H. L. Gelernter

N. Rochester

Intelligent Behavior in Problem-Solving Machines

Abstract: As one step in the study of intelligent behavior in machines, the authors consider the particular case of a machine that can prove theorems in elementary Euclidean plane geometry. The device uses no advanced decision algorithm, but relies rather on rudimentary mathematics and "ingenuity" in the manner, for example, of a clever high-school student.

This paper discusses heuristic methods and learning machines and introduces the concept of a theory machine as an extension of a theorem-proving machine.

Introduction

Modern machines execute giant tasks in arithmetic and carry out clerical operations that are far beyond human capacity, but we have not yet learned to apply computers to problems that require more than a barest minimum of ingenuity or resourcefulness. This paper reports some early results in an approach to the problem of learning how to use machines in these presently unmanageable areas. The goal of this research is the design of a machine whose behavior exhibits more of the characteristics of human intelligence.

We shall concern ourselves in particular with a single representative problem, one which contains in relatively pure form the difficulties we must understand and overcome in order to attain our stated goal. The special case we have chosen is the proof of theorems in Euclidean plane geometry in the manner, let us say, of a high-school sophomore. 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 to be developed will be no less valid for problem solving in systems where no such decision algorithm exists.

If the application of a decision algorithm is rejected as uninteresting (in the case of plane geometry) or impossible (for most problems of interest), there remain two alternative approaches to the proof of theorems in formal systems. The first consists in exhaustively developing the proof from the axioms and hypotheses of the system by systematically applying the rules of transformation until the required proof has been produced (the so-called "British Museum algorithm" of Newell and Simon[1]). There is ample evidence that this procedure would require an impossibly large number of steps for all but the most trivial theorems of the most trivial formal systems. The remaining alternative is to have the machine rely upon heuristic methods, as people usually do under similar circumstances.

Problems for which people use heuristic methods seem to have the following characteristic. The work begins routinely, and then suddenly the person experiences a flash of understanding. This is followed by the writing down and checking of the solution. Apparently the person first used heuristic methods to look for a solution. To each suggestion turned up by the heuristic methods he applies some sort of a test. The flash of understanding comes when a suggestion gets a high score on the test. The clerical task that follows is the transformation from *suggestion space*[2] to *problem space*. The transformation is possible, of course, only if a valid solution has been indicated. The geometry machine will behave in this way.

Instead of geometry we might have chosen a certain class of probability problems, proofs of theorems in projective geometry, proofs of trigonometric identities, proofs in part of number theory, or the evaluation of indefinite integrals. There were compelling reasons, however, for choosing plane geometry, the most important being the readily understood "suggestion space" offered by the diagram (the semantic interpretation of the formal system), and the ease of transforming "proof indications" into problem space. An important secondary reason was the fact that everyone who would be interested in our results has studied Euclid, so the results can be communicated more efficiently.

It should be noted here that the geometry project is a consequence of the Dartmouth Summer Research Project on Artificial Intelligence, standing on a foundation laid

by the members of the study,[3] and evolving from the pioneering work of Newell and Simon in heuristic programming.[4]

Not all problems whose solutions seem to be accompanied by a "flash of understanding" are elementary enough to lie within the scope of the methods described in this paper. Many have difficulties of a more profound nature. It will be possible to say a little more about this later, but a secure understanding of the nature of these harder problems will come only after more research has been done.

The explanation of the precise meaning of the term *heuristic method* is an important part of this paper. For the moment, however, we shall consider that a heuristic method (or a *heuristic,* to use the noun form) is a procedure that may lead us by a short cut to the goal we seek or it may lead us down a blind alley. It is impossible to predict the end result until the heuristic has been applied and the results checked by some formal process of reasoning. If a method does not have the characteristic that it may lead us astray, we would not call it a heuristic, but rather an algorithm.[5] The reason for using heuristics instead of algorithms is that they may lead us more quickly to our goal and they allow us to venture by machine into areas where there are no algorithms.[6]

Finally, since people seem to use heuristic reasoning in nearly every intelligent act, it is reasonable to ask why some task more familiar and natural for people was not chosen as representative of the class rather than plane geometry. Several alternatives to geometry were, in fact, considered and rejected for failing to satisfy one or more of the following requirements:

1. The task must include a kind of reasoning that we are not yet able to get our machines to do but which we think we can learn to manage.
2. It must not contain harder kinds of reasoning that are too far beyond our understanding.
3. It must not be too much cluttered with irrelevant work.

Most human acts fail to meet requirement (2). We have a long way to go before our machines can play Turing's "Imitation Game" and win.[7]

Geometry

A standard dictionary defines geometry as "the theory of space and of figures in space," and indeed, most people would offer a similar definition. To the mathematician, however, geometry represents a formal mathematical system within which proofs are possible, and which can be related to real space if this seems interesting for the purpose at hand, but which can alternatively be related to concepts having no physical reality or significance. The machine considers geometry primarily as a formal system but uses the interpretation in terms of figures in space for heuristic purposes.

A formal system such as geometry comprises:

1. Primitive symbols
2. Rules of formation
3. Well-formed formulas
4. Axioms
5. Rules of inference
6. Theorems.

The set of primitive symbols (or alphabet) for geometry is those characters which are interpreted as the names of points together with those interpreted as specifying relations between discrete sets of points, or between a given set and the universe of points (e.g., $=, \|, A, B, \Delta$). In order to make proofs in geometry it is not necessary, for example, to think of a line as something long, thin, and straight. It is sufficient to be able to recognize the symbol *line*.

The rules of formation specify how to assemble the primitive symbols into well-formed formulas (statements) which may be valid or invalid within the formal system. For example, "Two sides of every triangle are parallel" is a well-formed formula (although not valid), whereas "Two exists of obtuse every one point" is not a well-formed formula. We can ask the machine whether the first is true (interpreting formal validity as truth), but the second is gibberish because it does not obey the rules of formation. These rules are, in a sense, the grammar of a language whose vocabulary comprises the alphabet of primitive symbols.

The axioms are a set of well-formed formulas, such as "Through every pair of points there can be drawn one and only one straight line," which are selected to serve as a foundation on which to build. They are regarded as being true by definition, if you like.

The rules of inference are the means by which the validity of one well-formed formula can be derived from others that are already established. The new formula is said to be inferred immediately from the given one or set by the specified rule of inference.

A proof is a succession of well-formed formulas in which each formula (or line of proof) either follows by one of the rules of inference from the preceding formulas, or is an axiom or previously established theorem. A theorem is the last line in a proof.

To recapitulate, a problem presented to our machine is a statement in a formal logistic system, and the solution to that problem will be a sequence of statements, each of which is a string of symbols in the alphabet of that system. The last statement of the sequence will be the problem itself, the first will always be an axiom or previously established theorem of the system.[8] Every other formula will be immediately inferable from some set preceding it or will itself be an axiom or previously established theorem.

This simple and elegant description of geometry is essentially the one given the high-school sophomore. It will shortly be seen that this view is too naive to describe what really happens, but for the moment it will be expedient to continue the exposition as if it were true, because the idealization has a significance of its own. A number of things should be pointed out about this ideal view of geometry. First, there is a difference between finding a proof and checking it. To check a proof one merely fol-

lows some simple rules that are set down very precisely. To discover a proof, on the other hand, requires ingenuity and imagination. One must use good intuitive judgment in selecting which of many possible alternatives is a step in the right direction. The high-school sophomore does not have a complete set of explicit rules to guide him in finding a proof.

Since checking a proof is a clerical procedure there is no reason why a machine cannot easily do it. A well-formed formula (i.e., axiom, line of a proof, or theorem) would be a string of data words in memory, and a rule of formation or of inference would be a subprogram. There is nothing really new or difficult about this, and many programs have been written to make machines do jobs as difficult. The artificial geometer discussed here will have a subprogram which is an algorithm for checking a proof.

The process of discovering a proof is another matter, and the question of how to get a machine to do it is the subject of this paper. The student or the machine can be given some useful hints but must also be provided with a warning that these hints may be misleading. For example, it can be said that if the proposition to be proved involves parallel lines and equality of angles, there is a good chance that it will help to try the theorem: "If two parallel lines are intersected by a third line, the opposite interior angles are equal." This advice is a heuristic that can be given to the machine or student. It will lead to a proof in a good many cases but will as often lead nowhere at all.

Thus far, there has been no mention of drawing figures. It is, of course, quite possible to discover a proof in a formal system without interpreting that system, and in the case of geometry, except for the need to discover proofs efficiently, or for applying theorems to practical problems, one need never make a drawing. The creative mathematician, however, 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 plane goemetry, the model is a diagram, a semantic interpretation of the formal system in which, to quote Euclid, the symbol *point* stands for "that which has no parts," a *line* is "breadthless length," and so on. The model is so useful an aid for discovering proofs in geometry that few people would attempt a proof without first drawing a diagram, if not physically, then in view of the mind's eye. If a calculated effort is made to avoid spurious coincidences, then 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.

We cannot emphasize too strongly the following point. To serve as a heuristic device in problem solving, the model need not lie in rigorous one-to-one correspondence with the abstract system. *It is only necessary that they correspond in a sufficient number of ways to be useful.* The success of the model in designating correct solutions to problems in that system (solutions that will be checked within the framework of the abstract system) is the only criterion one need apply in judging the suitability of a given model.[9] If the model is indeed a semantic interpretation of a formal logistic system, then it is most desirable that the interpretation satisfy every axiom of the formal system. But should the interpretation be valid, too, for some richer or poorer formal system, its heuristic value might be impaired, but by no means eliminated.

Heuristic method

The proof of theorems in Euclidean plane geometry in the sense described above requires the extensive use of heuristic methods, and it is these methods rather than geometry that are of primary interest to us. The role of geometry is to provide a problem of the right difficulty to permit a thorough development and understanding of the class of heuristics involved.

The steps in a typical application of a heuristic method to theorem proving are the following:

1. Calculate the character[10] of the theorem.
2. Using the theorem character, calculate the applicable methods and estimate the merit of each.
3. Select the most appropriate method.
4. Try it.
5. In case of failure, cross off this method and return to step (3).
6. In case of success, print the proof and stop.

The character of a theorem (or of any problem) is in essence the machine's description of the theorem (or the problem). In its simplest form, the character may be represented by a vector, each element of which describes a given property of either the syntactic statement of the theorem or its semantic representation. The vector designating the applicable methods and estimated merit of each is a vector function of the character. The figures of merit are, of course, only guesses based initially on the judgment of the programmer, and subsequently modified by the machine in the light of its experience.

Defining the term *characteristic* as a given element of the character vector, the following might be introduced as syntactic characteristics of a theorem:

C_i = 1 if the hypotheses contain the symbol $\|$, 0 otherwise;

C_j = 1 if the consequents of the theorem contain the symbol $\|$, 0 otherwise;

C_k = 1 if there exists a permutation of the names of points in the hypotheses that leaves the set of hypotheses unchanged, 0 otherwise; and so on.

Examples of semantic characteristics are the following:

C_l = n, where n is the number of axes of symmetry in the diagram;

C_m = 1 if two angles of segments are to be proved equal, and they are corresponding elements of congruent triangles, 0 otherwise; and so on.

The rules formalized into the vector function that transforms the character of a problem into a sequence of designated methods of approach and the estimated merit of each will in general fall into two categories. The first

will contain those heuristics which operate on the syntactic characteristics of the problem. The second will, in the general case of a problem-solving machine, comprise those rules which operate on the characteristics of the model. For the artificial geometer, these are the semantic characteristics described above.

The problem of strategy and tactics in choosing methods is most important. One obvious strategy mentioned earlier is to explore all alternatives systematically. This is known to be inadequate for many problems and is considered by the authors to be uninteresting, and probably useless, for geometry. The strategy and tactics used by Newell and Simon in their achievement in theorem proving by machines are not adequate for this harder problem on present-day machines. Their proofs were, at most, three or four steps long, and machine time required is probably an exponential function of the number of steps. Clearly the ten-step proofs of geometry will require much more selective heuristics than those adequate for propositional calculus.

The authors have at present a system of strategy and tactics. It does not seem useful to report it in detail at this time because machine experience will probably induce major revisions and improvements. It is clear, however, that the skill with which the machine selects and manipulates methods will distinguish a good machine from a poor one. Since it is impossible to predict the detailed behavior of so complex an information-processing system as the artificial geometer, it is necessary to write the program and run the simulation before conclusions can be reached with confidence.

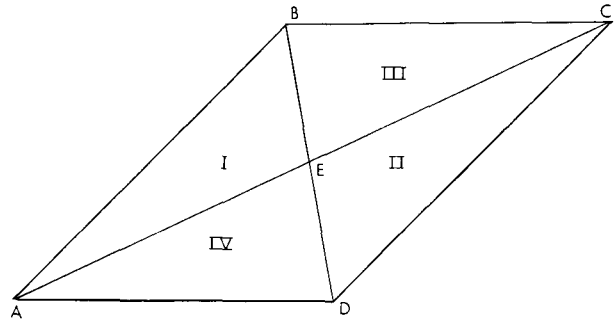
The speed with which a difficult problem can be solved is an essential factor in determining the usefulness of an intelligent machine. This speed cannot be achieved by little steps like inventing faster components. On the scale considered here, a factor of ten is a minor change in speed. Suppose, for example, that a given proof requires ten steps. If for each step, the machine must explore three alternatives, there will be about 20,000 things to consider. A slightly less intelligent machine that must explore six alternatives will have to consider 20,000,000 things. For problems having longer solutions, selectiveness becomes more important exponentially.

Syntactic symmetry

The formal system of plane geometry will be a difficult one for the machine to manipulate. Not only are the alphabet and axiom set both large, but geometry must be formalized in the lower functional calculus, at the very least. The difficulty is compounded, too, by the fact that the predicates of plane geometry exhibit a high degree of symmetry, and a given statement in the system will in general admit of a multiplicity of completely equivalent forms.

These symmetries are at times a painful thing to contend with; they make it necessary that a theorem be considered in every one of its equivalent forms in any attempt to establish a deduction by means of substitution. On the other hand, they are the basis of a powerful new rule, completely syntactic in nature, that simplifies immensely the search for a proof of a theorem displaying these symmetries. The rule will prevent the machine from searching in a circle for useful intermediate steps, or subgoals, to bridge the gap between antecedent and consequent of the theorem to be proved. In effect, it removes from consideration those subgoals which are formally equivalent to some subgoal already incorporated into the structure of the search for a proof.

Figure 1



We shall introduce the rule by an example. Let us consider the following theorem: "The diagonals of a parallelogram bisect one another" (Fig. 1). To solve the problem, the machine must establish the formulas $AE=EC$ and $BE=ED$. Now it would be most useful if the artificial geometer could recognize, as people usually do, that the proof for the second formula is essentially the same as that for the first, and therefore only one of the two need be established. But it is even more important that the machine not fall into the class of trap illustrated by the following redundant search process. The method chosen is that of congruent triangles, and in order to establish the formula $\Delta I \cong \Delta II$, from which the theorem may be immediately inferred, the machine sets at some later stage the subgoal $\Delta III \cong \Delta IV$. The geometer will, in fact, satisfy our requirements on both these points. The mechanism whereby this is accomplished is an embodiment of the theorem and rule specified below.

Consider first the following definition: Let π be a permutation on the names of the syntactic variables in a theorem. Then π is a *syntactic symmetry* of the theorem if its operation on the set of hypotheses leaves the set unchanged except for a possible transformation into an equivalent form with respect to the symmetries of the predicates (i.e., $\pi\{H\} \equiv \{H\}$ is valid. We can now state the required theorem thus:

"If Γ is a well-formed formula provable from the set of hypotheses $\{H\}$, and π is a syntactic symmetry of the set $\{H\}$, then $\pi\Gamma$ is a well-formed formula provable from the same set $\{H\}$. The formula $\pi\Gamma$ will be called a *syntactic conjugate* of Γ."

The proof of the theorem is quite trivial, and follows from the fact that the syntactic variables in a theorem

may be renamed without destroying the validity of the theorem. Thus, if

$\{H\} \supset \Gamma$ is valid, then
$\pi\{H\} \supset \pi\Gamma$ follows by the rule of substitution.
Since $\pi\{H\} \equiv \{H\}$,
$\{H\} \supset \pi\Gamma$ is valid.

The theorem itself grants the machine the same power the human mathematician has at his disposal when he recognizes the equivalence of two different statements with respect to a given formal system, for now it may establish the syntactic conjugate of any valid formula Γ by merely asserting "similarly, $\pi\Gamma$." The rule of syntactic symmetry follows from the theorem. It is used by the machine to construct, given the heuristics and methods at its disposal, the optimum *problem-solving graph,* and a description of such a graph follows. (See Fig. 2.)

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* 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 hypotheses and axioms.[11]

We can now specify the rule of syntactic symmetry thus: G_i is not a suitable subgoal to add to the problem-solving graph if it is the syntactic conjugate of any G_j for $i \geq j$, for any proof sequence leading to G_i is identical with a conjugate sequence leading to G_j with the variables renamed, and any mechanism leading to a proof of G_i would as well prove G_j. If $i=j$, the two subgoals are in effect redundant, and if $i>j$, the sequence leading to G_i leads to G_j when conjugated, and all the steps G_k, $i \geq k > j$ can be eliminated.

In the light of the above, we may now re-examine our introductory problem (Fig. 1). The machine must establish the following two goals:

G_0^1: $AE=EC$

G_0^2: $BE=ED$

By the theorem of syntactic symmetry, the machine will eliminate G_0^2 from the graph, since $G_0^2=\pi G_0^1$, where π is the transformation *A into B, B into C, C into D, and D into A,* and after proving G_0^1, will assert "similarly, G_0^2." Then, if at some point in the proof, $\triangle ABE \cong \triangle CED$ is a subgoal, it will eliminate the statement $\triangle BCE \cong \triangle DEA$ as a possible subgoal; if $AB=CD$ is a subgoal, $BC=DA$ will be removed from consideration. Clearly every directed path through the problem-solving graph from hypotheses to goal will be unique under the π-transformation, and will be the shortest one in that it will contain no redundant subgraphs (no two nodes will be linkable by a π-transformation).

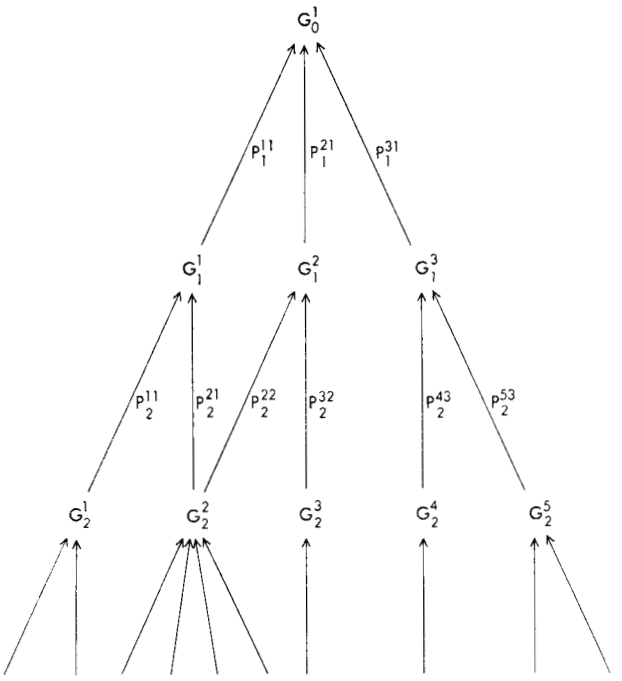


Figure 2 **Problem-solving graph.**
The nodes G_i^α represent subgoals of order i, with α numbering the subgoals of a given order. $P_i^{\alpha\beta}$ is a transformation on G_i^α into G_{i-1}^β.

Syntactic rules such as the above will be essential to the success of the plane geometry machine. But while they ease the labor of the geometer considerably as it threads a path from problem to solution, they are, except in the simplest cases, powerless to indicate which path, among the very many possible, does indeed lead to a solution, and which wander off into infinity, regressing farther from the goal with each step. The geometer will need more information about most problems before it can even begin to seek a solution. It will find this information as the mathematician does, in the diagram.

Semantic heuristic

Semantic heuristic is concerned with the body of pertinent and probably true statements that can be obtained by observing the diagram. For example, one of the first such rules to be applied by the geometer in a particular case will be the following:

If the diagram consists of a "bare" simple polygon, a construction will probably be required.

A rule to indicate which construction to make might be:
If the figure has one axis of symmetry, and it is not drawn, then draw it.

A most useful rule will be:
If the theorem asks that two line segments or angles be proved equal, determine by measuring whether these are corresponding parts of apparently congruent triangles. If so, attempt to prove the congruence. If necessary, draw lines connecting existing points in the diagram in order to create the congruent triangles.

Another frequently used heuristic will be:

If two apparently parallel lines are crossed by a transversal, attempt to establish the parallelism by considering the angles.

A more complete understanding and appraisal of the appropriate heuristics will be one of the major consequences of experimentation.

It should be clear that the best set of heuristic rules, the best compromise between conciseness and efficiency, should not be expected to yield the best proof in every case. Indeed in a number of awkward cases the rules will impede, rather than aid, the search for a concise proof. In some cases the machine will make a construction and produce an elaborate proof while missing a simple, elegant one. People, too, do this. But these awkward cases should be the exception, and the heuristic rules appear sufficiently powerful to make an efficient machine.

Rigor

Mathematical rigor becomes a significant matter in two different aspects of the artificial geometer. One of these is that machines can provide, in a sense, more rigorous proofs than have hitherto been available. More important than this is the second aspect—that our machine is like a good human mathematician in that it increases its output and improves its communication with other mathematicians by taking chances with rigor.

Axioms and theorems are objects that can be examined and manipulated by people and machines. These present no problem. However, methods of inference are instructions to do something. In the case of machines they are programs of instructions in machine language. In the case of people they are instructions expressed in a natural language and intended to control human behavior. Except for undetected blunders in the design of a machine or in the writing of a set of machine instructions, the machine and its instructions are fully understood. And when one of these blunders is detected, it causes merely annoyance and not bewilderment. Therefore when a machine proves a theorem, there is in principle no doubt about what is going on and, except for possible apprehensions about human blunders or undetected machine malfunctions, there is no doubt about rigor.

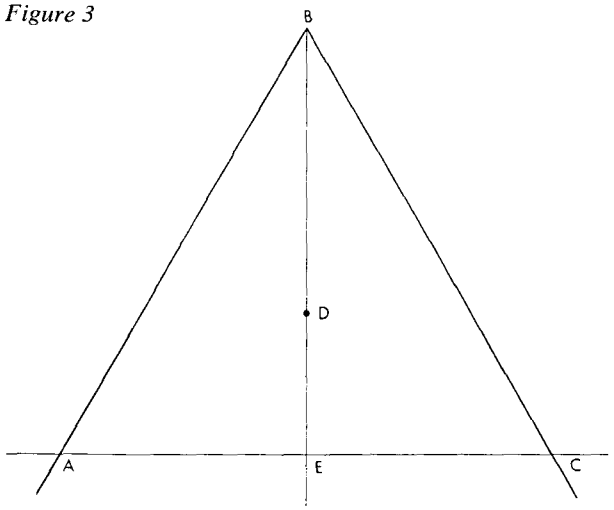
It is interesting to observe that the most rigorous treatments of the foundations of mathematics seem equivalent to designing a machine and a machine language and henceforth communicating in this language. In one case[12] the mathematician even uses the term *machine,* although his machines could not actually be built because they contain parts with infinite dimensions. Other really good treatments do not use the word *machine* but are essentially equivalent. It should be clear, then, that the translation of a formal system into a machine program is reasonable and natural.

The other aspect of rigor is quite different. Most elementary textbooks on geometry fail to prove betweenness relations. In Fig. 3, the acute angle ABC is bisected by the line segment BD. Then line segments BD and AC are extended to infinity, thus becoming lines BD and AC. Point E is defined as the intersection of lines BD and AC. Now how can it be determined whether point E lies between A and C or to the left of A or to the right of C?

Ordinarily this decision is made by looking at the figure. In rigorous treatments it is proven formally, but this is a tedious effort. Expediency dictates that the mathematician should neglect the possibility that a semantic heuristic will lead him seriously astray. Since people rarely get into trouble because of honest errors of this kind, traditional geometry excludes proofs of betweenness, and most mathematics appears to lack rigor because many matters are settled by heuristic methods rather than formal proofs. It seems clear that the machine must be able to work this way if it is to become proficient.

The artificial geometer decides questions of betweenness by measurements on the figures. But whenever it does so, it explicitly records the necessary assumptions for a given proof so as to leave a record of its guesses. There is, of course, a danger that the machine will be proving only a special instance of the theorem presented to it, but this danger can be minimized by having the machine draw alternate diagrams to test the generality of its assumptions when they are necessary.

Figure 3



Programming the geometer

The organization of the program falls naturally into three parts: a *syntax computer* and a *diagram computer* embedded in an executive routine, the *heuristic computer.* The flow of control is indicated in Fig. 4. The syntax computer contains the formal system, and its purpose is to establish the proof. The formal system manipulated by the syntax computer is expressed as a Post-Rosenbloom canonical language, and consequently the syntax computer should be useful for a wide range of logistic systems. The heuristic computer can submit any sequence of lines of proof to the syntax computer, which will test them to see that they are correct.

The diagram computer makes constructions and measures them. It does this by means of coordinate geometry

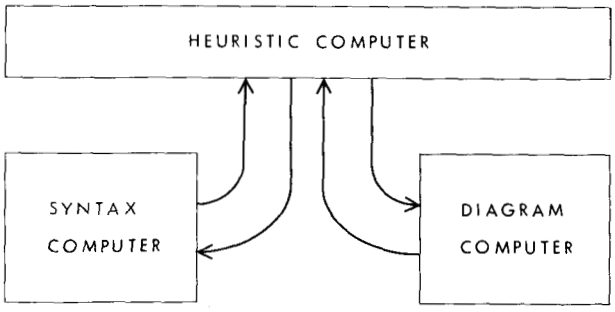


Figure 4 **Flow chart of the artificial geometer.**

and floating-point calculations. However, it keeps all this secret from the heuristic computer and reports only qualitative information of the type acquired by a mathematician in scanning a well-drawn figure. The behavior of the heuristic computer and the syntax computer would not be changed if the diagram computer were replaced by a machine that could draw diagrams on paper and observe them.

The heuristic computer does most of the things that have been discussed in this report. It contains the heuristic rules and decides what to do next. The subordinate computers only follow its instructions and answer its questions.

The program is being written in an information-processing language constructed by appending a large set of special functions to the Fortran compiler for the IBM 704. The language increases manyfold the ease of writing programs of the nature of the geometer, and will be reported upon in detail in a subsequent paper.

Learning in intelligent machines

The machine described thus far will exhibit intelligent behavior but will not improve its technique. Except for the annexation of previously proved theorems to its axiom list, its structure is static. A rigorous sequence of practice problems will not improve its performance at all in solving a given problem unless a usable theorem is among them. Such a machine, incapable of developing its own structure, will always be limited in the class of problems it can solve by the initial intent of its designer. It seems that the problem of designing a machine of general intelligence will be enormously greater, if at all possible, than designing one not so intelligent but with the capacity to learn.

One might attempt to endow an automaton like the geometer with the ability to learn at various levels of sophistication. Indeed, the behavior of the machine in storing away for future use each theorem it has proved may be interpreted as learning of a rudimentary sort. This might be refined by having the machine become selective in its choice of theorems for permanent storage, rejecting those which (by some well-defined criteria) do not seem to be sufficiently "interesting" or general to be useful later on. Similarly, instead of "forgetting" all lemmas it might have established as intermediate steps in the proof of the theorems offered to it, to be rederived when needed, the machine might select the especially interesting ones for its list of established theorems.

The next level of learning is indicated when the machine adjusts, on the basis of its experience, the probability for success it assigns to a given heuristic rule for a theorem with a given character. This is the learning involved when the machine uses results on one problem to improve its guesses about similar problems. As the geometer is given problems of a given class, say problems about parallelograms, its ability to handle them would improve. After it had been given a graded sequence of harder and harder problems, its performance should be much better and it could be said to have learned to prove parallelogram theorems. The highest level to which we aspire for an early model of the geometer will be achieved when it looks over the quality of its predictions and discards as irrelevant some of the criteria that comprise the problem character. The earliest models of the geometer will include only low levels; later models will be more complex.

Beyond these kinds of learning we can see other things. Before we come to them, however, we will probably be working on machines to solve harder problems than those of geometry. There are kinds of learning that are needed only by machines that take their environment more seriously than do theorem-proving machines. But we can hope that a theorem machine might some day be able to observe that a certain sequence of methods was effective in certain circumstances, and consequently streamline the sequence into a single method and in this way devise a new method.

But in still another vein there are possibilities for theorem machines. Instead of providing a machine with a formal system and a sequence of propositions to prove, one could give the machine a formal system and ask it to see what it could find. Here it would at least need criteria for the utility of theorems in proving other theorems and for the elegance of a proof in terms of large achievements in a small number of steps. New kinds of learning would be used here.

Before closing the subject of learning machines, there are some further considerations to deal with. A computer is, after all, merely a finite automaton, and, as such, its behavior is completely determined by its internal state at the beginning and by subsequent input information. This being the case, it can be argued that its response to any set of input signals is in principle predictable and is consequently uninteresting and not worthy of the description "intelligent." Another version of this objection is the following. The machine, endowed with heuristics and judgments of its designer, is but a trivial extension of that person, in principle no different from a slide rule in the hands of an engineer.

From a certain irrelevant point of view the objection is justified, but in practice the behavior of the machine is far from being predictable. That this is indeed the case is well illustrated by the fact that the geometer, its operation simulated "by hand," has on several occasions produced a proof that was a complete surprise to its programmers.

The nature of an intelligent program is such that unlike a conventional arithmetic computation, in which the branches are few and easily traceable, the number of conditional branches depending on the input are bewilderingly many and highly interdependent, rendering impossible any detailed attempt to trace its behavior. And of course, once learning is introduced into the program, it will constantly modify itself in a highly complex way, so that while its behavior is still in principle determined, one will become increasingly powerless to predict its response in any given case. In a very real sense, the machine's proofs will be no more or less trivial than those offered by the neophyte mathematician who is still under the influence of his professor.

One may view this machine in still another way. At any instant of time, the internal configuration of our machine is some particular state of a finite-state automaton. Then of the infinite number of sequences that one might ask the machine to establish as theorems, some infinite subset of these will be provable by it. At any given time, our machine represents a partial decision method over this infinite set of theorems, and this set will be richer in "interesting" theorems than a random subset of all theorems. The class of theorems considered "interesting" will determine the heuristics that control the partial decision method, and in turn, the density of interesting theorems in the set enumerated by the machine will depend on the apt choice of the heuristics. It is important to note that if even the most rudimentary learning behavior is built into the machine, its initial internal configuration will be different for each new problem presented to it, and consequently, the class of theorems decidable by the machine will be continually changing. And what is any human mathematician but a partial decision machine over some unknown class of theorems?

It is possible to approach the problem of theorem-proving by machine from a rather different direction. E. W. Beth[13] describes a method *(semantic tableaux)* for systematically constructing a counterexample for a proposed theorem if there is one, or else establishing the fact that none exists. If it can be shown that a counterexample cannot be constructed, an algorithm is given for converting the "closed" semantic tableau produced into a proof of the theorem in the formal system. But the method of semantic tableaux is essentially an enumeration procedure—in this case, it is the set of individual instances of the theorem that could possibly be counterexamples to the theorem that is being enumerated, and like all such procedures, the bulk of calculation required rapidly outdistances the capacity of conceivable computing machines. In order to make the procedure reasonably efficient, heuristic rules for the control of the enumeration must be introduced, and one is faced with essentially the same problem that concerns the body of this paper. The more or less anthropomorphic approach followed by the authors has the advantage that suitable heuristics are readily suggested by introspection, and the methods developed are more likely to be applicable to the solution of problems in nonformal systems.

The theory machine

At various points in the preceding discussion, a line of reasoning was terminated by the comment that harder problems exist but they are outside the scope of the matter being considered. This large new class of problems, and how a machine can handle them, is the subject of this section. We consider now a machine that takes its environment more seriously.

The subject will be introduced by an example of a more advanced kind of geometry machine, a machine that tries to learn what kind of geometry fits the environment it finds around it. The heuristic computer is provided with an environment by the diagram computer. It looks to the environment for heuristic—for clues about what to do next. However, if it learns that a measurement contradicts something it can prove in the syntax computer, it assumes the measurement is in error. In other words the formal system is sacrosanct.

Now suppose the diagram computer is replaced with another that does its drawings on the surface of a sphere. Suppose further that the priorities in the heuristic computer are readjusted so that it believes the diagram computer rather than the syntax computer when the two are in conflict. Suppose also that it is provided with the means to modify the formal system and additional heuristic to enable it to do so efficiently. It would be arranged so as to try to bring theory (the syntax computer) and experiment (the diagram computer) into harmony, and thereby discover what kind of a world it lives in. This is a theory machine.

There seems to be, in principle, no reason why a theory machine should not be fitted with the means to do experimentation, with a tool room, a stockroom, and an instrument room, and told to work out the theory of something or other. In practice, there is the familiar difficulty of speed and cost. Today it is cheaper and quicker to use people to do research, but perhaps some day machines will do the research and people will merely control the doing of research. This is precisely parallel to the digging of excavations. At one time excavating was manual labor, but now machines do the digging and people merely control the machines. The scientist using a machine to do research would have a role analogous to that of a university professor directing his graduate students.

A further conjecture along this line relates to programming. A person finds it much easier to communicate a complex message to another person than to a machine. Speaking is relaxed and easy, while writing a program of machine instructions is detailed and exacting. When one person listens to another he often fails to interpret some word correctly for a while, but later other words enable him to understand the earlier word. It seems as if the listener is continually generating hypotheses about what the speaker means and is continually checking these hypotheses and accepting them or rejecting them and casting about for others. In terms of human activity, theorizing is much too pretentious a word for this activity. However, from the point of view of machine design, it may be that only a theory machine will be easy for people to instruct.

The interaction between formal and heuristic procedures in a theory machine is more intricate than in a theorem machine. To determine the consequences of its present hypothesis the theory machine must use the methods of the theorem machine. Because of the different nature of the typical problems it will be solving, the theory machine must lean more heavily on semantic heuristic as a substitute for rigorous deduction. Then when it finds a discrepancy between theory and experiment it must use both rigorous deduction and heuristic procedures to modify its formal system. An interesting feature of such a machine is that the rules for formal deduction used to modify the formal system are actually part of the formal system. This is not an unreasonable situation; it is essentially what happens when the program for a calculator causes the calculator to modify the program. But it surely is complicated, and the complication does not end here.

The machine described so far resembles a theoretician with little or no experimental skill. Additional heuristic is required to enable the machine to select a "clean" experiment that will be an effective test of a theory. Contingencies will arise in the experimentation, and the machine must handle these as subproblems. In other words it must invoke this whole apparatus over again at a lower level.

The theory machine is a device that conjectures about its environment and tests its conjectures. In so doing, it gains an increased understanding of what is going on. It is hoped that not only will the theory machine be able to do research, but will also be easier to communicate with than a present-day automatic calculator.

Summary

In contrast to the present use of automatic calculators which outperform human beings in clerical tasks, the theorem machine is proposed as a device that reasons heuristically. It is therefore able to solve harder problems, and the study of it reveals some things about the nature of problems and of machines. The essential operating principle of this kind of artificial intelligence is that it has a formal part, a syntax computer that can make deductions, and a heuristic part that can make guesses. By using the syntax computer to test the guesses made on a heuristic basis, the machine is able to get results that are beyond the scope of a purely deductive machine.

Heuristic processes can be syntactic, whereby they depend on the language in which the problem is stated, and on the statement in that language, or they can be semantic and depend upon an interpretation or model of the formal system.

The artificial geometer is an example of a theorem machine. Geometry was chosen, not because of any inherent interest, but rather because it provides an example of a problem at the right level of difficulty that needs semantic heuristic in a major way. The geometer is being studied by simulation on the IBM 704 Electronic Data Processing Machine.

An interesting aspect of the geometry taught in high school is that it is not rigorous. Some facts are established by proving them and some by observing the figure (i.e., semantic heuristic). This is a powerful, effective method of reasoning used by people and by the artificial geometer. While it would be possible, and probably easier, to make the artificial geometer perfectly rigorous, it is more significant in the study of artificial intelligence to avoid the strictness of rigor that is a proper part of metamathematics but not efficient in mathematics.

Beyond the theorem machine is the theory machine which, by conjecturing and testing the conjectures, gains an understanding of its environment. Such a machine should be able to do research and should be easier to communicate with.

The largest obstacle to the development of useful theorem and theory machines is the problem of speed. This cannot be cured by faster components alone. The major contribution to speed must come from improved heuristic so that the machine will waste less time in fruitless endeavor. The nature of hard problems insures that the machine must waste some time on wrong hunches, but the waste must be kept within bounds. The machines themselves are expected to make a major contribution to the understanding of artificial intelligence because they learn as they work, and what they learn reveals much.

Acknowledgment

The authors wish to acknowledge the contributions of A. Newell, J. McCarthy, M. L. Minsky, and H. A. Simon, whose relation to the project has been indicated in the text, and to C. L. Gerberich, J. R. Hansen, and R. M. Krause, whose technical and programming contributions are making the project possible. Professor McCarthy, in particular, has been playing a continuous role as consultant to the authors.

References and footnotes

1. Newell, Shaw and Simon, "Empirical Explorations of the Logic Theory Machine. A Case Study of Heuristic," *Proceedings of the Western Joint Computer Conference,* p. 218 (February, 1957).
2. Newell and Simon have used the term *planning space.*
3. Particularly J. McCarthy, M. L. Minsky, and one of the authors (N.R.)
4. A. Newell and H. A. Simon, *I.R.E. Transactions on Information Theory,* **IT-2,** No. 3, 61 (September, 1956).
5. A decision procedure applied under the constraint of a time limit behaves as if it were a heuristic.
6. There are classes of problems, for example proofs of theorems in number theory, where it can be shown that no decision procedure can be devised. Heuristic pro-

cedures should enable machines to solve problems that are members of such classes. It should be evident that no set of heuristics together with the programs to employ them can guarantee that a machine will solve every member of such a class. All a machine can do is to probe around and perhaps come up with an answer. This, of course, is all that people can do. It should be evident, too, that a program utilizing heuristics can well be an algorithm that is guaranteed to solve any member of some class of problems. Such a class must, of course, be amenable to a decision procedure. The contribution of an individual heuristic here is that it may lead to a short cut. The geometry theorem machine will probably be an algorithm of this type.

7. A. M. Turing, "Computing Machinery and Intelligence," *Mind,* **59,** 433 (October, 1950).
8. In the case of a theorem contingent upon a set of hypotheses, the proof is developed in an extended system in which the hypotheses are appended to the original set of axioms. The transformation of this categorical proof to the desired hypothetical one is trivial.
9. Newell and Simon, in private communication with the authors, have described an abstract model for a propositional calculus which is not a semantic interpretation, but which, in fact, is another formal system in which it is trivially easy to prove the transformed theorems. Since this is a true heuristic, it is not always possible to transform back to the problem space.
10. The term *character* was introduced by Minsky (M. L. Minsky, "Heuristic Aspects of the Artificial Intelligence Problem," Lincoln Laboratory Report 34-55, December, 1956) and is to be understood in its dictionary sense. The particular machine representation of a theorem character selected by the authors differs somewhat from that of Minsky.
11. The completed proof will use a deduction metatheorem to get $\vdash \{H\} \supset G_0$ from $\{H\} \vdash G_0$.
12. A. M. Turing, *Proceedings of the London Mathematical Society, Series 2,* **24,** 230-265 (1936).
13. E. W. Beth, "Semantic Entailment and Formal Derivability," *Mededelingen der Koninklijke Nederlandse Akademie van Wetenschappen, afd. Letterkunde, Nieuwe Reeks,* **18,** No. 13, 309 (1955). See also: Abraham Robinson, "Proving a Theorem (as done by Man, Logician, or Machine)," *Transcription of the Proceedings of the 1957 Cornell Summer Institute of Logic,* Ithaca, New York.

Received June 4, 1958