34p.

Artificial Intelligence Project--RLE and MIT Computation Center
Memo 56--

A PROPOSAL FOR A GEOMETRY THEOREM PROVING PROGRAM

by

Timothy P. Hart

September 11, 1963

TABLE OF CONTENTS

	Introduction	1
1	The Historical Development of Mechanical Theorem Proving	2
1.1	Herbrand Expansion Method	2
1.1.1	Some Formalism	2
1.1.2	Herbrand's Theorem	3
1.1.3	The Free Variable Form	5
1.1.4	The Herbrand Universe	6
1.1.5	The Proof Procedure Programs	6
1.1.5.1	Gilmore's Program	7
1.1.5.2	The Davis-Putnam Algorithm	8
1.1.5.3	The Program of J. A. Robinson	8
1.2	Subgoal-Reduction Method	9
1.2.1	The Logic Theory Machine	9
1.2.1.1	Subgoal Reduction	10
1.2.1.2	Working Backwards	11
1.2.1.3	Matching	11
1.2.2	Gelernter's Geometer	11
1.2.2.1	The Diagram	12
1.2.2.2	Tree Pruning Heuristics	13
1.2.2.2.1	Interpretation of Goal Tree as a Propositional Formula	13
1.2.2.2.2	Syntactic Symmetry	14
1.2.2.3	J. A. Robinson's Paper	15
1.2.2.3.1	Proof Set	15
1.2.2.4	Summary	17

TABLE OF CONTENTS (cont'd)

2.	Proposal for a Theorem Prover	18
2.1	The Order of an Element of H	18
2.2	The Order of a Proof	19
2.3	The Interpretation of H in Geometry	20
2.4	Many-Sorted Quantification Theory	20
2.5	Examples	21
2.5.1	Proof without Construction	21
2.5.2	Proof with Construction	24
2.6	Finding a Proof Given a Proof Set	27
2.7	Proposal: A Program to Find High Order Proofs	28

/

A PROPOSAL FOR A GEOMETRY THEOREM PROVING PROGRAM

by

Timothy P. Hart

Introduction

During the last half of the nineteenth century the need for formal methods of proof became evident to mathematicians who were making such confidence-shaking discoveries as non-Euclidean geometry

> The demand is not to be denied; every jump must be barred from our deductions. That it is hard to satisfy must be set down to the tediousness of proceeding step by step. Every proof which is even a little complicated threatens to become inordinately long. [M1]
>
> G. Frege, 1884

This general desire for rigor has persisted since that time, and a great deal has been learned about formal methods. But, for the reason noted by Frege, very little of real mathematics has been done with full formal treatment. Our present hope is to use computers to take the drudgery out of formal demonstrations, just as they are taking it out of accounting.

Toward this end, several programs are under way. They vary in purpose; the Proofchecker [M8, M9] is to be capable of filling the gaps of a proof; the work of Mott et. al. [M10] aims to achieve the equivalent of a desk calculator ability as an aid to a mathematician doing formal proofs.

The most intriguing prospect, however, is that computers can eventually be made to both devise and prove interesting non-trivial theorems wholly on their own. The first of these desires, the invention of interesting conjectures, has not even been attempted. I believe, however, that we are at the verge of achieving the second of these ends, the mechanical proof of non-trivial theorems, a belief which I hope I can justify in the sequel.

1. The Historical Development of Mechanical Theorem Proving

This section discusses the relationship between the two separate approaches which have been taken toward mechanical theorem proving. We will not discuss the relatively trivial history of "logic machines" that are based on switching models of Boolean Algebra [M2]. The first significant approach, which I will term "subgoal reduction", was initiated by the work of Newell, Shaw and Simon described in the Logic Theory Machine papers (LTM) [H1, H2]. The other approach depends on a theorem due to Herbrand [L5]. This work is typified by the programs of Gilmore [P2], Wang [P3, P4, P5], and Davis and Putnam [P8, P9]. I will call this the "Herbrand expansion" approach.

1.1. Herbrand Expansion Method

The theorem of Herbrand alluded to above yields a proof procedure for quantification theory (first order/lower predicate/functional calculus). A proof procedure is an algorithm which can provide a proof for a formula which is valid, but may not terminate if the formula is not valid. (A valid formula is true no matter what predicates are substituted for its predicate variables.) Unfortunately, it is hopeless to search for an algorithm which will distinguish unfailingly between theorems and non-theorems, since quantification theory is undecidable.

1.1.1. Some Formalism

I will now describe the system of notation for quantification theory to be used:

1. Individual variables: $x, y, z, u, v, w, x_1, y_1, \ldots$

2. Individual constants: $a, b, c, \quad a_1, b_1, c_1, \ldots$

3. Predicate variables: $F, G, H, F_1, G_1, \ldots$

4. Function variables: $f, g, h, f_1, g_1, \ldots$

Some predicate constants and function constants will be introduced later.

A literal is an expression of the form:

$$P[A_1, A_2 \ldots A_n]$$

or of the form

$$\sim P[A_1, A_2 \ldots A_n]$$

where P is a predicate symbol and the A_i are terms.

A term is

1. an individual variable
2. an individual constant
3. an expression of the form

$$\phi[A_1, A_2 \ldots A_n]$$

where ϕ is a function symbol and the A_i are terms.

If R and S are wff's, then the following are well formed formulas (wff):

1. A literal

2. $(R \vee S)$, $(R \wedge S)$, $(R \supset S)$, $\sim R$, $(R \equiv S)$

3. $(x)R$, $(Ex)R$

I will omit extra parentheses when no confusion will result, and sometimes use an individual constant symbol in place of a function of no arguments.

1.1.2. Herbrand's Theorem

Herbrand's theorem defines a process which in effect translates a formula of quantification theory into a new formula (not equivalent to the original) which has no quantifiers, which I will call the free variable form (fvf).

The theorem also provides a method of generating an infinite, hierarchically-structured set of individuals, the Herbrand universe, which are to be substituted for the free variables in the free variable form of the formula.

The proof procedure can be described as follows:

1. Let us suppose there are n free variables in the formula. Form successive n-tuples of members of the Herbrand universe, exhausting all possible combinations of those elements already generated before generating a new element.

2. Substitute an n-tuple for the variables in the free variable form and form the disjunction of this sentence with that already accumulated.

3. If this formula is a tautology the original theorem is valid, if not, then select a new n-tuple and repeat the process.

Every formula which is valid will result in a tautology after some finite number of terms have been accumulated. There is, of course, no effective procedure for estimating the number of terms necessary in the general case. However, for certain forms of the original formula upper bounds can be set; these are the decidable cases.

The explanation of the Herbrand proof procedure in Davis and Putnam [P8] is far more satisfying intuitively than that which I give here. I recommend that the reader study that paper for the feel it gives for what is really going on. My purpose here is to summarize the process in terms

3. If x is a negative variable occurring in R, substitute $f_i(x_1, x_2, x_3, \ldots)$ for it everywhere it occurs, where f_i is a previously unused function symbol and the x_i are all the variables which were bound by a positive quantifier whose scope extended over R.

The resulting free variable form of the formula has no quantifiers, and may have some new function symbols.

1.1.4. The Herbrand Universe

The Herbrand universe, $\underline{H}$, is formed from the function symbols and individual constants appearing in the free variable form. Included among the individual constants for this purpose are these functions of no arguments which appear. If there are no constants of either variety then the single constant "a" is taken. These elements form the 0th level of the hierarchy which I will denote H_0.

The r + 1th level, H_{r+1}, is the set of all expressions of the form $f(K_1, K_2, \ldots)$ where f is a function symbol occurring in the fvf, and the K's are members of one of the sets H_j, $j \leq r$, and at least one of the K's is of level r.

H is just $\cup_{j=0}^{\infty} H_j$.

[Note that H_n $n > 0$ may be empty.]

1.1.5 The Proof Procedure Programs

All the proof procedure programs described in the literature so far use essentially the method described above: a growing propositional expression is generated and tested. The primary differences of these programs

Gilmore's program also included methods for recognizing when a formula was in certain decidable domains and would compute an upper bound to the number of instances of the fvf which must be investigated, in order to be able to establish that invalid formulas falling in this class were non-theorems.

1.1.5.2. The Davis-Putnam Algorithm

Davis and Putnam made a major improvement to proof procedure programs [P8, P9]. They deal with the negated fvf in conjunctive normal form. Although somewhat complicated reduction rules are necessary to recognize that the propositional formula is false when it is expressed this way, the addition of a new instance of the fvf to the accumulated expression doesn't involve any more work than simply "tacking it on". The resulting expression is still in conjunctive normal form. The reduction rules, while complicated, require a computation time which grows only linearly with the length of the expression.

1.1.5.3. The Program of J. A. Robinson

No improvement in proof procedures has come to my attention past the program of Davis and Putnam. (Note: I have not yet studied the papers of Joyce Friedman [P11, P12]). J. A. Robinson [P10], while presenting the problem in an enlightening fashion which will be discussed fully below, really has no heuristic which will speed up or reduce the search for a proof. He also proposes a proof procedure which is essentially an unjustified reordering of the usual search (although he doesn't seem to realize this). I will deal more fully with the question of reordering the search in the next section.

It will be evident from the examples in the second section that proofs in the Herbrand theorem form and some more familiar form such as in a natural deduction system are readily comparable, and indeed could be translated either one to the other by a suitable (complex but efficient) algorithm. This fact was pointed out by A. Robinson [P1], and elaborated on by J. A. Robinson [P10], who has written a program which translates a Herbrand Expansion into a deductive proof.

This completes my precis of the Herbrand type programs. I have not mentioned several works here, because they are essentially only trivial variations of the themes which have been presented. The papers of Wang [P3, P4, P5], however, contain much more than descriptions of proof procedures, and I recommend that they be read carefully.

1.2. Subgoal-Reduction Method

In general when embarking on a search it is advisable to choose carefully the order in which possibilities are examined. The important question of how much time should be spent planning is one of economics, and must be decided on this basis. It is in this matter as to how much effort should be alloted to ordering the search that the two groups of machine theorem provers are at odds. An important point which is not generally recognized is that the subgoal reduction programs are searching essentially the same space as those which take the Herbrand approach, but spend more time deciding where to look next.

1.2.1. The Logic Theory Machine

The LTM papers [H1, H2] represent the most significant results so far achieved in the field of artificial intelligence. Three key ideas to be found here are subgoal reduction, list structure programming technique, and recursive subroutine capability. Unfortunately the authors' logic was unsophisticated, and I feel this has caused most logicians working in the machine theorem-proving area to discount the powerful techniques of heuristic programming which have sprung from this work.

NSS seemed to imply that a program that would examine more likely proofs of a formula first, would lose cognizance of some possible proof sequences and cease to be a proof procedure. Indeed, their program did have this property, not because they used search reducing heuristics but because their ad hoc logic was incomplete. From a practical standpoint the whole issue is immaterial since computer limitations prevent any program from being a true proof procedure.

1.2.1.1. Subgoal Reduction

The LTM, a program to prove theorems in propositional calculus, uses the technique of separating a problem into pieces and attacking each in the same fashion as the whole. Eventually, it is hoped, the problem will be reduced to a large number of "atomic" problems which the machine can deal with as wholes. This is the process I call subgoal reduction.

The LTM uses five axioms and three rules of inference. The latter are:

1. Substitution: any proposition may be substituted for a propositional variable.
2. Replacement: an expression may be replaced by its definition.
3. Modus-ponens: from p and $p \supset q$ infer q.

Actually, instead of (3) they use "chaining", a metatheorem based on (3).

1.2.1.2. Working Backwards

In the LTM the initial goal is the formula to be proved, and the method of reducing the problem to pieces is to find one or two axioms or previously proved theorems which can be transformed into the problem at hand by one of the three rules of inference. Upon application of modus ponens the premise usually becomes a new subgoal. The process begins with the conclusion of the theorem as the initial problem and ends with atomic problems which are axioms and premises; thus it is termed working backwards.

1.2.1.3. Matching

The process of working backwards requires that a means be available to determine whether a formula is an instance of some schema. LTM employed a matching process for this purpose. This is a recursive search procedure which will determine what substitutions must be made for variables in the schema to convert it to the specific formula desired.

1.2.2. Gelernter's Geometer

Gelernter's geometry program's [H4, H5, H6] overall organization is the same as the LTM. It uses all the techniques described above which were developed for the LTM, such as subgoal reduction and working backward. It eventually proved moderately difficult theorems at the level of high school geometry.

From a private discussion with Gelernter, I have the impression that the program utilizes what is equivalent to a negation-and-quantifier-free logic. (A system in which neither denials nor quantifiers appear, but with the interpretation that free variables be regarded as universally quantified.) The fact that certain statements are not expressible in this calculus is reflected in a limitation on the kinds of proof methods which Gelernter's program can use. For example, it cannot make a reductio ad absurdum argument.

1.2.2.1. The Diagram

The most important heuristic included in Gelernter's program (due to Minsky) was the interpretation of formulae in an example diagram. Since most false propositions in geometry are false in a specific example (such as the proposition "Two arbitrary lines form a right angle"), an invalid modus ponens step ($p \supset q$ where p is false) can be eliminated as a means of deriving q without searching for a proof of its premise, simply by consulting the diagram.

This principle can be extended to other domains of theorem proving besides logic. A. Robinson proposes a theorem prover in number theory [H7] which tries out a number of instances of a proposed formula to attempt to refute it before trying to prove it. This process could have been applied as a heuristic in LTM, where the machine could have been assigned random truth values to propositional variables, and evaluated the resulting Boolean expressions in an attempt to refute an inference. Indeed, this might be a good heuristic to add in Davis and Putnam's proof procedure program.

1.2.2.2. Tree Pruning Heuristics

The subgoal reduction method of problem solving generates a branching tree structure of subproblems, called the goal tree, whose extremities represent atomic problems. Sometimes whole limbs of this tree can be eliminated because for some reason they really don't help to solve the problem. There are several techniques for finding the redundant goals; these are called "tree pruning" heuristics.

1.2.2.2.1. Interpretation of the Goal Tree as a Propositional Formula

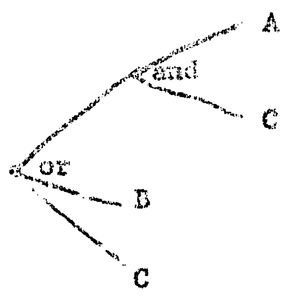
The intermediate branch points of the goal tree indicate an "and" or an "or" type relation; i.e., all the subproblems eminating from an "and" branch must be solved to achieve the goal for which the branch point stands, whereas only a single subproblem's solution suffices for the achievement of "or" branch points.

This similarity between goal tree and logical relations suggests interpreting the goal tree as a propositional formula.

A propositional variable corresponding to each goal at the tip of a branch is to stand for the proposition that that goal has been achieved and the propositions "and" and "or" are to have their normal interpretations. If the propositional expression so derived can be shown to be true, then the goal at the root of the tree is achievable.

Certain algebraic transformations on the propositional formula correspond to simplifying the goal tree.

These simplifications consist of the elimination of goals which are redundant. For example, if the goal tree is:



then the propositional formula corresponding to the tree is $B \vee AC \vee C$. This can be simplified to $B \vee C$. In this case, A is a redundant goal. The problem of eliminating redundant literals in a propositional expression involves finding the prime implicants of a formula, and is treated in the switching circuit literature [M3, M4, M5].

Gelernter's program uses these Boolean function reductions on his goal tree (though without recognizing this nice interpretation).

A great number of comparisons need to be made to recognize the simplifications. In general some problems which require little effort for their straightforward solution will not warrant the search to see if they are redundant.

1.2.2.2.2. Syntactic Symmetry

The notion of syntactic symmetry is a valuable heuristic used in Gelernter's program. It is a formalization of the informal "and such-and-such follows similarly".

A syntactic symmetry is a permutation operator on the names of the variables in a formula which leaves the formula unchanged except for rearrangement. If π is a syntactic symmetry of H then $\pi(H) \equiv (H)$ is valid. The utility of this concept depends on this theorem: if K is provable from H and π is a syntactic symmetry of H then $\pi(K)$ is also provable from H. Having once computed π and proved $H \supset K$, the theorem

$H \supset \pi(K)$ is free.

If let run unchecked a problem solving program might well be expected to eventually generate P, or some formula which is syntactically symmetric to P, as a subgoal in establishing P. The program would probably then waste a lot of effort before trying some other line of attack. One way to circumvent this trouble is to make certain that no subgoal which is identical to or syntactically symmetric to some goal dependent on it is expanded.

The main trouble in the application of this principle is that not only must the ss's of a formula which one hopes will reappear be computed and stored, but also all newly generated formulae must be compared with all the old formulae and their ss's to detect a reoccurrence. Storage and running time requirements demand a careful regulation of the use of this principle. Its unrestricted application was probably an inhibiting factor in Gelernter's program.

1.2.2.3. J. A. Robinson's Paper

The work of J. A. Robinson [P10] describes the Herbrand proof procedure in a manner which points out the essential components of a proof arrived at by this process. These are the key concepts of a proof set and a verification set. Using these notions I will attempt to point out in Section 2, how to combine the best ideas in each of the two approaches to mechanical theorem proving.

1.2.2.3.1. Proof Set

Let us consider the fvf in disjunctive normal form. I will use a matrix such as:

$$\begin{bmatrix} L_{11} & L_{12} & \cdots & L_{1m_1} \\ L_{21} & L_{22} & \cdots & L_{2m_2} \\ \vdots & & & \\ L_{k1} & L_{k2} & \cdots & L_{km_k} \end{bmatrix}$$

to represent the fvf in disjunctive normal form, where the L's are literals and the logical expression

$$(L_{11} \vee L_{12} \vee \ldots) \wedge (L_{21} \vee L_{22} \vee \ldots) \wedge \ldots$$

is its interpretation.

A proof of a theorem consists of a tautologous disjunction of substitution instances of the rows of L, where the objects substituted are elements of H. (Notice that a substitution instance of a single row may be included without the same substitution in the rest of L necessarily being included.) A proof set is a set of elements from H which results in a tautology upon complete instantiation of L (substitution instances of all possible legal n-tuples) and such that this property is lost if any element is deleted from the set.

A verification set of conjunctions is a set of instances of rows of L whose disjunction is tautologous, and such that if any conjunct is deleted, the disjunction is not tautologous. The Herbrand proof procedure is really a search for a verification set (which defines an accompanying proof set). The procedure calls for a complete search of the solution space of instantiations of rows of L to locate a verification set. For theorems of common interest the verification set is quite small (usually contains less than 100 elements), compared to the astronomical size of sets generated by instantiating the fvf over all the elements of H whose level is less than or equal to that of the appropriate proof set.

J. A. Robinson [P10] has written a program which searches for a verification set when given a proof set. Although he gives no times for the search which is made in this case, I believe it must be quite slow, when operating on moderately large problems, for reasons I will state in Section 2.

1.2.2.4. Summary

We have seen that the Herbrand expansion proof procedure is a search for a small set of substitution instances of rows of the fvf matrix. The proof procedures which have used this expansion have attempted to find this set by substituting successive elements from a fixed ordering of H. This technique has failed on large problems because the elements of H which are needed come from too far along in the enumeration. Methods have been developed which reorder the enumeration to suit the theorem. These are the heuristic techniques of LTM.

Thus far they have been used with the greatest effectiveness in Gelernter's geometer. That this work did not attempt problems of a very difficult level will be the contention of the next section.

2. Proposal for a Theorem Prover

In this section I propose a program for geometry theorem proving which will be attacking what I believe to be the obstacle which is immediately before us on the way toward mechanical theorem proving: the selection of a proof set. I believe that the solution of this problem will then permit work on the much more difficult one of generating significant and true conjectures as candidate theorems to be proved. I think that another problem of considerable importance might possibly be solved by the device next to be discussed; this is the problem of deciding what already proved theorems to "remember" to use as lemmas in further work.

2.1. The Order of an Element of H

The notion of the order of an element of H is closely related to that of level. H is again regarded as structured hierarchically by order, but the subclasses are slightly different than the levels.

A composite form is a term of the form:

$\theta[A_1; A_2; \ldots A_n]$

where θ is a function symbol, and the A's are either individual variables, constants, or composite forms.

The 0th order of H is the set consisting of:

1. individual constants appearing in the fvf
2. composite forms appearing in fvf containing no individual variable symbols.

H^{r+1}, the r+1th order of H, is the set of all expressions of the following forms, which haven't appeared in a lower order:

1. $\phi[A_1; A_2 \ldots A_n]$

   where ϕ is a function symbol appearing in the fvf and the A's are members of H^i for $i \leq r$ and at least one of the A's is from H^r.

2. $\Theta[A_1; A_2 \ldots A_n]$

where $\Theta[A_1; A_2 \ldots A_n]$ is a composite form appearing in the fvf with elements of H^i for $i \leq r$ substituted for its free variables, and at least one of the substituted elements is from H^r.

It is easy to show that $H = U_{j=0}^{\omega} H_j = U_{j=0}^{\omega} H^j$ justifying the phrase "nth order of H" since by the definition of elements of H^i all elements of H_i are contained in $U_{j=0}^{i} H^i$ so $H \subseteq U_{j=0}^{\infty} H^j$; and every element of H^i appears in $U_{j=0}^{i \cdot k} H_j$ where k is the maximum depth of nesting of function symbols in the fvf.

2.2. The Order of a Proof

Let us define the order of a proof of a formula in Herbrand expansion form as the order of the highest element of H which appears in verification set for the formula.

The concept of the order of a proof suggests itself as a measure of difficulty of a proof of a theorem. Of course, whether or not this is a good measure depends on what we consider to be difficult; it seems reasonable in the sense that it agrees with my own notions of what are more or less difficult proofs.

I will attempt to evaluate this measure's performance as a filter for significant theorems. I have hopes that it will be the key to enabling the computer to decide what theorems will be useful as lemmas.

2.3. The Interpretation of H in Geometry

In the case of theorems in geometry there is an interesting interpretation of the Herbrand universe. The elements of H whose order is 0 correspond to the parts of the diagram one draws when given a statement of a theorem. The first order of the hierarchy consists of all the elements which may be made in one application of some existential postulate to the elements of level 0. The higher orders are produced in a similar fashion from elements of the lower orders.

The fact is almost indisputable that those theorems in geometry which require more complicated added constructions for their proof are more difficult to prove. The order of an element in the added constructions can be estimated by finding the minimum number of operations (such as connecting two points, laying off a length) necessary to get it.

The examples to follow will help to justify and illustrate this point. First, though, it is necessary to develop some more formalism.

2.4. Many-Sorted Quantification Theory

In geometry we speak of many kinds of entities: points, lines, angles, circles etc. The use of a many-sorted quantification theory greatly facilitates proofs about systems of this kind.

Certain modifications must be made to my formalism to adapt it to a many-sorted theory. Individual variables and constants will henceforth be given a number of primes to indicate the sort of object they stand for, e.g., x'' will be an angle variable.

To the definition of wff we must add the restriction (the type substitution criterion) that each predicate symbol or function symbol appearing in a formula must have a consistent assignment of sort of variable to each argument position, that is, the same sort of object must appear in the nth argument position of ϕ in its every occurrence. In addition, each expression $\phi[A_1; A_2 \ldots]$ where ϕ is a given function symbol will always denote the same sort of individual.

2.5. Examples

The angles opposite the equal sides of an isoceles triangle are equal.

This theorem of elementary Euclidean geometry has several simple proofs [L1]. We shall examine two proofs of this theorem, only one of which requires a construction. The proofs will be given in the Herbrand expansion form. In doing this I hope to convince the reader of two things; (1) that the interpretation I give of H is correct, and (2) that the order of the proof set is a significant measure of the difficulty of a proof.

The two proofs require different lemmas, and only those actually necessary will be included among the postulates in each proof. (I intend the term postulate to mean either an axiom or a previously proved theorem.) The sorts we are dealing with in these proofs are: points, line segments, angles, and triangles. These will be sorts 0, 1, 2, 3 respectively. "$\triangle$" is a function from three points to a triangle, "C" is the predicate "colinear" and "$\angle$" a function from three points to an angle. The notation "xy" is to indicate a function which maps two points into a segment. "$\cong$" will denote all the relations of congruence (there can be no confusion about which one is meant).

2.5.1. Proof without Construction

In this example the theorem is proved by showing that the triangle is congruent to itself flipped over by SSS. The theorem in quantification theory is:

$$\begin{aligned}&\Big\{ (x)(y)(z)(u)(v)(w)[\triangle[x;y;z] \cong \triangle[u;v;w] \supset \angle[x;y;z] \cong \angle[u;v;w]] \wedge \\ &(x)(y)(z)(w)(v)(w)[\sim C[x;y;z] \wedge \sim C[u;v;w] \wedge xy \cong uw \wedge zy \cong wv \wedge \\ &xy \cong uv \supset \triangle[x;y;z] \cong \triangle[u;v;w]] \wedge (x)(y)(z)[\sim C[x;y;z] \supset \sim C[x;z;y]] \wedge \\ &(x)(y)[xy \cong yx]\Big\} \supset (x)(y)(z)[\sim C[x;y;z] \wedge xz \cong xy \supset \angle[x;y;z] \cong \angle[x;z;y]]\end{aligned}$$

The matrix derived from the fvf contains three functions of no arguments coming from the quantifiers in the conclusion of the main implication. I will write them as "a", "b", and "c". I have not rewritten bound variables since no conflicts are present.

$$\begin{array}{r|llll|}
1. & \triangle[x;y;z] \cong \triangle[u;v;w] & \sim(\angle[x;y;z] \cong \angle[u;v;w]) & & \\
2. & \sim C[x;y;z] & \sim C[u;v;w] & xz \cong ww & zy \cong wv \\
 & xy \cong uv & \sim(\triangle[x;y;z] \cong \triangle[u;v;w] & & \\
3. & \sim C[x;y;z] & C[x;z;y] & & \\
4. & \sim(xy \cong yx) & & & \\
5. & x' \cong y' & \sim(y' \cong x') & & \\
6. & C[a;b;c] & & & \\
7. & \sim(ac \cong ab) & & & \\
8. & \angle[a;b;c] \cong \angle[a;c;b] & & &
\end{array}$$

A single instance of each row, or 8 conjuncts, is sufficient to establish the theorem. These are:

$\triangle[a;b;c] \cong \triangle[a;c;b]$	$\sim\angle[a;b;c] \cong \angle[a;c;b]$	1: x/a y/b z/c u/a v/c w/b
$\sim C[a;b;c]$	$\sim C[a;c;b]$	ac$\cong$ab ab$\cong$bc ab$\cong$ac
$\sim\triangle[a;b;c] \cong \triangle[a;c;b]$		2: same
$\sim C[a;b;c]$	$C[a;c;b]$	3: same
$\sim cb \cong bc$		4: x/c y/b
$ac \cong ab$	$\sim ab \cong ac$	5: x'/ac y'/ab

and rows 6, 7, 8.

Abbreviating:

$\triangle[a;b;c] \cong \triangle[a;c;b]$	by	A
$\angle[a;b;c] \cong \angle[a;c;b]$	by	B
$C[a;b;c]$	by	C
$C[a;c;b]$	by	D
$ac \cong ab$	by	E
$cb \cong bc$	by	F
$ab \cong ac$	by	G

gives:

$$(A \wedge \sim B) \vee (C \wedge D \wedge E \wedge F \wedge G \wedge \sim A) \vee (\sim C \wedge D) \vee \sim F \vee (E \wedge \sim G) \vee C \vee \sim E \vee B$$

which is a tautology.

The 0th order elements of H for this proof are:

a b c ac ab $\angle[a;b;c]$ $\angle[a;c;b]$.

The 1st order elements are:

ba ca cb bc $\triangle[a;b;c]$ $\triangle[a;c;b]$ $\triangle[b;a;c]$ $\triangle[b;c;a]$ $\triangle[c;a;b]$ $\triangle[c;b;a]$ $\angle[b;a;c]$ $\angle[b;c;a]$ $\angle[c;a;b]$ $\angle[c;b;a]$.

The higher orders are vacuous.

Since the proof requires elements from the 1st order, it is a 1st order proof.

1. $y \neq z \quad \sim B[y;f[x;y;z];z]$
2. $y \neq z \quad f[x;y;z] = y$
3. $y \neq z \quad f[x;y;z] = z$
4. $y \neq z \quad \sim f[x;y;z]y \cong f[x;y;z]z$
5. $\sim C[x;y;z] \quad \sim C[u;v;w] \quad xz = uw \quad zy = wr \quad xy = uv \quad \sim \Delta[x;y;z] \cong \Delta[u;v;w]$
6. $u \neq y \quad u \neq z \quad \sim C[x;y;z] \quad C[u;y;z] \quad C[x;y;u]$
7. $u \neq y \quad u \neq z \quad \sim C[x;y;z] \quad C[u;y;z] \quad C[x;z;u]$
8. $B[y;u;z] \quad \sim C[u;y;z]$
9. $\Delta[x;y;z] \cong \Delta[u;v;w] \quad \sim \angle[x;y;z] \cong \angle[u;v;w]$
10. $\sim C[x;y;z] \quad y = z$
11. $B[y;u;z] \quad \sim \angle[x;y;u] \cong \angle[x;y;z]$
12. $B[y;u;z] \quad \sim \angle[x;z;u] \cong \angle[x;z;y]$
13. $x^2 = y^2 \quad u^2 = v^2 \quad x^2 = u^2 \quad y^2 \neq u^2$ [illegible]
14. $x' \neq x'$
15. $C[a;b;c]$
16. $\sim ab \cong ac$
17. $\angle[a;b;c] \cong \angle[a;c;b]$

The function symbol "f" which has been introduced into the fvf comes from the existential quantifier in the first postulate. Three constant symbols, "a", "b", and "c" come from the universal quantifiers in the statement of the theorem. I will abbreviate f[a;b;c] by s.

$b \neq c$ $\quad B[b;s;c]$	1. $x/a \; y/b \; z/c$
$b \neq c$ $\quad s = b$	2. same
$b \neq c$ $\quad s = c$	3. same
$b \neq c$ $\quad \sim sb \cong sc$	4. same
$\sim C[a;b;s] \quad \sim C[a;c;s] \quad as \cong as \quad sb \cong sc \quad ab \cong ac$	
$\sim \triangle[a;b;s] \cong \triangle[a;c;s]$	5. $x/a \; y/b \; z/s \quad a/a \; v/c \; w/s$
$s \neq b \quad s \neq c \quad \sim C[a;b;c] \quad C[s;b;c] \quad C[a;b;s]$	
	6. $x/a \; y/b \; z/c \; u/s$
$s \neq b \quad s \neq c \quad \sim C[a;b;c] \quad C[s;b;c] \quad C[a;c;s]$	
	7. $y/b \; u/s \; z/c$
$B[b;s;c] \quad \sim C[s;b;c]$	8. same
$\triangle[a;b;s] \cong \triangle[a;c;s] \quad \sim \angle[a;b;s] \cong \angle[a;c;s]$	
	9. $x/a \; y/b \; z/c \; u/a \; v/c \; w/b$
$\sim C[a;b;c] \quad b = c$	10. $x/a \; y/b \; z/c$
$B[b;s;c] \quad \sim \angle[a;b;s] \cong \angle[a;b;c]$	11. $x/a \; y/b \; z/c \; u/s$
$B[b;s;c] \quad \sim \angle[a;c;s] \cong \angle[a;c;b]$	12. same
$\angle[a;b;s] \cong \angle[a;b;c] \quad \angle[a;c;s] \cong \angle[a;c;b] \quad \angle[a;b;s] \cong \angle[a;c;s]$	25''
$\sim \angle[a;b;c] \cong \angle[a;c;b]$	13. $x^2/\angle[a;b;s] \quad y^2/\angle[a;b;c]$
	$u^2/\angle[a;c;s] \quad v^2/\angle[a;c;b]$
$as \cong as$	14. x'/as
$C[a;b;c]$	15.
$\sim ab \cong ac$	16.
$\angle[a;b;c] \cong \angle[a;c;b]$	17.

The disjunction of these row instances can be verified as tautologous.

In the matrix there are no true constants. There are, however, functions of no arguments: "a", "b", and "c". Among the composite forms are:

f[x;y;z]   f[x;y;z]y   xy   △[x;y;z]   ∠[x;y;z]

Elements of H which appear in the verification set are:

a   b   c   ab   ac   ∠[a;b;c]   ∠[a;c;b]

(the entire 0th order of H).

f[a;b;c] (abbreviated as "s"), a single 1st order element

and

as   sb   sc   △[a;b;s]   △[a;c;s]   ∠[a;b;s]   ∠[a;c;s]

from the 2nd order.

This is a 2nd order proof.

2.6. Finding a Proof Given a Proof Set

Since only the postulates which were actually necessary for the proofs were included in these examples, the Herbrand universe was much smaller than it would ordinarily be (even finite in the first example) and the problem of selecting which elements to work with in searching for a proof is correspondingly less difficult.

Gelernter's program is given a list of all the points, line segments etc. involved in the proof it is to generate, with one minor exception: the program can add a line segment which connects two points which it already knows about. Essentially, then, Gelernter's program searches for a proof already knowing the proof set (with the noted exception), i.e., those elements it is given along with the theorem to be proved.

J. A. Robinson has also written a program to find a proof given a proof set. My feeling is that this heuristically unsophisticated program which generates all substitution instances of the fvf to produce proofs of 0th order theorems would be incapable of proving theorems in a reasonable time when large numbers of postulates are available. The fact that his program proved some interesting theorems when the set of postulates was restricted (as in my examples) misleadingly seems to indicate that the task is easy. The fact that Gelernter's program (which I consider an able effort) takes many minutes to solve problems of comparable difficulty is also evidence that the problem is not trivial.

2.7. <u>Proposal: A Program to Find High Order Proofs</u>

I propose to write a program which will produce high order proofs in geometry. It will work by first selecting a set of elements which seem likely to be useful, and then apply the techniques which have been developed for finding a proof given a proof set to search for a verification set.

The goal I hope to reach is the ability to produce proofs of the theorems in <u>Foundations of Euclidean Geometry</u> [L1]. [L6 contains a formalization of some of L1.] This contains an axiomatic development of plane geometry up to about the level of Euclid's <u>Elements</u>.

There are three major areas in which my efforts must be spent. First, the 0th order proof heuristics must be translated so as to apply to formulas expressed in Herbrand's form. Second, a diagram generator must be programmed. This was never done by Gelernter, and is not a simple task. Last, and most difficult, will be the development of effective techniques for finding the set of elements necessary for a proof.

References

Heuristic Theorem Proving

H1. Newell, A., J. C. Shaw and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic," Proc. WJCC, 1957.

H2. Newell, A. and J. C. Shaw, "Programming the Logic Theory Machine," Proc. WJCC, 1957.

H3. Robinson, A., "A Basis for the mechanization of the theory of equations," Computer Programming and Formal Systems, North-Holland Pub. Co., Amsterdam, 1963.

H4. Gelernter, H., J. R. Hansen, and D. W. Loveland, "Empirical Explorations of the Geometry Theorem Proving Machine, Proc. WJCC, 1960.

H5. Gelernter, H. and N. Rochester, "Intelligent Behavior in Problem-Solving Machines," IBM J. Res. and Dev., 2, No. 4, October 1958.

H6. Gelernter, H., "Realization of a Geometry Theorem Proving Machine," Proc. Int'l Conf. on Inform. Processing (ICIP), UNESCO House, Paris, 1959.

H7. Machover, M. and A. Robinson, " On the Mechanization of the Theory of Numbers," USONR Inform. Sys. Br., Contract 62558-2214, Hebrew Univ. of Jerusalem. (ASTIA)

H8. McCarthy, J., "A Basis for a Math. Theory of Computation," Computer Prog. and Formal Sys., North-Holland Pub. Co., 1963.

H9. Abrahams, P. W., "Machine Verification of Mathematical Proof," Int'l Electric Corp., Tech. Rept. P-AA-TR-(0045), 1963.

H10. Mott, T. H., J. R. Guard, J. H. Bennett, and W. B. Easton, "Introduction to Semi-Automated Mathematics," Int'l Res. Inst., Final Rept. AF 19(628)-468.

References (cont'd)

Proof Procedure and Related Literature

P1. Robinson, A., "Proving a Theorem (as done by man, logician or machine)," Proc. of Summer Inst. of Symbolic Logic, Cornell Univ., 1957.

P2. Gilmore, P. C., "A Proof Method for Quantification Theory: Its Justification and Realization," IBM J. Res. and Dev., 4, No. 1, January 1960.

P3. Wang, Hao, "Toward Mechanical Mathematics," IBM J. Res. and Dev., 4, No. 1, January 1960.

P4. _____, "Proving Theorems by Pattern Recognition I," Comm. ACM, 3, April 1960.

P5. _____, "Proving Theorems by Pattern Recognition II," Bell Sys. Tech. Journal, 40, January 3, 1961.

P6. Prawitz, Dag, "An Improved Proof Procedure," Theoria, 26, pp. 102-139, 1960.

P7. Kanger, Stig, "A Simplified Proof Method for Elementary Logic," Comp. Programming and Formal Systems, North-Holland Pub. Co., 1963.

P8. Davis, M. and H. Putnam, "A Computing Procedure for Quantification Theory," J. ACM, 7, No. 3, July 1960.

P9. Davis, M., G. Logemann and D. Loveland, "A Machine Program for Theorem Proving," Comm. ACM, 5, No. 7, July 1962.

P10. Robinson, J. A., "Theorem-Proving on the Computer," J. ACM, 10, pp. 163-174.

P11. Friedman, J., "A Semi-Decision Procedure for the Functional Calculus," J. ACM, 10, No. 1.

P12. _____, "A Computer Program for a Solvable Case of the Decision Problem," J. ACM, 10, No. 3.