grounds later.

2. Atomic sentences and predicates and negations thereof: literals

Let us for convenience use the word "literal" to denote indifferently an atomic sentence, an atomic predicate, or a negation of either. The term is due to Davis and Putnam.

The basic 36-bit 704 word provides the frame for the structure of atomic sentences and predicates in GAMMA I's language. The word is divided into four fields of 9 bits each (see diagram). Each field is subdivided into two subfields, the first being the leftmost bit, the second being the other eight bits.



In each of the four eight-bit fields, we may put any one of the 255 eight-bit patterns, 00000001 through 11111111, to indicate which symbol (relational variable, individual variable, or individual constant, depending on which field and, in the second two cases, on which of '0' or '1' occupies the leftmost bit-position in the field) we have selected to put there. Singulary and binary predicates use only one (the first) and only two (the first and second), respectively, of the three argument fields, the unused ones being left with binary zeros in each bit-position. Negation of the whole atomic sentence is indicated by a '1' in the first bit position; lack of negation by a '0' there.

Which eight-bit patterns are used to represent which relational variables or which individual variables has been left to the discretion of the user of the program: but the individual constants are considered by the program to be ordered in their natural order from 00000001 to 11111111, for the purposes of carrying out the instantiation process within the procedure H. If the program is allowed to run for so long a time that the instantiation process calls for the substitution of an individual constant beyond 11111111 in this ordering, it terminates at that point with a printed explanation of its reason for having stopped. Its capacity has been reached in this direction. (There are other ways in which its capacity can become exhausted also: these will be explained in the appropriate place.)

3. Disjunctions of literals

Since the sentences manipulated by GAMMA I are at all times in prenex conjunctive normal form, we are able to represent them without explicitly employing symbols for either disjunction or conjunction.

To represent a disjunction of N literals we simply construct a sequence containing the N words encoding the literals, and prefix at the front of the sequence a further word containing the integer N in the FORTRAN integer word format, viz., with low-order bit in the 18th bit position (a position helpfully designated as the seventeenth, under IBM's conventions, which involve denoting the first position as 'S' (for 'sign'), the second as first, the third as second, and so on).

A conjunction of M disjunctions is then represented by a sequence of M such sequences as were defined in the previous paragraph, the whole sequence being prefixed by a word containing an integer, in FORTRAN integer word form, giving the total number of words which are contained in the M disjunctions. (It would have been nicer if the integer to be specified were M; but life is not like that, always.) Included in this count must be the words prefixing each disjunction. As a schematic illustration, consider

$$(14)\quad \overbrace{(1)(A)}\ \overbrace{(3)(A)(B)(D)}\ \overbrace{(2)(C)(D)}\ \overbrace{(4)(A)(G)(H)(K)}\quad .$$

The letters represent literals: there are four disjunctions, containing, respectively, one, three, two, and four literals, as indeed their respective "counters" (as we shall henceforward often refer to them) indicate. The total number of words, including the four counters as well as the literals, is fourteen, and the leading word so indicates.

By this means we avoid the necessity of employing special symbols for conjunction and disjunction, at the expense of having to use the counters; these, however, facilitate the internal computer processing of the sentences enormously.

4. The Quantifier Prefix

The remaining portion of a sentence in prenex conjunctive normal form, over and above the conjunction of disjunctions of literals which comprises its matrix, consists of the initial sequence of zero, one or more quantifiers which bind the individual variables within the matrix.

It turned out to be far more efficient for GAMMA I to put its quantifier prefixes not at the beginning, and in the natural order, but rather at the end, and in the reverse order. In constructing sentences for input to GAMMA I, therefore, that is where we put, and that is the order in which we put, the "prefix" (a designation no longer very appropriate).

If the total number of quantifiers in the prefix is K (including the case K = 0), we first put K, as a FORTRAN integer-word, immediately after the last word of the matrix. We then put successively a FORTRAN integer-word for each quantifier, with positive sign indicating universal, negative sign existential, quantifiers. The integer used in each quantifier

is that corresponding to the bit pattern representing the individual variable belonging to the quantifier, i.e., we simply place the bit pattern itself with its rightmost bit in the 18th bit position of the quantifier word. Of course, if K = 0, we do not put any quantifier words after the prefix counter. But the zero word is mandatory, for the counter itself. GAMMA I takes the prefix counters quite seriously.

5. The Input to GAMMA I

Thus we construct the sequence of 704 words which is a representation of a sentence in prenex conjunctive normal form for GAMMA I. A set of such sentences, comprising the initial list for the procedure H, is represented by sticking the respective word sequences end-to-end to form one single sequence containing, let us say, W words in all (including all of the various counters). The number W is supplied to GAMMA I as the value of a FORTRAN integer variable MATEND. The number of sentences in the list (i.e., the number of the prenex conjunctive normal form sentences) is likewise supplied, as the value of a variable JLINE. The highest individual constant which occurs anywhere in the input list of sentences is considered as an integer in the obvious way and supplied as the value of a variable LPHI.

The sequence of words comprising the list of sentences after the above fashion is given to GAMMA I as a one-dimensional FORTRAN array MATRIX. MATRIX (1) is thus the first word of this array, and MATRIX (MATEND) the last word of this array.

The final major piece of input information required by GAMMA I is a list of numbers stating respectively in which word of the array MATRIX the successive prenex conjunctive normal form sentences begin. (The first such number clearly is always 1.) This list of numbers, in ascending order, is given as a one-dimensional FORTRAN array LINE. LINE (1) is thus the first word of this array; LINE(JLINE) is the last.

We have also provided, as a convenience, for up to 120 words of BCD comment data, which is read in along with all the other information at input time and used essentially as a label for the problem. Any material capable of being printed may be put in the comment array, which is formally a FORTRAN one-dimensional array LEAD. The number of words actually used is supplied as the value of a variable LEADMX, and thus the array will begin with LEAD(1) and end with LEAD(LEADMX).

Physically, an input deck of cards is prepared as follows:

1st card:

In four successive 6-column fields, beginning with column 1, the values of MATEND, JLINE, LPHI, and LEADMX are punched, hard over to the right of each field, with leading zeros either left blank or not, as one pleases.

2nd through kth cards:

(Where k is no greater than it must be in relation to the value assigned to LEADMX.) The comment is punched, 6 characters to a word, 12 words to a card. It is not mandatory that LEADMX be a multiple of 12.

(k+1)st card onwards:

Each card has five successive fields of 14 columns, starting with column 1; in the rightmost 12 columns of each field are punched, in octal form, the words of the array LINE followed by the words of the array MATRIX. There will therefore be (JLINE + MATEND) octal words in all.

Such a deck is the entire input information required for a problem. GAMMA I will handle one problem after another, and we simply stack the respective decks on top of each other, in the desired order, in the card reader. After having processed the last deck in such a batch of problems, GAMMA I selects the card reader in quest of another; finding none, it stops, and this is the normal manner for a run to terminate. This is indeed the only stop not deemed worthy of a printed comment at the on-line printer.

After having processed a problem, GAMMA I prints the entire comment array LEAD at the on-line printer, followed by its verdict (INCONSISTENT or CONSISTENT), followed by the number of minutes which were required to complete the procedure H and find the proof. If a problem were submitted which was not in fact inconsistent (a synonym for unsatisfiable), and which was not in the category for which the instantiation process terminates, then, whereas the theoretical procedure H for such a problem goes on for all eternity, GAMMA I goes on until its capacity is exhausted in one or other of the several ways in which this can occur. Since this could be quite a long time, a way has been provided to terminate a problem arbitrarily and peremptorily from the console: sense switch 2 is pushed down, and GAMMA I prints out an appropriate comment at the on-line printer and moves on to the next problem, if any. When this "get-off" facility is used, sense switch 2 should be placed UP again before the next problem deck has been entirely read in - otherwise it too will be summarily terminated.

In addition to the information supplied to the on-line printer at the end of each problem, fuller information is written on TAPE 2 pertaining to the problem just completed. In particular, a copy is provided of the quantifier-free conjunction of sentences which was found to be inconsistent (or found to be consistent, in the case of a problem for which the instantiation process happens to terminate, and which, in addition, happens to be consistent). The remaining information consists of the number of times the Davis-Putnam truth-functional analysis was performed, and the number of iterations of it which were required in the performance of its final, decisive application.

After these notes about the observable behavior of GAMMA I, and the discussion of its input requirements, we now pass to an examination of the program itself.

6. GAMMA I itself

It seems reasonable to discuss GAMMA I's organization and structure entirely on the FORTRAN symbolic statement level, the precise details of the compiled object program in SAP machine language not being necessary for a knowledge of the program's logical properties.

The flow of events takes place essentially just as is specified in the theoretical algorithm which we have been calling procedure H. The original input array MATRIX grows longer, for we add to it further sentences obtained by instantiation, provided they have one or more quantifiers surviving in their prefixes. Those that do not, which are therefore quantifier-free sentences, we segregate, and stack them end to end in a new array called MODEL. In doing so, we omit both the initial counter (which, one recalls, gives the total number of words in the matrix part of the sentence) and the terminal (zero) prefix counter. We can do without the former because we no longer need to preserve the separate identity of each sentence making up the quantifier-free conjunction, for we are building up but a single, long, sentence; and we can do without the latter because in the present context it tells us only something we already know - that there are no quantifiers in the prefix. The array MODEL therefore grows longer as the procedure continues. Its length is at all times stored as the value of a variable, MODEND. As the array MATRIX grows, so does the array LINE, whose successive entries tell where, in MATRIX, the successive sentences begin. The values of MATEND and JLINE, respectively, at all times tell how long each of these arrays is.

Eight thousand words of memory are available for the array MATRIX, 18000 for the array MODEL, and 2000 for the array LINE. If at any point these storage areas are about to be exceeded, GAMMA I interrupts its orderly processes, prints out a comment appropriate to the occasion, and proceeds to the next problem, if any, in the input deck.

There are two modes in which GAMMA I operates as far as concerns the matter of when to test MODEL by the Davis-Putnam process for satisfiability or unsatisfiability. The first mode is automatic, in which GAMMA I chooses for itself when to test, by a criterion to be explained in a moment. The second is manual, whereby the decision when to test is made at the console. Which mode is operative is determined by the setting of sense switch 4: UP for automatic mode, DOWN for manual mode. In the manual mode, the decision to test is effected by depressing sense switch 5 and raising it again when the READ-WRITE SELECT light is lit up. This phenomenon will occur after a second or two, and indicates that the Davis-Putnam test procedure has gotten under way.

The points at which testing is done when in the automatic mode are determined as follows: whenever, in the instantiation process, the sentence in MATRIX which is about to be instantiated is the first, the instantiation process is interrupted and a Davis-Putnam test of the current MODEL is performed. Whenever this point is found to have been reached, a test is also made to see whether the individual constant, which is about to be used to instantiate the first sentence with, is greater than the largest one introduced so far. If it is, then the instantiation algorithm calls for termination. Therefore this fact is noted whenever it comes about, and GAMMA I does not proceed with the instantiation process after the Davis-Putnam test has been performed. Otherwise (unless the Davis-Putnam test turns up an inconsistency) the instantiation process is then resumed right where it was interrupted.

Since the Davis-Putnam test is "destructive" in the sense that MODEL is successively reduced, perhaps to nothing, during the test, and since, should the instantiation process have to be resumed, MODEL must at that point be what it was before Davis-Putnam havoc was wrought upon it, we write it out on TAPE 3 prior to testing, along with the entire array MATRIX. The latter array must also be "saved" because the Davis-Putnam test requires its storage area as an extensive "scratch pad" on which to make notes. Specifically, the Davis-Putnam process involves the construction of an array LIST during each iteration, the entries whereof are literals which are either the lone occupants of a disjunction, or are such as occur only unnegated, or only negated, throughout the entire MODEL. LIST is assigned the same 8000-word storage area as is MATRIX.

Certain advantages are obtained in carrying out the Davis-Putnam test procedure if at all times the literals within each disjunction can be assumed to be in a fixed, known order. The handiest ordering to employ was found to be that obtained by pretending that each literal is really a 35-bit binary number with a plus or minus sign attached, and then ordering the literals within each disjunction in ascending absolute magnitude. The necessary sorting is done immediately prior to appending a new quantifier-free sentence to the end of MODEL. At this point is also performed the task

of deleting, within any one disjunction, all but one copy of any literal which happens to have one or more duplicates of itself as colleagues in the disjunction; and if a disjunction is found to have mutually contradictory literals within it (i.e., literals exactly alike except that one has a negation sign while the other does not), then the entire disjunction is deleted. (For such disjunctions are true no matter what truth values are assigned to their atomic constituents; hence a conjunction containing such a disjunction is inconsistent if and only if the remainder is.)

The flow of events within the Davis-Putnam test procedure is, again, essentially given by the statement of the theoretical process described earlier, which process the program carries out as there stated. The precise details of the actual steps which are executed are best ascertained from the symbolic FORTRAN program listed in the Appendix, where the liberal comments there provided tell the story plainly enough. The present overall remarks are intended as no more than a helpful guide and companion during a scrutiny of the FORTRAN program.

A word of running commentary on the way in which the instantiation process is done by GAMMA I. An array LSTUPE is constructed during the process; its kth entry tells which sentence of MATRIX was last instantiated by the kth individual constant. In assessing the question "Which sentence of MATRIX should next be instantiated, and with what individual constant?", GAMMA I exploits LSTUPE as follows: one selects the first entry in LSTUPE which does not "point to" the final sentence in MATRIX. This entry is then increased by 1, and the sentence then indicated is selected for instantiation. If the entry is the ith in the array LSTUPE, then the ith individual constant is used to do the instantiation of the selected sentence. However, if all the entries in the array LSTUPE point to the final sentence in MATRIX, then a new entry numerically equal to zero is added. This indicates that none of the sentences of MATRIX have yet been instantiated by the corresponding individual constant, but that the first sentence of MATRIX is just about to be.

Finally, a useful feature has been incorporated into GAMMA I to enable the user to have a picture of what is going on during internal processing. The register MQ is not required during the processing, either in the instantiation section of the program or in the Davis-Putnam section of the program. Two numbers are therefore displayed on the console MQ neons, and their behavior indicates how far GAMMA I has progressed with the problem at the time.

During the instantiation process, we display in the left half of MQ the number of sentences currently in MATRIX, and in the right half of MQ the number of quantifier-free sentences which have so far been added to MODEL.

During the Davis-Putnam process we display in the left half of MQ the current length (total number of words) of MODEL, and in the right half of MQ the number of iterations so far carried out of the Davis-Putnam procedure.

Depressing sense switch 3 at any time causes GAMMA I to report at the on-line printer the sentences which it is getting via the instantiation process.

Depressing sense switch 1 causes GAMMA I to report, during the Davis-Putnam tests, the successive appearances of MODEL as it is reduced at each iteration.

CHAPTER IV. SHORTCOMINGS OF GAMMA I. GAMMA II.

In the Introduction the remark was made that, owing to theoretical properties of the procedure H, the program GAMMA I has marked limitations as to what kinds of problem it is capable of handling in a reasonable span of time. Specifically, this is due to the fact that, for most "interesting" (and therefore sufficiently complex-structured) axiom sets, and, for a given interesting axiom set, for most interesting theorems deducible from it, the value of x guaranteed to exist (where x, it will be recalled, is the earliest step of the instantiation process at which a truth-functionally unsatisfiable sentence is obtained) by the theory is a sickeningly large number.

An example of this situation was met early in the testing of GAMMA I. The axiom system which was formalized was that for elementary abstract group theory. The axioms are three in number, and simple-looking ("o" is a binary functional variable):

$$(Ax)(Ay)(Ez)(x = zoy) \tag{1}$$

$$(Ax)(Ay)(Ez)(x = yoz) \tag{2}$$

$$(Ax)(Ay)(Az)(xo(yoz) = (xoy)oz) \quad . \tag{3}$$

From these axioms we sought to prove the theorem that an identity element exists; indeed, we contented ourselves with the weaker theorem that there exists a left identity element:

$$(Ex)(Az)(z = xoz) \quad . \tag{4}$$

Algebraically humble though this example be, it is not without some interest; the proof of it, while not difficult, is not trivial either. All in all, it was felt that it was a reasonable example of a nontrivial theorem which might be within the range of GAMMA I.

It is not. If appears, by an argument which will not be given here, that the earliest value of x to which GAMMA I would have to go in order to get a proof of (4) from [(1), (2), (3)] is at least 57^6, or about 2.10^{12}.

The irony of the situation is that, in the couple of trillion or so quantifier-free sentences which GAMMA I would have to generate by instantiation, only four are actually required to produce the requisite contradiction. It is these four, together with a mere handful of others, which any good student of modern algebra would select as a proof of (4) from [(1), (2), (3)].

Procedure H is in fact what one might call an exhaustion algorithm: the desired entity is, if it exists at all, certainly a member of an effectively

enumerable set; very well then, says an exhaustive algorithm, let us list the set, member by member, and see if the entity turns up. [Actually, the term 'algorithm' is a misnomer, since the process described will not terminate if (a) the enumeration does not terminate (i.e., the set is not finite) and (b) the desired entity is not in the set.]

The contrast is between methods calling for the examination of "all" possibilities, on the one hand, and methods which somehow select from the totality of possibilities a subset thereof which contains only the likelier possibilities. Clearly, the second category of methods embraces those distinguished by their employment of so-called strategies. At the very least, such methods are less uniform, more flexible, than exhaustive methods, and in some sense the flow of events ensuing when such a method is applied to a particular problem is very much a function of the specific properties of the particular problem.

Apart from its being a uniform, exhaustive method, procedure H also is formulated within a fairly spartan syntactical structure. It is in fact less easy to "spot" proofs, within the structure operated on by procedure H, than it is to spot them in the richer (though not more powerful) languages in which intuitive deductions are made, and then to transcribe them, or otherwise use them, to discover corresponding proofs within the more austere system.

The next program which is planned, GAMMA II, will embody some ideas, still somewhat in the formative stage, for selecting paths of instantiation on the basis of the particular structure of each problem, which should contain the desired contradiction if indeed there is one contained in the single, uniform path of instantiation followed by procedure H. This problem is much easier to handle within a system which explicitly contains functional signs (variables and constants) and which also contains the identity relation as part of the underlying logical machinery, with associated rules of deduction. It already is clear that, for instance, the group theory problem can be solved by a fairly simple generalization of the instantiation procedure, carried out within a language possessing function signs and the identity relation as a part of its deductive machinery. But it is not yet clear to what level of difficulty of theorems such a generalized and richer procedure will be able to penetrate.

APPENDIX. THE SYMBOLIC FORTRAN PROGRAM GAMMA I.

1. Annotated glossary of FORTRAN symbols occurring in GAMMA I.

The following complete alphabetically ordered list of FORTRAN symbols which occur in GAMMA I, together with the definitions and explanations attached to each, should facilitate the task of understanding the program listing. In the cases where synonyms occur, they were introduced into the program because the information they carried had sometimes to be treated as a FORTRAN integer (and as such had to be named by a symbol beginning with I, J, K, L, M, or N) and other times as a word of "Boolean" information, for which a symbol not beginning with I, J, K, L, M, or N is required.

ALIST	Synonym for LIST.
ATRIX	Synonym for MATRIX.
FIRST	Used to store the variable which is going to be instantiated; bit pattern is adjusted so as to be in alignment with first argument field of the literal.
GEORGE	Used during the construction of LIST in connection with the tagging of literals which are to be deleted from LIST before LIST is actually used.
I	General-purpose indexing variable used frequently in array-manipulation.
IMAX	The counter whose value is the highest individual constant which is used next to instantiate a universally quantified variable.
INSTA	Indicates the location of the matrix counter of the sentence being processed, during instantiation procedure.
INSTB	Indicates the location of the prefix counter of the sentence being processed, during instantiation procedure.
INSTC	Indicates the location of the last quantifier of the sentence being processed, during the instantiation procedure.
INSTD	Indicates the location of the first literal in the sentence being processed, during the instantiation procedure.
ITERAT	Contains the count of the number of iterations so far, in the current Davis-Putnam test.
J	General-purpose index variable, used frequently in array manipulation.
JLINE	Contains the number of sentences in MATRIX.
JUNK	A scratch-pad variable used in assembling word to be displayed in MQ.

K	General-purpose index variable, used frequently in array manipulation.
L	General-purpose index variable.
LA	Used as counter in the test to see how many universal quantifiers are in prefix of sentence about to be instantiated.
LAPSED	The number of minutes taken by a completed problem.
LAST	Indicates the last literal in a disjunction, during Davis-Putnam test.
LASTM	Indicates last literal in a disjunction, during the sorting operation on the disjunctions of a sentence about to be added to MODEL.
LATER	Time, in minutes elapsed since previous midnight, at which a problem was finished.
LEAD	The 120-word-maximum array containing BCD material of the comment accompanying the problem.
LEADMX	The number of BCD words in the comment.
LEAST	Indicates the first literal of a disjunction, during Davis-Putnam test.
LEASTM	Indicates first literal in a disjunction, during sorting procedure.
LENGTH	The number of words in the matrix of the current sentence; in instantiation procedure.
LFIRST	Synonym of FIRST.
LGEORGE	Synonym of GEORGE.
LINE	2000-word-maximum array whose entries give the locations of the beginnings of successive sentences in MATRIX.
LINES	Counter: number of quantifier-free sentences obtained so far.
LIST	8000-word-maximum array whose entries are literals which are to be eliminated from MODEL; in Davis-Putnam test.
LITS	Counter: number of literals in a disjunction; in Davis-Putnam test.
LN	Indicates which sentence of MATRIX is about to be instantiated next.
LNEG	Is equal to zero if all disjunctions in MODEL contain a negated literal; equals one otherwise.
LOST	Indicates last literal in a disjunction; in Davis-Putnam test.

LPHI	Highest individual constant used so far to instantiate an existentially quantified variable.
LPIVOT	The literal to be eliminated by step (6) of Davis-Putnam procedure.
LPOS	Is equal to zero if all disjunctions in MODEL contain an unnegated literal; equals one otherwise.
LSCOND	Synonym of SECOND.
LSTUPE	100-word-maximum array whose i^{th} entry gives which sentence was last instantiated by i^{th} individual constant.
LTEST	Counter: the number of times the Davis-Putnam test has been performed so far in the current problem.
LTHIRD	Synonym for THIRD.
LUNGTH	Indicates location of last literal in the matrix part of a sentence; during instantiation process.
LUST	Indicates the last literal of a disjunction; in Davis-Putnam testing process.
LVALUE	Synonym for VALUE.
LVBLE	Indicates the individual variable with respect to which instantiation will be done; during instantiation process.
LWORD1	Synonym for WORD1.
LWORD2	Synonym for WORD2.
LWORD3	Synonym for WORD3.
M	General purpose indexing variable.
MATEND	Gives the length of the array MATRIX, viz., the total number of words in the array.
MATRIX	The 8000-word-maximum array containing all the sentences which still have quantifiers left in their prefixes.
MAXK	Gives the length of the array LIST.
MODEL	The 18000-word-maximum array containing all the quantifier-free sentences.
MODEND	Gives the length of the array MODEL.
N	General-purpose indexing variable.
NAXT	Indicates the location of the counter immediately in front of a disjunction; in Davis-Putnam test process.

NEXT	Indicates the location of the counter immediately in front of a disjunction; in Davis-Putnam test-process.
NEXTM	Indicates the location of the counter immediately in front of a disjunction in the sorting process.
NOW	The time, in minutes elapsed since previous midnight, at which a problem is begun.
NOXT	Indicates the location of the counter immediately in front of a disjunction; in Davis-Putnam process.
ODEL	Synonym for MODEL.
SECOND	Used to store the variable which is going to be instantiated upon; bit pattern is adjusted so as to be in alignment with the second argument field of the literal.
THIRD	Used to store the variable which is going to be instantiated upon; bit pattern is adjusted so as to be in alignment with the argument field of the literal.
VALUE	The individual constant which is going to be used for instantiation, with bit pattern adjusted so as to be in alignment with first argument field of literal.
VALUE2	The individual constant which is going to be used for instantiation, with bit pattern adjusted so as to be in alignment with second argument field of literal.
VALUE3	The individual constant which is going to be used for instantiation, with bit pattern adjusted so as to be in alignment with third argument field of literal.
WORD1	The contents of the first argument field of a literal, context stripped away.
WORD2	The contents of the second argument field of a literal, context stripped away.
WORD3	The contents of the third argument field of a literal, context stripped away.
[X,Y,Z]	Three BCD words which state the date and time, whenever the subroutine MINUTE is called in. No use is actually made of these three variables beyond their function as the dummy variables in the calling line of the subroutine.

2. Listing of GAMMA I FORTRAN Symbolic program, with explanatory comments.

```
C*************************************************************************     0000
C*    GAMMA I.     A GENERAL THEOREM-PROVING PROGRAM.                     *     0001
C*************************************************************************     0002
      DIMENSION MODEL(18000),ODEL(18000),MATRIX(8000),ATRIX(8000),              0003
     1LINE(2000),LSTUPE(100),LIST(8000),ALIST(8000),LEAD(120)                   0004
      EQUIVALENCE (VALUE,LVALUE),(WORD1,LWORD1),(WORD2,LWORD2),                 0005
     1(WORD3,LWORD3),(MATRIX,ATRIX,LIST,ALIST),(MODEL,ODEL),                    0006
     2(FIRST,LFIRST),(SECOND,LSCOND),(THIRD,LTHIRD),(GEORGE,LGORGE)             0007
C*************************************************************************     0008
C*  OBTAIN INPUT FOR NEXT PROBLEM FROM CARD READER.                        *     0009
C*************************************************************************     0010
    1 READ 1200, MATEND,JLINE,LPHI,LEADMX                                       0011
      READ 1201,(LEAD(I),I=1,LEADMX)                                            0012
      READ 1001,(LINE(I),I=1,JLINE),(MATRIX(I),I=1,MATEND)                      0013
C*************************************************************************     0014
C*  FIND OUT WHAT TIME IT IS.                                              *     0015
C*************************************************************************     0016
      NOW = MINUTE(X,Y,Z)                                                       0017
C*************************************************************************     0018
C*  INITIALIZE THE COUNTERS WHICH WILL GROW, AND BE SURE ALL THE           *     0019
C*  SENSE LIGHTS ARE TURNED OFF.                                           *     0020
C*************************************************************************     0021
      SENSE LIGHT 0                                                             0022
    4 LSTUPE(1) = 0                                                             0023
      LINES = 0                                                                 0024
      LTEST = 0                                                                 0025
      MODEND = 0                                                                0026
      IMAX = 1                                                                  0027
C*************************************************************************     0028
C*  TRY INSTANTIATING A SENTENCE WITH 1.                                   *     0029
C*************************************************************************     0030
    5 INS = 1                                                                   0031
C*************************************************************************     0032
C*  SET UP THE LATEST PROGRESS REPORT IN THE MQ CONSOLE NEONS.             *     0033
C*************************************************************************     0034
S       CLA LINES                                                               0035
S       ARS 18                                                                  0036
S       ADD JLINE                                                               0037
S       STO M                                                                   0038
S       LDQ M                                                                   0039
C*************************************************************************     0040
C*  QUIT IF FORCED TO DO SO.                                               *     0041
C*************************************************************************     0042
      IF (SENSE SWITCH 2) 1900,6                                                0043
C*************************************************************************     0044
C*  CAN ANY SENTENCE BE INSTANTIATED WITH INS, WE ASK.                     *     0045
C*************************************************************************     0046
    6 IF (LSTUPE(INS) - JLINE) 9,7,9                                            0047
C*************************************************************************     0048
C*  IF NOT, STEP UP INS, AND THEN UNLESS INS IS NOW UP TO TESTING          *     0049
C*  SIZE, GO SEE WHETHER WE CAN INSTANTIATE A SENTENCE WITH THIS           *     0050
C*  HIGHER VALUE. IF INS IS UP TO TESTING SIZE, GO REPORT TO STATEMENT     *     0051
C*  8000, WHERE THE MATTER WILL BE MORE CLOSELY PURSUED.                   *     0052
C*************************************************************************     0053
```

```
    7 INS = INS + 1                                                     0054
      IF (INS - IMAX) 6,6,8000                                          0055
C*************************************************************************0056
C*  WE MAY PROCEED, EVIDENTLY.  THE TESTING CRISIS MUST BE OVER.       *0057
C*************************************************************************0058
    8 LSTUPE(INS) = 0                                                   0059
      IMAX = INS                                                        0060
      GO TO 6                                                           0061
C*************************************************************************0062
C*  WE CAN INSTANTIATE WITH INS. SET UP THE SENTENCE DUE FOR            *0063
C*  INSTANTIATION WITH INS.                                            *0064
C*************************************************************************0065
    9 LSTUPE(INS) = LSTUPE(INS) + 1                                     0066
      LN = LSTUPE(INS)                                                  0067
      INSTA = LINE(LN)                                                  0068
      INSTB = MATRIX(INSTA) + INSTA + 1                                 0069
      INSTC = MATRIX(INSTB) + INSTB                                     0070
      LVBLE = XABSF(MATRIX(INSTC))                                      0071
      INSTD = INSTB + 1                                                 0072
C*************************************************************************0073
C*  WAIT. IF WE INSTANTIATE THIS SENTENCE, WE WANT TO KNOW WHETHER      *0074
C*  WE WILL WIND UP WITH A QUANTIFIER-FREE SENTENCE OR NOT.  IF THIS    *0075
C*  SENTENCE HAS LESS THAN TWO UNIVERSAL QUANTIFIERS IN ITS PREFIX,     *0076
C*  THE RESULT OF INSTANTIATION WILL BE A QUANTIFIER-FREE SENTENCE.     *0077
C*  SO WE COUNT THE NUMBER OF UNIVERSAL QUANTIFIERS IN THE PREFIX....   *0078
C*************************************************************************0079
      LA = 0                                                            0080
      DO 11 K = INSTD, INSTC                                            0081
      IF (MATRIX(K)) 11, 10, 10                                         0082
   10 LA = LA + 1                                                       0083
   11 CONTINUE                                                          0084
C*************************************************************************0085
C*  ....AND IF THERE ARE LESS THAN TWO WE PREPARE TO ADD THE RESULT     *0086
C*  TO MODEL....                                                       *0087
C*************************************************************************0088
      IF (LA - 2) 14, 12, 12                                            0089
   12 M = MATEND                                                        0090
C*************************************************************************0091
C*  ...BUT IF THERE ARE TWO OR MORE WE PREPARE TO ADD THE RESULT TO     *0092
C*  MATRIX...                                                          *0093
C*************************************************************************0094
      DO 13 N = INSTA, INSTC                                            0095
      M = M + 1                                                         0096
   13 MATRIX(M) = MATRIX(N)                                             0097
C*************************************************************************0098
C*  ...AND DEPART FOR STATEMENT 38 WHERE THIS WILL BE DONE.            *0099
C*************************************************************************0100
      INSTB = MATEND + MATRIX(INSTA) + 2                                0101
      INSTA = MATEND + 1                                                0102
      LVALUE = INS                                                      0103
      GO TO 38                                                          0104
C*************************************************************************0105
C*  SET UP ON TO THE END OF MODEL THE SENTENCE TO BE INSTANTIATED.      *0106
C*************************************************************************0107
```

```
   14 M = MODEND                                                          0108
      LENGTH = MATRIX(INSTA)                                              0109
      INSTD = INSTA + 1                                                   0110
      DO 15 N = INSTD,INSTC                                               0111
      M = M + 1                                                           0112
   15 MODEL(M) = MATRIX(N)                                                0113
      INSTB = MODEND + MATRIX(INSTA) + 1                                  0114
      INSTA = MODEND + 1                                                  0115
   16 LVALUE = INS                                                        0116
C*************************************************************************0117
C*  GET RID OF ALL THE EXISTENTIAL QUANTIFIERS AS WELL AS THE          *  0118
C*  UNIVERSAL QUANTIFIER.                                              *  0119
C*************************************************************************0120
S  17  CLA LVBLE                                                         0121
S      STO FIRST                                                         0122
S      ARS 9                                                             0123
S     STO SECOND                                                         0124
S      ARS 9                                                             0125
S      STO THIRD                                                         0126
S      CLA LVALUE                                                        0127
S      ARS 9                                                             0128
S      STO VALUE2                                                        0129
S      ARS 9                                                             0130
S      STO VALUE3                                                        0131
B      FIRST = FIRST + 000400000000                                      0132
B      SECOND = SECOND + 000000400000                                    0133
B      THIRD = THIRD + 000000000400                                      0134
      DO 23 L = INSTA, INSTB                                             0135
B      WORD1 = ODEL(L) * 000777000000                                    0136
B      WORD2 = ODEL(L) * 000000777000                                    0137
B      WORD3 = ODEL(L) * 000000000777                                    0138
      IF (LWORD1 - LFIRST) 19, 18, 19                                    0139
B  18  ODEL(L) = ODEL(L) * 777000777777 + VALUE                          0140
   19 IF (LWORD2 - LSCOND) 21, 20, 21                                    0141
B  20 ODEL(L) = ODEL(L) * 777777000777 + VALUE2                          0142
   21 IF (LWORD3 - LTHIRD) 23, 22, 23                                    0143
B  22  ODEL(L) = ODEL(L) * 777777777000 + VALUE3                         0144
   23 CONTINUE                                                           0145
      MODEL(INSTB) = MODEL(INSTB) - 1                                    0146
      LASTM = MODEL(INSTB) + INSTB                                       0147
      IF (MODEL(INSTB)) 25, 25, 24                                       0148
   24 LVALUE = LPHI + 1                                                  0149
      LPHI = LVALUE                                                      0150
      LVBLE = XABSF(MODEL(LASTM))                                        0151
      GO TO 17                                                           0152
   25 LASTM = INSTA - 1                                                  0153
      LUNGTH = MODEND + LENGTH                                           0154
C*************************************************************************0155
C*  PRINT OUT THE RESULT IF REQUESTED.                                 *  0156
C*************************************************************************0157
      IF (SENSE SWITCH 3) 252, 251                                       0158
  252 PRINT 1004, LVALUE, (MODEL(N),N=INSTA,INSTB)                       0159
C*************************************************************************0160
C*  SORT EACH DISJUNCTION WITHIN THE NEW QUANTIFIER-FREE SENTENCE.     *  0161
```

```
C*************************************************************************     0162
  251 NEXTM = LASTM + 1                                                        0163
      IF (NEXTM - (MODEND + LENGTH + 1)) 26, 35, 35                            0164
   26 LEASTM = NEXTM + 1                                                       0165
      LASTM = MODEL(NEXTM) + NEXTM                                             0166
      K = LASTM - 1                                                            0167
      IF (MODEL(NEXTM) - 1) 251, 251, 27                                       0168
   27 DO 31 L = LEASTM, K                                                      0169
      IF(XABSF(MODEL(L)) - XABSF(MODEL(L + 1))) 31, 29, 28                     0170
   28 M = MODEL(L + 1)                                                         0171
      MODEL (L + 1) = MODEL(L)                                                 0172
      MODEL(L) = M                                                             0173
      SENSE LIGHT 1                                                            0174
      GO TO 31                                                                 0175
   29 IF (MODEL(L) + MODEL(L + 1)) 30, 32, 30                                  0176
   30 MODEL(L) = 0                                                             0177
      MODEL(NEXTM) = MODEL(NEXTM) - 1                                          0178
   31 CONTINUE                                                                 0179
      IF (SENSE LIGHT 1) 27, 251                                               0180
   32 IF(MODEL(L)) 33, 31, 33                                                  0181
C*************************************************************************     0182
C*  IF A DISJUNCTION IS LOGICALLY TRUE, DELETE IT.                       *     0183
C*************************************************************************     0184
   33 DO 34 L = NEXTM, LASTM                                                   0185
   34 MODEL(L) = 0                                                             0186
      GO TO 251                                                                0187
C*************************************************************************     0188
C*  PACK DOWN THE POSSIBLY DEPLETED SENTENCE.                            *     0189
C*************************************************************************     0190
   35 DO 37 N = INSTA,LUNGTH                                                   0191
      IF (MODEL(N)) 36, 37, 36                                                 0192
   36 MODEND = MODEND + 1                                                      0193
      MODEL(MODEND) = MODEL(N)                                                 0194
   37 CONTINUE                                                                 0195
C*************************************************************************     0196
C*  PRINT IT OUT IF REQUESTED.                                           *     0197
C*************************************************************************     0198
      IF (SENSE SWITCH 3) 371, 372                                             0199
  371 PRINT 1006, MODEND                                                       0200
      PRINT 1000, (MODEL(N), N=INSTA,MODEND)                                   0201
  372 CONTINUE                                                                 0202
C*************************************************************************     0203
C*  QUIT, IF WE ARE OUT OF CAPACITY FOR MODEL.  OTHERWISE, GO            *     0204
C*  INSTANTIATE THE NEXT SENTENCE.                                       *     0205
C*************************************************************************     0206
      LINES = LINES + 1                                                        0207
      IF (MODEND - 17950) 5, 1900, 1900                                        0208
C*************************************************************************     0209
C*  INSTANTIATE THE LEADING UNIVERSALLY QUANTIFIED VARIABLE, AND ANY     *     0210
C*  EXISTENTIAL QUANTIFIERS WHICH ARE THEREBY EXPOSED.                   *     0211
C*************************************************************************     0212
S  38  CLA LVRLE                                                              0213
S      STO FIRST                                                              0214
S      ARS 9                                                                  0215
```

```
S       STO SECOND                                                      0216
S       ARS 9                                                           0217
S       STO THIRD                                                       0218
S       CLA LVALUE                                                      0219
S       ARS 9                                                           0220
S       STO VALUE2                                                      0221
S       ARS 9                                                           0222
S       STO VALUE3                                                      0223
B       FIRST = FIRST + 000400000000                                    0224
B       SECOND = SECOND + 000000400000                                  0225
B       THIRD = THIRD + 000000000400                                    0226
        DO 44 L = INSTA,INSTB                                           0227
B       WORD1 = ATRIX(L) * 000777000000                                 0228
B       WORD2 = ATRIX(L) * 000000777000                                 0229
B       WORD3 = ATRIX(L) * 000000000777                                 0230
        IF (LWORD1 - LFIRST) 40, 39, 40                                 0231
B   39  ATRIX(L) = ATRIX(L) * 777000777777 + VALUE                      0232
    40  IF (LWORD2 - LSECND) 42, 41, 42                                 0233
B   41   ATRIX(L) = ATRIX(L) * 777777000777 + VALUE2                    0234
    42  IF (LWORD3 - LTHIRD) 44, 43, 44                                 0235
B   43  ATRIX(L) = ATRIX(L) * 777777777000 + VALUE3                     0236
    44  CONTINUE                                                        0237
        MATRIX(INSTB) = MATRIX(INSTB) - 1                               0238
        MATEND = MATRIX(INSTB) + INSTB                                  0239
        IF (MATRIX(MATEND)) 45,46,46                                    0240
    45  LVALUE = LPHI + 1                                               0241
        LPHI = LVALUE                                                   0242
        LVBLE=XABSF(MATRIX(MATEND))                                     0243
        IF (MATEND - 7750) 38, 1900,1900                                0244
    46  JLINE = JLINE + 1                                               0245
        LINE(JLINE) = INSTA                                             0246
C*****************************************************************     0247
C*  PRINT OUT THE RESULT, IF REQUESTED.                          *     0248
C*****************************************************************     0249
        IF (SENSE SWITCH 3) 461, 462                                    0250
  461   PRINT 1005, JLINE,LINE(JLINE),LVBLE,LVALUE                      0251
        PRINT 1000, (MATRIX(N),N=INSTA,MATEND)                          0252
  462   CONTINUE                                                        0253
C*****************************************************************     0254
C*  QUIT IF CAPACITY IS EXCEEDED. OTHERWISE GO INSTANTIATE THE NEXT    *     0255
C*  SENTENCE.                                                          *     0256
C*****************************************************************     0257
        IF (JLINE - 2000) 5, 1900,1900                                  0258
C*****************************************************************     0259
C*  STATEMENTS 80 THRU 240 + 3 COMPRISE THE DAVIS-PUTNAM TEST PROCESS. *     0260
C*****************************************************************     0261
    80  LTEST = LTEST + 1                                               0262
        ITERAT = 0                                                      0263
C*****************************************************************     0264
C*  WE RETURN HERE FOR EACH NEW ITERATION.  HERE WE FIRST INITIALIZE   *     0265
C*  THE NECESSARY COUNTERS AND TESTING VARIABLES.                      *     0266
C*****************************************************************     0267
 7115   LNEG = 0                                                        0268
        LPOS = 0                                                        0269
```

```
      ITERAT = ITERAT + 1                                                0270
      K = 0                                                              0271
      LAST = 0                                                           0272
C*********************************************************************  0273
C*  DISPLAY THE LATEST PROGRESS REPORT ON THE MQ CONSOLE NEONS.       *  0274
C*********************************************************************  0275
S       CLA ITERAT                                                       0276
S       ARS 18                                                           0277
S       ADD MODEND                                                       0278
S       STO JUNK                                                         0279
S       LDQ JUNK                                                         0280
C*********************************************************************  0281
C*  PRINT OUT MODEL AS IT NOW STANDS, IF REQUESTED TO DO SO.          *  0282
C*********************************************************************  0283
      IF (SENSE SWITCH 1) 3339, 110                                      0284
 3339 PRINT 3997                                                         0285
      PRINT 3001, MODEND                                                 0286
      PRINT 1001, (MODEL(I),I=1, MODEND)                                 0287
C*********************************************************************  0288
C*  MAKE A LIST OF ALL THE UNIT DISJUNCTIONS, I.E., THOSE CONTAINING  *  0289
C*  ONLY ONE LITERAL.                                                 *  0290
C*********************************************************************  0291
  110 NEXT = LAST + 1                                                    0292
      IF (NEXT -  MODEND) 120, 130, 130                                  0293
  120 LEAST = NEXT + 1                                                   0294
      LAST = MODEL(NEXT) + NEXT                                          0295
      IF (MODEL(NEXT) - 1)100,140,100                                    0296
  140 K = K + 1                                                          0297
      LIST(K) = MODEL(LEAST)                                             0298
C*********************************************************************  0299
C*  AS EACH NEW UNIT DISJUNCTION IS ADDED TO THE LIST, CHECK IT AGAINST*  0300
C*  THE EARLIER ONES TO SEE IF IT CONTRADICTS ANY OF THEM.  IF IT DOES,*  0301
C*  TERMINATE INCONSISTENT.                                           *  0302
C*********************************************************************  0303
      DO 50 N = 1, K                                                     0304
      IF (LIST(N) + LIST(K)) 50,250,50                                   0305
   50 CONTINUE                                                           0306
C*********************************************************************  0307
C*  SINCE WE ARE SCANNING ALL OF MODEL ANYWAY, LET US CHECK TO SEE IF  *  0308
C*  (A) EACH DISJUNCTION CONTAINS AN UNNEGATED LITERAL, OR (B) EACH    *  0309
C*  DISJUNCTION CONTAINS A NEGATED LITERAL.                           *  0310
C*********************************************************************  0311
  100 DO 106 J = LEAST, LAST                                             0312
      IF (MODEL(J)) 101, 106, 102                                        0313
  101 SENSE LIGHT 1                                                      0314
      GO TO 106                                                          0315
  102 SENSE LIGHT 2                                                      0316
  106 CONTINUE                                                           0317
      IF (SENSE LIGHT 1) 108, 107                                        0318
  107 LNEG = 1                                                           0319
  108 IF (SENSE LIGHT 2) 110, 109                                        0320
  109 LPOS = 1                                                           0321
      GO TO 110                                                          0322
C*********************************************************************  0323
```

```
C*   IF EITHER (A) OR (B) IS THE CASE, WE TERMINATE CONSISTENT.          *    0324
C*   OTHERWISE, IF WE HAVE ANY UNIT DISJUNCTIONS AT ALL ON THE LIST,     *    0325
C*   SET UP THE SITUATION TO DELETE THE FIRST OF THEM.                   *    0326
C*************************************************************************    0327
  130 IF (LPOS) 131, 260, 131                                                 0328
  131 IF (LNEG) 4997, 260, 4997                                               0329
 4997 IF (K)132,132,4998                                                      0330
 4998 MAXK = 1                                                                0331
      GO TO 1412                                                              0332
C*************************************************************************    0333
C*   IF WE HAVE NO UNIT DISJUNCTIONSLET US MAKE A LIST OF PURE LITERALS.*    0334
C*   A LITERAL IS SAID TO BE PURE IF EITHER ALL ITS OCCURRENCES IN THE   *    0335
C*   MODEL ARE UNNEGATED OR ALL ARE NEGATED.                             *    0336
C*************************************************************************    0337
  132 MAXK = 0                                                                0338
      K = 0                                                                   0339
      LAST = 0                                                                0340
C*************************************************************************    0341
C*   TURN ON THE SIGN BIT IN MQ.                                         *    0342
C*************************************************************************    0343
S       STQ JUNK                                                            0344
S       CLA JUNK                                                            0345
S       SSM                                                                 0346
S       STO JUNK                                                            0347
S       LDQ JUNK                                                            0348
C*************************************************************************    0349
C*   IF WE FIND THAT A LITERAL IS MIXED, I.E., NOT PURE, WE OMIT IT      *    0350
C*   FROM THE LIST. BUT WE PUT A LITERAL ON THE LIST AS BEING PURE,      *    0351
C*   UNTIL IT IS PROVED TO BE MIXED.  WHEN WE FIND THAT A LITERAL IS     *    0352
C*   MIXED, WE TAG ITS OCCURRENCE IN LIST (BY PUTTING A BINARY ONE IN    *    0353
C*   IN THE SECOND BIT POSITION) SO AS TO KNOW IT MUST BE LATER          *    0354
C*   REMOVED FROM THE LIST.                                              *    0355
C*************************************************************************    0356
 8300 NEXT = LAST + 1                                                         0357
      IF (NEXT - MODEND) 8301, 8320, 8320                                     0358
 8301 LEAST = NEXT + 1                                                        0359
      LAST = MODEL(NEXT) + NEXT                                               0360
      DO 8303 I = LEAST, LAST                                                 0361
      DO 8302 J = 1, MAXK                                                     0362
B      GEORGE = ALIST(J)*577777777777                                         0363
      IF (LGORGE - MODEL(I)) 8305,8303,8305                                   0364
 8305 IF (LGORGE + MODEL(I)) 8302,8304,8302                                   0365
B8304  ALIST(J) = ALIST(J) + 200000000000                                     0366
      GO TO 8303                                                              0367
 8302 CONTINUE                                                                0368
      MAXK = MAXK + 1                                                         0369
      LIST(MAXK) = MODEL(I)                                                   0370
 8303 CONTINUE                                                                0371
      GO TO 8300                                                              0372
C*************************************************************************    0373
C*   DELETE FROM THE LIST ALL THE LITERALS WHICH ARE TAGGED.             *    0374
C*************************************************************************    0375
 8320 DO 8325 I=1,MAXK                                                        0376
B      GEORGE = ALIST(I)*200000000000                                         0377
```

```
      IF (LGORGE) 8325,8325,8324                                        0378
 8324 LIST(I) = 0                                                       0379
 8325 CONTINUE                                                          0380
C*********************************************************************  0381
C*  THEN PACK DOWN THE LIST TO REMOVE ANY GAPS.                      *  0382
C*********************************************************************  0383
      J = 0                                                             0384
      DO 8330 I = 1, MAXK                                               0385
      IF (LIST(I)) 8327, 8330, 8327                                     0386
 8327 J = J + 1                                                         0387
      LIST(J) = LIST(I)                                                 0388
 8330 CONTINUE                                                          0389
C*********************************************************************  0390
C*  IF WE THEREBY WIND UP WITH AN EMPTY LIST (SO THAT ALL THE LITERALS *  0391
C*  IN MODEL ARE MIXED) WE GO TO THE BLASTING PROCEDURE. OTHERWISE WE  *  0392
C*  GO TO THE DELETION PROCEDURE.                                    *  0393
C*********************************************************************  0394
      MAXK = J                                                          0395
      IF (MAXK) 160, 160, 1412                                          0396
C*********************************************************************  0397
C*  INITIALIZE, PREPARATORY TO SCANNING EACH DISJUNCTION IN MODEL TO   *  0398
C*  SEE WHETHER LITERALS OCCUR IN THEM WHICH ARE ON LIST, OR WHICH ARE *  0399
C*  NEGATIONS OF LITERALS WHICH ARE ON LIST.                         *  0400
C*********************************************************************  0401
 1412 LAST = 0                                                          0402
C*********************************************************************  0403
C*  TURN OFF THE SIGN BIT IN MQ                                      *  0404
C*********************************************************************  0405
S       STQ JUNK                                                        0406
S       CLA JUNK                                                        0407
S       SSP                                                             0408
S       STO JUNK                                                        0409
S       LDQ JUNK                                                        0410
C*********************************************************************  0411
C*  PREPARE TO SCAN THE NEXT DISJUNCTION.                            *  0412
C*********************************************************************  0413
 1411 NEXT = LAST + 1                                                   0414
      IF (NEXT - MODEND) 142, 150, 150                                  0415
  142 LEAST = NEXT + 1                                                  0416
      LAST = MODEL(NEXT) + NEXT                                         0417
C*********************************************************************  0418
C*  IF THE DISJUNCTION CONTAINS THE NEGATION OF A LITERAL WHICH IS ON  *  0419
C*  LIST, DELETE THAT NEGATION FROM THE DISJUNCTION.                 *  0420
C*********************************************************************  0421
      DO 143 J = LEAST, LAST                                            0422
      DO 144 K = 1, MAXK                                                0423
      IF (MODEL(J) - LIST(K)) 145, 147, 145                             0424
  145 IF (MODEL(J) + LIST(K)) 144, 146, 144                             0425
  146 MODEL(J) = 0                                                      0426
      MODEL(NEXT) = MODEL(NEXT) - 1                                     0427
      GO TO 143                                                         0428
C*********************************************************************  0429
C*  IF THE DISJUNCTION CONTAINS A LITERAL WHICH IS ON LIST, DELETE THE *  0430
C*  ENTIRE DISJUNCTION.                                              *  0431
```

```
C*************************************************************************     0432
  147 DO 148 M = NEXT, LAST                                                     0433
  148 MODEL(M) = 0                                                              0434
      GO TO 1411                                                                0435
  144 CONTINUE                                                                  0436
  143 CONTINUE                                                                  0437
      GO TO 1411                                                                0438
C*************************************************************************     0439
C*  THE DELETION PROCESS BEING OVER, PACK DOWN MODEL SO AS TO CLOSE UP  *      0440
C*  ANY GAPS.                                                           *      0441
C*************************************************************************     0442
  150 J = 0                                                                     0443
      DO 154 I = 1, MODEND                                                      0444
      IF (MODEL(I)) 153, 154, 153                                               0445
  153 J = J + 1                                                                 0446
      MODEL(J) = MODEL(I)                                                       0447
  154 CONTINUE                                                                  0448
      MODEND = J                                                                0449
C*************************************************************************     0450
C*  IF THE ENTIRE MODEL HAS VANISHED, TERMINATE CONSISTENT, OTHERWISE   *      0451
C*  RETURN FOR ANOTHER DAVIS-PUTNAM ITERATION.                          *      0452
C*************************************************************************     0453
      IF (MODEND) 260,260,7115                                                  0454
C*************************************************************************     0455
C*  IT IS NECESSARY TO APPLY STEP (6) OF THE DAVIS-PUTNAM PROCESS, AND  *      0456
C*  BLAST OUT A LITERAL FROM MODEL. WE CHOOSE THE FIRST LITERAL IN THE  *      0457
C*  FIRST DISJUNCTION AS THE PIVOTAL LITERAL TO BE BLASTED OUT.         *      0458
C*************************************************************************     0459
  160 LPIVOT = MODEL(2)                                                         0460
      K = MODEND                                                                0461
      LAST = 0                                                                  0462
C*************************************************************************     0463
C*  SEARCH FOR THE NEXT DISJUNCTION WHICH CONTAINS AN UNNEGATED         *      0464
C*  OCCURRENCE OF THE PIVOTAL LITERAL.                                  *      0465
C*************************************************************************     0466
  200 NEXT = LAST + 1                                                           0467
      IF (NEXT - MODEND) 208, 210, 210                                          0468
  208 LEAST = NEXT + 1                                                          0469
      LAST = MODEL(NEXT) + NEXT                                                 0470
      DO 201 M = LEAST, LAST                                                    0471
      IF (MODEL(M) - LPIVOT) 201, 202, 201                                      0472
  201 CONTINUE                                                                  0473
      GO TO 200                                                                 0474
C*************************************************************************     0475
C*  HAVING FOUND SUCH A DISJUNCTION (CALL IT A), PREPARE TO FIND        *      0476
C*  EACH DISJUNCTION IN MODEL WHICH CONTAINS AN OCCURRENCE OF THE       *      0477
C*  NEGATION OF THE PIVOTAL LITERAL.                                    *      0478
C*************************************************************************     0479
  202 LOST = 0                                                                  0480
C*************************************************************************     0481
C*  SEARCH FOR THE NEXT DISJUNCTION WHICH CONTAINS AN OCCURRENCE OF     *      0482
C*  THE PIVOTAL LITERAL.                                                *      0483
C*************************************************************************     0484
  203 NAXT = LOST + 1                                                           0485
```

```
      IF (NAXT - MODEND) 207, 200, 200                                  0486
  207 LUST = NAXT + 1                                                   0487
      LOST = MODEL(NAXT) + NAXT                                         0488
      DO 204 N = LUST, LOST                                             0489
      IF (MODEL(N) + LPIVOT) 204, 205, 204                              0490
  204 CONTINUE                                                          0491
      GO TO 203                                                         0492
C*************************************************************************0493
C*  HAVING FOUND THE NEXT SUCH DISJUNCTION, (CALL IT B) WE NOW MERGE   *  0494
C*  A WITH B, DELETING ANY DUPLICATIONS OF LITERALS IN THE RESULT,     *  0495
C*  AND DELETING THE WHOLE DISJUNCTION IF TWO CONTRADICTORY LITERALS   *  0496
C*  SHOW UP, ONE FROM A AND THE OTHER FROM B.  IF THE LENGTH OF MODEL  *  0497
C*  EXCEEDS CAPACITY DURING THIS PROCESS, TERMINATE BY FLEEING TO      *  0498
C*  STATEMENT NUMBER 1900.  STATEMENTS 174 THRU 231 INVOLVE INTRICATE  *  0499
C*  HOUSEKEEPING CHORES CONNECTED WITH THIS MERGING OPERATION.         *  0500
C*************************************************************************0501
  205 NOXT = K + 1                                                      0502
      K = NOXT                                                          0503
      I = LEAST                                                         0504
      J = LUST                                                          0505
      LITS = 0                                                          0506
  191 IF (I - LAST) 190, 190, 193                                       0507
  190 IF (J - LOST) 169, 169, 195                                       0508
  169 IF (XABSF(MODEL(I)) - XABSF(MODEL(J))) 1692, 170, 171             0509
  170 IF (MODEL(I) + MODEL(J)) 172, 173, 172                            0510
  173 IF (MODEL(I) - LPIVOT)  175, 174, 175                             0511
  175 K = NOXT - 1                                                      0512
      GO TO 203                                                         0513
  174 I = I + 1                                                         0514
      J = J + 1                                                         0515
      GO TO 191                                                         0516
  172 J = J + 1                                                         0517
 1692 K = K + 1                                                         0518
      IF (K - 18000) 1691, 1691, 1900                                   0519
 1691 LITS = LITS + 1                                                   0520
      MODEL(K) = MODEL(I)                                               0521
      I = I + 1                                                         0522
      GO TO 191                                                         0523
  171 K = K + 1                                                         0524
      IF (K - 18000) 1711, 1711, 1900                                   0525
 1711 LITS = LITS + 1                                                   0526
      MODEL(K) = MODEL(J)                                               0527
      J = J + 1                                                         0528
      GO TO 191                                                         0529
  193 IF (J - LOST) 1933, 1933, 197                                     0530
 1933 DO 194 N = J, LOST                                                0531
      K = K + 1                                                         0532
      IF (K - 18000) 1931, 1931, 1900                                   0533
 1931 LITS = LITS + 1                                                   0534
  194 MODEL(K) = MODEL(N)                                               0535
      GO TO 197                                                         0536
  195 IF (I - LAST) 1953, 1953, 197                                     0537
 1953 DO 196 M = I, LAST                                                0538
      K = K + 1                                                         0539
```

```
      IF (K - 18000) 196, 196, 1900                                      0540
  196 MODEL(K) = MODEL(M)                                                0541
  197 MODEL(NOXT) = LITS                                                 0542
      M = 0                                                              0543
  198 N = M + 1                                                          0544
      IF (N - MODEND) 199, 203, 203                                      0545
  199 L = N + 1                                                          0546
      M = MODEL(N) + N                                                   0547
      IF (MODEL(N) - MODEL(NOXT)) 230, 198, 230                          0548
  230 J = NOXT + 1                                                       0549
      DO 231 I = L,M                                                     0550
      IF (MODEL(I) - MODEL(J)) 198, 232, 198                             0551
  232 J = J + 1                                                          0552
  231 CONTINUE                                                           0553
      GO TO 175                                                          0554
C*************************************************************************0555
C*  SET UP THE NEW DISJUNCTIONS SO THAT ONE PASS THROUGH THE DELETION  *  0556
C*  PROCESS WILL ELIMINATE EACH DISJUNCTION WHICH, IN THE OLD MODEL,   *  0557
C*  CONTAINED EITHER THE PIVOTAL LITERAL OR ITS NEGATION.  THEN        *  0558
C*  PROCEED TO THE DELETION PROCESS.                                   *  0559
C*************************************************************************0560
  210 MODEND = K                                                         0561
      DO 240 I = 1, MODEND                                               0562
      IF (MODEL(I) + LPIVOT) 240, 241, 240                               0563
  241 MODEL(I) = LPIVOT                                                  0564
  240 CONTINUE                                                           0565
      LIST(1) = LPIVOT                                                   0566
      MAXK = 1                                                           0567
      GO TO 1412                                                         0568
C*************************************************************************0569
C*  WE HAVE COME HERE BECAUSE THE INSTANTIATION PROCEDURE HAS REACHED  *  0570
C*  A DAVIS-PUTNAM TESING POINT. IF WE ARE IN MANUAL MODE, WITH NO     *  0571
C*  REQUEST FOR A TEST, WE RETURN TO THE INSTANTIATION PROCESS.        *  0572
C*  OTHERWISE WE FIRST DETERMINE WHETHER THE UPCOMING TEST WILL BE     *  0573
C*  THE LAST, SAVE MODEL AND MATRIX, AND SEND CONTROL TO THE DAVIS-    *  0574
C*  PUTNAM TESTING PROCESS.                                            *  0575
C*************************************************************************0576
 8000 IF (SENSE SWITCH 4) 8004,8003                                      0577
 8004 IF (SENSE SWITCH 5)8003,8                                          0578
 8003 WRITETAPE3,(MODEND,(MODEL(I),I=1,MODEND),(MATRIX(I),I=1,MATEND))   0579
      IF (INS - LPHI) 80,80,8001                                         0580
 8001 SENSE LIGHT 4                                                      0581
      GO TO 80                                                           0582
C*************************************************************************0583
C*  THE PROBLEM IS CONSISTENT.  OUTPUT TO ON-LINE PRINTER AND TAPE 2,  *  0584
C*  AND RETURN FOR THE NEXT PROBLEM.                                   *  0585
C*************************************************************************0586
  260 BACKSPACE 3                                                        0587
       READTAPE3,(MODEND,(MODEL(I),I=1,MODEND),(MATRIX(I),I=1,MATEND))   0588
      IF (SENSE LIGHT 4) 8002,8                                          0589
 8002 LATER = MINUTE(X,Y,Z)                                              0590
      LAPSED = LATER - NOW                                               0591
      PRINT 1202,(LEAD(I),I=1,LEADMX)                                    0592
      PRINT 1210                                                         0593
```

```
      PRINT 1204,LAPSED                                                  0594
      WRITE OUTPUT TAPE 2,1202,(LEAD(I),I=1,LEADMX)                      0595
      WRITE OUTPUT TAPE 2,1210                                           0596
      WRITE OUTPUT TAPE 2, 1204,LAPSED,ITERAT,LTEST                      0597
      WRITE OUTPUT TAPE 2,1000,(MODEL(I),I=1,MODEND)                     0598
      GO TO 1                                                            0599
C*********************************************************************  0600
C*  THE PROBLEM IS INCONSISTENT.  OUTPUT TO THE ON-LINE PRINTER AND   *  0601
C*  TAPE  2, AND RETURN FOR NEXT PROBLEM.                             *  0602
C*********************************************************************  0603
  250 LATER = MINUTE(X,Y,Z)                                              0604
      LAPSED = LATER - NOW                                               0605
      PRINT 1202, (LEAD(I),I=1,LEADMX)                                   0606
      PRINT 1203                                                         0607
      PRINT 1204, LAPSED                                                 0608
      WRITE OUTPUT TAPE 2,1202,(LEAD(I),I=1,LEADMX)                      0609
      WRITE OUTPUT TAPE 2,1203                                           0610
  291 WRITE OUTPUT TAPE 2,1204,LAPSED,ITERAT,LTEST                       0611
      BACKSPACE 3                                                        0612
      READ TAPE 3, (MODEND,(MODEL(I),I=1,MODEND))                        0613
      WRITE OUTPUT TAPE 2,1000,(MODEL(I),I=1,MODEND)                     0614
      GO TO 1                                                            0615
C*********************************************************************  0616
C*  CAPACITY HAS BEEN EXCEEDED.  OUTPUT REMARK TO ON-LINE PRINTER, AND *  0617
C*  RETURN FOR NEXT PROBLEM.                                          *  0618
C*********************************************************************  0619
 1900 LATER = MINUTE(X,Y,Z)                                              0620
      LAPSED = LATER - NOW                                               0621
      PRINT 1202,(LEAD(I),I=1,LEADMX)                                    0622
      PRINT 1205,LAPSED                                                  0623
      GO TO 1                                                            0624
 1000 FORMAT (7014)                                                      0625
 1001 FORMAT (5014)                                                      0626
 1002 FORMAT (7014)                                                      0627
 1003 FORMAT (7014)                                                      0628
 1004 FORMAT (6H025,  , 8014)                                            0629
 1005 FORMAT (6H046,  , 4I6)                                             0630
 1006 FORMAT (6H037,  , I6)                                              0631
 1200 FORMAT (12I6)                                                      0632
 1201 FORMAT (12A6)                                                      0633
 1202 FORMAT (1H0, 19A6)                                                 0634
 1203 FORMAT (14H0INCONSISTENT.)                                         0635
 1204 FORMAT (28H0TIME ELAPSED, IN MINUTES = , 3I4)                      0636
 1205 FORMAT (20H0FORCED STOP AFTER  , I4,31H  MINUTES, WITH NO PROOF FO 0637
     1UND.)                                                              0638
 1210 FORMAT (12H0CONSISTENT.)                                           0639
 3001 FORMAT (1H0,I5)                                                    0640
 3997 FORMAT (8H0TF TEST)                                                0641
      END (0,1,0,0,1)                                                    0642
```