# REALIZATION OF A GEOMETRY-THEOREM PROVING MACHINE

*by H. Gelernter*

## *Introduction*

Few of those who have seen a modern high-speed digital computer digest and transform a mass of data in less time than it takes to follow the process in the mind can suppress a certain amount of speculation concerning the future of such machines. Under the assumption that the computer is operating at the mere threshhold of its capacity in performing the tasks we have thus far delegated to it, a long-range program directed at the problem of "intelligent" behavior and learning in machines has been established at the IBM Research Center in New York (Gelernter and Rochester, 1958). In particular the technique of heuristic programming is under detailed investigation as a means to the end of applying large-scale digital computers to the solution of a difficult class of problems currently considered to be beyond their capabilities; namely those problems that seem to require the agent of human intelligence and ingenuity for their solution. It is difficult to characterize such problems further, except, perhaps, to remark rather vaguely that they generally involve complex decision processes in a potentially infinite and uncontrollable environment.

If, however, we should restrict the universe of problems to those that amount to the discovery of a proof for a theorem in some well-defined formal system, then the distinguishing characteristics of those problems of special interest to us are brought clearly into focus. We should like our machine to be able to prove many of the theorems presented to it in a formal system that is manifestly undecidable. Further, as the machine

gains "experience" in proving theorems, we should expect it to be able to solve problems that were earlier beyond its capabilities.

The requirement that a machine should deal with undecidable systems places a fundamental restriction on its modus operandi. Finding a suitable algorithm, the obvious technique for the solution of problems on a digital computer, is no longer acceptable for the simple reason that no such algorithm exists. An exhaustive search for the initial axioms and theorems of the proof, combined with exhaustive development of the proof sequence by systematically applying the rules of transformation until the required proof has been produced, has been shown to be much too time-consuming for so simple a logic as propositional calculus (Newell, Shaw and Simon, 1957*a*). It is a fortiori out of the question for any of the more interesting logics. A remaining alternative is to have the machine rely upon heuristic methods, as people usually do under similar circumstances.

### *Heuristic Methods*

A heuristic method is a provisional and plausible procedure whose purpose is to discover the solution of a particular problem at hand. The use of heuristic methods by the human mathematician is quite well understood, at least in its less subtle forms. The reader is referred to the excellent two-volume treatise by Prof. G. Polya (1954) for a definitive treatment of heuristics and mathematical discovery. A machine that functioned under the full set of principles indicated by Polya would be a formidable problem-solver in mathematics, and would be well on the way toward satisfying Turing's requirements for a machine able to compete successfully in the "imitation game" (1950). Such a machine, however, lies in the indefinite future, for the art of instructing a computer is yet in too primitive a state to consider translating Polya into machine language. As a representative problem more in keeping with the present state of computer technology, we have selected the discovery of proofs for theorems in elementary euclidean plane geometry in the manner, let us say, of a high-school sophomore. This problem contains in relatively pure form the difficulties we must surmount in order to attain our stated goal. It must be emphasized that although plane geometry will yield to a decision algorithm, the proofs offered by the machine will not be of this nature. The methods developed will be no less valid for problem-solving in systems where no such decision algorithm exists.

Although we have narrowed the scope of our study to include only those machines that deal with formal systems, there is ample justification for such a restriction. First, the concept of a problem is now well defined, as is the concept of a solution for that problem. Second, our ultimate goal stands clearly before us; it is the design of an efficient theorem-prover in some un-

decidable system. And, finally, just as manipulation of numbers in arithmetic is the fundamental mode of operation in contemporary computers, manipulation of symbols in formal systems is likely to be the fundamental operating mode of the more sophisticated problem-solving computers of the future. It seems clear that while the problems of greatest concern to lay society will be, for the most part, not completely formalizable, they will have to be expressed in some sort of formal system before they can be dealt with by machine.[1]

Our problem, then, is a statement (or string) in some formal logistic system. A solution for the problem will be a sequence of statements, each of which comprises a string of symbols of the alphabet of the system. The last string of the solution will be the problem itself; the first will always be an axiom or previously established theorem of the system. Every other string will be immediately inferable from some set preceding it or will itself be an axiom or previously established theorem.[2] It is the task of the machine to choose from its stock of axioms and theorems the appropriate ones for the base of the proof, and to generate from these the remaining strings necessary to complete the proof.

The problem of theorem-proving is, in a sense, of a particularly simple nature. Once a sequence of expressions is found that passes the test for a proof of the theorem (such a test always exists), one may, so to speak, "close the book" on that problem, provided that no stipulations have been made concerning the elegance required of the proof. But, basing our estimate on the work of Newell, Shaw, and Simon (1957), any computer extant would require times of the order of a thousand years to prove a not uncommon ten-step geometry theorem by exhaustively developing sequences until one emerged that passed the test for a proof. What is clearly called for is a technique for generating sequences with a much higher *a priori* probability of being the solution to the problem than those generated by an exhaustion algorithm.

As did the *Logic Theorist* of Newell, Shaw, and Simon, the geometry machine relies upon the well-known analytic method to achieve this end. By working backward, the machine is assured that every sequence it considers does indeed terminate in the required theorem. This in itself, however, represents no striking improvement over exhaustion without additional heuristics, for the advantages of working backward are purchased at a steep price; each sequence generated, while terminating properly, is no longer guaranteed to be a proof of anything at all. Indeed, most of the strings generated in this way will be false! But it is here that the great

[1] For a critique of some attempts to formalize scientific, but nonmathematical theories, see Dunham, Fridshal, and Sward (1959).

[2] The machine will use the deduction theorem to get $\vdash \{\text{premises}\} \supset \{\text{conclusions}\}$ from $\{\text{premises}\} \vdash \{\text{conclusions}\}$.

power of the analytic method lies, for if one could find a way of making their falseness manifest, such sequences could be immediately rejected, allowing most of the deadwood to be pruned away from the highly branched problem-solving tree. The set of sequences generated under such a process would contain fewer members by many orders of magnitude by the time the search reached any depth, and the density of possible proofs for the theorem among them would be proportionately greater. It is here, too, that the geometry machine finds the additional theorem-proving power necessary for the complex formal system assigned to it; theorem-proving power that was not necessary, and therefore not sought for in the propositional calculus machine of Newell, Shaw, and Simon (Polya, 1954). Like the human mathematician, the geometry machine makes use of the potent heuristic properties of a diagram to help it distinguish the true from the false sequences. Although the diagram is useful to the machine in other ways as well, the single heuristic "Reject as false any statement that is not valid in the diagram" is sufficient to enable the machine to prove a large class of interesting theorems, some of which contain a certain trivial kind of construction.

Before examining the internal structure of the geometry machine in some detail, we remark on two fundamental, if obvious, principles that must guide the choice of heuristics for any problem-solving machine. A heuristic is, in a very real sense, a filter that is interposed between the solution generator and the solution evaluator for a given class of problems. The first requirement for such a filter is a consequence of the fact that its introduction into the system is never costless. It must, therefore, be sufficiently "nonporous" to result in a net gain in problem-solving efficiency. Secondly, a heuristic will generally remove from consideration a certain number of sequences that are quick and elegant solutions, if not indeed all solutions, to some potential problems within the domain of the problem-solving machine. The filter must, then, be carefully matched to that subclass of problems in the domain containing those that are considered "interesting," and are therefore likely to be posed to the machine. For a given class of heuristics, the balance between these essentially opposing requirements is largely a function of the organization and computing power of the machine, and can under certain rather easily attainable conditions be quite critical. In the case of the *Logic Theorist*[3] experiments with varying "strengths" of a particular heuristic (the similarity test) indicated that the optimum porosity of that heuristic varied markedly with the length of the

[3] The designers of the *Logic Theorist* were not unaware of this heuristic device. In a later version of that machine, they did, in fact, include some syntactic heuristics to reject false subgoals. To use a semantic interpretation of the propositional calculus (a truth table, for example) for this purpose would have reduced the *Logic Theorist* to triviality.

problem and the number of theorems already established in the theorem memory, a consequence of the limited storage capacity of the computer.

### *The Geometry Machine*

With the object of our research program clearly determined, there were a number of specific alternatives to theorem-proving in Euclidean geometry that might have been adopted as a test problem; the evaluation of indefinite integrals, for example, or theorem-proving in the pure functional calculus. The decisive point in favor of geometry was the great heuristic value of the diagram. The creative scientist generally finds his most valuable insights into a problem by considering a model of the formal system in which the problem is couched. In the case of Euclidean geometry, the semantic interpretation is so useful that virtually no one would attempt the proof of a theorem in that system without first drawing a diagram; if not physically, then in the mind's eye. If a calculated effort is made to avoid spurious coincidences in the figure, one is usually safe in generalizing any statement in the formal system that correctly describes the diagram, with the notable exception of those statements concerning inequalities. Further geometry provides illustrative material in treatises and experiments in human problem-solving. It was felt that we could exchange valuable insights with behavioral scientists during the course of our research. In any event, elementary Euclidean geometry is comprehensible to every segment of the scientific community to which we should wish to communicate our results. Finally, it should not be a difficult task to generalize our machine to include the more interesting case of the non-Euclidean geometrics. A program of the same theorem-proving power as our Euclidean theorem-prover should be sufficient to prove a large class of non-obvious theorems in non-Euclidean geometry. A machine furnished with a non-Euclidean diagram (no more difficult to supply than the Euclidean one in suitable analytic form) encounters none of the assault on rationality experienced by a human mathematician searching from some heuristic insight into a theorem by considering a non-Euclidean diagram.

The formalization of geometry must be carried out within the framework of the lower functional calculus. Since we are interested in having the machine produce proofs comparable to those of a high-school student, we have preferred to construct a more or less *ad hoc* system following the scheme of most elementary texts, rather than to adopt as a primitive basis the fundamental axiomatization of Tarski, Hilbert, or Forder. No attempt has been made to provide a formalization that is either complete or non-redundant. If at some later time, the machine is able to prove one axiom from the others, that axiom will be discarded and we shall applaud the elegance displayed by our automaton. With regard to completeness, the

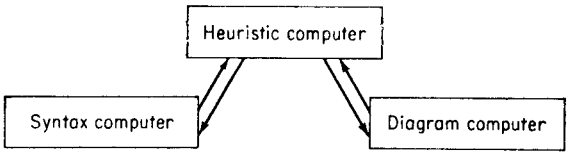


Figure 1.

machine is granted the same privileges enjoyed by the high-school student who is always assuming (*i.e.*, introducing as additional axioms) the truth of a plethora of "obviously self-evident" statements concerning, for example, the ordering properties of points on a line and the intersection properties of lines in a plane. Some of these statements are indeed independent of his original axioms, and must be introduced to complete the system. Most could be derived (but usually with some difficulty) from what he already has. There is nothing essentially wrong with this procedure of extracting assumptions from the model, provided that one is fully aware that this is being done (of course, this is rarely the case for the average student), and it simplifies the proof considerably without invalidating it. The geometry machine explicitly records its assumptions for a given proof. It could, if necessary, minimize the danger that it is proving a specific instance of a given theorem by drawing alternate diagrams to test the generality of its assumptions.

The geometry machine is in reality a particular state configuration of the IBM 704 electronic Data Processing Machine specified by a rather long and complex program written for the computer. Its organization falls naturally into three parts: a *syntax computer* and a *diagram computer* both embedded in an executive routine, which is a *heuristic computer*. The flow of control is indicated in Fig. 1.

Manipulation of the formal system is relegated to the syntax computer, which has within it the equivalent of most of the syntactic heuristics used by the *Logic Theorist*.[4] The diagram computer contains a coordinate representation of the theorem to be established together with a series of routines that produce a qualitative description of the diagram. It is important to point out that although the procedures of analytic geometry are used to generate the description, the only information transmitted to the heuristic computer (there is no direct link between the diagram and the formal system) is of the form: "Segment AB appears to be equal to segment CD in the diagram," or "Triangle ABC does not contain a right angle in the diagram." The behavior of the system would not be changed if the diagram computer were replaced by a device that could draw figures on paper and scan them.

[4] The process of *chaining* as defined by Newell et al. is under the control of the heuristic computer.

The major function of the heuristic computer for our first system, the subject of this report, is to compare strings generated by the syntax computer (working backward) with their interpretation in the diagram, rejecting those sequences that are not supported by the model. In addition to the above, the heuristic computer performs several other tasks. Among these are the organization of the proof-search process and the recognition of the syntactic symmetry of certain classes of strings. The latter function produces behavior equivalent to that of the human mathematician who, when A and B are syntactically symmetric, and both must be established, will merely prove A, and say "Similarly, B." It is an important feature, and is described in detail in an earlier report (Gelernter, 1959*a*). The procedures above are clearly independent of geometry; they are applicable to any formal system with its corresponding interpretation. The heuristic computer applies some additional semantic heuristics that are not indepedent of geometry. These may be "switched off" so that the behavior of the machine can be observed with and without specific geometry heuristics.

The character of the theorem-proving machine is determined largely by the heuristic computer. Modifications and improvements in the system (the introduction of learning processes, for example) will be made by modifying this part of the program.

Our first system does not "draw" its own initial figure, but is, instead, supplied with the diagram in the form of a list of possible coordinates for the points named in the theorem. This point list is accompanied by another list specifying the points joined by segments. Coordinates are chosen to

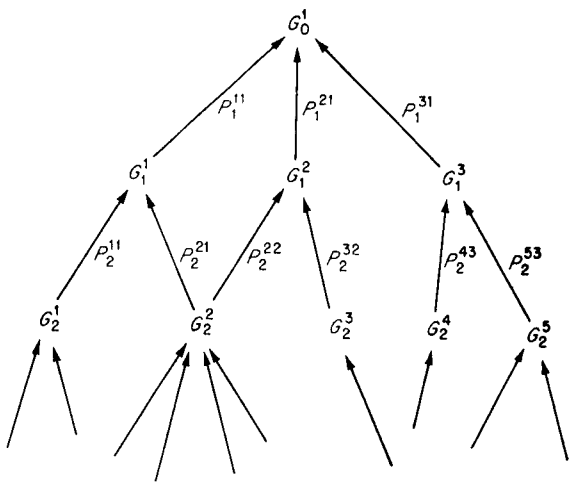


Figure 2. Problem-solving graph. The nodes $G_i^{\alpha}$ represent subgoals of order $i$, with $\alpha$ numbering the subgoals of a given order. $P_i^{\alpha\beta}$ is a transformation on $G_i^{\alpha}$ into $G_{i-1}^{\beta}$.

reflect the greatest possible generality in the figures. Later systems will construct their own interpretation of the premises, but since most problems for high school students are accompanied by a diagram, it was felt that we could dispense with this additional spate of programming at the current stage. When the machine is drawing its own figures, points will be chosen at random, subject to the constraints of the premises.

In working backward, the system generates a problem-solving graph, defined in the following way: Let $G_0$ be the formal statement to be established by the proof. It will be called the problem goal. If $G_i$ is a formal statement with the property that $G_{i-1}$ may be immediately inferred from $G_i$, then $G_i$ is said to be a subgoal of order i for the problem. All $G_j$ such that $j < i$ are higher subgoals than $G_i$, where $G_0$ is considered to be a subgoal of order zero. The problem-solving graph (Fig. 2) has as nodes the $G_i$, with each $G_i$ joined to at least one $G_{i-1}$ by a directed link. Each link represents a given transformation from $G_i$ to $G_{i-1}$. The problem is solved when any $G_i$ can be immediately inferred from the premises and axioms. If, as is generally the case in geometry, a given subgoal is a conjunction of statements, the graph splits at that point, and each parallel subgoal must be separately established. At any given time, the problem-solving graph is a complete representation of the current status of the proof-search process.

The organization of the heuristic computer (which is also the organization of the entire system) is displayed in greatly simplified form in Fig. 3. The diagram and syntax computers are accessible as subroutines to the heuristic computer. In operation, the machine executes the following processes (numbered below to correspond with like-numbered blocks in the flow chart).

1. The diagram is scanned to construct three lists, one containing every segment in the figure, one the angles, and one the triangles. Each element on a list is followed by a sublist describing that element.

2. The initial configuration of the system is set up, with the premises placed on a list of established formulas, and the conclusion on the problem-solving graph as a zero-order subgoal.

3. Definitions of nonprimitive predicates in the premises are added to the list of established formulae.

4. A subgoal to be established (the generating subgoal) is chosen from the problem-solving graph.

5. Appropriate axioms and theorems are selected from the theorem memory and, by working backward, a set of lower subgoals is generated such that if any one of these is established, the generating subgoal may be established by modus ponens and the generating axiom (or theorem). If the generating subgoal was labeled "provisionally fruitless" (see step 8), constructions are attempted (see below, p. 144).

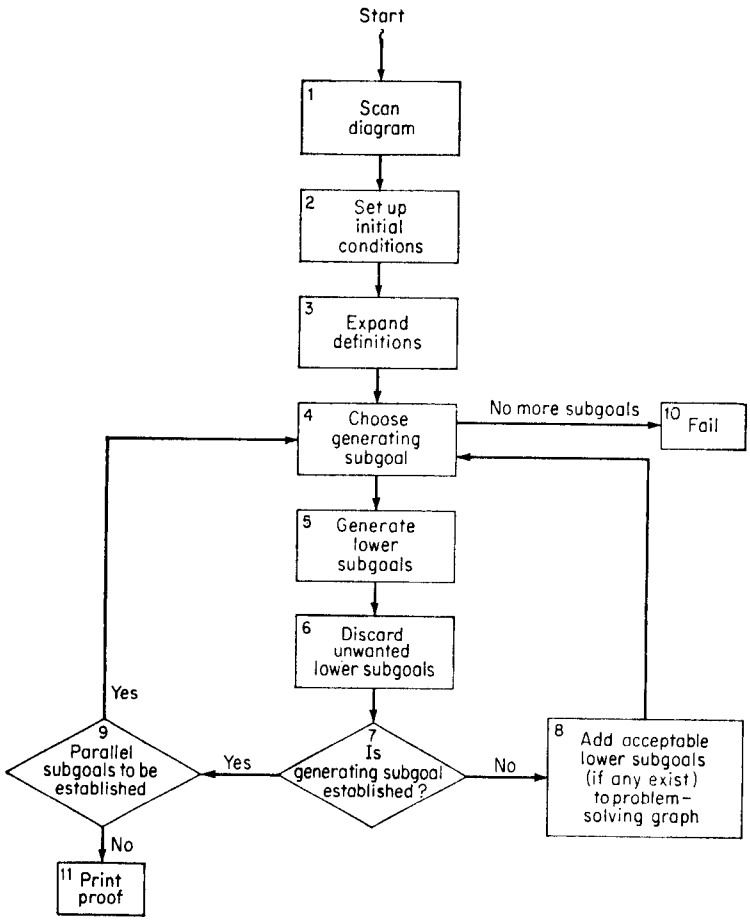


Figure 3. Simplified flow chart for the geometry-theorem proving machine.

6. Subgoals that are not valid in the diagram are rejected, as are those that appear as higher subgoals on the graph (or are syntactically symmetric to some higher subgoal).

7. If any lower subgoal is valid by virtue of its instance on the list of established formulas or if it may be assumed from the diagram, the generating subgoal is established; otherwise—

8. Acceptable nonredundant lower subgoals are added to the graph, and a new subgoal generator is chosen (4). If there are no acceptable lower subgoals and a construction is possible at this point, the generating subgoal is designated as provisionally fruitless. If a construction is not possible, or if the machine has tried and failed to find one, the generating subgoal is designated as fruitless.

9. If the generating subgoal is established, it is added to the list of established formulas, together with all of its higher consequences as determined by the graph. If there are no parallel subgoals remaining to be established, the machine reconstructs the proof from the problem-solving graph and prints it (11).

10. If at any time, every free subgoal on the graph is fruitless, the machine fails, providing it has not previously exhausted its available storage or the patience of the operator.

It is within blocks 4, 5 and 6, where subgoals are chosen, developed, and discarded, that the major heuristics reside. These subprograms represent, if you like, the seat of our artificial intelligence.

### *Some Early Results*

The geometry machine is able to prove many of the theorems within the scope of its ad hoc formal system using the diagram only to indicate which subgoals are probably valid. In this way, the following theorem is proved in less than a minute.[5]

Theorem: A point on the bisector of an angle is equidistant from the sides of the angle (see Fig. 4 in Appendix A).

In less than five minutes, the machine is able to find the attached proof, which requires the construction of an auxiliary segment.

Theorem: In a quadrilateral with one pair of opposite sides equal and parallel, the other pair of sides are equal (see Fig. 5 in Appendix B).

Although the introduction of a new element by the machine is impressive, the construction in this proof is essentially trivial, for the new segment merely joins two already existing points. It was discovered by the following process. In attempting to develop subgoals for the string "AB = CD," the machine could find none that were valid in the diagram. The normal procedure at this point is to seek an alternative path on the problem-solving graph. But when none is available (as was the case here, since the offending string is a zero-order subgoal), the machine reexamines those of the previously rejected subgoals containing instances of predicates for which there was no representation in the diagram. The machine then considers for each one an augmented set of premises such that the

[5] In the proofs displayed herein, the nonobvious predicates have the following interpretations:

| | |
|---|---|
| OPP-SIDE XYUV | Points X and Y are on opposite sides of the line through points U and V. |
| SAME-SIDE XYUV | Points X and Y are on the same side of the line through points U and V. |
| PRECEDES XYZ | Points X, Y, and Z are collinear in that order. |
| COLLINEAR XYZ | Points X, Y, and Z are collinear. |

interpretation does contain a representation of the predicate. If the string is valid in the augmented system, and there exist theorems permitting the required additional premises to be derived from the original set, then the string becomes a subgoal in the augmented system. The added premises specify a construction in the diagram that is permitted by virtue of the theorems through which they were derived. Returning to our example, the subgoal "$\Delta ABD \cong \Delta CDB$" is generated for the string "AB = CD," but the required triangles are not represented in the diagram until the premise "Segment BD exists" is added. The axiom "Two distinct points determine a segment" justifies the construction.[6] The entire process is a variant of the major heuristic above, and is clearly independent of the particular formal system under consideration. Note, too, that the process is finite, since no new points are introduced into the predicates; the old ones are merely reconsidered.

Our second example illustrates one further point. Although it is clear in the diagram (Fig. 5) that the transversal BD makes alternate interior angles with sides BC and AD, this is a consequence of the theorem "Opposite vertices of a convex quadrilateral fall on opposite sides of the diagonal through the other vertices." That this is not true of a general quadrilateral becomes clear when one considers the outside diagonal of a reflex quadrilateral. A completely rigorous solution, then, requires that one prove the lemma above if it is not already available, and that one demonstrate that the quadrilateral ABCD can only be convex. Rather than do this, the machine makes the usual assumption that the diagonal forms alternate interior angles with the opposite sides of the quadrilateral. Unlike the usual high-school text, however, the assumption is made explicit in the proof.

The theorem-proving system described thus far is adequate for many problems of greater complexity than the ones cited above. However, with a linear increase in the number of individual points mentioned in the premises, the rate of growth of the problem-solving graph increases exponentially and the time required to explore the graph increases correspondingly. If the machine were able to select those among a given set of subgoals that were more likely to lead to a solution, much of the wasted search time could be eliminated. Two specific geometry heuristics have been introduced to enable the machine to do this. The first is a routine that recognizes certain of the subgoals that are usually established in just one step. Identities are in this category, for example, as are equalities between angles that are observed to be vertical angles in the diagram. Such subgoals

[6] Our ad hoc formal system requires that the segments joining the vertices of a triangle be specified, as well as the vertices themselves, to define the triangle. This is necessary in order to avoid the difficulties that would otherwise arise when the theorem names a large number of noncollinear points. If our formal system were a true point geometry, all such constructions would be implicit in the diagram.

are placed on a priority list and developed before any of the others are considered. The second specific heuristic is a routine that assigns a "distance" between each subgoal string and the set of premise strings in some vaguely defined formula space. After those on the priority list have been developed, the next subgoal chosen is that which is "closest" to the premise set in formula space.

It is instructive to examine the machine's behavior in proving complex theorems both with and without the expanded set of semantic heuristics. For the theorem "Two vertices of a triangle are equidistant from the median to the side determined by those vertices," the machine finds a proof in about eight minutes with the basic heuristics alone (see Fig. 6 in Appendix C). The expanded set of heuristics produces a proof in one minute. In addition, the second proof is quite short and to the point, while the first proof meanders blindly about the direct path to the goal before reaching it.

Reflecting the greater efficiency with which the machine attacked the problem in the second trial, only four circuits of the subgoal-generating loop were required compared with twenty-four circuits required without the extended heuristics. Twenty-one intermediate subgoals were generated, compared with sixty-one in the first case, and the problem-solving graph extended to a depth of only three levels, rather than twelve levels for the proof with basic heuristics alone.

For a particular case of a problem taken from a Brooklyn technical high school final examination in plane geometry a solution was found with the extended heuristics in less than five minutes. With the basic heuristics alone, the machine exhausted its working storage in half an hour without having completed the problem. On the other hand, there are problems for which the machine achieves no net gain by applying the additional heuristics. The theorem: "Diagonals of a parallelogram bisect one another" was proved in about three minutes in either mode. The proofs produced in each trial were equivalent, though not the same. A Brooklyn technical high school final examination supplied an example of an intermediate case, where the machine found identical proofs in both modes, but took almost three times as long with the basic heuristics alone (eight minutes, compared with three minutes with extended heuristics). We shall undoubtedly encounter cases for which the application of the extended set will result in a net loss of efficiency, although none has appeared yet in our limited tests.

## *Conclusion*

It is well at this point in our discussion to reemphasize the fact that the object of this research has not been the design of a machine capable of proving theorems in Euclidean plane geometry, or even one able to prove

theorems in some undecidable system such as number theory. We are, rather, interested in understanding the use of heuristic methods (or strategies) by machines for the solution of problems that would otherwise be inaccessible to them. Theorem-proving machines in themselves are objects of much interest to mathematicians and logicians, and important work at IBM is being done on this approach by Wang and by Gilmore. Wang (1960*a*) has written a program for the IBM 704 that is able to prove all theorems in propositional logic offered by Russell and Whitehead in the *Principia Mathematica,* whereas the Logic Theorist could master only about 38 of the 52 theorems appearing in chap. 2 of that volume. Also, the time required by the latter machine was far in excess of that used by the former. Newell, Shaw, and Simon, however, were interested in heuristic methods, whereas Wang, and also Gilmore, whose machine deals with the first order predicate calculus, are searching for algorithms, which, though less than a decision procedure, will produce "interesting" proofs within a reasonable amount of time. Both Wang and Gilmore find that for more complex formal systems, heuristics are required (they prefer the word "strategies") to make their algorithms sufficiently selective to produce, within acceptable bounds on space and time, proofs of any great interest.

The work of Wang and Gilmore is most relevant to a new branch of applied logic first characterized by Wang. He names this discipline "inferential analysis," and defines it to include "treatment of proofs as numerical analysis does calculations" (1960*a*). The results of inferential analysis are expected to "lead to mechanical checks of new mathematical results," and ultimately "lead to proofs of difficult new theorems by machine." The present author feels that inferential analysis is relevant, too, to the problem of intelligent behavior in machines. An automaton confronted with the real world, however, will certainly have to rely heavily on heuristics, for the unorthodox formal systems describing its environment will probably be far from amenable to the traditional methods of mathematical logic.

In conclusion, we should like to specify the course of this research for the immediate future. The machine described above is purely a problem-solving system. Except for the annexation of new theorems to the list of axioms, its structure is static. A sequence of practice problems given to the machine will not improve its performance unless a usable theorem is among them. Because it is incapable of developing its own structure, the machine will always be limited in the class of problems it can solve by the initial intent of the designer. It seems that the problem of designing a more general problem-solving machine will be enormously greater than that of designing one not so intelligent but with the capacity to learn.

An immediately obvious approach to the problem of introducing learning into the geometry machine is to allow the machine to adjust all of the parameters that determine its specific semantic heuristics, maximizing the

predicted utility of those subgoals that prove to be useful in practice. The machine will thus improve the match of its heuristic filters to the class of problems considered interesting enough to be presented to it for solution. Of greater significance would be the introduction of routines enabling the machine to recognize recurring patterns in its proof-search procedure. Once discovered, such a pattern would enable the machine to construct its own heuristics designed to induce a repetition of the pattern in later proofs. For example, the machine might notice that certain classes of premise strings are regularly followed by the same first step in a proof. The heuristic derived from this pattern would search the premises for such strings and perform the first deduction before starting on the problem-solving graph. The difficult subject of abstract pattern recognition must be understood first, however, and the transformation of pattern to effective heuristic is by no means trivial. But whatever approach to learning is considered most worthwhile to explore, the geometry machine should serve as an excellent framework within which the explorations may be pursued.

*Appendix A*

**Premises**

Angle ABD equals angle DBC
Segment AD perpendicular segment AB
Segment DC perpendicular segment BC

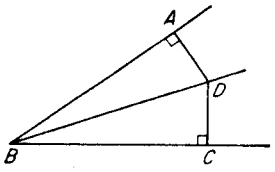


Figure 4.

**Definition**

Right angle DAB
Right angle DCB

**Syntactic Symmetries**

CA, BB, AC, DD

**Goals**

Segment AD equals segment CD

**Solution**

Angle ABD equals angle DBC
  Premise
Right angle DAB
  Definition of perpendicular
Right angle DCB
  Definition of perpendicular
Angle BAD equals angle BCD
  All right angles are equal

Segment DB
  Assumption based on diagram
Segment BD equals segment BD
  Identity
Triangle BCD
  Assumption based on diagram
Triangle BAD
  Assumption based on diagram
Triangle ADB congruent triangle CDB
  Side-angle-angle
Segment AD equals segment CD
  Corresponding elements of congruent triangles are equal

Total elapsed time = 0.3200 minute

*Appendix B*

**Premises**

Quad-lateral ABCD
Segment BC parallel segment AD
Segment BC equals segment AD

| | | |
|---|---|---|
| CA | BA | DA |
| DB | AB | CB |
| AC | DC | BC |
| BD | CD | AD |

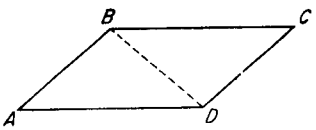

Figure 5.

**Goals**

Segment AB equals segment CD

**I am stuck, elapsed time = 0.88 minute**

Construct segment DB
Add premise segment DB
Restart problem

**Solution**

Segment BC parallel segment AD
  Premise
Opp-side CADB
  Assumption based on diagram
Segment DB
  Premise
Angle ADB equals angle CBD
  Alternate interior angles of parallel lines

Segment BC equals segment AD
  Premise
Segment BD equals segment DB
  Identity
Triangle CDB
  Assumption based on diagram
Triangle ABD
  Assumption based on diagram
Triangle ABD congruent triangle CDB
  Side-angle-side
Segment AB equals segment CD
  Corresponding elements of congruent triangles

Total elapsed time = 4.06 minutes

*Appendix C*

**Premises**

Triangle ABC
Precedes BMC
Segment BM equals segment MC
Precedes ADM
Precedes DME
Segment BD perpendicular segment AM
Segment CE perpendicular segment ME

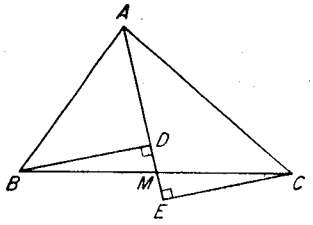

Figure 6.

**Definition**

Right angle MEC
Right angle BDM
Right angle BDA
Angle BDA equals angle BDM

**No syntactic symmetries**

**Goals**

Segment BD equals segment EC

**Solution**

Precedes BMC
  Premise
Segment EC
  Assumption based on diagram

Angle ECM equals angle BCE
  Same angle
Precedes DME
  Premise
Angle CED equals angle MEC
  Same angle
Right angle BDM
  Definition of perpendicular
Right angle MEC
  Definition of perpendicular
Angle BDM equals angle CEM
  Right angles are equal
Angle CED equals angle BDM
  Angles equal to the same angle are equal
Same side MEDB
  Assumption based on diagram
Same side DACE
  Assumption based on diagram
Collinear EDM
  Ordered collinearity implies collinearity
Precedes ADM
  Premise
Precedes EDA
  Combinatorial properties of ordered collinearity
Collinear EDA
  Ordered collinearity implies collinearity
Angle BDE equals angle AEC
  Different names for equal angles
Precedes EMA
  Combinatorial properties of ordered collinearity
Angle CEM equals angle AEC
  Same angle
Angle BDE equals angle MEC
  Angles equal to the same angle are equal
Angle CED equals angle EDB
  Angles equal to the same angle are equal
Opp side CBED
  Assumption based on diagram
Segment ED
  Assumption based on diagram
Segment EC parallel segment BD
  Segments are parallel if alternate interior angles are equal
Opp side EDCB
  Assumption based on diagram

Segment CB
  Assumption based on diagram
Angle BCE equals angle DBC
  Alternate interior angles of parallel lines
Angle ECM equals angle DBC
  Angles equal to the same angle are equal
Same side CMBD
  Assumption based on diagram
Same side MBEC
  Assumption based on diagram
Collinear CMB
  Ordered collinearity implies collinearity
Angle DBM equals angle BCE
  Different names for equal angles
Angle MBD equals angle MCE
  Angles equal to the same angle are equal
Angle DMB equals angle EMC
  Vertical angles
Segment BM equals segment MC
  Premise
Triangle BDM
  Assumption based on diagram
Triangle CEM
  Assumption based on diagram
Triangle BDM congruent triangle CEM
  Angle-side-angle
Segment BD equals segment EC
  Corresponding elements of congruent triangles

Total elapsed time = 8.08 minutes

WITH BASIC HEURISTICS

**Solution**

Precedes DME
  Premise
Precedes BMC
  Premise
Angle DMB equals angle EMC
  Vertical angles
Right angle BDM
  Definition of perpendicular
Right angle MEC
  Definition of perpendicular

Angle BDM equals angle CEM
  Right angles are equal
Segment BM equals segment MC
  Premise
Triangle CEM
  Assumption based on diagram
Triangle BDM
  Assumption based on diagram
Triangle BDM congruent triangle CEM
  Side-angle-angle
Segment BD equals segment EC
  Corresponding elements of congruent triangles

Total elapsed time = 1.06 minutes

WITH EXTENDED HEURISTICS

# EMPIRICAL EXPLORATIONS OF THE GEOMETRY-THEOREM PROVING MACHINE

*by H. Gelernter, J. R. Hansen, & D. W. Loveland*

## *Introduction*

In early spring, 1959, an IBM 704 computer, with the assistance of a program comprising some 20,000 individual instructions, proved its first theorem in elementary Euclidean plane geometry (Gelernter, 1959*b*). Since that time, the geometry-theorem proving machine (a particular state configuration of the IBM 704 specified by the afore mentioned machine code) has found solutions to a large number of problems[1] taken from high-school textbooks and final examinations in plane geometry. Some of these problems would be considered quite difficult by the average high-school student. In fact, it is doubtful whether any but the brightest students could have produced a solution for any of the latter group when granted the same amount of prior "training" afforded the geometry machine (*i.e.*, the same vocabulary of geometric concepts and the same stock of previously proved theorems).

The research project which had as its consequence the geometry-theorem proving machine was motivated by the desire to learn ways to use modern high-speed digital computers for the solution of a new and difficult class of problems; a class heretofore considered to be beyond the capabilities of a finite-state automaton. In particular, we wished to make our computer perform tasks which are generally considered to require the intervention of human intelligence and ingenuity for their successful completion. The reasons behind our choice of theorem proving in geometry as a representative task are set forth in detail in an earlier study (1958). We

[1] More than fifty proofs are on file at the present time.

only remark here that problem-solving in geometry satisfies our definition of an intellectual activity, while being at the same time especially well suited to the approach we wished to explore. The fact that geometry is decidable is irrelevant for the purpose of our investigation. The methods employed by the machine are suitable as well for the proof of theorems in systems for which no decision algorithm can exist.

We shall not labor the question as to whether our machine is indeed behaving intelligently in performing a task for which humans are credited with intelligence. The psychologists offer us neither aid nor comfort here; they have yet to satisfactorily characterize such behavior in humans, and have rarely considered the abstract concept of intelligence independent of its agent. In the final analysis, people are occasionally observed to do things that may best be described as intelligent, however vague the connotations of the word. These are, in general, tasks involving highly complex decision processes in a potentially infinite and uncontrollable environment. We should be most happy to have our machine duplicate this kind of behavior, whatever label is affixed to it.

### *Heuristic Programming and the Geometry Machine*

The geometry machine is able to discover proofs for a significant number of interesting theorems within the domain of its *ad hoc* formal system (comprising theorems on parallel lines, congruence, and equality and inequality of segments and angles) without resorting to a decision algorithm or exhaustive enumeration of possible proof sequences. Instead, the theorem-proving program relies upon heuristic methods to restrain it from generating proof sequences that do not have a high *a priori* probability of leading to a proof for the theorem in question.

The general problem of heuristic programming has been discussed by Minsky (1959*a*) and Newell, Shaw, and Simon (1959*a*). The particular approach pursued by the authors has been described at length in the papers to which we have already referred (Gelernter et al., 1958, 1959*b*). We shall therefore defer to the presentation of the machine's detailed results in the full study summarized here for a description of how these results were achieved. It should be recorded here, however, that the geometry machine operates principally in the analytic mode (reasoning backward). At each stage of the search for a proof, a goal exists which must be "connected" with the premises for the problem by a bridge of axioms and previously established theorems of lemmas. If the connection cannot be made directly, then a set of "subgoals" is generated and the process is repeated for one of the subgoals. Heuristic rules are used to reject subgoals that are not likely to prove useful, to select one from those remaining to work on, and to choose particular axioms and theorems to use in generat-

ing new subgoals. The machine does depart from this procedure in a number of circumstances (in setting up an indirect proof, for example), but these cases account for only a small fraction of the total search time.

The computer program itself was written within the framework of the so-called Newell-Shaw-Simon list memory (1957*b*). In order to ease the task of writing so massive and complex a machine code, a convenient special-purpose list processing language was designed to be compiled by the already available FORTRAN system for the IBM 704 computer (Gelernter et al., 1960*b*). The authors feel that had they not made free use of an intermediate programming language, it is likely that the geometry program could not have been completed.

### *Summary of Results*

Since its initial solo performance, the geometry machine has existed in several different configurations. In its earliest and most primitive form, the system was equipped with a single major semantic heuristic.[2] That first system was, however, able to prove a large number of interesting, though admittedly simple theorems in elementary plane geometry.[3] The heuristic rule in question, which is independent of the particular formal system under consideration, may be described in the following way. All subgoal formulas that are generated at a given stage of the proof search are interpreted in a model of the formal system; in our case, the model is a diagram, a formal semantic interpretation. If the interpreted subgoal is valid in the diagram, it is accepted as a possible step in the proof, provided that it is noncircular (Gelernter, 1959*a*). Otherwise, it is rejected.

As an experiment, a number of attempts were made to prove extremely simple theorems with the latter heuristic "disconnected" from the system (*i.e.*, all noncircular subgoals generated were accepted). In each case, the computer's entire stock of available storage space was quickly exhausted by the initial several hundreds of first level subgoals generated, and, in fact, the machine never finished generating a complete set of first level subgoals. We estimate conservatively that on the average, a number of the order of 1000 subgoals are generated per stage by the decoupled system. If one compares the latter figure with the average of 5 subgoals per stage accepted when the diagram is consulted by the machine, it is easy to see that the use of a diagram is crucial for our system. (Note that the total number of subgoals appearing on the problem-solving graph grows exponentially with the number accepted per stage.)

Since the procedure described above is a heuristic one, errors are oc-

[2] A semantic heuristic is one based on an interpretation of the formal system rather than on the structure of the strings within that system.

[3] A number of these proofs are reproduced in Gelernter, 1959*b*.

casionally made in the selection or rejection of formulas as subgoals. The diagram is made available to the machine in coordinate representation to finite precision. Formulas are interpreted by transforming them into an appropriate calculation on the numerical coordinates representing the point variables. For example, to check the validity of a statement concerning the equality of two segments, the length of each segment in the figure is calculated, and they are then compared to a certain preassigned number of decimal places. If, instead, the statement concerned parallel segments, the slopes would be calculated and compared. In a small number of cases, round-off error has propagated beyond the allowed value, so that valid subgoals were rejected, or invalid ones accepted. It is important to point out, however, that in no case could this effect result in a false proof. Where valid subgoals were rejected, the machine found alternate paths to the solution. Where invalid ones were accepted, the machine failed, of course, to establish them within the formal system. In the worst possible case, the interpretation error could prevent the computer from finding any solution at all, but never could it lead to an invalid proof.

It should be clear at this point that the diagram is used only to guide the search for a proof by supplying yes or no answers to questions of the form: "Is segment AB equal to segment CD in the figure?", or "Is angle ABC a right angle in the figure?". There is no direct link between the diagram and the formal system in the geometry machine. The behavior of the machine would not be changed if the coordinate representation were replaced by a device capable of drawing figures on paper and scanning them.

In the basic theorem-proving system described above, after a set of subgoals has been generated, each member of the set is explored in order. The next subgoal in line is not examined until the one preceding it has been followed down to a dead end. Too, in generating the next level for a given subgoal, every applicable theorem available is pressed into service.

This system was soon extended by the introduction of selection heuristics for both subgoals and subgoal-generating theorems. The subgoal selection heuristic assigns a "distance" between each subgoal string and the set of premises in a vaguely defined *ad hoc* formula space. At each stage, the next subgoal selected is that which is "closest" to the premises in formula space. The generator selection routine recognizes certain classes of subgoals that are usually established in one step. For such "urgent" subgoals, the appropriate generator is withdrawn immediately, and an attempt is made for a one-step proof (of that particular subgoal) before generating the full set for that formula.

The extended system is able to prove a number of somewhat more difficult theorems that are beyond the capacity of the basic machine. For those problems within the range of both systems, the former is, on the

average, about three times faster, and generates about two-thirds the total number of subgoals in half as many subgoal generation cycles as required by the basic system. The average depth of the problem-solving graph for the refined system, about seven to nine levels, is two-thirds the average depth for the basic system.

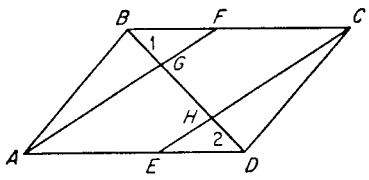

Figure 1.

By the addition of a simple construction routine, the theorem-proving power of the machine is expanded to include an entirely new class of problem, hitherto logically unattainable. The routine, called upon only when all other attempts have failed, allows the machine to join two previously unconnected points in the diagram, and extends the newly created segment to its intersections with all other segments in the figure. The new segment, when it intersects previously given ones, introduces new points into the problem which are named by the machine and become part of the problem system.

At this stage in its development, the geometry machine was capable of producing proofs that were quite impressive (Appendix 1).[4] Its performance, however, fell off rapidly as the number of points in the diagram increased. This effect was due largely to the fact that unlike humans, who generally identify angles visually by their vertices and rays, the computer specifies an angle by a predicate on three variables, the vertex and a point on each ray. Consequently, the equality of angles 1 and 2 in Fig. 1 may be represented in thirty-six different ways, since each angle has six different names. Formal rigor demands, too, that the equality of angles ADH and EDG, for example, be proved rather than taken for granted. It should be clear that where the condition above exists, the search for a proof quickly bogs down in a mass of uninteresting detail.

In the current system, the angle problem is solved by allowing the machine to use the diagram to identify a given angle with its full set of names, and to assume the equality relationship between different names for the same angle, as does its human counterpart. The geometry machine in its present configuration is able to find proofs for theorems of the order of difficulty represented by the following:

Theorem: If the segment joining the midpoints of the diagonals of a

[4] In the proofs appended to this paper, the nonobvious predicates have the following interpretations:

| | |
|---|---|
| OPP-SIDE XYUV | Points X and Y are on opposite sides of the line through points U and V. |
| SAME-SIDE XYUV | Points X and Y are on the same side of the line through points U and V. |
| PRECEDES XYZ | Points X, Y, and Z are collinear in that order. |
| COLLINEAR XYZ | Points X, Y, and Z are collinear. |

trapezoid is extended to intersect a side of the trapezoid, it bisects that side (Appendix 2).

### *Limitations of the System*

It will be immediately evident to those familiar with the properties of formal logistic systems that unless a construction which generates a new point is introduced by the machine, all problems are solved within the framework of a propositional calculus, however complex its structure. Although the machine's present construction routine can and does generate new points, we could not expect our results to be of great interest to logicians until a full set of possible constructions (corresponding to a complete set of existentially quantified axioms) is made available to the system to abet its search for a proof.

An equally serious limitation on the formal generality of the theorem-proving machine is imposed by our method for determining the well-formedness of strings within the logical system. In order to attain the necessary speed and efficiency in processing, well-formed formulas are defined by schema rather than recursively. The kind of statement that can be made in the system is then determined by the schema available to the machine. The practical effect of this loss in generality is to restrict rather severely the freedom with which algebraic statements in geometry may be manipulated.

In addition to the above, there are a number of nonessential bounds on the theorem-proving ability of the machine. These are a consequence of the limited speed and memory capacity of the computer for problems of such highly combinatorial character. Improvements in either of the above will be immediately effective in extending the class of machine-solvable problems in both quantity and difficulty.

### *Conclusion*

The initial goal of our research program in machine intelligence has been attained. If the interrogator were to restrict his probing to the area of theorem-proving in elementary Euclidean plane geometry, our machine could be expected to give an excellent account of itself in competition with a human in Turing's well-known "imitation game" (1950). Of course there are many other problem areas (solving arithmetic problems, for example) where computers have always been able to compete successfully with humans. The significant point is that a knowledgeable interrogator would certainly avoid such areas in his questioning, while he might well (until now, at any rate) introduce a plane geometry problem in a cal-

culated attempt to separate the men from the machines.[5] Although the stage is now set for the argument that any distinct area of human intellectual activity will in the same way succumb to the inexorable logic of electrons, switches, and gates, we defer to our philosopher colleagues for debate on the implications of that contention, at least until the time that computers have been programmed to consider such issues.

There are a number of consequences of our work that are, fortunately, more concrete than that alluded to above. Perhaps the most important are those relating to inferential analysis, a new branch of applied logic first characterized by Wang (1960*a*). Inferential analysis "treats proofs as numerical analysis does calculations," and is expected to "lead to mechanical checks of new mathematical results" and, more important, "lead to proofs of difficult new theorems by machine." It is expected that our techniques for the manipulation and efficient search of problem-solving trees and our results concerning syntactic symmetry will prove to be useful tools in pursuing the goals of inferential analysis.

Contributions have been made, too, in the area of techniques for computer implementation of complex information processes. Results pertaining to the design and use of intermediate languages for the specification of list manipulation processes have been reported elsewhere (Gelernter et al., 1960*b*). The latter work indicates clearly the requirements of a digital computer system designed for optimum execution of such list processes. In brief, a list processing computer should possess hardware facilities for:

1. Generalized indirect addressing; specified in the indirectly addressed instruction to arbitrary depth and in arbitrary order from either the left or the right field of a two-address register,
2. Effective address recovery; making available the terminal content of the address register (the final address in a long and complex indirect address chain, for example) as the address field for a subsequent operation,
3. Field logic; a greatly expanded set of interfield operations within a full register sectioned according to some previously established convention, and
4. List search operations; a list equivalent of the conventional table look-up instruction.

The bulk storage input-output requirements for a list processing computer are severe, and are not included in the enumeration above. The system

[5] It may be argued (and undoubtedly, it *will* be argued) that the truly knowledgeable interrogator, cognizant of the decidability of geometry, would certainly avoid this area as well, perhaps preferring the manifestly undecidable parts of the predicate calculus or number theory to effect the distinction between man and machine. We recall here that our methods are independent of the decidability of the formal system, and, in fact, Wang (1960*a*) and Gilmore (1960) have developed proofs for theorems in the undecidable area of the predicate calculus.

design of a digital computer for the manipulation of list structures will be described in detail in a subsequent paper.

Finally, we consider the implications of our work for the basic problem of machine intelligence. The geometry machine, we feel, offers convincing evidence of the power and fruitfulness of heuristic programming for the solution of problems of a certain class by computer. In our experience, the theorem-proving power of the machine has often been extended by the addition of a single heuristic to a degree equivalent to a three-to-fivefold increase in the speed or storage capacity of the computer.

Our program has proved to be disappointing as a tool for the study of the more elementary trial-and-error types of machine learning, largely because of the rather low rate at which it accumulates experience. It is reasonable to expect, however, that the geometry machine might yet be pressed into service in an investigation of the higher, conceptual types of machine learning, providing that one will someday know how to formulate the problem.

If nothing else, our work offers some qualitative indication of the order of magnitude of difficulty for problems that could be expected to yield to contemporary computer technology. Three years ago, the dominant opinion was that the geometry machine would not exist today. And today, hardly an expert will contest the assertion that machines will be proving interesting theorems in number theory three years hence.

## *Appendix 1*

**Premises**

Quad-lateral ABCD
Point E midpoint segment AB
Point F midpoint segment AC
Point G midpoint segment CD
Point H midpoint segment BD

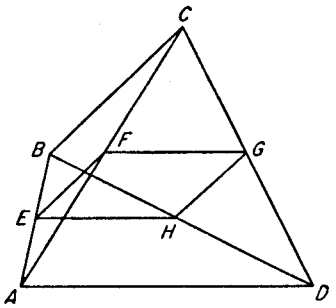

Figure 2.

**To Prove**

Parallelogram EFGH

**Syntactic Symmetries**

BA, AB, DC, CD, EE, HF, GG, FH, CA, DB, AC, BD, GE, FF, EG, HH, DA, CB, BC, AD, GE, HF, EG, FH

**Proof**

Segment DG equals segment GC
  Definition of midpoint

Segment CF equals segment FA
  Definition of midpoint
Triangle DCA
  Assumption based on diagram
Precedes DGC
  Definition of midpoint
Precedes CFA
  Definition of midpoint
Segment GF parallel segment AD
  Segment joining midpoints of sides of triangle is parallel to base
Segment HE parallel segment AD
  Syntactic conjugate
Segment GF parallel segment EH
  Segments parallel to the same segment are parallel
Segment HG parallel segment FE
  Syntactic conjugate
Quad-lateral HGFE
  Assumption based on diagram
Parallelogram EFGH
  Quadrilateral with opposite sides parallel is a parallelogram

Total elapsed time = 1.03 minutes

## *Appendix 2*

**Premises**

Quad-lateral ABCD
Segment BC parallel segment AD
Point E midpoint segment AC
Point F midpoint segment BD
Precedes MEF
Precedes AMB

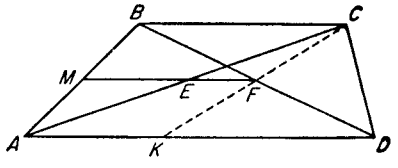

Figure 3.

**To prove**

Segment MB equals segment MA

**No Syntactic Symmetries**

**I Am Stuck, Elapsed Time = 8.12 Minutes**
Construct segment CF
Extend segment CF to intersect segment AD in point K

**Add to Premises the Following Statements**

Precedes CFK
Collinear AKD

**Proof**

Segment BC parallel segment AD
  Premise
Collinear AKD
  Premise
Segment KD parallel segment BC
  Segments collinear with parallel segments are parallel
Opp-side KCDB
  Assumption based on diagram
Segment DB
  Assumption based on diagram
Angle KDB equals angle CBD
  Alternate interior angles of parallel lines are equal
Precedes CFK
  Premise
Precedes DFB
  Definition of midpoint
Angle KFD equals angle CFB
  Vertical angles are equal
Segment DF equals segment FB
  Definition of midpoint
Triangle FDK
  Assumption based on diagram
Triangle FBC
  Assumption based on diagram
Triangle FDK congruent triangle FBC
  Two triangles are congruent if angle-side-angle equals angle-side-angle
Segment KF equals segment CF
  Corresponding segments of congruent triangles are equal
Segment CE equals segment EA
  Definition of midpoint
Triangle AKC
  Assumption based on diagram
Precedes CEA
  Definition of midpoint
Segment EF parallel segment AK
  Segment joining midpoints of sides of triangle is parallel to base
Segment EF parallel segment KD
  Segments collinear with parallel segments are parallel
Segment FE parallel segment BC
  Segments parallel to the same segment are parallel

Precedes MEF
  Premise
Collinear MEF
  Ordered collinear points are collinear
Segment FM parallel segment BC
  Segments collinear with parallel segments are parallel
Segment FM parallel segment DA
  Segments parallel to the same segment are parallel
Triangle DBA
  Assumption based on diagram
Precedes AMB
  Premise
Segment MB equals segment MA
  Line parallel to base of triangle bisecting one side bisects other side

Total elapsed time = 30.68 minutes