The gallant tailor: seven (flies) in one blow.
IBM 704: 220 theorems (in the propositional calculus) in three minutes.

Hao Wang

Toward Mechanical Mathematics

Abstract: Results are reported here of a rather successful attempt at proving all theorems, totalling near 400, of *Principia Mathematica* which are strictly in the realm of logic, viz., the restricted predicate calculus with equality. A number of other problems of the same type are discussed. It is suggested that the time is ripe for a new branch of applied logic which may be called "inferential" analysis, which treats proofs as numerical analysis does calculations. This discipline seems capable, in the not too remote future, of leading to machine proofs of difficult new theorems. An easier preparatory task is to use machines to formalize proofs of known theorems. This line of work may also lead to mechanical checks of new mathematical results comparable to the debugging of a program.

Introduction

If we compare calculating with proving, four differences strike the eye: (1) Calculations deal with numbers; proofs, with propositions. (2) Rules of calculation are generally more exact than rules of proof. (3) Procedures of calculation are usually terminating (decidable, recursive) or can be made so by fairly well-developed methods of approximation. Procedures of proof, however, are often nonterminating (undecidable or nonrecursive, though recursively enumerable), indeed incomplete in the case of number theory or set theory, and we do not have a clear conception of approximate methods in theorem-proving. (4) We possess efficient calculating procedures, while with proofs it frequently happens that even in a decidable theory, the decision method is not practically feasible. Although short-cuts are the exception in calculations, they seem to be the rule with proofs in so far as intuition, insight, experience, and other vague and not easily imitable principles are applied. Since the proof procedures are so complex or lengthy, we simply cannot manage unless we somehow discover peculiar connections in each particular case.

Undoubtedly it is such differences that have discouraged responsible scientists from embarking on the enterprise of mechanizing significant portions of the activity of mathematical research. The writer, however, feels that the nature and the dimension of the difficulties have been misrepresented through uncontrolled speculation and exaggerated because of a lack of appreciation of the combined capabilities of mathematical logic and calculating machines.

Of the four differences, the first is taken care of either by quoting Gödel representations of expressions or by recalling the familiar fact that alphabetic information can be handled on numerical (digital) machines. The second difference has largely been removed by the achievements of mathematical logic in formalization during the past eighty years or so. Item (3) is not a difference that is essential to the task of proving theorems by machine. The immediate concern is not so much theoretical possibility as practical feasibility. Quite often a particular question in an undecidable domain is settled more easily than one in a decidable region, even me-

chanically. We do not and cannot set out to settle all questions of a given domain, decidable or not, when, as is usually the case, the domain includes infinitely many particular questions. In addition, it is not widely realized how large the decidable subdomains of an undecidable domain (e.g., the predicate calculus) are. Moreover, even in an undecidable area, the question of finding a proof for a proposition known to be a theorem, or formalizing a sketch into a detailed proof, is decidable theoretically. The state of affairs arising from the Gödel incompleteness is even less relevant to the sort of work envisaged here. The purpose here is at most to prove mathematical theorems of the usual kind, e.g., as exemplified by treatises on number theory, yet not a single "garden-variety" theorem of number theory has been found unprovable in the current axiom system of number theory. The concept of approximate proofs, though undeniably of a kind other than approximations in numerical calculations, is not incapable of more exact formulation in terms of, say, sketches of and gradual improvements toward a correct proof.

The last difference is perhaps the most fundamental. It is, however, easy to exaggerate the degree of complexity which is necessary, partly because abstract estimates are hardly realistic, partly because so far little attention has been paid to the question of choosing more efficient alternative procedures. There will soon be occasion to give illustrations to these two causes of exaggeration. The problem of introducing intuition and experience into machines is a bit slippery. Suffice it to say for the moment, however, that we have not realized that much of our basic strategies in searching for proofs is mechanizable, because we had little reason to be articulate on such matters until large, fast machines became available. We are in fact faced with a challenge to devise methods of buying originality with plodding, now that we are in possession of slaves which are such persistent plodders. In the more advanced areas of mathematics, we are not likely to succeed in making the machine imitate the man entirely. Instead of being discouraged by this, however, one should view it as a forceful reason for experimenting with mechanical mathematics. The human inability to command precisely any great mass of details sets an intrinsic limitation on the kind of thing that is done in mathematics and the manner in which it is done. The superiority of machines in this respect indicates that machines, while following the broad outline of paths drawn up by man, might yield surprising new results by making many new turns which man is not accustomed to taking.

It seems, therefore, that the general domain of algorithmic analysis can now begin to be enriched by the inclusion of inferential analysis as a younger companion to the fairly well established but still rapidly developing leg of numerical analysis.

The writer began to speculate on such possibilities in 1953, when he first came into contact with calculating machines. These vague thoughts were afterwards appended to a paper on Turing machines.[1] Undoubtedly many people have given thought to such questions. As far as the writer is aware, works more or less in this area include Burks-Warren-Wright,[2] Collins,[3] Davis,[4] Newell-Shaw-Simon, Gelernter.[5] Of these, the most extensively explained and most widely known is perhaps that of Newell-Shaw-Simon, a series of reports and articles[6] written since 1956. Their work is also most immediately relevant to the results to be reported in this paper. It will, therefore, not be out of place if we indicate the basic differences in the respective approaches and the specific advances beyond their work.

They report[7] that their program LT on JOHNNIAC was given the task of proving the first 52 theorems of *Principia Mathematica* of Whitehead and Russell: "Of the 52 theorems, proofs were found for a total 38. . . . In 14 cases LT failed to find a proof. Most of these unsuccessful attempts were terminated by time or space limitations. One of these 14 theorems we know LT cannot prove, and one other we believe it cannot prove." They also give as examples that a proof for *2.45 was found in 12 minutes and a report of failure to prove *2.31 was given after 23 minutes.

The writer wrote three programs last summer on an IBM 704. The first program provides a proof-decision procedure for the propositional calculus which prints out a proof or a disproof according as the given proposition is a theorem or not. It was found that the whole list of over 200 theorems of the first five chapters of *Principia Mathematica* were proved within about 37 minutes, and 12/13 of the time is used for read-in and print-out, so that the actual proving time for over 200 theorems was less than 3 minutes. The 52 theorems chosen by Newell-Shaw-Simon are among the easier ones and were proved in less than 5 minutes (or less than ½ minute if not counting input-output time). In particular, *2.45 was proved in about 3 seconds and *2.31 in about 6 seconds. The proofs for these two theorems and some more complex proofs are reproduced in Appendix I as they were printed out on the machine.

The other two programs deal with problems not considered by Newell-Shaw-Simon in their published works. The second program instructs the machine to form propositions of the propositional calculus from basic symbols and select nontrivial theorems. The speed was such that about 14,000 propositions were formed and tested in one hour, storing on tape about 1000 theorems. The result was disappointing in so far as too few theorems were excluded as being trivial, because the principles of triviality actually included in the program were too crude.

The third program was meant as part of a larger program for the whole predicate calculus with equality which the writer did not have time to complete during 1958. The predicate calculus with equality takes up the next five chapters of *Principia Mathematica* with a total of over 150 theorems. The third program as it stands can find and print out proofs for about 85% of these theorems in about an hour. The writer believes that slight modifications in the program will enable the ma-

chine to prove all these theorems within 80 minutes or so. The full program, as envisaged, will be needed only when we come to propositions of the predicate calculus which are much harder to prove or disprove than those in this part of *Principia Mathematica*.

It will naturally be objected that the comparison with the program of Newell-Shaw-Simon is unfair, since the approaches are basically different. The writer realizes this but cannot help feeling, all the same, that the comparison reveals a fundamental inadequacy in their approach. There is no need to kill a chicken with a butcher's knife. Yet the net impression is that Newell-Shaw-Simon failed even to kill the chicken with their butcher's knife. They do not wish to use standard algorithms such as the method of truth tables,[8] because "these procedures do not produce a proof in the meaning of Whitehead and Russell. One can invent 'automatic' procedures for producing proofs, and we will look at one briefly later, but these turn out to require computing times of the orders of thousands of years for the proof of *2.45." It is, however, hard to see why the proof of *2.45 produced by the algorithms to be described in this paper is less acceptable as a proof, yet the computing time for proving *2.45 is less than ¼ second by this algorithm. To argue the superiority of "heuristic" over algorithmic methods by choosing a particularly inefficient algorithm seems hardly just.

The word "heuristic" is said to be synonymous with "the art of discovery," yet often seems to mean nothing else than a partial method which offers no guarantees of solving a given problem. This ambiguity endows the word with some emotive meaning that could be misleading in further scientific endeavors. The familiar and less inspiring word "strategy" might fare better.

While the discussions by Newell-Shaw-Simon are highly suggestive, the writer prefers to avoid hypothetical considerations when possible. Even though one could illustrate how much more effective partial strategies can be if we had only a very dreadful general algorithm, it would appear desirable to postpone such considerations till we encounter a more realistic case where there is no general algorithm or no efficient general algorithm, e.g., in the whole predicate calculus or in number theory. As the interest is presumably in seeing how well a particular procedure can enable us to prove theorems on a machine, it would seem preferable to spend more effort on choosing the more efficient methods rather than on enunciating more or less familiar generalities. And it is felt that an emphasis on mathematical logic is unavoidable, because it is just as essential in this area as numerical analysis is for solving large sets of simultaneous numerical equations.

The logical methods used in this paper are along the general line of cut-free formalisms of the predicate calculus initiated by Herbrand[9] and Gentzen.[10] Ideas of Hilbert-Bernays,[11] Dreben,[12] Beth,[13] Hintikka,[14] Schütte[15] and many others on these formulations, as well as some from standard decision methods for subdomains of the predicate calculus presented by Church[16] and Quine,[17] are borrowed. The special formulations actually used seem to contain a few minor new features which facilitate the use on machines. Roughly speaking, a complete proof procedure for the predicate calculus with equality is given which becomes a proof-decision procedure when the proposition to be proved or disproved falls within the domain of the propositional calculus or that of the "*AE* predicate calculus"* which includes the monadic predicate calculus as a subdomain.

The treatment of the predicate calculus by Herbrand and Gentzen enables us to get rid of every "Umweg" (cut or *modus ponens*) so that we obtain a cut-free calculus in which, roughly speaking, for every proof each of the steps is no more complex than the conclusion. This naturally suggests that, given any formula in the predicate calculus, we can examine all the less complex formulae and decide whether it is provable. The reason that this does not yield a decision procedure for the whole predicate calculus is a rule of contraction which enables us to get rid of a repetition of the same formula. As a result, in searching for a proof or a disproof, we may fail in some case because we can get no proof, no matter how many repetitions we introduce. In such a case, the procedure can never come to an end, although we do not know this at any finite stage. While this situation does not preclude completeness, it does exclude a decision procedure.

Now if we are interested in decidable subdomains of the predicate calculus, we can usually give suitable reformulations in which the rule of contraction no longer occurs. A particularly simple case is the propositional calculus. Here we can get a simple system which is both a complete proof procedure and a complete decision procedure. The completeness receives a very direct proof, and as a decision procedure it has an advantage over usual procedures in that if the proposition tested is provable, we obtain a proof of it directly from the test. This procedure is coded in Program I. Moreover, it is possible to extend the system for the propositional calculus to get a proof-decision procedure for the *AE* predicate calculus, which has the remarkable feature that in searching for a proof for a given proposition in the "miniscope" form, we almost never need to introduce any premise which is longer than its conclusion. This procedure is coded in Program III.

A rather surprising discovery, which tends to indicate our general ignorance of the extensive range of decidable subdomains, is the absence of any theorem of the predicate calculus in *Principia* which does not fall within the simple decidable subdomain of the *AE* predicate calculus. More exactly, there is a systematic procedure of separating variables to bring a proposition into the "miniscope" form, a term to be explained below. Since this procedure can be easily carried out by hand or by machine for these particular theorems, every theorem in the predicate calculus part of *Principia* can then be proved by the fairly simple Program III.

*Those propositions which can be transformed into a form in which no existential quantifier governs any universal quantifier.

Originally the writer's interest was in formalizing proofs in more advanced domains, such as number theory and differential calculus. It soon became clear that for this purpose a pretty thorough mechanization of the underlying logic is a necessary preliminary step. Now that this part is near completion, the writer will discuss in the concluding part of this paper some further possibilities that he considers to be not too remote.

Work for this paper was done at the Poughkeepsie IBM Research Laboratory in the summer of 1958. The writer much appreciates the satisfactory working conditions, especially the stimulating suggestions of friends in the Laboratory and the easy access to a good calculating machine.

The propositional calculus (System *P*)

Since we are concerned with practical feasibility, it is preferable to use more logical connectives to begin with when we wish actually to apply the procedure to concrete cases. For this purpose we use the five usual logical constants $\sim$ (not), & (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\equiv$ (biconditional), with their usual interpretations.

A propositional letter P, Q, R, M or N, et cetera, is a formula (and an "atomic formula"). If ϕ, ψ are formulae, then $\sim\phi$, $\phi\,\&\,\psi$, $\phi\vee\psi$, $\phi\supset\psi$, $\phi\equiv\psi$ are formulae. If π, ρ are strings of formulae (each, in particular, might be an empty string or a single formula) and ϕ is a formula, then π, ϕ, ρ is a string and $\pi\rightarrow\rho$ is a sequent which, intuitively speaking, is true if and only if either some formula in the string π (the "antecedent") is false or some formula in the string ρ (the "consequent") is true, i.e., the conjunction of all formulae in the antecedent implies the disjunction of all formulae in the consequent.

There are eleven rules of derivation. An initial rule states that a sequent with only atomic formulae (proposition letters) is a theorem if and only if a same formula occurs on both sides of the arrow. There are two rules for each of the five truth functions—one introducing it into the antecedent, one introducing it into the consequent. One need only reflect on the intuitive meaning of the truth functions and the arrow sign to be convinced that these rules are indeed correct. Later on, a proof will be given of their completeness, i.e., all intuitively valid sequents are provable, and of their consistency, i.e., all provable sequents are intuitively valid.

P1. Initial rule: if λ, ζ are strings of atomic formulae, then $\lambda\rightarrow\zeta$ is a theorem if some atomic formula occurs on both sides of the arrow.

In the ten rules listed below, λ and ζ are always strings (possibly empty) of atomic formulae. As a proof procedure in the usual sense, each proof begins with a finite set of cases of *P1* and continues with successive consequences obtained by the other rules. As will be explained below, a proof looks like a tree structure growing in the wrong direction. We shall, however, be chiefly interested in doing the steps backwards, thereby incorporating the process of searching for a proof.

The rules are so designed that given any sequent, we can find the first logical connective, i.e., the leftmost symbol in the whole sequent that is a connective, and apply the appropriate rule to eliminate it, thereby resulting in one or two premises which, taken together, are equivalent to the conclusion. This process can be repeated until we reach a finite set of sequents with atomic formulae only. Each connective-free sequent can then be tested for being a theorem or not, by the initial rule. If all of them are theorems, then the original sequent is a theorem and we obtain a proof; otherwise we get a counterexample and a disproof. Some simple samples will make this clear.

P2a. Rule $\rightarrow\sim$: If ϕ, $\zeta\rightarrow\lambda$, ρ, then $\zeta\rightarrow\lambda$, $\sim\phi$, ρ.

P2b. Rule $\sim\rightarrow$: If λ, $\rho\rightarrow\pi$, ϕ, then λ, $\sim\phi$, $\rho\rightarrow\pi$.

P3a. Rule $\rightarrow\&$: If $\zeta\rightarrow\lambda$, ϕ, ρ and $\zeta\rightarrow\lambda$, ψ, ρ, then $\zeta\rightarrow\lambda$, $\phi\,\&\,\psi$, ρ.

P3b. Rule $\&\rightarrow$: If λ, ϕ, ψ, $\rho\rightarrow\pi$, then λ, $\phi\,\&\,\psi$, $\rho\rightarrow\pi$.

P4a. Rule $\rightarrow\vee$: If $\zeta\rightarrow\lambda$, ϕ, ψ, ρ, then $\zeta\rightarrow\lambda$, $\phi\vee\psi$, ρ.

P4b. Rule $\vee\rightarrow$: If λ, ϕ, $\rho\rightarrow\pi$ and λ, ψ, $\rho\rightarrow\pi$, then λ, $\phi\vee\psi$, $\rho\rightarrow\pi$.

P5a. Rule $\rightarrow\supset$: If ζ, $\phi\rightarrow\lambda$, ψ, ρ, then $\zeta\rightarrow\lambda$, $\phi\supset\psi$, ρ.

P5b. Rule $\supset\rightarrow$: If λ, ψ, $\rho\rightarrow\pi$ and λ, $\rho\rightarrow\pi$, ϕ, then λ, $\phi\supset\psi$, $\rho\rightarrow\pi$.

P6a. Rule $\rightarrow\equiv$: If ϕ, $\zeta\rightarrow\lambda$, ψ, ρ and ψ, $\zeta\rightarrow\lambda$, ϕ, ρ, then $\zeta\rightarrow\lambda$, $\phi\equiv\psi$, ρ.

P6b. Rule $\equiv\rightarrow$: If ϕ, ψ, λ, $\rho\rightarrow\pi$ and λ, $\rho\rightarrow\pi$, ϕ, ψ, then λ, $\phi\equiv\psi$, $\rho\rightarrow\pi$.

For example, given any theorem of *Principia,* we can automatically prefix an arrow to it and apply the rules to look for a proof. When the main connective is $\supset$, it is simpler, though not necessary, to replace the main connective by an arrow and proceed. For example:

*2.45. $\vdash : \sim(P \vee Q) \cdot \supset \cdot \sim P$,

*5.21: $\vdash : \sim P \,\&\, \sim Q \cdot \supset \cdot P \equiv Q$

can be rewritten and proved as follows.

*2.45 $\sim(P \vee Q) \rightarrow \sim P$ (1)

(1) $\rightarrow \sim P, P \vee Q$ (2)

(2) $P \rightarrow P \vee Q$ (3)

(3) $P \rightarrow P, Q$

VALID

QED

*5.21. $\rightarrow \sim P \,\&\, \sim Q \cdot \supset \cdot P \equiv Q$ (1)

(1) $\sim P \,\&\, \sim Q \rightarrow P \equiv Q$ (2)

(2) $\sim P, \sim Q \rightarrow P \equiv Q$ (3)

(3) $\sim Q \rightarrow P \equiv Q, P$ (4)

(4) $\rightarrow P \equiv Q, P, Q$ (5)

(5) $P \rightarrow Q, P, Q$

VALID

(5) $Q \rightarrow P, P, Q$

VALID

QED

These proofs should be self-explanatory. They are essentially the same as the proofs printed out by the machine, except that certain notational changes are made both to make the coding easier and to avoid symbols not available on the machine printer. The reader may wish to read the next section, which explains these changes, and then compare these with more examples of actual print-outs reproduced in Appendix I. It is believed that these concrete examples will greatly assist the understanding of the procedure if the reader is not familiar with mathematical logic.

- *Program I: The propositional calculus P*

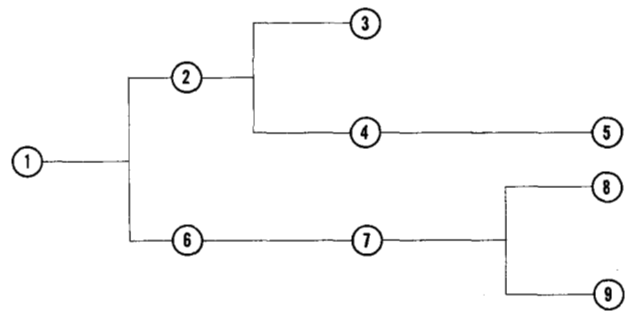
There is very little in the program which is not straightforward. To reserve the dot for other purposes and to separate the numbering from the rest, we write, for example, 2*45/ instead of *2.45.

For the other symbols, we use the following dictionary:

–	$\rightarrow$
B	$\equiv$
C	&
D	$\vee$
F	$\sim$
I	$\supset$

Moreover, we use a modified Polish notation by putting, for example, *CFP . FQ* instead of $\sim P \,\&\, \sim Q$. By putting the connective at the beginning, we can more easily search for it. The use of dots for grouping makes it easier to determine the two halves governed by a binary connective. The reader will have no difficulty in remembering these notational changes if he compares the examples *2.45 and *5.21 with the corresponding proofs in the new notation given in Appendix I.

With the longer examples in Appendix I, the reader will observe that the numbers on the right serve to identify the lines, while the numbers on the left serve to identify the conclusions for which the numbered lines are premises. Essentially each proof is a tree structure. Since we have to arrange the lines in a one-dimensional array, there is a choice among various possible arrangements. The one chosen can be seen from the example 4*45 given in Appendix I. The tree structure would be:



and the one-dimensional arrangement we use is:
(1), (1,2), (2,3), (2,4), (4,5), (1,6), (6,7), (7,8), (7,9).

The whole program has about 1000 lines. The length of the sequents to be tested is deliberately confined to 72 symbols, so that each sequent can be presented by a single punched card. Although this restriction can be removed, it makes the coding considerably easier and gives ample room for handling the problems on hand. Thus, for instance, the longest theorem of the propositional calculus in *Principia,* 5*24, has only 36 symbols. When presented with any punched card, the program enables the machine to proceed as follows.

Copy the card into the reserved core storage COL1 to COL72 (72 addresses in all) in the standard *BCD* notation, i.e., a conventional way of representing symbols by numbers, one symbol in each address. Append the number 1 at the last address, viz., COL72. Search for the arrow sign. If it does not occur, then the line is regarded as ordinary prose, printed out without comment, and the machine begins to study the next card. In particular, the machine stops if the card is blank. If the arrow sign occurs, then the machine marks all symbols before the arrow sign as negative and proceeds to find the earliest logical connective. According as it is *F, C, D, I,* or *B,* the machine turns to *RTNF, RTNC, RTND, RTNI,* or *RTNB.* In each case the proper rule is applied according as whether the connective is before or after the arrow.

After COL1 to COL72, 144 addresses are reserved for getting the one or two premises. As soon as the premises are found according to the proper rule, the original line is printed out and the first premise is shifted into COL1 to COL72 and gets the next number for its identification. If there is a second premise, it has to be shifted away to the idle section and wait for its turn. When the line in COL1 to COL72 contains no more logical connectives, the machine goes to a COMPARE routine to determine whether there is a formula occurring on both sides of the

arrow or not and prints the line out with VALID or NOT VALID appended to it. Then it looks at the idle section to see whether any earlier premises remain there. If there is, it moves the first line there into COL1 to COL72 and pushes the remaining lines of the idle section to fill up the vacancy. If there is no more line left then it concludes, according as whether all final sequents are valid, that the original sequent is a theorem (QED) or not (NOT VALID). When the original sequent is not a theorem, the conjunction of all the resulting nonvalid connective-free sequents amounts to a conjunctive normal form of the original sequent.

Several alternatives are permitted by putting down suitable sense switches. If the interest is to determine merely whether the given sequent is a theorem, it is natural to stop as soon as a nonvalid, connective-free sequent is found. This is indeed taken as the normal procedure in the program. Another permissible choice is to omit the proofs or disproofs altogether but only print out the final answer. A third possible choice is to give the output by punching cards rather than by printing. The possibility of omitting the proof or the disproof enables us to separate the calculating time from the input-output time.

While the program is sufficiently fast for testing propositions we ordinarily encounter, it is not the most efficient testing procedure for more complex propositions. If the purpose is to find an isolated fast-test procedure just for the propositional calculus, and not to obtain at the same time a proof procedure which can be combined naturally with proofs in more advanced domains, it is possible to find much more efficient methods. For example, B. Dunham, R. Fridshal, and G. Sward† have one which is being coded by them.

On the other hand, the storage needed is not large. Not only is each theorem proved from scratch so that earlier theorems need not be kept in the store, but in each proof there is no need to keep all the intermediate lines. In fact, at any time the machine needs to keep in its store at most one line from each level of the proof in its tree form.

This can be made clear by the following example:

*6*6/–BBBBBQR . P . . BQR . . . BBPQ . . BP . BQR . . . . BP . . BQ . BQR.*

There are 13 occurrences of B (if and only if) in this theorem. Since every elimination of B gives two new branches, the complete proof consists of $2^{14}-1$, or about 16,000 lines. Yet at no time need the machine keep in the store for the idle section more than 13 lines. If we use one address for each symbol, we need 72×13 addresses for this; if we pack up these idle lines, we need only 12×13 addresses, a very small number for such a long proof. The length of the proof, incidentally, illustrates how inefficient the procedure can be for certain long propositions. While the complexity of the ordinary truth-table test is determined by the number m of distinct proposition letters (2^m rows) and the number n of distinct subformulae (n columns), the length of proof of the present approach is determined by the number k of occurrences of propositional connectives, i.e., $2^{k+1}-1$ lines or less, since in many cases a conclusion has only one instead of two premises.

The estimated running time for a complete proof of the above theorem with all steps printed out on line is about five hours (about 2500 of the 16,000 lines were printed out in 48 minutes or so). On the other hand, if the proof is not printed out, it takes the machine less than 30 minutes to get the answer. Hence, about 12/13 of the time is spent in reading and printing out. Since axioms and definitions are not used in this approach, they are taken as theorems to be proved, and one arrives at about 220 theorems from the first five chapters of *Principia*. These were proved in about 37 minutes. When only the theorems themselves and the answers (i.e., QED) are printed out, it takes 8 minutes. The actual calculating time, i.e., not counting the input-output time, is less than 3 minutes.

• *Program II: Selecting theorems in the propositional calculus*

A natural question to ask is "Even though the machine can prove theorems, can it select the theorems to be proved?" A very crude experiment in this direction was made with some quite preliminary results. These will be reported here, not for their intrinsic interest, but for suggesting further attempts on the same line. The motive is quite simple: by including suitable principles of triviality, the machine will only select and print out less trivial theorems. These may in turn suggest further principles of triviality; after a certain stage, one would either arrive at essentially the same theorems which have already been discovered and considered interesting, or find in addition a whole crowd of interesting new theorems.

The machine has been made to form a fairly large class of propositions (sequents) and select "interesting" theorems from them. At first all formulae with exactly six symbols containing at most the propositional letters P, Q, R are formed. These come to a total of 651, of which 289 are basic and 362 are trivial variants obtainable from the basic ones by renaming the propositional letters. Of these 651 formulae, 107 are theorems. The program enables the machine to form these formulae, one stored in a single address, and select the theorems among them, prefixing each with a minus sign.

Then the machine is to form all non-ordered pairs (π, ρ) such that either π or ρ is (or both are) among the 289 basic formulae and for each pair (π, ρ), the sequents $\pi \rightarrow \rho$, $\rho \rightarrow \pi$, $\pi, \rho \rightarrow$, $\rightarrow \pi, \rho$, when neither π nor ρ is a theorem. When π, ρ are the same, $\pi \rightarrow \rho$, $\rho \rightarrow \pi$, $\rightarrow \pi$, ρ are not formed. These are the only principles of triviality which are included. Thus, the distinction between basic formulae and their variants avoids the necessity of testing that large number of sequents which are variants of other tested sequents. Moreover, if either π or ρ is a theorem, $\rightarrow \pi$, ρ is a trivial consequence, π, $\rho \rightarrow$ is

†All at the IBM Poughkeepsie Research Laboratory.

a trivial variant of $\rho, \rho\rightarrow$ or $\pi, \pi\rightarrow$; moreover, then $\pi\rightarrow\rho$ (or $\rho\rightarrow\pi$) is a theorem, and a trivial one, if and only if ρ (or π) is a theorem. Finally, $\pi\rightarrow\pi$ is always a trivial theorem.

It was at first thought that these crude principles are sufficient to cut down the number of theorems to a degree that only a reasonably small number of theorems remain. It turns out that there are still too many theorems. The number of theorems printed out after running the machine for a few hours is so formidable that the writer has not even attempted to analyze the mass of data obtained. The number of sequents to be formed is about half a million, of which about 1/14 are theorems. To carry out the whole experiment would take about 40 machine hours.

The reason that such a high portion are theorems comes from the bias in our way of forming sequents. If we view an arbitrary truth table with n proposition letters, since the table gives a theorem if and only if every row gets the value true and there are 2^n rows, the probability of getting a theorem is $\frac{1}{2}^{(2^n)}$. In particular, if $n=3$, we get 1/256. However, the sequents $\pi\rightarrow\rho, \rho\rightarrow\pi, \rightarrow\pi, \rho, \pi, \rho\rightarrow$ amount to $\sim\pi\vee\rho, \sim\rho\vee\pi, \rho\vee\pi, \sim\rho\vee\sim\pi$, each being a disjunction. Hence, the probability is much higher since the probability of $\phi\vee\psi$ being true is $\frac{3}{4}$. If there are three proposition letters, we have $(\frac{3}{4})^8$, which is about 1/10. The few crude principles of triviality, besides cutting the sequents to be tested to less than half, reduces this percentage to about 1/14. It would seem clear that other principles of triviality should be devised and included, e.g. if $\pi\equiv\rho$ and $\Theta(\pi)$ are theorems, $\Theta(\rho)$ is a trivial consequence which need not be recorded.

In the actual program, input cards are used to assign the region of sequents to be formed and tested, since otherwise the machine would simply run continuously for about 40 hours until all sequents are formed and tested. Each sequent formed is retained in a reserved region of the core memory or simply thrown away, according as it has been found to be a theorem or a non-theorem. When all sequents required by the input card have been tested or when the reserved region has been filled up, the theorems obtained to date are transferred onto a tape which afterwards is printed out off-line. Just as a curiosity, a very small random consecutive sample of the print-out is reproduced in Appendix II.

- *Completeness and consistency of the systems P and P_s*

A simple proof for the consistency and the completeness of the system P is possible. Since, however, such considerations become even shorter if fewer truth-functional connectives are used, a system P_s based on the single-stroke | connective (not both) will be given and proved consistent, as well as complete. It will then be clear that a similar proof applies to the system P.

The formulae and sequents are specified as with the system P except that the clause on forming new formulae is now merely: if ϕ and ψ are formulae, then $\phi|\psi$ is a formula. There are only three rules:

P_s1. Same as $P1$.

If λ and ζ are (possibly empty) strings of atomic formulae, then:

P_s2. If $\phi, \psi, \zeta\rightarrow\lambda, \rho$, then $\zeta\rightarrow\lambda, \phi\,|\,\psi, \rho$.

P_s3. If $\lambda, \rho\rightarrow\pi, \phi$ and $\lambda, \rho\rightarrow\pi, \psi$, then $\lambda, \phi\,|\,\psi, \rho\rightarrow\pi$.

Consistency of the calculus P_s. One can easily verify by the intended interpretation of the arrow and the comma that P_s1 is valid, i.e., true in every interpretation of the atomic formulae, that the conclusion of P_s2 is valid if (and only if) its premise is, and that the conclusion of P_s3 is valid if (and only if) its premises are. It follows that every provable sequent is valid.

Completeness of the calculus P_s. We wish to prove that every sequent, if valid, is provable. Given any sequent, we find the earliest non-atomic formula, if any, in the antecedent, and apply P_s3 in reverse direction, thereby obtaining two premises, each with less occurrences of | . If there is no | in the antecedent, we find the earliest, if any, occurrence of | in the consequent and apply P_s2 in reverse direction. We then repeat the same procedure with the results thus obtained. This process will be continued with each sequent until | no longer occurs. Since there are only finitely many occurrences of | in each sequent, this process always comes to an end and then we have a finite class of sequents in which only atomic formulae occur. Now the original sequent is valid if and only if every sequent in the class is. But a sequent with only atomic formulae is valid if and only if it is a case of P_s1, and we can decide effectively in each case whether this is so. Hence, the calculus is complete and we have a decision procedure for provability which yields automatically a proof for each provable sequent.

The propositional calculus with equality (System P_e)

It is convenient, though not necessary, to add the equality sign $=$ before introducing quantifiers. This procedure serves to stress the fact that equality is more elementary than quantifiers, even though customarily quantifiers are presented prior to equality. The changes needed to reach this system from the system P are rather slight. Variables X, Y, Z, S, T, U, V, W, et cetera, are now taken as terms, and the domain of atomic formulae are extended to include all expressions of the form $\alpha=\beta$ when α and β are terms.

The only additional rules necessary for equality are an extension of the initial rule $P1$ so that in addition to the rules $P1$ to $P6b$ of the system P, we now have: If λ, ζ are strings (possibly empty) of atomic formulae, then:

$P7$. $\lambda\rightarrow\zeta$ is a theorem if there is a term α such that $\alpha=\alpha$ occurs in ζ.

$P8$. $\lambda\rightarrow\zeta$ is a theorem if $\alpha=\beta$ occurs in λ and $\lambda\rightarrow\zeta'$ is a theorem, where ζ' is obtained from ζ by substituting α (or β) for some or all occurrences of β (or α).

It is quite easy to extend Program I to obtain a program for the system P_e. The writer, however, did not write a separate program for P_e but includes such a program as a part in Program III. This part enables the machine to proceed exactly as in Program I except that, in testing whether a sequent of atomic formulae is a theorem, the machine does not stop if the sequent is not a theorem by the initial rule *P1* but proceeds to determine whether the sequent can be shown to be a theorem by using the additional initial rules *P7* and *P8*. To distinguish sequents of atomic formulae which are valid truth-functionally from those which become valid only after applying *P7* or *P8*, the former case is marked with VA only, while the latter case is marked in addition by $=$. Examples of print-outs are given in Appendix III.

The longish example *13.3 is included to make a minor point. It has been suggested that it would be interesting if the machine discovers mistakes in *Principia*. This example may be said to reveal a mistake in *Principia* in the following sense. The authors of *Principia* proved this theorem by using *10.13 and *10.221 from the predicate calculus. From the discussion attached to the proof of this theorem, it seems clear that the authors considered this as a theorem which presupposes the predicate calculus. Yet in the proof printed out by the machine, no appeal to anything beyond the system *P*, i.e., no appeal even to the additional rules *P7* and *P8* is made. This is revealed by the fact that all the sequents of atomic formulae, viz., lines 6, 10, 14, 18, 24, 28, 32, 36 in the proof, are marked with VA without the additional $=$ sign. At first the writer thought this indicates a mistake in the program. An examination of the theorem shows, however, that *13.3 is indeed a theorem of the propositional calculus. In fact, Program I alone would yield essentially the same proof.

- *Preliminaries to the predicate calculus*

Thus far an attempt has been made to avoid heavy technicalities from symbolic logic. A few more exact definitions seem necessary, however, when one comes to the predicate calculus.

Formulae, terms, and sequents of the full predicate calculus are specified as follows. Basic symbols are $=$; the five truth-functional connectives; the two quantification symbols for "all" and "some"; proposition letters *P, Q, R,* et cetera; predicate letters *G, H, J, K,* et cetera; variables *X, Y, Z,* et cetera; function symbols *f, g, h,* et cetera; numerals 1, 2, . . . , 9, et cetera; dots or parentheses for grouping. Terms are: (i) a variable is a term; (ii) a numeral is a term; (iii) if α, β, et cetera are terms and σ a function symbol, then $\sigma\alpha$, $\sigma\alpha\beta$, et cetera, are terms. A variable or a numeral is a simple term; other terms are composite. The five truth-functional connectives and two quantification symbols are called logical constants. Atomic formulae are: (i) proposition letters; (ii) $\alpha=\beta$, when α, β are terms; (iii) $G\alpha$, $H\alpha\beta$, et cetera, where α, β, et cetera are terms. Formulae are: (i) atomic formulae are formulae; (ii) if ϕ, ψ are formulae, $\sim\phi$, $\phi\vee\psi$, $\phi\supset\psi$, $\phi\equiv\psi$, $\phi\&\psi$, $(E\alpha)\phi$, $(\alpha)\phi$ are formulae, where α is a variable. A string may be empty or a single formula, and if π, ρ are nonempty strings π, ρ is a string. Given any two strings π and ρ, $\pi\rightarrow\rho$ is a sequent.

Intuitively the scope of a logical constant is clear. In mechanical terms, the method of finding the scope of a logical constant in a given formula depends on the notation. According to the notation actually chosen for the machine, $\phi\vee\psi$, $\phi\&\psi$, $\phi\supset\psi$, $\phi\equiv\psi$ are written as $D\phi\_\_\psi$, $C\phi\_\_\psi$, $I\phi\_\_\psi$, $B\phi\_\_\psi$, with the blank filled in by a string of dots whose number is one larger than the longest string in ϕ and ψ except that no dot is used when both ϕ and ψ are proposition letters. These and related details in the notation can be understood easily from the Appendices. Given a sequent, the scope of a logical constant Γ standing at the beginning of a whole formula in the sequent is the entire formula minus Γ. If Γ is singulary, that is, $\sim$ or one of the two quantification symbols, the scope of the next logical constant Γ' in the formula, if any, is the whole remaining part of the formula minus Γ'. If Γ is binary, then its scope breaks into two parts at the longest string of dots in the scope and each part, if containing a logical constant at all, must begin with one, say Γ', whose scope is the whole part minus Γ'. This gives a mechanizable inductive definition of the scope of every logical constant in any sequent. A logical constant Γ is said to "govern" a logical constant Γ', if Γ' falls within the scope of Γ.

To avoid the explicit use of the prenex normal form, i.e., the form in which all quantifiers in a formula stand at its beginning, it is desirable to introduce, after Herbrand,[18] the sign of every quantifier in a sequent. Two simple preliminary operations will be performed on a given sequent before calculating the signs of the quantifiers in it. First, distinct quantifiers are to get distinct variables, even when one quantifier does not govern the other; moreover, the free variables in the sequent are not used as variables attached to explicit quantifiers. This simplifies the elimination of quantifiers afterwards. Second, all occurrences of $\equiv$ which govern any quantifiers at all are eliminated by either of two simple equivalences: $\phi\equiv\psi$ if and only if $(\phi\&\psi)\vee(\sim\phi\&\sim\psi)$, or, alternatively, $(\sim\phi\vee\psi)\ \&\ (\sim\psi\phi)$.

The positive and negative parts of any formula in the sequent are defined thus: (i) (an occurrence of) ϕ is a positive part of (the same occurrence of) ϕ; (ii) if ϕ is a positive (a negative) part of ψ, then ϕ is a negative (a positive) part of $\sim\psi$; (iii) if ϕ is a positive (a negative) part of ψ or of χ, then ϕ is a positive (a negative) part of $\psi\vee\chi$; (iv) similarly with ϕ and $\psi\&\chi$; (v) if ϕ is a positive (a negative) part of ψ, then ϕ is a positive (a negative) part of $(\alpha)\psi$; (vi) similarly with ϕ and $(E\alpha)\psi$; (vii) if ϕ is a positive (a negative) part of ψ, then ϕ is a positive (a negative) part of $\chi\supset\psi$, and a negative (a positive) part of $\psi\supset\chi$. Any formula ϕ in a sequent is a positive or negative part of the sequent according as (i) it is a positive (a negative) part of a whole formula in the consequent (the antecedent), or (ii) it is a negative (a positive) part of a whole formula

in the consequent (the antecedent). Any quantifier (α) with the scope ϕ in a given sequent is positive (negative) in the sequent if and only if $(\alpha)\ \phi$ is a positive (a negative) part of the sequent; $(E\alpha)$ is positive (negative) if and only if $(E\alpha)\ \phi$ is a negative (a positive) part of the sequent; the different occurrences of a same free variable α in the sequent also make up a positive quantifier (as if (α) were put at the head of the whole sequent).

This involved definition can be illustrated by an example:

$$\text{Ex.0.}(X)\ (GXY \supset (\sim GXX\ \&\ (EZ)\ HXZ)),$$
$$(W)((\sim GWW\ \&(EU)HWU) \supset GWY)$$
$$\rightarrow \sim (EV)HYV.$$

In this example, (X) is a negative quantifier, (EZ) is positive, (W) is negative, (EU) is negative, (EV) is positive. For instance, $(EU)HWU$ is positive in $\sim GWW\ \&(EU)HWU$ but negative in $(\sim GWW\ \&(EU)HWU) \supset GWY$ and $(W)((\sim GWW\ \&\ (EU)HWU) \supset GWY)$, which is a whole formula in the antecedent. Hence, $(EU)HWU$ is positive in the sequent. Hence, (EU) is negative. The assignment of signs to quantifiers coincides with the result in a prenex normal form; positive for universal, negative for existential. Thus, one prenex form of Ex. 0 is:

$$(Y)\ (EX)\ (EW)\ (EU)\ (Z)\ (V)\ \{[(GXY \supset$$
$$(\sim GXX\ \&\ HXZ))$$
$$\&((\sim GWW\ \&\ HWU) \supset GWY)] \supset \sim HYV\}.$$

Another useful but involved concept is "miniscope" forms of a formula of the predicate calculus. It is in a sense the opposite of the prenex form, which generally gives every quantifier the maximum scope. Since the interweaving of quantifiers and variables is the main factor determining the complexity of a formula of the predicate calculus, it is not hard to see that separating variables and reducing the ranges of quantifiers may help to simplify the problem of determining whether a formula is a theorem. The unfortunate part is that sometimes it can be a very complicated process to get a formula into the miniscope form.

It is easier to explain the notion for a formula in which $\equiv$ and $\supset$ no longer occur (say, eliminated by usual definitions) so that the only truth-functional connectives are $\sim$, &, v. Such a formula is said to be in the miniscope form if and only if: (i) an atomic formula ϕ is in the miniscope form; (ii) if ϕ in $(\alpha)\phi$ (or $(E\alpha)\phi$) is a disjunction (a conjunction) of formulae, each of which is in the miniscope form and either contains α or contains no free variable at all, then $(\alpha)\phi$ (or $(E\alpha)\phi$) is in the miniscope form; (iii) if ϕ and ψ are in the miniscope form, so are $\sim\phi$, $\phi \vee \psi$, $\phi\ \&\ \psi$; (iv) if ϕ in $(\alpha)\phi$ (or $(E\alpha)\phi$) begins with $(E\beta)$ (or (β)) and is in the miniscope form, so is $(\alpha)\phi$ (or $E\alpha)\phi$); (v) a formula beginning with a string of quantifiers of the same kind is in the miniscope form if every formula obtained by permuting these quantifiers and then dropping the first, is in the miniscope form. One procedure for bringing a formula into the miniscope form is explained in detail by Quine.[10] In what follows, only parts of Quine's procedure will be used and explained, the machine will follow quite different procedures if the formulae in a given sequent are not easily brought into the miniscope form.

The system Qp and the AE predicate calculus

A specially simple decision procedure is available for many of those sequents not containing function symbols in which each formula is in the miniscope form and in the AE form, i.e., no positive quantifier is governed by a negative quantifier. The procedure can be extended by two preliminary steps and described as follows.

Step 1. Bring every formula into the miniscope form and at the same time apply the truth-functional rules *P2 - P6b*, whenever possible. In general, we obtain a finite set of sequents which all are theorems if and only if the original sequent is.

Step 2. Test each sequent and decide whether it is in the AE form. If this is so for all the sequents, then they and the original sequent all fall within the AE predicate calculus, and we proceed to decide each sequent by continuing with Step 3. If this is not so for some sequent, then the original sequent does not belong to the AE predicate calculus and has to be treated by appealing to a richer system Q to be described below.

Step 3. For a sequent in the AE predicate calculus, drop all quantifiers and replace all the variables attached to negative quantifiers by numerals, one numeral for each quantifier. The resulting sequent contains no more quantifiers.

Step 4. Apply the truth-functional rules to obtain a finite set of sequents which contain no more logical constants. Test each sequent by the initial rules and retain only the non-valid ones.

Step 5. List all the variables and numerals occurring in this last set of sequents of atomic formulae, make all possible substitutions of the variables for the numerals in the sequents, (substitute X for the numerals if no variables occur) and test each time whether the resulting sequents are all valid. The initial sequent of Step 3 is a theorem if there is a substitution which makes all the sequents in the finite set theorems.

Step 6. The original sequent is a theorem if all the sequents obtained by Step 1 are theorems by Steps 3 to 5.

This completes the description of Qp. It is possible to formulate this system more formally, as derived from the basic system P_e, by adding additional explicit rules. But the result would be rather lengthy and a bit artificial. Since the above less explicit formulation conforms to the general theoretical requirements of a formal system, we shall not give a formally more pleasing description. This remark applies also to the systems to be given below.

Here are a few simple examples which illustrate this procedure and some minor modifications in it:

*10.25. $(X)GX \rightarrow (EY)GY$ (1)

(1) $G1 \rightarrow G2$ NOT (2)

(1) $GX \rightarrow GX$ VA (2)

QED

*11.21. $\rightarrow (X)(Y)(Z)GXYZ \equiv (V)(U)(W)GUVW$ (1)

(1) $(X)(Y)(Z)GXYZ \rightarrow (V)(U)(W)GUVW$ (2)

(1) $(V)(U)(W)GUVW \rightarrow (X)(Y)(Z)GXYZ$ (3)

(2) $G123 \rightarrow GUVW$ NOT (4)

(2) $GUVW \rightarrow GUVW$ VA (4)

PQED

(3) $G213 \rightarrow GXYZ$ NOT (5)

(3) $GXYZ \rightarrow GXYZ$ VA (5)

QED

*11.57 $(X)GX \equiv (Y)(Z)(GY \,\&\, GZ)$ (1)

(1) $(X)GX \rightarrow (Y)(Z)(GY \,\&\, GZ)$ (2)

(1) $(Y)(Z)(GY \,\&\, GZ) \rightarrow (X)GX$ (3)

(3) $G1 \,\&\, G2 \rightarrow GX$ (4)

(4) $G1, G2 \rightarrow GX$ NOT (5)

(4) $GX, GX \rightarrow GX$ VA (5)

PQED

(2) $(X)GX \rightarrow (Y)GY \,\&\, (Z)GZ$ (6)

(6) $(X)GX \rightarrow (Y)GY$ (7)

(6) $(X)GX \rightarrow (Z)GZ$ (8)

(7) $G1 \rightarrow GY$ NOT (9)

(7) $GY \rightarrow GY$ VA (9)

PQED

(8) $G1 \rightarrow GZ$ NOT (10)

(8) $GZ \rightarrow GZ$ VA (10)

QED

*9.22. $(X)(GX \supset HX) \rightarrow (EY)GY \supset (EZ)HZ$ (1)

(1) $(X)(GX \supset HX),\ (EY)GY \rightarrow (EZ)HZ$ (2)

(2) $G1 \supset H1, GY \rightarrow H2$ (3)

(3) $H1, GY \rightarrow H2$ NOT (4)

(3) $GY \rightarrow H2, G1$ NOT (5)

(3) $GY \rightarrow HY, GY$ VA (5)

QED.

(3) $HY, GY \rightarrow HY$ VA (4)

These examples are intended to show several things. In the first place, for example, in line (5) of *11.21, the possible substitutions for (1,2,3) are (X,X,X), (X,X,Y), (X,X,Z), (X,Y,X), (X,Y,Y), (X,Y,Z), $(Y,X,X,)$ (Y,X,Y), (Y,X,Z), (Y,Y,X), (Y,Y,Y), (Y,Y,Z), et cetera, 27 in all. If one tries out the substitutions one by one, as was done in Program III, it will take some time before one reaches the correct substitution (Y,X,Z). It is, however, clear that an equally mechanizable procedure is to single out occurrences of the same predicate letter on both sides of the arrow and select the substitutions which would make all the sequents in question theorems. This is one minor change which will be made in Program III. Incidentally, this is also an instance of a simple strategy which improves the program.

In the second place, *9.22 shows that there is no need to eliminate $\supset$ because the definition of formulae in the miniscope form can easily be modified to include formulae containing $\supset$ in addition to $\sim$, v, &. All that is needed is to remember that $\phi \supset \psi$ is the same as $\sim \phi \vee \psi$.

In the third place, as will be proved later on, the Steps 3 to 5 given above are applicable to a sequent in the *AE* form which contains at most one positive quantifier even if it is not in the miniscope form. This is why in the proof of *11.57, (3) does not have to be transformed into the miniscope form, while (2) has to. Thus if the quantifiers in (2) are not separated as in line (6), one gets:

(2) $G1 \rightarrow GY \,\&\, GZ$ (6)

(6) $G1 \rightarrow GY$ NOT (7)

(6) $G1 \rightarrow GZ$ NOT (8)

No possible substitution can make both (7) and (8) theorems. Hence, although *11.57 is a theorem, no proof would be obtained in this way, unless a reduction to the miniscope form is made first. On the other hand, in an alternative procedure to be described below, Step 5 in the above procedure is replaced by a different substitution, performed before Step 4, so that $G1$ is replaced by GY,GZ in the above example and it is no longer necessary to have the sequent in the miniscope form to begin with. The relative merits of the two procedures will be compared below.

- *Program III: The AE predicate calculus*

This program was originally intended to embody the procedure *Qp*. But the preliminary part of bringing a formula into the miniscope form has not been debugged. It is now clear that just for the purpose of proving all the theorems of the predicate calculus in *Principia,* it is not necessary to include all the rules for bringing a formula into a miniscope form. Indeed, only about 5% of the theorems need such rules at all, and only rather simple ones.

There are, however, a few other differences between Program III and the procedure described above. Instead of eliminating all quantifiers at once according to their signs, quantifiers are treated on the same basis as the truth-functional connectives with two rules for each. If λ and ζ are (possibly empty) strings of atomic formulae and i is a new numeral, ν is a new variable:

Rule $\rightarrow\forall$: If $\zeta \rightarrow \lambda, \phi\, \nu, \pi$, then $\zeta \rightarrow \lambda, (\alpha)\, \phi\, \alpha, \pi$.

Rule $\rightarrow\exists$: If $\zeta \rightarrow \lambda, \phi\, i, \pi$, then $\zeta \rightarrow \lambda, (E\,\alpha)\, \phi\, \alpha, \pi$.

Rule $\forall\rightarrow$: If $\lambda, \phi\, i, \rho \rightarrow \pi$, then $\lambda, (\alpha)\, \phi\, \alpha, \rho \rightarrow \pi$.

Rule $\exists\rightarrow$: If $\lambda, \phi\, \nu, \rho \rightarrow \pi$, then $\lambda, (E\,\alpha)\, \phi\, \alpha, \rho \rightarrow \pi$

These rules make for uniformity in the whole procedure except that precaution should be taken that the same quantifier, when recurring at different places on account of truth-functional reductions, should still be replaced by the same variables or numerals, although when the replacement is by a numeral, the difference is not vital.

When this precaution is not taken, it can happen that certain theorems of the AE predicate calculus do not get proofs. This in fact happened with Program III, which failed to yield proofs for *10.3, *10.51, *10.55, *10.56, *11.37, *11.52, *11.521, *11.61, for no other reason than this.

A less serious defect in Program III is that truth-functional reductions are not always made as often as possible before eliminating quantifiers. This has the defect that several separate problems are sometimes treated as one whole problem and the running time required for getting a proof becomes unnecessarily long. In four special cases, viz., *10.22, *10.29, *10.42, *10.43, this defect in fact results in the failure to get a proof, even though a proof for each can be found by Program III, if all possible truth-functional reductions are made before eliminating quantifiers. Both this and the preceding defects of Program III can easily be amended. The reason for dwelling so long on them is to illustrate how machines can assist mathematical research in revealing theoretical defects in preliminary formulations of general procedures.

Since the present methods do not use axioms and definitions, the axioms and definitions of *Principia* are rewritten as theorems. The resulting augmented list of theorems in *Principia* (*9 to *13) from the predicate calculus with equality, has a total number of 158 members. Of these, 139 can be proved by Program III as it stands, although some of them require unnecessarily long running time, e.g., *11.21 and *11.24. If we make the few minor modifications mentioned above, the running time for all becomes reasonably short and the 12 theorems listed in the last two paragraphs become provable. Altogether there are only 7 of the 158 theorems which stand in need of some preliminary simple steps to get the formulae into the miniscope form: *11.31, *11.391, *11.41, *11.57, *11.59, *11.7, *11.71. The rules needed to take care of these cases are three:

(i) Replace $(\alpha)(\phi\alpha \,\&\, \psi\alpha)$ by $(\alpha)\,\phi\alpha \,\&\, (\alpha)\,\psi\alpha$.

(ii) Replace $(E\alpha)(\phi\alpha \vee \alpha)$ by $(E\alpha)\,\phi\alpha \vee (E\alpha)\,\psi\alpha$.

(iii) Replace $(\alpha)(\chi\alpha \supset (\phi\alpha \,\&\, \psi\alpha))$ by
$(\alpha)(\chi\alpha \supset \phi\alpha) \,\&\, (\alpha)(\chi\alpha \supset \psi\alpha)$.

Hence, to summarize, Program III can be somewhat modified to prove all the 158 theorems of *Principia,* with the modified program doing the following. Given a sequent, see whether the rules (i), (ii), (iii) are applicable and apply them if so. Then make all truth-functional simplifications by the rules *P2 - P6b*. This in general yields a finite set of sequents. If every one is in the AE form and either contains no more than one positive quantifier or is in the miniscope form, then the original sequent is often decidable by the method; otherwise it is beyond the capacity of the method. If the former is the case, proceed to decide each sequent either by eliminating all quantifiers at once, as in Step 3 of the preceding section, or by mixing the application of the rules *P2 - P6b* with the rules $\rightarrow\forall$, $\rightarrow\exists$, $\forall\rightarrow$, $\exists\rightarrow$. Finally, use Steps 4 and 5 of the preceding section.

A sample of the print-outs by Program III without the modifications is given in Appendix IV. In this connection, it may be of interest to report an amusing phenomenon when Program III is in operation. The machine usually prints out the lines of a proof in quick succession, and then there is a long pause, as if it were thinking hard, before it prints out the substitution instance which makes all the nonvalid sequents of atomic formulae valid. For example, the first 15 lines of *11.501 in Appendix IV were printed out in quick succession, followed by a long pause of over 2 minutes, and then the remaining few lines were printed out.

• *Systems Qq and Qr: Alternative formulations of the AE predicate calculus*

As remarked in connection with the proof of *11.57, one can replace Step 5 by a different way of substitution and then the new method Qq becomes applicable even when the formulae in the sequent are not in the miniscope form, as long as the sequent is in the AE form.

More exactly, the new method of substitution is as follows. Consider the sequent obtained immediately after the elimination of quantifiers, and determine all the occurring variables and numerals. If $\alpha_1, \ldots, \alpha_n$ are the variables, a formula ϕi, where i is a numeral, is replaced by $\phi\alpha_1, \ldots, \phi\alpha_n$. For example, take (2) of *11.57:

	$(X)GX \rightarrow (Y)(Z)(GY \,\&\, GZ)$	(1)
(1)	$G1 \rightarrow GY \,\&\, GZ$	(2)
(2)	$GY, GZ \rightarrow GY \,\&\, GZ$	(3)
(3)	$GY, GZ \rightarrow GY$	VA (4)
(3)	$GY, GZ \rightarrow GZ$	VA (5)

QED.

What are the comparative merits of Qp and Qq? According to Qq, only one substitution is made, and at the beginning rather than at the end, but unlike Qp, the results obtained may be sequents longer than any previous sequents in the proof. To apply Qq it is not necessary that the formulae be first brought to the miniscope form—the procedure is applicable as long as the sequent is in the AE form. The method QP, however, has the compensating advantage that sometimes a sequent not in the AE form can be reduced to sequents in this form by bringing it into the miniscope form. For example, all sequents of the monadic predicate calculus are decidable by Qp, but not by Qq. Church's example[20] in Appendix VI is such a case. On the other hand, the same example also shows that the procedure of bringing a sequent into the miniscope form can get very involved, so that other methods become preferable. While both Qp and Qq can be incorporated in the system Q for the whole predicate calculus, to be described below, a third method Qr is closer to Q in spirit. Hence, Q will be presented as an extension of Qr rather than one of Qp or Qq.

The method Qr proceeds in the same way as Qp except that at the step of substitution, the disjunction of all substitution instances is tested for truth-functional validity. Thus, take (2) of *11.57 again:

$(X)GX \rightarrow (Y)(Z)(GY \,\&\, GX)$ (1)

(1) $G1 \rightarrow GY \,\&\, GZ$ (2)

(2) $G1 \rightarrow GY$ (3)

(2) $G1 \rightarrow GZ.$ (4)

Now we test whether the disjunction of the conjunction of $GY \rightarrow GY$ and $GY \rightarrow GZ$, and that of $GZ \rightarrow GY$ and $GZ \rightarrow GZ$, is truth-functionally valid. This is indeed so, because if the first disjunctant is false, then $GY \rightarrow GZ$ is false and then $GZ \rightarrow GY$ is true and therewith the second disjunctant is true. Alternatively, this disjunction can also be expressed as a conjunction of four clauses:

(i) $GY, GZ \rightarrow GY, GY$

(ii) $GY, GZ \rightarrow GY, GZ$

(iii) $GY, GZ \rightarrow GZ, GY$

(iv) $GY, GZ \rightarrow GZ, GZ.$

Now we have three alternative methods, *Qp*, *Qq*, *Qr* for the *AE* predicate calculus. In Appendix V, we give an example with its three disproofs.

How do we justify the methods *Qp*, *Qq*, and *Qr*? First, if there is no proof, then the original sequent is not valid. The proof for this is easiest for *Qq*. In the example in Appendix V, line (3) under method *Qq* is not valid if and only if the original sequent (1) is not valid in the domain $\{X, Z\}$. The fact that (4), (5), (6), (7) are not all valid shows that (3) is not valid. In fact, if GXX and GXZ are true but GZX and GZZ are false, (3) is not valid. Therefore, (1) is false under the particular interpretation and hence not valid. More exactly, we wish to find an interpretation under which $(EX)(Y)(GXY \vee GYX) \,\&\, \sim (Z)(EW)GZW$, or more simply, $(Y)\ (GXY \vee GYX)\ \&\ (W)\ \sim GZW$, is satisfiable in $\{X, Z\}$. That is to say, to find an interpretation under which the conjunction of $GXX \vee GXX$, $GXZ \vee GZX$, $\sim GZX$, $\sim GZZ$ is true, or its negation (3) is false. The interpretation of G given above serves this purpose. It is not hard to generalize the argument to all sequents in the *AE* form. Indeed, such considerations are familiar from standard decision procedures.[21]

The justification of *Qp* consists in the fact that if an *AE* sequent is in the miniscope form, then all the negative quantifiers essentially govern a disjunction in the consequent and a conjunction in the antecedent. As a result, the substitution at the end is often equivalent to that obtained by the method *Qq*. For example, in Appendix V, the disjunction under method *Qp* is of:

(i) $GXX \rightarrow GZX;\ GXX \rightarrow GZX$ or simply $GXX \vee GXX \rightarrow GZX$

(ii) $GXX \rightarrow GZZ;\ GXX \rightarrow GZZ$ or simply $GXX \vee GXX \rightarrow GZZ$

(iii) $GXZ \rightarrow GZX;\ GZX \rightarrow GZX$ or simply $GXZ \vee GZX \rightarrow GZX$

(iv) $GXZ \rightarrow GZX;\ GZX \rightarrow GZZ$ or simply $GXZ \vee GZX \rightarrow GZZ$

Hence, the disjunction is equivalent to:

$\sim (GXX \vee GXX) \vee \sim (GXZ \vee GZX) \vee GZX \vee GZZ,$

which, in turn, is equivalent to (3) under method *Qq*.

When a sequent has no more than one positive quantifier, whether in the miniscope form or not, it is quite obvious that a proof by *Qq* is obtainable if and only if one by *Qp* is, since in either method there is only a single possible substitution. In the general case, however, as Mr. Richard Goldberg† has pointed out in correspondence, there are *AE* sequents in the miniscope form which are provable by *Qq* but not by *Qp*. He gives the following valid sequent as example:

$(X)GXU \rightarrow (EW)(GYW \,\&\, GZW).$

It follows that *Qp* is applicable only to a subclass of *AE* sequents in the miniscope form. Hence, it would seem that the correct course is to modify Program III to embody the procedure *Qq* or the procedure *Qr*. The modifications needed are, fortunately, again not extensive.

With some care, it is possible to prove by an inductive argument that a proof by *Qq* is obtainable if and only if one by *Qr* is. This is so because the substitutions at the beginning of *Qq* give the same result as the taking of disjunctions at the end of *Qr*. Hence, it is true that all valid sequents of the *AE* predicate calculus are provable in *Qq* and in *Qr*.

It is easier to prove that if there is a proof by any of the methods, then the sequent is valid. In the case of *Qp*, one can simply replace all numerals throughout by the correct variables found at the end, and the result would be a quite ordinary proof which can easily be seen to yield only valid results. In the case of *Qq*, one can again make the replacement throughout, so that the result is a proof which, instead of $\rightarrow\forall$, $\rightarrow\exists$, $\forall\rightarrow$, $\exists\rightarrow$, uses $\rightarrow\forall$, $E\rightarrow$, and:

Rule $\rightarrow\exists^*$: If $\zeta \rightarrow \lambda, \phi\,\alpha_1, \ldots, \phi\,\alpha_n\,\pi$, then $\zeta \rightarrow \lambda, (E\,\alpha)\,\phi\,\alpha, \pi.$

Rule $\forall\rightarrow^*$: If $\lambda, \phi\,\alpha_1, \ldots, \phi\,\alpha_n, \rho \rightarrow \pi$, then $\lambda, (\alpha)\,\phi\,\alpha, \rho \rightarrow \pi.$

Finally, since there is a proof by *Qr* if and only if there is one by *Qq*, every theorem of *Qr* is also valid.

• *System Q: The whole predicate calculus with equality*

Thus far we have considered only *AE* sequents and have used no function symbols. Now we shall consider arbitrary sequents of the predicate calculus and make use of function symbols from time to time.

The method *Q*, an extension of *Qr*, can be explained as follows. It is desirable (for shorter running time) but not necessary, to make preliminary truth-functional reductions so that one problem is broken up into several simpler problems. For each problem, the following steps are used.

Step I. Eliminate all occurrences of $\equiv$ whose scopes contain quantifiers. Determine the positive quantifiers, the negative quantifiers, and for every positive quantifier governed by negative quantifiers, if there is any, the negative quantifiers which govern it. Use distinct variables for all the free variables and positive quantifiers.

† IBM Research Laboratory, Yorktown Heights, N. Y.

Step II. Drop all quantifiers and replace all variables attached to a negative quantifier by a distinct numeral, all variables attached to a positive quantifier governed by negative quantifiers by a function symbol, followed by the numerals for the governing negative quantifiers.

Step III. Make truth-functional simplifications until all logical constants are eliminated and a finite set of sequents of atomic formulae is obtained.

Step IV. Make all possible substitutions on these sequents obtaining results S_1, S_2, S_3, et cetera. The original sequent is a theorem if and only if there is a truth-functional tautology among S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$, et cetera.

The substitutions to be made are a bit more complex than those in the AE predicate calculus. The basic terms consist of not only all the occurring variables, say X and Y, but also instances of the composite terms such as, say, fX, ffX or f^2X, fY, f^2Y, f^3X, f^3Y, f^4X, et cetera. The numerals are to be substituted by all possible selections from these basic terms. If no variables occur in the sequents, a single variable X is added.

We give a few simple examples. More complex examples are included in Appendices VI and VII.[22]

Ex. 1. $\rightarrow (EY)(Z)(GXZ \supset GXY)$ (1)
(1) $\rightarrow GXf1 \supset GX1$ (2)
(2) $GXf1 \rightarrow GX1$ (3)
(2) $GXfX \rightarrow GXX$ (4)
(2) $GXf^2X \rightarrow GXfX$ (5)
(2) $GXf^3X \rightarrow GXf^2X$ (6)
and so on.

Since the disjunction of (4) and (5) is already valid, this is a theorem. In this simple case, (4), (5), (6) are S_1, S_2, S_3, and S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$ can be simply rewritten as S_1 and:

$GXfX, GXf^2X \rightarrow GXX, GXfX$
$GXfX, GXf^2X, GXf^3X \rightarrow GXX, GXfX, GXf^2X.$

Ex. 2 $(X)(EY)(GXY \rightarrow (EZ)(W)GZW$ (1)
(1) $G1f1 \rightarrow G2g2$ (2)
$GXfX \rightarrow GXgX$ (S_1)
$GfXf^2X \rightarrow GXgX$ (S_2)
$GXfX \rightarrow GfXgfX$ (S_3)
$GfXf^2X \rightarrow GfXgfX$ (S_4)
and so on.

In this example, the basic terms are X, fX, gX, fgX, gfX, f^2X, g^2X, fg^2X, gf^2X, f^2gX g^2fX, f^3X, g^3X, et cetera. Intuitively it can be seen that, no matter what basic terms we substitute for 1 and 2, we can never arrive at a tautologous disjunction. Thus, no matter what we do, we can never get an antecedent of the form $G\alpha g\beta$, or a consequent of the form $G\alpha f\beta$. Hence, this is not a theorem of Q.

An alternative procedure Q', which is more directly related to the standard method initiated by Herbrand, is to make the substitutions immediately after the quantifiers are eliminated. In Appendix VI, a proof of Ex. 3 is given by this method. Clearly Q' is an extension of Qq, just as Q is an extension of Qr. While it is not clear whether Q or Q' is superior if the problem is done by hand, it is conjectured that Q is mechanically less cumbersome, especially if the final test procedure is programmed along the line suggested in Appendix VII.

To justify the procedures Q and Q', that is, to prove their correctness and completeness, one need introduce only slight modifications into standard arguments of Skolem and Herbrand. The equivalence of the two procedures is established by the same kind of argument as that for the equivalence of Qq and Qr. Hence, it suffices to prove the correctness and completeness of Q'.

The correctness, i.e., that every provable sequent is valid, is apparent. Given a proof of Q', i.e., a disjunction of substitution instances which is tautologous, we can derive a sequent which is equivalent to the original sequent to be proved, except that every whole formula in the sequent is in the prenex form. Take Ex. 1. The proof of Q' consists merely in replacing the lines (3)–(6) in the above proof by the tautologous line:

$\rightarrow GXfX \supset GXX, GXf^2X \supset GXfX.$ (i)

From this line, we can by the usual rules of quantification infer:

$\rightarrow GXfX \supset GXX, (Z)(GXZ \supset GXfX)$
$\rightarrow GXfX \supset GXX, (EY)(Z)(GXZ \supset GXY)$
$\rightarrow (Z)(GXZ \supset GXX), (EY)(Z)(GXZ \supset GXY)$
$\rightarrow (EY)(Z)(GXZ \supset GXX), (EY)(Z)GXZ \supset GXY)$
$\rightarrow (EY)(Z)(GXZ \supset GXX).$

It may be remarked that since a new term for the universal quantifier is introduced every time, viz., fX, f^2X, et cetera, we can always reintroduce it successively without violating the restriction that the term to be replaced by Z is not free elsewhere. Strictly speaking, the terms fX, f^2X, et cetera should first be replaced by new variables, say U, V, et cetera. Then the line (i) remains valid truth-functionally, and the resulting proof of the original sequent of Ex. 1 would conform entirely to usual rules for quantifiers.[23]

The completeness of Q′ can be proved by using ideas familiar in mathematical logic.[24] We wish to prove that if there is no proof, then its negation is satisfiable in an enumerable domain, and hence the original sequent is not valid. Consider Ex. 2 for which none of S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$, et cetera, is valid. In other words, for each disjunction, there are truth-value assignments which would make it false. Since every later disjunction contains all earlier disjunctions as part, a falsifying assignment of $S_1 \vee \ldots \vee S_n$ (call it D_n) also falsifies all D_i, $i < n$. In other words, there is a truth-assignment which falsifies D_1, and for every n, there exists a falsifying assignment for D_n which has an extension that falsifies D_{n+1}. In the simple example Ex. 2, we have:

D_1 is falsified if $GXfX$ is true but $GXgX$ is false,
D_2 is falsified if, in addition, $GfXf^2X$ is true,
D_3 is falsified if, in addition, $GfXgX$ is false,
D_4 is falsified if, in addition, $GfXf^2X$ is true,
and so on.

In general, we have an infinite tree structure such that there is a finite set of nodes falsifying D_1, of which at least one has extensions or nodes on the second level, which falsify D_2. Among the nodes on the second level,

i.e., truth-value assignments which falsify D_2, at least one has extensions which falsify D_3, and so on. It then follows by the *Unendlichkeitslemma*[25] that there exists an infinite path, or an infinite truth-value assignment which falsifies D_1, $D_2, \cdots$ simultaneously. Thus, since each node originates only finitely many (possibly zero) immediate branches and there are infinitely many paths, there must be one node a_1 at the first level which occurs in infinitely many paths, and among the finitely many nodes of the second level joined to a_1, there must be at least one node a_2 which occurs in infinitely many paths. This is true for every level, and hence $a_1, a_2, a_3 \cdots$ determines an infinite path. This is no longer true generally when there may be infinitely many branches for a given node. For example, there is no infinite path in a "spread" in which there is one node of the first level (the origin) and a path from it of length n, for every n, all disjoint except for the origin.

An assignment which falsifies D_1, $D_2, \cdots$ simultaneously is a model of the negation of, say, Ex. 2:

$$(X)(EY)GXY \,\&\, \sim (EZ)(W)GZW,$$

or $$(X)[(EY)GXY \,\&\, (EW) \sim GXW]. \qquad \text{(N)}$$

Thus the individuals are X, fX, gX, f^2X, g^2X, fgX, et cetera, and we have found an interpretation of G such that for every individual a, there is an individual b, viz. fa, and an individual c, viz. ga, such that $Gafa \,\&\, \sim Gaga$. Hence, in this domain with this G, the negation N of Ex. 2 is true, and Ex. 2 is not valid.

A program for the method Q or the method Q' has not yet been written. It seems clear that certain auxiliary procedures will be useful in reducing the running time and extending the range of application. For example, it seems desirable to separate scopes of different quantifiers when possible, although it is not immediately obvious whether always bringing a sequent into the miniscope form first is feasible on the whole. Other simplifications such as dropping tautologous or repetitive conjunctants could easily and profitably be included. In general, the practical limitation of the machine will necessarily impose certain restrictions on the solvable problems. The machine will have to concede defeat when the running time is too long or the easily available storage is exhausted. When such a situation arises, it seems desirable to try some alternative procedure before giving up the problem entirely.

An intrinsic limitation of the methods Q and Q' is the following. There are various sequents which amount to an axiom of infinity, i.e., a proposition satisfiable in an infinite domain but in no finite domain. If the machine is given the negation of such a sequent, the method Q or the method Q' will never give the desired negative answer since it is, being the negation of an axiom of infinity, valid in every finite domain, though not a theorem of the predicate calculus. Simple examples of this type are:

Ex. 6. $\rightarrow (EX)GXX, (EX)(Y) \sim GXY,$
$(EX)(EY)(EZ)(GXY \,\&\, GYZ \,\&\, \sim GXZ)$

Ex. 7. $\rightarrow (EX)GXX, (EX)(Y) \sim GXY,$
$(EX)(Y)(EZ)(GYZ \,\&\, \sim GXZ).$

This class of propositions may be of special interest if we wish to test whether a formal system is consistent. Most of the interesting formal systems are intended to be satisfiable only in infinite domains. Hence, if the system is consistent, then its negation, though not a theorem of the predicate calculus, is valid in every finite domain. Hence, even theoretically, the methods Q and Q' can at most discover contradictions in an inconsistent formal system but cannot ascertain that an interesting formal system is indeed consistent.

This suggests the desirability of adding special decision procedures which cover some propositions in this class. Such results are rather scarce. The only one seems to be Ackermann's, which is applicable only to a rather special subclass.[26]

A question concerning the efficiency of the methods Q and Q' is the obvious remark that if some of D_1, D_2, $\cdots$ is indeed a tautology, the human being often finds such a disjunction in the sequence without actually examining all the preceding disjunctions. Hence, it may be possible to include suitable strategies for choosing such disjunctions. The difficult problem here is to find suitable strategies of sufficient generality. This is in part related to the larger questions of making use of previously proved theorems. The methods considered in this paper so far all begin from scratch. When we get into more advanced disciplines, it seems unlikely that the machine can feasibly avoid reference to previously proved theorems. Yet there is the analogous situation in ordinary calculations where it is often faster for the machine to calculate known results on the spot rather than look them up in tables stored in some remote corner of the machine. On account of questions of storage and access time, some golden mean has to be struck between the knowledgeable pedant and the prodigy who turns out to be somewhat ignorant.

Conclusions

The original aim of the writer was to take mathematical textbooks such as Landau on the number system,[27] Hardy-Wright on number theory,[28] Hardy on the calculus,[29] Veblen-Young on projective geometry,[30] the volumes by Bourbaki, as outlines and to make the machine formalize all the proofs (fill in the gaps). The purpose of this paper is to report work done recently on the underlying logic, as a preliminary to that project.

The restricted objective has been met by a running program for the propositional calculus and a considerable portion of the predicate calculus. Methods for dealing with the whole predicate calculus by machine have been described fairly exactly. A summary of results and a comparison with previous work in this field were given in the introductory section and will not be repeated here.

The writer sees the main interest of the work reported here, not so much in getting a few specific results which in some ways are stronger than expected (e.g., the fast speed attained and the relatively small storage needed), as in illustrating the great potentiality of machines in an

apparently wide area of research and development. Various problems of the same type come to mind.

Decision procedures for the intuitionistic and modal propositional calculi are available but often too lengthy to be done by hand.[31] It seems possible and desirable to code these procedures in a manner similar to the classical systems of logic. The intuitionistic predicate calculus with its decidable subdomains, such as all those propositions which are in the prenex form, may also be susceptible to analogous treatment. Since the efficiency of the proof-decision procedure in Program I depends on the elimination of *modus ponens* (rule of detachment), a related question of logic is to devise cut-free systems for various partial and alternative systems of the propositional calculus.

A good deal of work has been spent in constructing various systems of the propositional calculus and of modal logic. The questions of completeness and independence are often settled by methods which are largely mechanizable and even of no great complexity. This suggests that many of the results in this area, such as those reported by Prior,[32] can be obtained by mechanical means. Given a system, in order to determine the independence and completeness (i.e., nonindependence of all axioms of some given complete system) of its axioms, we may simultaneously grind out proofs and matrices used for independence proofs and stop when we have either obtained a derivation or a matrix that establishes the independence of the formula under consideration. It is true that Linial and Post[33] have proved the undecidability of this class of problems so that we cannot be sure that we can always settle the particular question in each case. Nonetheless, we may expect this procedure to work in a large number of cases. The only practical difficulty is that, in grinding out proofs, the rules of *modus ponens* makes the matter rather unwieldy. When equivalent cut-free formulations are available, this mechanical aid to such simple mathematical research would become more feasible. Alternatively, the strategies devised by Newell-Shaw-Simon may find here a less wasteful place of application.

A mathematically more interesting project is to have machines develop some easy number theory. Here there are two possible alternative approaches: use quantifiers or avoid quantifiers. It is known in mathematical logic that ordinary number theory can be developed largely without appeal to quantifiers. Thus, from the discussions in the body of the paper, it is clear that quantifiers serve essentially to replace an indeterminate class of function symbols. In number theory, these function symbols can usually be replaced by specific function symbols introduced by recursive definitions. Since these are more specific and often intuitively more familiar, it seems quite plausible that avoiding quantifiers would be an advantage. On the other hand, it may be better to use existential quantifiers but avoid mixing quantifiers of both kinds ("all" and "some"), since that is the main source of the complexity of the predicate calculus.

If one wishes to prove that the square root of 2 is not a rational number, this can be stated in the free-variable form as: $2Y^2 \neq X^2$, and a proof can be written out without use of quantifiers. On the other hand, if one wishes to prove that there are infinitely many primes, it seems natural to state the theorem as:
$(EX)(Y < X \,\&\, X$ is a prime).
Essentially, the usual proof gives us a simple recursive function f, such that
$Y < fY \,\&\, fY$ *is a prime*
is true. But before we get the proof and the required function, it is convenient to use the quantifier (EX) which serves to express the problem that a yet unknown function is being sought for.

In this connection, it may be of interest to make a few general remarks on the nature of expansive features in different proof procedures. The attractive feature of the system P as a proof procedure is that, given a sequent, all the lines in a proof for it are essentially parts of the sequent. As a result, the task of searching for a proof is restricted in advance so that, at least in theory, we can always decide whether a proof exists or not. This contrasts sharply with those proof procedures for the propositional calculus which make use of the *modus ponens*. There, given q, we wish to search for p, such that p and $p \supset q$ are theorems. There is no restriction on the length and complexity of p. The cut-free formulation achieves a method such that for every proof by the expansive method there is a corresponding proof in this method without expansion, and vice versa.

Since there is no decision procedure for the predicate calculus or current number theory, it follows that expansive features cannot be eliminated entirely from these disciplines. The cut-free formulation for the predicate calculus concentrates the expansive feature in one type of situation: viz., a conclusion $(EX)FX$ may come from $F1$ or $F2$ or et cetera. The method Q given above further throws together all such expansions for a given sequent to be proved or disproved at the end of the process. These devices have the advantage that, for more efficient partial methods or strategies, one may direct the search mainly to one specific region which contains the chief source of expansion.

If number theory is developed with no appeal to quantifiers, the above type of expansion is avoided. It is not possible, however, to avoid in general another type of expansion. Thus we can conclude $X=Y$ from $fX=fY$, but given X and Y, there are in general infinitely many candidates for the function f, so that trying to find an f which leads to $X=Y$ through $fX=fY$ is an expansive procedure. So much for different expansive features.

Other possibilities are set theory and the theory of functions. In these cases, it seems desirable to use a many-sorted predicate calculus[34] as the underlying logic. While this is in theory not necessary, it will presumably make for higher efficiency.

As is well known, all standard formal systems can be formulated within the framework of the predicate calculus. In general, if a theorem p is derived from the axioms $A_1, \cdots, A_n$, then the sequent $A_1, \cdots, A_n \rightarrow p$ is

a theorem of the predicate calculus. In particular, if a system with finitely many axioms is inconsistent, the negation of the conjunction of all its axioms is a theorem of the predicate calculus. (The restriction on finitely many axioms is, incidentally, not essential since in most cases we can reformulate a formal system to use only finitely many axioms, with substantially the same theorems.) Specker has proved[35] that Quine's *New Foundations* plus the axiom of choice is inconsistent. Hence, the negation of the conjunction of these (finitely many) axioms is a theorem of the predicate calculus. If a sufficiently efficient program for the predicate calculus on a sufficiently large machine yields, unaided, a proof of this, we would be encouraged to try to see whether the system without the axiom of choice might also be inconsistent. If a system is indeed inconsistent, then there would be a chance that a proof of this fact can be achieved first by a machine.

So far little is said about specific strategies. In number theory, we are often faced with the problem of choosing a formula to make induction on. Here an obvious strategy would be to try first to use as the induction formula the whole conclusion, and then the various subformulae of the conclusion to be established. When faced with a conclusion $(EX)FX$, it seems usually advantageous to try terms occurring elsewhere in the known part of the proof, or their variants, in order to find α such that $F\alpha$. Polya's book[36] contains various suggestions on strategies for developing number theory which will presumably be useful when one gets deeper into the project of mechanizing number theory. Efficient auxiliary procedures such as the one already mentioned by Dunham-Fridshal-Sward for the propositional calculus will undoubtedly be of use in shortening running time, when one tries to formalize proofs or prove theorems in more advanced domains.

While formalizing known or conjectured proofs and proving new theorems are intimately related, it is reasonable to suppose that the first type of problem is much easier for the machine. That is why the writer believes that machines may become of practical use more quickly for mathematical research, not by proving new theorems, but by formalizing and checking outlines of proofs. This proof formalization could be developed, say, from textbooks to detailed formulations more rigorous than *Principia,* from technical papers to textbooks, or from abstracts to technical papers.

The selection of interesting conjectures or theorems and useful definitions is less easily mechanizable. For example, Program II described above gives only very crude results. It should be of interest to try to get better results along the same line. In more advanced domains, however, the question seems to have a complexity of a different order.

If we use a machine to grind out a large mass of proofs, then there seems to be some mechanical test as to the importance and centrality of concepts and theorems. If a same theorem or a same expression occurs frequently, then we may wish to consider the theorem interesting or introduce a definition for the expression. This is, however, a rather slippery criterion. The finite number of proofs printed out at a given time may form a class that is determined on the ground of some formal characteristic of an accidental nature. Unless there is some acceptable norm in advance for ordering the proofs to be obtained, one can hardly justify in this way the claim that certain theorems are interesting.

A more stable criterion may be this: A formula which is short but can be proved only by long proofs is a "deep" theorem. A short expression which is equivalent only to very long expressions is a "rich" concept.

In the normal situations, of course, we have less restricted objective guidance. There is a fixed body of concepts and theorems which is for good reasons regarded as of special interest (the "archive of mathematical knowledge built up by the cumulative effort of the human intellect"). For such a body it is theoretically possible to select important theorems and concepts mechanically, as well as to find elegant alternative proofs. However, even in this case, one is looking backwards. It is not easy to find a forward-looking mechanizable criterion for mathematical centrality. For example, the nice criterion of ranges of application is hard to render articulate.

In one special kind of mathematics, one discipline is developed from another. For example, theories of natural numbers and real numbers can be developed from set theory. If theorems are generated mechanically from set theory, then any set of theorems isomorphic with the axioms for real numbers (or natural numbers) determines expressions which may be taken as definitions for the basic concepts of the theory of real numbers (or natural numbers). In such a case, one can claim that machines can discover definitions too.

It has often been remarked that the machine can do only what it is told. While this is true, one might be misled by an ambiguity. Thus the machine can be told to make a calculation, find a proof, or choose a "deep" theorem, et cetera. The main problem of using rather than building machines is undoubtedly to express more things in mechanical terms.

The limitation of machines has been seen as revealed by its inability to write love letters. That depends on the quality of the love letters to be composed. If one takes the common sort of love letter taught in manuals of effective letter-writing, the machine can certainly write some useful love letters more quickly than it can prove an interesting theorem. If the image of Don Juan in some films is to be believed, the machine can surely be taught to repeat the few sentences of flattery to every woman.

If experimenting with a machine to see what it can do is compared with the usual type of scientific research, it seems more like engineering than physics, in so far as we are not dealing with natural objects but man-made gadgets, and we are applying rather than discovering theories. On the other hand, calculating machines are rather unique among man-made things in that their po-

tentialities are far less clear to the maker than are other gadgets. In trying to determine what a machine can do, we are faced with almost the same kind of problem as in animal or human psychology. Or, to quote Dunham, we are almost trying to find out what a machine is.

The suspiciously aggressive term "mechanical mathematics" is not unattractive to a mathematical logician. A common complaint among mathematicians is that logicians, when engaged in formalization, are largely concerned with pointless hairsplitting. It is sufficient to know that proofs can be formalized. Why should one take all the trouble to show exactly how such formalizations are to be done, or even to carry out actual formalizations? Logicians are often hard put to give a very convincing justification of their occupation and preoccupation. One lame excuse which can be offered is that they are of such a temperament as to wish to tabulate all scores of all baseball players just to have a complete record in the archives. However, the machines seem to supply, more or less after the event, one good reason for formalization. While many mathematicians have never learned the predicate calculus, it seems hardly possible for the machine to do much mathematics without first dealing with the underlying logic in some explicit manner. While the human being gets bored and confused with too much rigour and rigidity, the machine requires entirely explicit instructions.

It seems as though logicians had worked with the fiction of man as a persistent and unimaginative beast who can only follow rules blindly, and then the fiction found its incarnation in the machine. Hence, the striving for inhuman exactness is not pointless, senseless, but gets direction and justification. One may even claim that a new life is given to the Hilbert program of the *Entscheidungsproblem* which von Neumann thought was thoroughly shattered by Gödel's discoveries. Although a universal decision procedure for all mathematical problems is not possible, formalization does seem to promise that machines will do a major portion of the work that takes up the time of research mathematicians today.

Note added in proof: Recently the writer has succeeded by improved methods to have the machine prove the 158 theorems of *9 to *13 in *Principia* in about four minutes. This line includes the time needed for writing tapes but uses the off-line printer. The output is about 47 pages of 60 lines each. (November 10, 1959)

Appendix I: A sample from print-outs by Program I

```
2*45/FDPQ–FP                                    1
1/–FP, DPQ                                      2
2/P–DPQ                                         3
3/P–P, Q                                        4
   VALID                                        4
                                      QED

5*21/–ICFP . FQ .. BPQ                          1
1/CFP . FQ–BPQ                                  2
2/FP, FQ–BPQ                                    3
3/FQ–BPQ, P                                     4
4/–BPQ, P, Q                                    5
5/P–Q, P, Q                                     6
   VALID                                        6
5/Q–P, P, Q                                     7
   VALID                                        7
                                      QED

2*31/DP . DQR–DPQ, R                            1
1/P–DPQ, R                                      2
2/P–P, Q, R                                     3
   VALID                                        3
1/DQR–DPQ, R                                    4
4/Q–DPQ, R                                      5
5/Q–P, Q, R                                     6
   VALID                                        6
4/R–DPQ, R                                      7
7/R–P, Q, R                                     8
   VALID                                        8
                                      QED

4*45/–BP .. CP . DPQ                            1
1/P–CP . DPQ                                    2
2/P–P                                           3
   VALID                                        3
2/P–DPQ                                         4
4/P–P, Q                                        5
   VALID                                        5
1/CP . DPQ–P                                    6
6/P, DPQ–P                                      7
7/P, P–P                                        8
   VALID                                        8
7/P, Q–P                                        9
   VALID                                        9
                                      QED

5*22/–BFBPQ ... DCP . FQ .. CQ . FP             1
                                      QED

5*23/–BBPQ ... DCPQ .. CFP . FQ                 1
                                      QED

5*24/–BFDCPQ .. CFP . FQ ... DCP . FQ .. CQ . FP 1
                                      QED

7. PRELIMINARY TO PREDICATE CALCULUS            1
7*1/IG2 . H2, GX, KX–CH3 . K3                   1
1/H2, GX, KX–CH3 . K3                           2
2/H2, GX, KX–H3                                 3
   NOT VALID                                    3
2/H2, GX, KX–K3                                 4
   NOT VALID                                    4
1/GX, KX–CH3 . K3, G2                           5
5/GX, KX–H3, G2                                 6
   NOT VALID                                    6
5/GX, KX–K3, G2                                 7
   NOT VALID                                    7
                                  NOT VALID
```

Appendix II. Sample from print-outs by Program II

/–BDPR . R, CDPQ . P
/CCPQ . R–BIPR . P
/CCPQ . P–BIPR . R
/CBPQ . Q–CBPP . Q
/CCPP . Q–CIPP . Q
/CCPP . R–CIPP . P
/CDPP . R–CDPP . P
/BBPP . P–DCPQ . P
/DCPQ . P–BBPP . P
/BIPP . P–DDPP . P
/DDPP . P–BIPP . P
/CIPR . P–BDPQ . P
/BIPQ . P–CDPR . P
/CCPR . R–BIPQ . Q
/CCPR . Q–BIPQ . R
/CCPR . P–BBPR . P
/CBPR . P–BCPR . P
/CIPQ . P–BDPR . P
/BIPR . P–CDPQ . P
/CCPQ . R–BIPR . Q
/CCPQ . Q–BIPR . R
/CCPQ . P–CBPP . P
/CBPQ . P–CCPP . P
/CDPP . P–CIPP . P
/CIPP . P–CDPP . P
/BBPP . R–DBPQ . R
/BCPP . Q–DBPQ . P
/–BDPP . R, DDPP . R
/BIPP . P–DDPP . Q
/BIPP . R–DCPP . R
/BBPQ . P–DCPP . Q

/BBPQ . Q–DCPP . P
/DCPP . P–BBPQ . Q
/–BDPQ . P, CIPR . Q
/BIPQ . P–CDPR . Q
/CDPR . P–BIPQ . Q
/CCPR . Q–BBPR . P
/CBPR . R–BBPR . R
/CBPR . Q–BCPR . P
/–BCPR . R, CIPQ . R
/CIPQ . P–BDPR . Q
/CDPQ . R–BDPR . R
/CCPQ . R–BIPR . R
/CCPQ . Q–CBPP . P
/CCPQ . P–CBPP . Q
/CBPQ . R–CBPP . R
/CBPQ . Q–CCPP . P
/CCPP . Q–CBPQ . P
/CBPQ . P–CCPP . Q
/CCPP . R–CIPP . R
/CDPP . Q–CIPP . P
/BBPP . Q–DCPQ . Q
/DCPQ . Q–BBPP . Q
/BCPP . Q–DBPQ . Q
/BIPP . P–DDPP . R
/BIPP . Q–DDPP . Q
/BBPQ . Q–DCPP . Q
/CIPR . Q–BDPQ . Q
/CIPR . P–BDPQ . R
/CDPR . Q–BIPQ . Q
/CCPR . R–BBPR . P
/CCPR . Q–BBPR . Q

/CCPR . P–BBPR . R
/CBPR . R–BCPR . P
/CBPR . P–BCPR . R
/–BDPR . P, CIPQ . R
/BIPR . P–CDPQ . R
/CDPQ . P–BIPR . R
/CCPQ . R–CBPP . P
/CCPQ . Q–CBPP . Q
/CCPP . Q–CBPQ . Q
/CBPQ . Q–CCPP . Q
/CDPP . Q–CIPP . Q
/CDPP . R–CIPP . P
/BBPP . P–DDPQ . P
/BCPP . Q–DBPQ . R
/BBPQ . Q–DCPP . R
/–BCPQ . P, DCPP . P
/CCPR . Q–BBPR . R
/CCPR . P–BCPR . P
/CBPR . Q–BCPR . R
/CBPR . P–BDPR . P
/CBPP . P–CDPQ . P
/CDPQ . P–CBPP . P
/CCPQ . R–CBPP . Q
/CCPQ . P–CCPP . P
/CBPQ . P–CDPP . P
/BBPP . R–DCPQ . R
/BDPP . Q–DBPQ . P
/BIPP . R–DDPP . R
/BBPQ . P–DDPP . Q

/BBPQ . Q–DDPP . P
/DDPP . P–BBPQ . Q
/–BCPQ . P, DCPP . Q
/DCPP . P–BCPQ . Q
/CIPR . P–BIPQ . Q
/CCPR . R–BBPR . R
/CCPR . Q–BCPR . P
/CBPR . R–BCPR . R
/CBPR . Q–BDPR . P
/CIPQ . R–BDPR . R
/CDPQ . R–BIPR . R
/CCPQ . R–CBPP . R
/CCPQ . Q–CCPP . P
/CCPP . Q–CCPQ . P
/CCPQ . P–CCPP . Q
/CBPQ . Q–CDPP . P
/CDPP . Q–CBPQ . P
/CBPQ . P–CDPP . Q
/CDPP . R–CIPP . R
/BBPP . P–DDPQ . R
/BBPP . Q–DDPQ . Q
/BDPP . Q–BDPQ . Q
/BBPQ . Q–DDPP . Q
/–BCPQ . P, DCPP . R
/–BCPQ . Q, DCPP . Q
/CIPR . Q–BIPQ . Q
/BBPR . P–CDPR . R
/CDPR . R–BBPR . P

Appendix III: Sample of print-outs by Program III (no quantifiers)

*13. IDENTITY 1
13*1/=XY–IGX . GY 1
1/GX, =XY–GY =VA 2
QED

13*12/=XY–BGX . GY 1
1/GX, =XY–GY =VA 2
1/GY, =XY–GX =VA 3
QED

13*13/GX, =XY–GY =VA 1
QED

13*14/GX, FGY–F=XY 1
1/GX–F=XY, GY 2
2/=XY, GX–GY =VA 3
QED

13*15/–=XX =VA 1
QED

13*16/–B=XY . =YX 1
1/=XY–=YX =VA 2
1/=YX–=XY =VA 3
QED

13*17/=XY, =YZ–=XZ =VA 1
QED

13*171/=XY, =XZ–=YZ =VA 1
QED

13*172/=YX, =ZX–=YZ =VA 1
QED

13*18/=XY, F=XZ–F=YZ 1
1/=XY–F=YZ, =XZ 2
2/=YZ, =XY–=XZ =VA 3
QED

13*194/–*BCGX . =XY ... CGX .. CGY . =XY* 1
1/*CGX . =XY–CGX .. CGY . =XY* 2
2/*GX, =XY–CGX .. CGQ . =XY* 3
3/*GX, =XY–GX* VA 4
3/*GX, =XY–CGY . =XY* 5
5/*GX, =XY–GY* =VA 6
5/*GX, =XY–=XY* VA 7
1/*CGX .. CGY . =XY–CGX . =XY* 8
8/*GX . CGY . =XY–CGX . =XY* 9
9/*GX, GY, =XY–CGX . =XY* 10
10/*GX, GY, =XY–GX* VA 11
10/*GX, GY, =XY–=XY* VA 12
QED

13*3/*DGY . FGY–BDGX . FGX .. D=XY . F=XY* 1
1/*GY–BDGX . FGX .. D=XY . F=XY* 2
2/*DGX . FGX, GY–D=XY . F=XY* 3
3/*GX, GY–D=XY . F=XY* 4
4/*GX, GY–=XY, F=XY* 5
5/*=XY, GX, GY–=XY* VA 6
3/*FGX, GY–D=XY . F=XY* 7
7/*GY–D=XY . F=XY, GX* 8
8/*GY–=XY, F=XY, GX* 9
9/*=XY, GY–=XY, GX* VA 10
2/*D=XY . F=XY, GY–DGX . FGX* 11
11/*=XY, GY–DGX, FGX* 12
12/*=XY, GY–GX, FGX* 13
13/*GX, =XY, GY–GX* VA 14
11/*F=XY, GY–DGX . FGX* 15
15/*GY–DGX . FGX, =XY* 16
16/*GY–GX, FGX, =XY* 17
17/*GX, GY–GX, =XY* VA 18
1/*FGY–BDGX . FGX .. D=XY . F=XY* 19
19/–*BDGX . FGX .. D=XY . F=XY, GY* 20
20/*DGX . FGX–D=XY . F=XY, GY* 21
21/*GX–D=XY . F=XY, GY* 22
22/*GX–=XY, F=XY, GY* 23
23/*=XY, GX–=XY, GY* VA 24
21/*FGX–D=XY . F=XY, GY* 25
25/–*D=XY . F=XY, GY, GX* 26
26/–*=XY, F=XY, GY, GX* 27
27/*=XY–=XY, GY, GX* VA 28
20/*D=XY . F=XY–DGX . FGX, GY* 29
29/*=XY–DGX . FGX, GY* 30
30/*=XY–GX, FGX, GY* 31
31/*GX, =XY–GX, GY* VA 32
29/*F=XY–DGX . FGX, GY* 33
33/–*DGX, FGX, GY, =XY* 34
34/–*GX, FGX, GY, =XY* 35
35/*GX–GX, GY, =XY* VA 36
QED

Appendix IV: Sample of print-outs by Program III (the *AE* predicate calculus)

10*25/*AXGX–EXGX* 1
1/*G1–EXGX* 2
2/*G1–G2* NOT 3
2/*GS–GS* VA 3
QED

11*26/*EXAYGXY–AYEXGXY* 1
1/*AYGSY–AYEXGXY* 2
2/*GS1–AYEXGXY* 3
3/*GS1–EXGXT* 4
4/*GS1–G2T* NOT 5
4/*GST–GST* VA 5
QED

11*501/–*BEXFAYGXY . EXEYFGXY* 1
1/*EXFAYGXY–EXEYFGXY* 2
2/*FAYGSY–EXEYFGXY* 3
3/–*EXEYFGXY, AYGSY* 4
4/–*EYFG1Y, AYGSY* 5
5/–*FG12, AYGSY* 6
6/*G12–AYGSY* 7
7/*G12–GST* NOT 8
1/*EXEYFGXY–EXFAYGXY* 9
9/*EYFGUY–EXFAYGXY* 10
10/*FGUW–EXFAYGXY* 11
11/–*EXFAYGXY, GUW* 12
12/–*FAYG3Y, GUW* 13
13/*AYG3Y–GUW* 14
14/*G34–GUW* NOT 15
14/*GUW–GUW* VA 15
7/*GST–GST* VA 8
QED

13*22/–*BEZEWC=ZX .. C=WY . GZW ... GXY* 1
1//*EZEWC=ZX .. C=WY . GZW–GXY* 2
2/*EWC=SX .. C=WY . GSW–GXY* 3
3/*C=SX .. C=TY . GST–GXY* 4
4/*=SX . C=TY . GST–GXY* 5
5/*=SX, =TY, GST–GXY* =VA 6
1/*GXY–EZEWC=ZX .. C=WY . GZW* 7
7/*GXY–EWC=1X .. C=WY . G1W* 8
8/*GXY–C=1X .. C=2Y . G12* 9
9/*GXY–=1X* NOT 10
9/*GXY–C=2Y . G12* 11
11/*GXY–=2Y* NOT 12
11/*GXY–G12* NOT 13
11/*GXY–GXY* VA 13
11/*GXY–=YY* =VA 12
9/*GXY–=XX* =VA 10
QED

Appendix V: Different methods for the *AE* predicate calculus

$(EX)\ (Y)\ (GXY \vee GYX) \rightarrow (Z)(EW)GZW$ (1)

Method Qp.

(1) $GX1 \vee G1X \rightarrow GZ2$ (2)
(2) $GX1 \rightarrow GZ2$ NOT (3)
(2) $G1X \rightarrow GZ2$ NOT (4)

This is not valid since (3) and (4) are not both valid no matter whether, for (1, 2), we substitute (X, X), (X, Z), (Z, X) or (Z, Z).

Method Qq.

(1) $GX1 \vee G1X \quad GZ2$ (2)
(2) $GXX \vee GXX,\ GXZ \vee GZX \rightarrow GZX,\ GZZ$ (3)
(3) $GXX, GXZ \rightarrow GZX, GZZ$ NOT (4)
(3) $GXX, GZX \rightarrow GZX, GZZ$ VA (5)
(3) $GXX, GXZ \rightarrow GZX, GZZ$ NOT (6)
(3) $GXX, GZX \rightarrow GZX, GZZ$ VA (7)

This immediately suggests a simplification since (4) and (6), (5) and (7) are the same. The sequent (1) is not a theorem since (4) and (6) are not valid.

Method Qr. Proceed as in *Qp*, and then test the disjunction of the two conjunctions which is equivalent to the conjunction of (4), (5), (6), (7). Hence, again, (1) is not valid.

Appendix VI: An example of Church

Ex. 3.

$(X)(EY)\{[JX \equiv (JY \supset GY)] \& [GX \equiv (JY \supset HY)] \& [HX \equiv ((JY \supset GY) \supset HY)]\} \rightarrow (Z)(JZ \& GZ \& HZ)$ (1)

This is a sequent in the monadic predicate calculus. It is not of the *AE* form but its miniscope forms must be in the *AE* form. As a result, this can be proved either by *Qp* or by *Q*, though not by *Qq* or *Qr*. It seems more tedious to use *Qp* than *Q*.

To bring (1) into the miniscope form, we have to eliminate the occurrences of $\equiv$. Then we have to bring the quantifier-free part of the antecedent into a disjunctive normal form, distribute (EY) and separate out those parts of the scope of every occurrence of (EY) which contains the variable X. Then we have to bring the new antecedent minus the initial (X) into a conjunctive normal form and distribute (X) in the same way. The reader may wish to convince himself how complex the whole procedure is. The separation of quantifiers in the consequent is, of course, easy. After the separation of quantifiers, the resulting sequent in the miniscope form is very long. The remaining steps, while easy, are also tedious.

On the other hand, if method *Q* is used instead, the proof is not so lengthy. By hand, it is even easier by *Q'*.

(1) $J1 \equiv (Jf1 \supset Gf1), G1 \equiv (Jf1 \supset Hf1),$
$H1 \equiv ((Hf1 \supset Gf1) \supset HF1) \rightarrow JX \& GX \& HX$ (2)

We may substitute for 1, X, fX, f^2X, f^3X, et cetera to get S_1, S_2, S_3, S_4, et cetera and test for validity S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$, $S_1 \vee S_2 \vee S_3 \vee S_4$, et cetera. For this purpose it is sufficient to find a disjunction such that any interpretation (truth-value assignment) which makes the antecedents of all the disjunctants true, will make the consequents of all the disjunctants true. It turns out that the disjunction of S_1 to S_7 does this. If one is doing this by hand, it is easier to argue, for example from $JX \equiv (JfX \supset GfX)$ that if JX is false, then JfX must be true and GfX must be false. In this way, one would see that it is not possible to make all the antecedents of S_1 to S_7 true without also making all their consequents true.

A simple consequence of Ex. 1 may illustrate that sometimes *Qq* is preferable to *Qp*:

Ex. 4.

$(EY)(X)\{[JX \equiv (JY \supset GY)] \& [GX \equiv (JY \supset HY)] \& [HX \equiv ((JY \supset GY) \supset HY)]\} \rightarrow (Z)(JZ \& GZ \& HZ).$

This is in the *AE* form, though not in the miniscope form. The proof of this is easier than Ex. 3 no matter which method we use. Nonetheless the proof by *Qq* seems considerably shorter than the proof by *Qp* which still includes similar (though somewhat simpler) steps, as with Ex. 3. The proof by *Qq* is quite easy since we need only drop quantifiers, substitute 1 for Z and verify that:

$JY \equiv (JY \supset GY), GY \equiv (JY \supset HY),$
$HY \equiv ((JY \supset GY) \supset HY), JZ \equiv (JY \supset GY),$
$GZ \equiv (JY \supset HY), HZ \equiv ((JY \supset GY) \supset HY) \rightarrow$
$JZ \& GZ \& HZ.$

The verification for this is quite easy. Thus, we wish to show that if all clauses of the antecedent are true, then JZ, GZ, HZ are all true. By the first clause, JY and GY are true. Hence, by the fourth clause, JZ is true; by the second clause, HY is true. Hence, by the fifth and the sixth clauses, GZ and HZ are true.

Appendix VII: An example of Quine

Ex. 5. $\rightarrow (EY)(Z)(EW)\{[GYX \,\&\, (GYW \,\&\, GWY)] \vee [\sim GYX \,\&\, \sim (GYZ \,\&\, GZY)]\}$ (1)

(1) $\rightarrow G1X \,\&\, (G12 \,\&\, G21), \sim G1X \,\&\, \sim (G1f1 \,\&\, Gf11)$ (2)

(2) $G1X \rightarrow G1X$ (3)

(2) $G1f1, Gf11 \rightarrow G1X$ (4)

(2) $G1X \rightarrow G12$ (5)

(2) $G1X \rightarrow G21$ (6)

(2) $G1f1, Gf11 \rightarrow G12$ (7)

(2) $G1f1, Gf11 \rightarrow G21$ (8)

Now we wish to make substitutions on (3)–(8). Since (3) is valid for all substitutions, it can be omitted. We obtain S_1, S_2, S_3, S_4 et cetera by substituting, for (1, 2), (X, X), (X, fX), (fX, X), (fX, fX), et cetera. In forming each substitution instance, valid sequents and repetitions of a same sequent can be omitted so that, for example, the conjunctive clauses of S_1, S_2, S_3 are as follows.

S_1: $GXfX, GfXX \rightarrow GXX.$

S_2: $GXfX, GfXX \rightarrow GXX; GXX \rightarrow GXfX; GXX \rightarrow GfXX.$

S_3: $GfXf^2X, Gf^2XfX \rightarrow GfXX; GfXX \rightarrow GfXX; GfXX \rightarrow GXfX; GfXf^2X, Gf^2XfX \rightarrow GXfX.$

It can be verified that $S_1 \vee S_2 \vee S_3$ is a tautology. If GXX is true, then S_1 is true. If GXX is false but $GXfX$ or $GfXX$ is false, then S_2 is true. If both $GXfX$ and $GfXX$ are true, then S_3 is true.

While the verification is easy here, it appears that as the number of conjunctions increases, the test for the disjunction of all conjunctions can get mechanically cumbersome. A presumably more manageable method of testing suggests itself.

Make two lists for S_1, one for the antecedents, one for the consequents:

(A_1) $GXfX, GfXX.$

(C_1) $GXX.$

Test whether every string in (C_1) is contained in some string in (A_1). If yes, a proof is obtained. If not, form two lists for S_2:

(A_2) $GXfX, GfXX; GXX.$

(C_2) $GXX; GXfX; GfXX.$

Test whether every string obtained by joining a string of (C_1) to one of (C_2) is contained in every string obtained by combining a string from (A_1) with a string from (A_2). If yes, a proof is obtained. Otherwise, form two lists for S_3 and continue.

It is not hard to convince oneself that this procedure is equivalent to the usual procedure for testing the validity of S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$, et cetera.

References

1. H. Wang, "A Variant to Turing's Theory of Computing Machines," *Journal ACM*, **4**, 88-92 (January 1957).
2. A. W. Burks, D. W. Warren, and J. B. Wright, "An Analysis of a Logical Machine Using Parenthesis-Free Notation," *Mathematical Tables and Other Aids to Computation*, **8**, 53-57 (April, 1954).
3. G. E. Collins, "Tarski's Decision Method for Elementary Algebra," *Proceedings of the Summer Institute of Symbolic Logic at Cornell University*, p. 64 (1957).
4. M. Davis, "A Program for Presburger's Algorithm," *Ibid.*, p. 215.
5. H. Gelernter, "Theorem Proving by Machine," *Ibid.*, p. 305.
6. For example, A. Newell, J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," Report P-951, Rand Corporation, March, 1957, 48 pp.
7. *Ibid.*, p. 26 and p. 28.
8. *Ibid.*, p. 8 and p. 10.
9. J. Herbrand, *Recherches sur la Théorie de la Démonstration*, Travaux de la Société des Sciences de Varsovie, No. 33, 1930, 128 pp.
10. G. Gentzen, "Untersuchungen über das Logische Schliessen," *Math. Zeitschrift*, **39**, 176-210, 405-431 (1934-35).
11. D. Hilbert and P. Bernays, *Grundlagen der Mathematik*, vol. II, Berlin, 1939
12. B. Dreben, "On the Completeness of Quantification Theory," *Proc. Nat. Acad. Sci. U.S.A.*, **38**, 1047-1052 (1952).
13. E. W. Beth, *La Crise de la Raison et la Logique*, Paris et Louvain, 1957.
14. K. J. J. Hintikka, "Two Papers on Symbolic Logic," *Acta Philos. Fennica*, **8**, 7-55 (1955).
15. K. Schütte, "Ein System des Verknupfenden Schliessens," *Archiv f. Math. Logik u. Grundlagenforschung*, **2**, 375-387 (1955).
16. A. Church, *Introduction to Mathematical Logic, I.* Princeton, 1956.
17. W. V. Quine, *Methods of Logic*, New York, 1950.
18. Herbrand, *op. cit.*, p. 21.
19. Quine, *op. cit.*, pp. 101-107.
20. Church, *op. cit.*, p. 262, 46.12 (3).
21. Compare, e.g., Church, *op. cit.* p. 249.
22. The example in Appendix VII is from Quine, *Mathematical Logic*, *180 and *181, pp. 129-130.
23. Compare Herbrand, *op. cit.*, Ch. V.
24. Compare, e.g., the references to Schütte and Beth.
25. D. König, *Theorie der Graphen*, Leipzig, 1936, p. 81.
26. Church, *op. cit.*, Case X, p. 257.
27. E. Landau, *Grundlagen der Analysis*, Leipzig, 1930.
28. G. H. Hardy and E. M. Wright, *Introduction to the Theory of Numbers*, Oxford, 1954.
29. G. H. Hardy, *A Course of Pure Mathematics*, various editions.
30. O. Veblen and J. W. Young, *Projective Geometry*, 1910.
31. See, e.g., G. Kreisel and H. Putnam, "Ein Unableitbarsbeweismethode," *Arkiv f. Math. Logik u. Grundlagenforschung*, **3**, 74-78 (1957).
32. A. N. Prior, *Formal Logic*, Oxford, 1954.
33. S. Linial and E. L. Post, "Recursive Unsolvability of Axioms Problems of the Propositional Calculus," *Bull. Am. Math. Soc.*, **55**, 50 (1949).
34. See, e.g., Church, *op. cit.*, p. 339.
35. E. P. Specker, "The Axiom of Choice in Quine's New Foundations for Mathematical Logic," *Proc. Nat. Acad. Sci. U.S.A.* **39**, 972-975 (1953).
36. G. Polya, *Mathematics and Plausible Reasoning*, Oxford, 1954.

Received December 22, 1958