MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A.I. LABORATORY

Artificial Intelligence January 1974 Memo No. 303

Plane Geometry Theorem Proving Using Forward Chaining

by

Arthur J. Nevins

Abstract

A computer program is described which operates on a subset of plane geometry. Its performance not only compares favorably with previous computer programs, but within its limited problem domain (e.g. no curved lines nor introduction of new points), it also invites comparison with the best human theorem provers. The program employs a combination of forward and backward chaining with the forward component playing the more important role. This, together with a deeper use of diagrammatic information, allows the program to dispense with the diagram filter in contrast with its central role in previous programs. An important aspect of human problem solving may be the ability to structure a problem space so that forward chaining techniques can be used effectively.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0003.

1. Introduction

The use of the diagram as a semantic model has been the cornerstone of previous attempts to mechanize plane geometry theorem proving (e.g. the early pioneering work in [1] and [2] and more recently [3]). It also has been the object of attention elsewhere in the literature [4], [9], particulary since it suggests the use of analogous models for other problem domains.

There are four benefits which humans derive from a diagram in plane geometry. The first is mnemonic. There is growing evidence [5] that humans are severely limited in short term memory. The diagram makes it easier to remember and process new geometrical relationships. Second, it focuses attention on line segments that appear in the initial diagram and which therefore receive higher priority than those segments that have not yet been drawn. Third, the diagram contains useful and often essential non-numerical information concerning the relative orientation of points. Finally, the diagram acts as a filter to reject goals not consistent with its numerical representation (e.g. one would not attempt to prove two sides equal if they appear in the diagram to be manifestly unequal).

However, which of the above four uses of a diagram are applicable to a modern computer? The mnemonic role does not apply since the computer has ample short term memory. The heuristic governing the use of line segments definitely is applicable but this information can be obtained directly from the axioms without the need of an actual diagram, as will be shown in section 2. The computer program which will be described in this

paper employs diagrammatic information of a non-numerical nature as will be shown in section 2; but it does not use the numerical information in the diagram and it is an open question whether it has lost much in the process. This suggests the possibility that the use of the filter mechanism by humans might be conditioned in part by the mnemonic use of the diagram. Since the present program operates directly from the predicates of a problem, it does not require a human to provide it with a diagram.

There is a close connection between the prominence given to the diagram filter by previous computer programs and the heavy reliance of these programs on backward chaining as a method of inference. Early work in artificial intelligence [8] had emphasized the importance of backward chaining (i.e. reasoning backwards from the goal to the axioms) as opposed to forward chaining (i.e. reasoning forward from the axioms to the goal). Although most researchers admit the need for both forms of inference, there has been a certain timidity about using the forward mode. The fear is that forward chaining might allow the generation of a disastrous number of irrelevant statements, particularly when dealing with large data bases. Backward chaining at least guarantees that the computational effort is linked to the goal one is trying to prove. However, the comparison is not quite that simple. The size of the data base may be less important than the manner by which the data base is organized and the heuristics employed in its management. The

addition of new facts to the data base (as provided by forward chaining) can be a source of great strength if these facts are relevant to the goals of the problem solver. These facts will stand a much greater chance of being relevant if the search can be confined to a relatively small compartment specific to the given problem domain rather than from a homogeneous pool of knowledge. The present use of forward chaining in plane geometry was made effective (1) by an efficient representation of the data base and (2) by confining its normal use to those points and line segments that were implicit in the statement of the problem (and which ordinarily would appear in the initial diagram). In the opinion of this author, the inability to make meaningful use of forward chaining is a sign that the problem space may not have been structured properly; this is a heuristic that should not be ignored when designing future computer programs which aspire to humanlike intelligence.

A potentially dangerous use of backward chaining can develop if it leads to the generation of an AND/OR goal tree. Such a tree arises when the attempt to solve a goal results in an OR bundle consisting of several subgoals where (1) the satisfaction of any of these subgoals would solve the goal, and (2) at least one of these subgoals B generates an AND bundle consisting of several additional goals which must all be solved in order to satisfy B. Although some attempts have been made to devise procedures for handling such trees (10), it is often difficult to prevent an explosion of subgoals from taking place. In plane geometry, (1) the attempt to prove something by

congruent triangles could generate an OR bundle (i.e. any of several different methods or different pairs of triangles might be sufficient to prove the goal), and (2) the attempt to prove a pair of triangles congruent by a particular method (e.g. side-angle-angle) could generate an AND bundle. The reason the diagram filter was so vital to previous plane geometry theorem provers is that it helped to contain this explosion by rejecting those subgoals that were false in the diagram. By contrast, the present program does not need a diagram filter because its limited use of backward chaining does not generate an AND/OR goal tree. In particular, it eliminates the AND aspect of the tree by requiring that no subgoal of an AND bundle be attempted unless all the other subgoals of that bundle appear already as statements in the data base. The reason the program can afford to be so stringent in its use of backward chaining is that it relies heavily on forward chaining to add new statements to the data base.

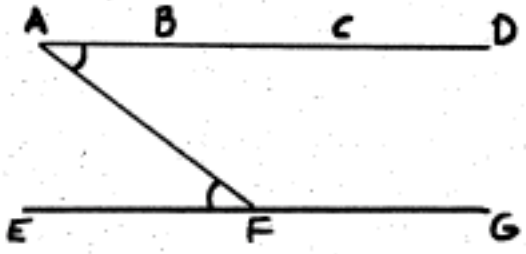
The success of the program rests in part upon its representation of the problem domain and this is described in section 2. The organization of the program is covered in sections 3 and 4; section 3 focuses on forward chaining and section 4 on backward chaining. The techniques in sections 3 and 4 make important and extensive use of paradigms; these paradigms (each based upon a mental picture of some possible situation from the plane geometry world) provide the motivation for different LISP routines used in the program.

The main limitations of the program are its neglect of

curved lines and the introduction of new points. Nevertheless, the problem domain is far from trivial as should be evident from examples of the program's performance presented in section 5. Some suggestions for future work are discussed in section 6.

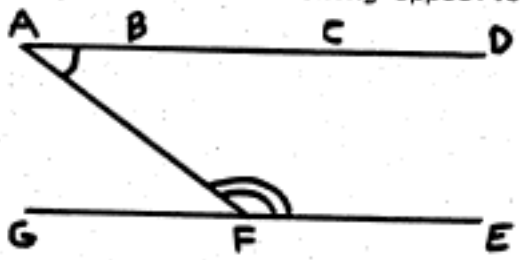
2. The representation of lines, angles, and equalities

The program regards a straight line as an ordered sequence of points. Thus, if it is told that (A B C D) is a straight line, it assumes that (1) B lies on this line between A and C and (2) C lies on this line between B and D. A line is represented by the longest line known to pass through the given set of points (i.e. the set of points (A B C) would not be regarded as a distinct line but rather as a subset of the line (A B C D)). The ordering of points on the line is primarily to determine what (if any) points lie between any two given points on the line rather than to determine whether one point comes before another. The only time when actual precedence matters is when one is dealing with parallel lines since parallel lines in a diagram reflect an implicit assumption concerning the ordering of points on these lines. Thus, in the following diagram



line ABCD can be represented either as (A B C D) or as (D C B A); however, the former representation would require line EFG to be represented as (E F G) whereas the representation (D C B A) would

require that line EFG be represented as (G F E). This is to inform the program about the relative orientation of the two parallel lines since the alternate interior angles BAF and AFE would be equal in the orientation shown in the above diagram but would not be equal in the following opposite orientation:



The program recognizes six predicates LN, PR, PRP, RT, ES and EA which will be described in this section. The statement (LN K) says that the list K is a straight line. The statement (PR K L) says that line K is parallel to line L and defines a relative orientation of the points in K with those in L as described above. The statement (PRP K L) says that line K is possibly parallel to line L. This is to give the program the ability to conclude for itself that line K is parallel to line L by providing it with the proper relative orientation of K and L as this is information that would be present in the diagram. Thus, if the problem had a diagram which showed two lines K and L that looked as if they conceivably could be parallel, the program would be told (PRP K L). The statement (RT A B C) says that ABC is a right angle. The statement (ES A B C D) says that side AB = side CD whereas the statement (EA A B C D E F) says that angle ABC = angle DEF.

The performance of a theorem prover can be affected greatly by the way it handles the equality relation. The success

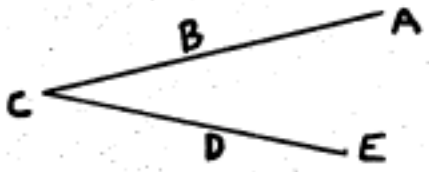
of (6) in domains such as group theory and number theory certainly was due in part to its treatment of equality and this included built in knowledge about associative and commutative operators. In plane geometry, even a simple statement like side AB = side CD has 8 equivalent representations (i.e. (ES A B C D), (ES A B D C), (ES B A C D), (ES B A D C), (ES C D A B), (ES D C A B), (ES C D B A) and (ES D C B A)). If one adds the statement side CD = side EF, the transitivity of equality would cause the number of statements to grow to 24 (i.e. 8 variations of AB = CD plus 8 variations of CD = EF plus 8 variations of AB = EF). The use of forward chaining might not be such a good idea if it were coupled with a representation that generated a host of redundant statements for each new statement added to the data base.

The present program uses the function ALPHALESSP available in LISP at M.I.T. to create an unambiguous representation for the expression "side AB." It does this by providing an alphanumeric ordering on points in the diagram. Thus, side AB would be represented as AB rather than BA since A appears before B in the alphabet. The representation for the equality of two sides is determined from a lexicographic ordering based upon the function ALPHALESSP; first, representations x and y are determined for each side and then the lexicographic ordering is applied to the lists x and y to determine which is to be regarded as the first side (i.e. the first elements of x and y are compared and whichever has the lower value based upon ALPHALESSP determines the choice between x and y; in case of a tie the second elements of x and y are compared). For example,

the statement (ES B C D A) would be replaced by (ES A D B C) since (1) side BC is represented as (B C), (2) side DA is represented as (A D) and (3) (A D) comes before (B C) in the lexicographic order as the first element of (A D) comes before the first element of (B C). Similarly, (ES D B B C) would be replaced by (ES B C B D).

The transitivity of equality is effected simply by grouping together all sides (or angles) known to be equal. Thus, the statements (ES C A D B) and (ES B D F E) would be replaced by (ES A C B D) and (ES B D E F) respectively and then gathered together as ((A C), (B D), (E F)) and inserted in the list that keeps track of all equal side relationships.

The representation of angles involves a complication not present in the representation of sides. Whereas there were only two possible ways to represent a side (depending upon which end point is listed first), there can be many possible ways to represent an angle. Thus, the following angle

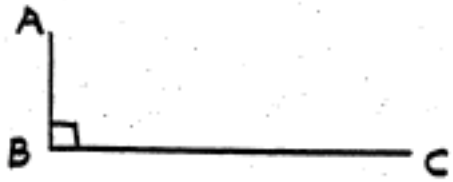


can be represented as ACD, DCA, ACE, ECA, BCD, DCB, BCE, or ECB. The program adopts the convention of selecting those points which lie closest to the vertex of the angle. In the above angle this would narrow the choice to BCD or DCB with the final selection determined from the alphanumeric ordering of the nonvertex points (i.e. BCD would be chosen over DCB since B appears before D in the alphabet). Otherwise, the representation of equal angle

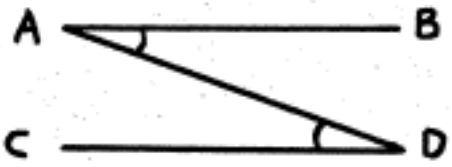
relationships is analogous to the treatment of equal sides.

The program gathers together homogeneous data. Thus, separate lists are constructed for (1) straight lines, (2) pairs of parallel lines, (3) pairs of potentially parallel lines, (4) right angles, (5) groups of equal sides, and (6) groups of equal angles. Suppose for example that the program suddenly discovers that angles DEF and GHI are equal. It first would determine the groups in which these angles appeared and then would merge the two groups. Thus, if the two groups were (ABC, DEF) and (GHI, JKL) (i.e. it having been determined previously that ABC = DEF and GHI = JKL), the new group would be (ABC, DEF, GHI, JKL). Next, the program would explore consequences of this new equality grouping using methods described in section 3.4. It not only would explore consequences of DEF = GHI, it also would explore consequences of DEF = JKL, ABC = GHI, and ABC = JKL. Furthermore, if (ABC, DEF) had been the right angle group, the consequences of the new discovery that GHI and JKL must be right angles would be explored using methods described in section 3.5.

The lines that appear in the initial diagram have heuristic value in that they are likely to be involved in the proof of the theorem. We already have seen how the program is told about lines from the predicates LN, PR and PRP. It also learns about lines from the predicates ES, RT and EA. Thus, (ES A B C D) tells the program that both AB and CD are lines (unless of course they are subsets of other lines). The statement (RT A B C) tells the program that AB and BC are lines, but it would not conclude from this that AC is a line since such an angle would be

drawn as 

Similarly, (EA A B C D E F) tells the program that AB, BC, DE and EF are lines. The program normally would not introduce any additional lines during the forward search (an exception to this rule is described in example 1 of section 5), but would leave such constructions to the backward search. For example, suppose AB were parallel to CD in the following diagram:



The program would conclude during the forward search that BAD = ADC since there is a transversal running through A and D but it would not conclude ABC = BCD since there is no transversal running through B and C; the conclusion ABC = BCD would have to await the backward search where it might be made in order to help prove some goal.

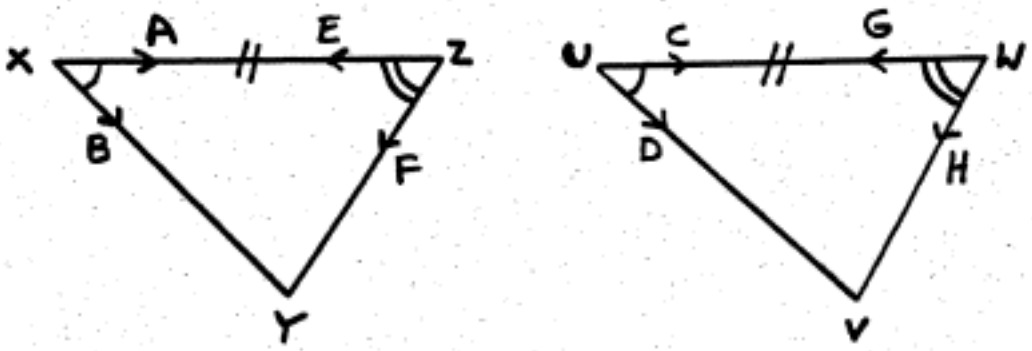
3. Forward chaining

Each time a new statement is added to the data base, the program asks whether it is identical to the goal it is trying to prove. If the answer is yes, the program terminates immediately with a successful proof. If the answer is no, the program would explore consequences of this new statement as described in sections 3.2 through 3.5.

The program first finds all the vertical angles and then examines each of the initial axioms applying the routines of sections 3.2 through 3.5 whichever section is relevant. If this fails to produce a proof, the program would proceed down the list of equal side groupings in a manner to be described in section 3.1 If an exit is made from the executive of section 3.1 and a proof still has not been found, the program would go directly to the backward search described in section 4 which if successful would solve the problem. Whenever the backward search ends in failure, the program asks whether any new facts had been added to the data base by the previous execution of section 3.1. If the answer is no, the program would terminate its search in failure. If the answer is yes, the program would loop back to another execution of section 3.1.

3.1 The equal side executive

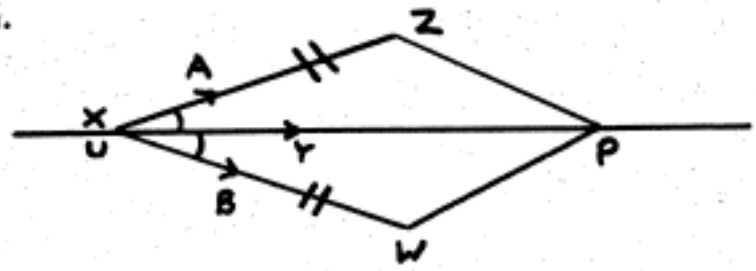
The program looks at the first group of equal sides and within this group it looks at each pair of sides such as XZ and UW. It attempts to find new congruent triangles based upon the known equality of XZ and UW. Each attempt is based upon a different paradigm. The first such paradigm is reflected by the following diagram



and has this explanation. The program has searched the equal angle groupings and found a pair of equal angles AXB and CUD where (1) point A lies along line XZ but X does not lie between A and Z and (2) C lies along line UW but U does not lie between C and W. It then continued its search among the equal angle groupings and found a pair of equal angles EZF and GWH where (1) E lies along line XZ but Z does not lie between E and X and (2) G lies along line UW but W does not lie between G and U. Finally, the program found that lines XB and ZF intersected at point Y and lines UD and WH intersected at point V where (1) X does not lie between B and Y, (2) Z does not lie between F and Y, (3) U does not lie between D and V, and (4) W does not lie between H and V. The program now is in a position to conclude that triangle XYZ is congruent to triangle UVW by angle-side-angle. If the program had not declared these two triangles congruent on some previous occasion, it then would conclude from their corresponding parts that side XY = side UV, angle XYZ = angle UVW, and side ZY = side WV. Each of these statements (if new) would be added to the data base and consequences of this addition would be explored using the routines from either sections 3.3 or 3.4 depending upon whether the new equality involved equal sides or equal angles. Although this paradigm was based upon a fairly simple mental picture, there still were a number of constraints that had to be satisfied in order for it to be realized. Behind each such paradigm is a LISP routine whose job is to satisfy the conditions that are implicit in the mental picture generated by the paradigm. This paper will describe (in varying degrees of

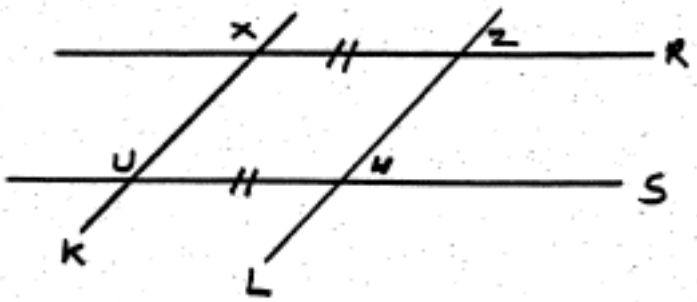
detail) the paradigms used by the program.

The next paradigm is invoked if the equal sides XZ and UW share a common endpoint (say X and U) as shown in the following diagram.



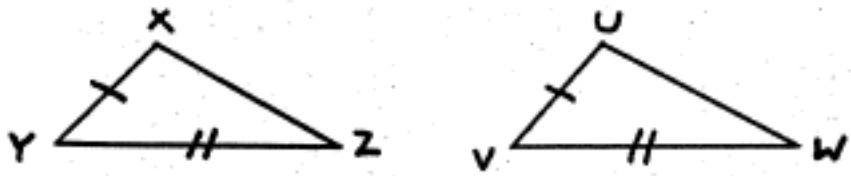
The program first found the closest point A to X along line XZ such that (1) A and X are distinct points and (2) X does not lie between A and Z. It likewise found a similar point B along line XW. It then searched for an equal angle group which contained AXY and BXY for some point Y (i.e. angle AXY = angle BXY). After finding such a point Y, it would conclude (for any point P distinct from X and situated on line XY) that triangle ZXP is congruent to triangle WXP by side-angle-side. As always, whenever two triangles are found to be congruent, the equality of corresponding parts (if new) get added to the data base and their consequences explored.

If the equal sides XZ and UW are subsets of lines R and S that are known to be parallel where X comes "before" Z on R and U "before" W on S, the program would look for lines K and L that potentially are parallel and where U comes "before" X on K and W "before" Z on L as shown below.



The program would declare UXZW to be a parallelogram and would explore its consequences (i.e. lines K and L being parallel and the opposite angles and sides of the parallelogram being equal).

For each pair of equal sides XY and UV, the program searches the equal side groupings for another equal pair YZ and VW.



The program never attempts to prove two triangles congruent if it is known that all three vertices of one of these "triangles" lie along the same line. On the other hand, if in addition to XY = UV and YZ = VW the program can find two lines K and L such that (1) (X Y Z) is a subset of K, (2) (U V W) is a subset of L, and (3) Y is between X and Z on line K and V is between U and W on line L, then it would conclude that XZ = UW since sums of equal sides are equal; it would reach this same conclusion if it found that Y was not between X and Z on line K and V was not between U and W on line L since differences of equal sides are equal.

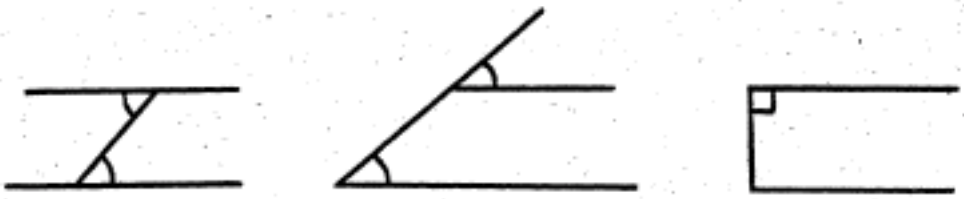
However, in the absence of any known collinearity, the triangles XYZ and UVW stand a good chance of being congruent since two of their sides already are known to be equal (i.e. XY = UV and YZ = VW). First, the program asks whether XZ and UW appear in the same equal side group and if the answer is yes, it

would conclude that the two triangles are congruent by side-side-side. Failing in this, the program examines the data base to see whether angle XYZ = angle UVW (i.e. it asks whether the unique representations for these angles appear in in the same equal angle group), and if the answer is yes, it concludes that the two triangles are congruent by side-angle-side. Otherwise, it asks whether the unique representations for either (1) angles YXZ and VUW or (2) angles YZX and VWU appear in the group which consists only of right angles, and if the answer is yes, it would declare the two triangles congruent by hypotenuse-arm. If all the above attempts fail, it would try to establish XZ = UW as the sum (or difference) of sides known already to be equal so that it could conclude congruence by side-side-side. Its final attempt would be to establish angle XYZ = angle UVW as the sum (or difference) of angles known already to be equal.

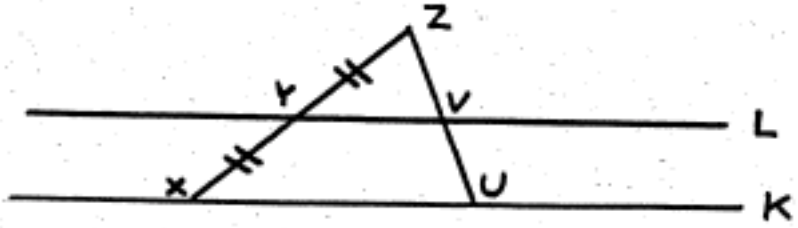
If the program has been unable to prove the theorem after having examined all such couplings XY = UV and YZ = VW, it would exit from the forward search and enter the backward search described in section 4.

3.2 Consequences of new parallel lines

Once the program has determined that two lines are parallel, it looks for transversals that cut these lines. There are three possible situations as shown in the following diagram:



The first is the normal situation where the program can determine the equality of alternate interior angles. In the second situation, the transversal has cut both lines at the "beginning" of each line and therefore the program must determine the equality of corresponding angles. The third situation is the same as the second except that there are no points on the transversal except those that lie either on or between the two parallel lines; however, since the transversal cuts one of the parallel lines at a right angle, the program can conclude that the other also must be cut at a right angle. Next the program attempts to employ the theorem that a line which bisects one side of a triangle and is parallel to the base must bisect also the other side. It does this by finding sides XY = YZ where (1) Y lies between X and Z on line XZ, (2) X lies on one of the parallel lines K, and (3) Y lies on the other line L.



For each point V (except Y) on line L, the program finds the line through Z and V and if this line intersects line K at point U, it concludes that side UV = side VZ. This last theorem actually is an instance of a more general theorem involving proportional line segments; the more general theorem could not be employed since the program had no predicate expressing the concept of "proportionality," a deficiency however which could be corrected without too much difficulty.

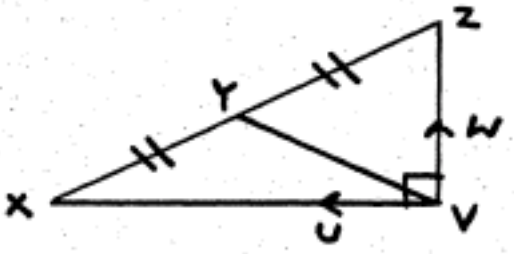
3.3 Consequences of new equal sides

This section describes immediate action taken by the program when it learns that two sides are equal. Later action taken by the program has been described already in section 3.1.

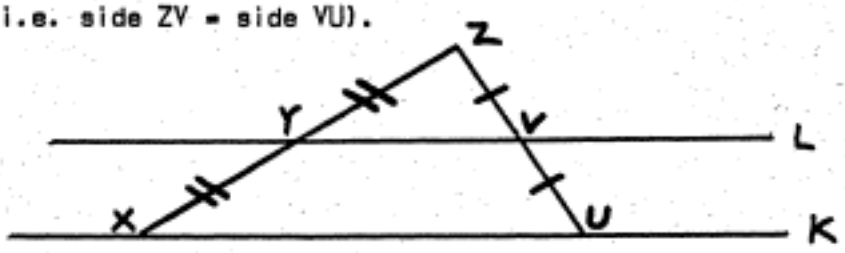
Suppose the program has just learned that side XZ = side UW. First, the program searches the equal side groupings for an XY = YZ for which Y lies between X and Z on line XZ (i.e. the line which passes through X and Z). It then looks at points V that lie between U and W on line UW. If UV and VW appear in the same equal side group, the program would conclude that side XY = side UV since halves of equal sides are equal. If either UV or VW appears in the same equal side group with XY, it would conclude that UV = VW since differences of equal sides are equal.

Suppose XZ = UW where Z and U are distinct points which lie between X and W on line XW. It then would conclude XU = ZW since either (1) XU = XZ + ZU = ZU + UW = ZW if Z lies between X and U or (2) XU = XZ - UZ = UW - UZ = ZW if U lies between X and Z.

If it learns that sides XY and YZ are equal, then (unless it knows that Y lies on line XZ) it would conclude that angle YXZ = angle YZX since base angles of an isoceles triangle are equal. However, suppose it knows that Y lies between X and Z on line XZ. First, it examines the group of right angles for a right angle UVW where (1) X lies on line UV, (2) Z lies on line VW, and (3) V does not lie either between U and X or between W and Z.



It then would conclude that XY = YV since the median to the hypotenuse of a right triangle is equal to one half of the hypotenuse. Second, it would look at the equal side group of XZ in order to find equal sides AC (i.e. XZ = AC) for which there exists at least one point B between A and C on line AC; it then would make another attempt to apply the theorems that halves (and differences) of equal sides are equal. Third, it would examine the known parallel lines in another attempt to employ the theorem that a line which bisects one side of a triangle and is parallel to the base must bisect also the other side. Finally, it would look for potentially parallel lines such as K and L where K passes through X and L passes through Y and where there is a point V (other than Y) on line L for which (1) line ZV intersects line K at a point U and (2) ZV and VU belong to the same equal side group (i.e. side ZV = side VU).



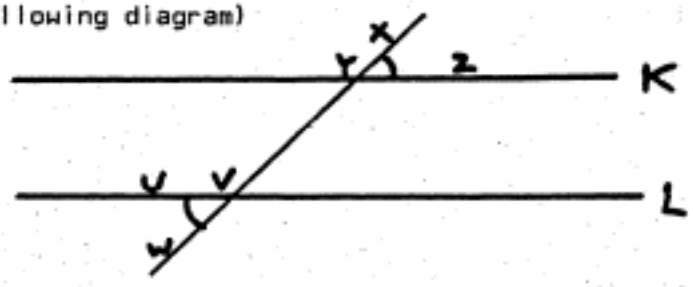
If these conditions are satisfied, it would conclude that line L is parallel to line K since a line which bisects both sides of a triangle must be parallel to the base.

3.4 Consequences of new equal angles

Throughout this section it will be assumed that the program has just learned that angle XYZ = angle UVW. If first

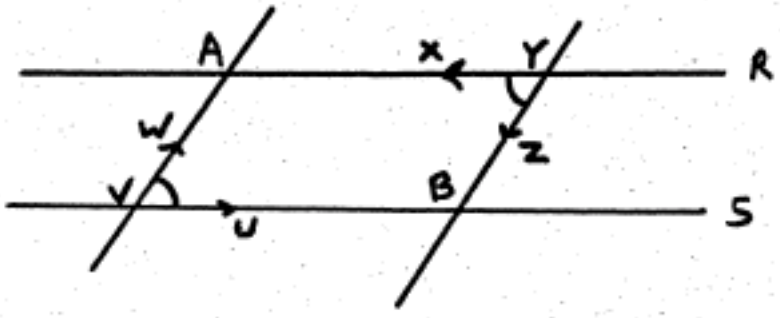
would conclude that the supplements of these angles also must be equal.

Suppose Y and V were distinct points. The program would examine all potentially parallel lines K and L where Y appears in K and V appears in L. It would attempt to show that K and L are parallel by means of the transversal through Y and V. Thus, if (as in the following diagram)

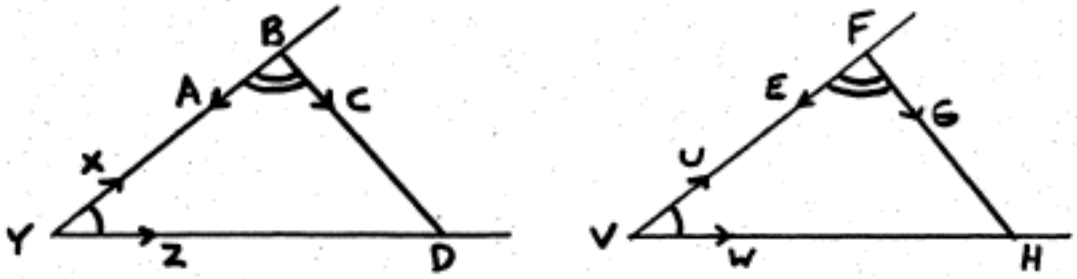


(1) Y appears "before" Z on line K, (2) V appears "after" U on line L, and (3) Y appears between X and V on the transversal, then in order for the program to conclude from XYZ = UVW that lines K and L are parallel it would be necessary that (4) V appear between W and Y on the transversal (unless UVW were known to be a right angle in which case W could appear anywhere on the transversal).

The equality of XYZ and UVW for distinct Y and V also would produce an attempt to generate a new parallelogram AYBV from known parallel lines R and S by using a paradigm which attempts to satisfy the conditions implicit in the following diagram.

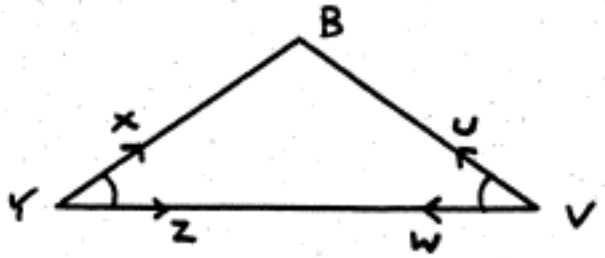


The following very powerful paradigm (which does not require that Y and V be distinct points) is illustrated in the following diagram and is based on the theorem that the sum of the angles of a triangle is the same for all triangles.



The program has searched the equal angle groupings and found a pair of equal angles ABC and EFG for which (1) A and B lie on line YX but B is not between A and Y and Y is not between X and B and (2) E and F lie on line VU but F is not between E and V and V is not between U and F. Lines BC and YZ intersect at point D where B does not lie between C and D and Y is not between Z and D. Lines FG and VW intersect at point H where F does not lie between G and H and V is not between W and H. The program then would conclude that triangle YBD is similar to triangle VFH since two pairs of equal angles must imply a third pair (i.e. angle BDY must equal angle FHV). If in addition a pair of corresponding sides are coincident, it would declare these triangles congruent by side-angle-angle; it does not attempt to establish congruence by an appeal to the equal side groupings since this eventually would be pursued in section 3.1.

Finally, the program would conclude that triangle BYV is isosceles if the conditions implicit in the following diagram are satisfied.



3.5 Consequences of new right angles

If the program discovers that UVW is a right angle, then the group of equal angles to which it belongs would be merged with the group of right angles and the consequences of any new equal angles to result from this merger would be explored by the methods described in section 3.4.

The program keeps a special list of all equal sides XY = YZ for which Y lies between X and Z on line XZ. When it discovers that UVW is a right angle, it searches this list in another attempt to apply the theorem that the median to the hypotenuse of a right triangle is equal to one half the hypotenuse.

Finally, the program looks for a supplement to the right angle UVW and (if found) declares it to be a right angle.

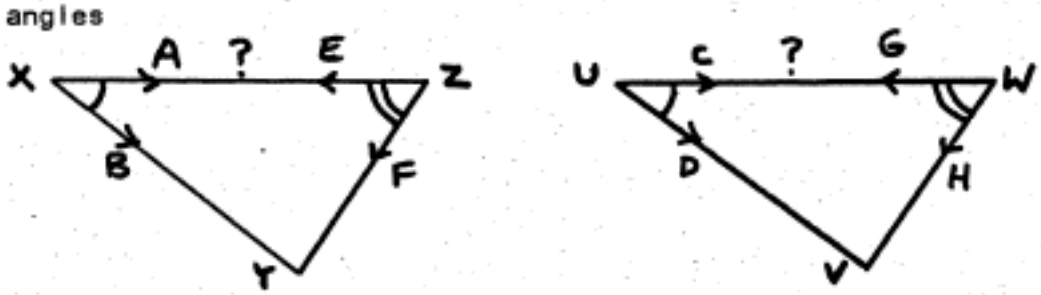
4. Backward chaining

The attempt to prove two lines parallel (or that a given angle is a right angle) is left entirely to the forward search. The backward chaining attempt to prove two sides equal (or two angles equal) depends primarily upon the ability to generate pairs of triangles that stand a good chance of being either congruent or at at least similar.

4.1 Proof of equal sides

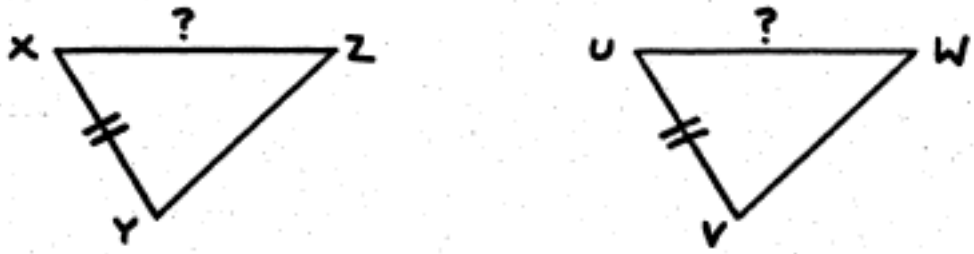
In attempting to prove that side XZ = side UW, the program first asks whether the unique representations for these sides appear in the same equal side group or else whether these two sides can be expressed as a sum (or difference) of sides known to be equal).

If this fails, it would try to generate the following triangles



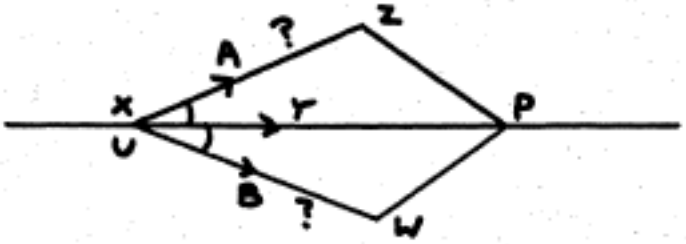
and then attempt to prove them congruent. This paradigm generates the pair of triangles in exactly the same manner as the paradigm in section 3.1. However, whereas in section 3.1 sides XZ and UW were known to be equal (so that the immediate conclusion could be reached that these two triangles were congruent by angle-side-angle), in this section it is the equality of sides XZ and UW that we wish to establish. On the other hand, if the goal XZ = UW is capable of fulfillment, then these two triangles will indeed be congruent. The program would attempt to establish this congruence by proving either XY = UV or ZY = WV.

The next paradigm (assuming failure of the previous one) searches the equal side groupings in order to find points Y and V such that the distance between Y and an end point of side XZ (say X) is equal to the distance between V and an endpoint of side UW (say U). This results in the pair of triangles XYZ and UVW

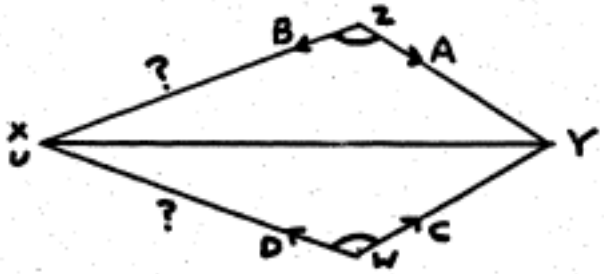


where XY = UV; the program would like to prove these two triangles congruent so that it can conclude XZ = UW. First, the program asks whether the unique representations for angles XYZ and UVW appear in the same equal angle group. If the answer is yes, then it need only establish either (1) YZ = VW, (2) XZY = UWV, or (3) YXZ = VUW in order to prove congruence. Before attempting to prove any of the above three statements (i.e. by generating a new subgoal) the program checks to see whether any can be demonstrated from an inspection of the data base; if none appear in the data base, it attempts the proof of each in turn. If it fails on all three subgoals, then it would abandon the attempt at proving the two triangles congruent. However, if the representations for angles XYZ and UVW had not appeared in the same equal angle group, the program would have asked the same question for angles XZY and UWV. An affirmative answer to this last question would mean that the establishment of any of the following three goals (1) YXZ = VUW, (2) XZY appears already in the right angle group and YZ = VW, or (3) XYZ = UVW would prove congruence. The general approach to proving two triangles congruent by now should be clear. The idea is to defer generating new subgoals until their selection can be narrowed down on the basis of information already present in the data base.

If the previous paradigm fails to prove XZ = UW, the program asks whether these two sides have a common end point. If the answer is no, it abandons the attempt at proving this goal. However, suppose the answer is yes where say X and U are identical. First, it employs the following paradigm



described in section 3.1 in order to generate pairs of triangles XZP, UWP. The only difference is that the equality of XZ and UW was given data in section 3.1 whereas we now wish to establish this equality by proving triangle XZP congruent to triangle UWP. The next paradigm is based upon the following picture



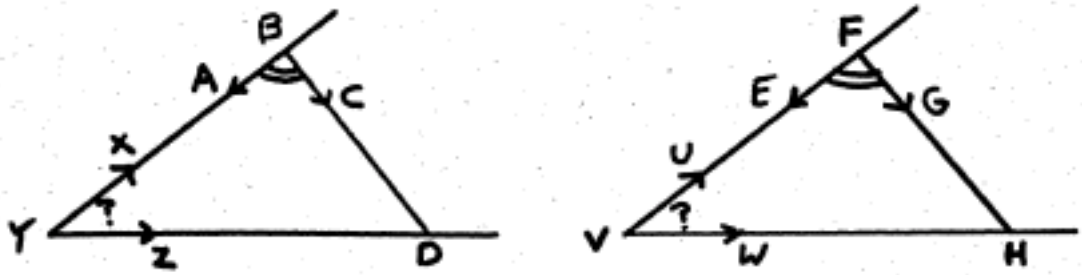
where one attempts to prove that triangles XZY and XWY are congruent. The final attempt to prove XZ = UW = XW would be to prove that the base angles of triangle XZW are equal (i.e. prove angle XZW = angle XWZ).

4.2 Proof of equal angles

In attempting to prove angle XYZ = Angle UVW, the program first asks whether the unique representations for these angles appear in the same equal angle group or else whether these two angles can be expressed as a sum (or difference) of angles known

to be equal.

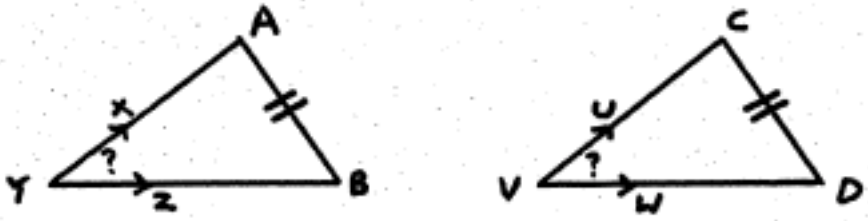
If this fails, it would employ the similar triangle generator described in section 3.4.



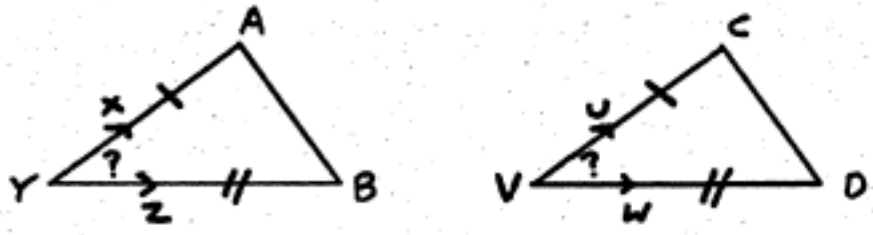
The only difference is that now we wish to prove that angle XYZ = angle UVW. Consequently, the program would attempt to prove that angle BDY = Angle FHV since if successful it then could conclude that triangles BYD and FVH were similar and hence Angle XYZ = UVW. Failing in this, it would attempt to prove triangles BYD and FVH congruent by a narrowing down process analogous to that described in section 4.1.

Failure of the previous paradigm would be followed by the following five paradigms:

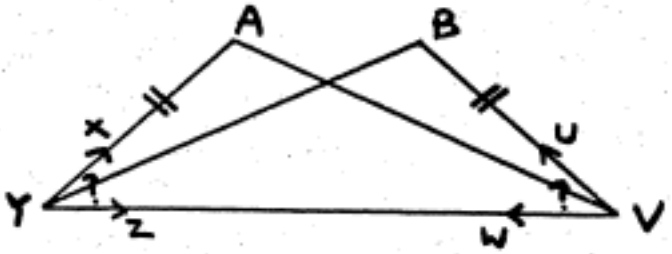
1. Prove triangle AYB congruent to triangle CVD.



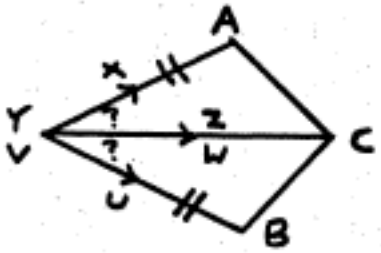
2. Prove triangle AYB congruent to Triangle CVD.



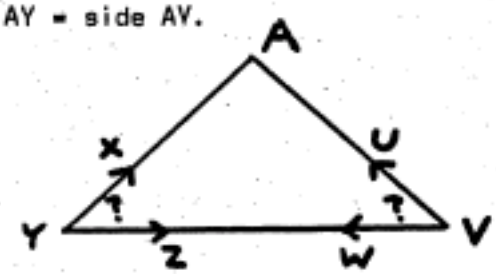
3. Prove triangle AYV congruent to triangle BVY.



4. Prove triangle AYC congruent to triangle BYC.



5. Prove side AY = side AV.



5. Computational experience

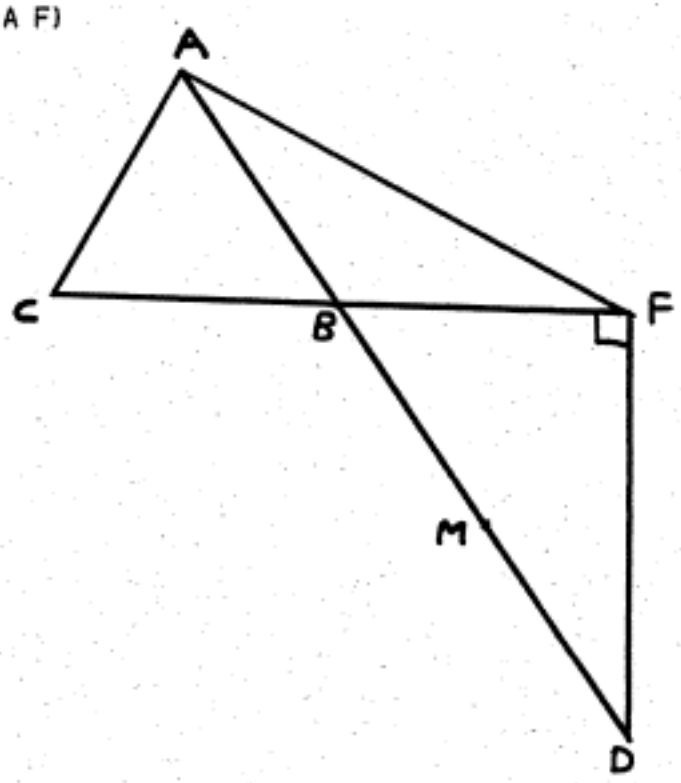
The program was written in LISP with the aid of a previous theorem proving program (7). It has solved numerous plane geometry problems on the PDP-10 computer at M.I.T. It proved all the problems reported by Gelernter (1), (2) and none of these took longer than five seconds. It also proved all the problems reported by Goldstein (3) and only one took more than about five seconds. Wong (11) described a more subtle set of geometry problems but did not present an actual program which solved these problems as his interest was in devising heuristics for introducing new points into a diagram. When provided with

the necessary additional points, the present program proved all the examples in (11) except those which depended upon algebraic facilities not possessed by the program, as in proving inequalities. Examples 10 and 30 from (11) took 85 and 40 seconds respectively. None of the other problems took more than about 20 seconds. The following five examples should provide some illustration of the program's capabilities. The reader should refer to section 2 for the meaning of the predicates LN, PR, PRP, RT, ES, and EA since the problems are presented to the program in terms of these predicates. The attendant diagrams are included solely for the benefit of the reader.

Example 1

GIVEN: (ES A C C B), (ES C B B A), (ES A B B M), (ES B M M D), (RT C F D), (LN (C B F)), (LN (A B M D))

PROVE: (RT C A F)



The following is a paraphrased version of the actual printout generated by the program during the course of obtaining a proof to example 1. It should be emphasized that this printout represents all the steps (at a certain level of detail) produced by the computer rather than just those steps that were used in the proof.

1. angle ABF = angle CBM. Vertical angles are equal.
2. angle ABC = angle FBM. Vertical angles are equal.
3. side AC = side BC. GIVEN.
4. angle ABC = angle BAC. Base angles of an isoceles triangle are equal.
5. side BC = side AB. GIVEN.
6. angle ACB = angle ABC. From steps 3 and 5 the program realizes that AC = AB and therefore concludes step 6 since base angles of an isosceles triangle are equal.
7. side AB = side BM. GIVEN.
8. side BM = side DM. GIVEN.
9. side AM = side BD. Sums of equal sides are equal.
10. right angle BFD. GIVEN.
11. side BM = side FM. The median to the hypotenuse of a right triangle is equal to one half of the hypotenuse. The construction of medians such as FM is an exception to the rule that does not allow the introduction of new line segments during the forward search. It is justified on the grounds that such a median (whose length can be linked to an already existing line segment) is relatively rare and therefore worth pursuing.
12. angle FDM = angle DFM. The program has realized immediately

from step 11 that FM and DM are both part of the same equal side group consisting of AC, BC, AB, BM, DM, FM and therefore concludes that base angles of an isoceles triangle are equal.

13. angle FBM = angle BFM. Base angles of an isosceles triangle are equal.

14. angle BAC = angle BMF. Triangles BAC and BMF must be similar since angle ABC = FBM and angle BFM = FBM = ABC = angle ACB.

15. side BF = side BM. A triangle with equal base angles is isosceles.

16. angle BAF = angle AFB. Since BF = BM = AB, it concludes that base angles of an isoceles triangle are equal.

17. angle AFM = angle CAF. Triangles AFM and CAF must be similar since angle BMF = BAC = ABC = angle ACB and angle BAF = AFB.

18. triangle AFM congruent to triangle CAF by side-angle-angle. Side AF is opposite both angles BMF and ACB.

19. side AM = side CF. Corresponding parts of congruent triangles are equal.

20. angle CBM = angle DMF. CBM is the supplement of ABC and DMF is the supplement of BMF = BAC = ABC.

21. triangle BAC congruent to triangle BFM by angle-side-angle. Angle ABC = angle FBM, side AB = side BM, angle BAC = angle BMF.

22. Triangle ABF congruent to triangle DMF by side-angle-side. Side AB = BM = FM, angle ABF = CBM = DMF, side BF = BM = DM.

23. side AF = side DF. Corresponding parts of congruent triangles are equal.

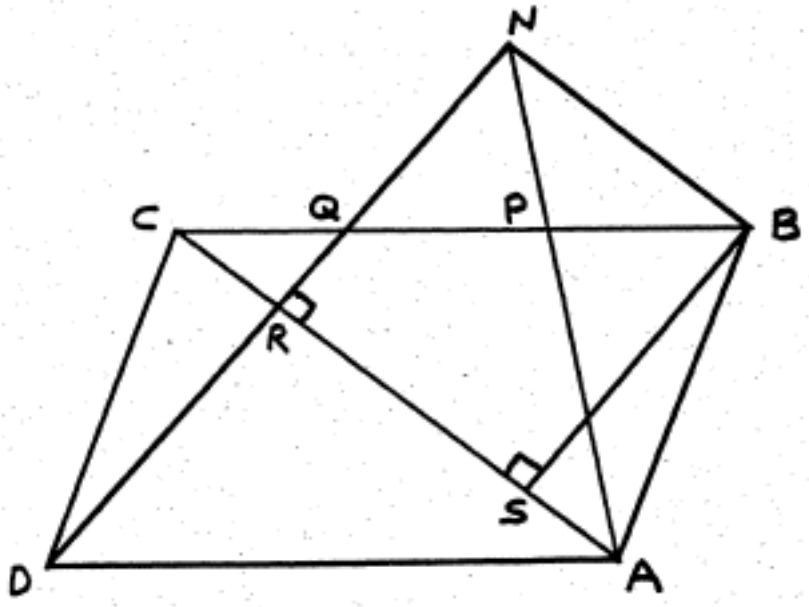
24. angle BAF = angle FDM. Base angles of an isosceles triangle are equal.

25. angle CAF = angle BFD. Triangles CAF and BFD must be similar since angle AFB = BAF = FDM and angle ACB = ABC = FBM.
26. CAF is a right angle, which completes the proof. CAF had just been identified with a known right angle BFD.

The elapsed time for the proof of example 1 was 13 seconds. One cannot help but observe the extremely high density of relevant statements generated by the program in the course of proving this theorem. High densities are characteristic of the program's performance and are probably representative of most human problem solving as well.

Example 2

GIVEN: (PR (D C) (A B)), (PR (C Q P B) (D A)), (LN (N P A)) (PR (D R Q N) (S B)), (PR (N B) (C R S A)), (RT N R A).
PROVE: (ES P Q P B).



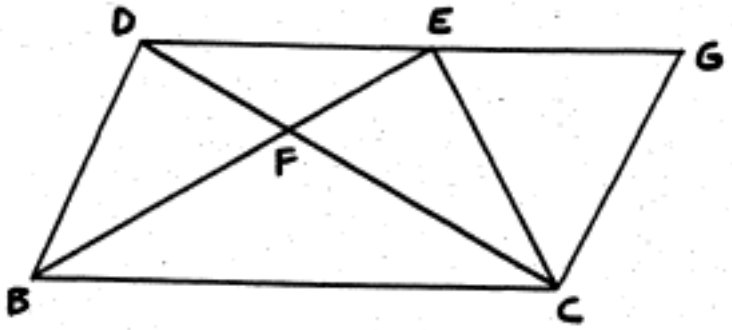
It should be noted that the point defined by the intersection of lines (N P A) and (S B) is not needed in the

proof. The computer produced proof of example 2 took 21 seconds.

Example 3

GIVEN: (ES B E C D), (EA D B F F B C), (EA E C F F C B), (EA D B E G D C), (ES E C C G), (LN (B F E)), (LN (D F C)), (PRP (B D) (C G)), (PRP (D E G) (B C)).

PROVE: (ES B C D G).

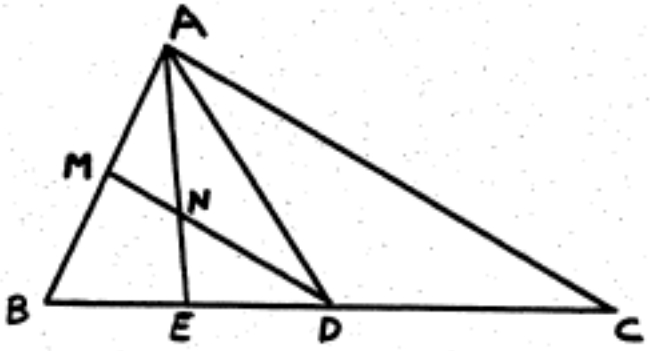


The proof of example 3 took 5 seconds.

Example 4

GIVEN: (ES B E E D), (ES B D D C), (ES A B D C), (LN (B E D C)), (LN (B M A)), (PR (M N D) (A C)), (LN (A N E)).

PROVE: (EA E A D D A C).

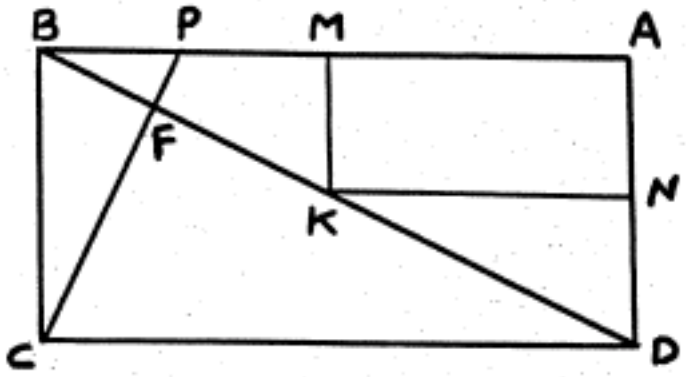


The proof of example 4 took 3 seconds.

Example 5

GIVEN: (RT B C D), (PR (B P M A) (C D)), (PR (B C) (A N D)), (LN (B F K D)), (LN (C F P)), (PR (B C) (M K)), (PRP (M K) (A N D)), (ES B M M A), (PR (K N) (B P M A)), (PRP (K N) (C D)), (ES A M B C), (ES B P P M).

PROVE: (RT B F C).



The proof of example 5 took 22 seconds.

6. Suggestions for future work

One possibility would be to introduce new predicates so as to provide the program with a richer representation of knowledge. It should not be too difficult to represent proportional (as opposed to equal) line segments but curved lines would be a more formidable task.

However, perhaps the most fundamental question concerns the introduction of new points into the diagram. Most of the paradigms described in this paper do in fact find points, although these points already were present in the diagram. Nevertheless, the significant thing about these paradigms is that they find points by a process of deduction (i.e. by reasoning from facts in the data base to points that appear relevant to the

satisfaction of a particular goal). This would suggest that similar paradigms might be constructed which deduce points that had not been present in the initial diagram.

As an illustration, point S was not essential to the statement of example 2 in section 5 but was utilized in the proof. Nevertheless, the construction of point S could have been arrived at by a paradigm which noticed that since vertical angles NPQ and APB are equal, the goal of proving PQ = PB could be expressed as corresponding parts of (potentially) congruent triangles found by drawing a line L through B that is parallel to line DRQN. The reason for drawing a line L in this manner is that its intersection with line NPA at a point M would mean that angle QNP = angle BMP so that triangles QNP and BMP are similar (and also congruent if the goal PQ = PB is true). The construction of point S then would be obtained merely by extending line L until it intersected line CRA.

References

1. Gelernter, H. Realization of a geometry theorem proving machine, Proc. Int. Conf. on Information Processing, Paris: UNESCO House, 1959, pp. 273-282; also in Computers and Thought, Feigenbaum, E. and Feldman, J. (Eds.), McGraw-Hill, New York, 1963, pp. 134-152.

2. Gelernter, H., Hansen, J.R. and Loveland, D.W. Empirical explorations of the geometry theorem proving machine, Proc. Western Joint Computer Conf., Vol. 17, pp. 143-147. Also in Computers and Thought, Feigenbaum, E. and Feldman, J. (Eds.), McGraw-Hill, New York, 1963, pp. 153-163.

3. Goldstein, I. Elementary geometry theorem proving, A.I. Memo No. 280, Massachusetts Institute of Technology, April 1973.

4. Lindsay, R.K. Inferential memory as the basis of machines which understand natural language, Computers and Thought, Feigenbaum, E. and Feldman, J. (Eds.), McGraw-Hill, New York, 1963, pp. 217-233.

5. Neisser, U. Cognitive Psychology, Appleton-Century-Crofts, New York, 1967.

6. Nevins, A.J. A human oriented logic for automatic theorem proving, J. Assoc. Computing Machinery (forthcoming).

7. Nevins, A.J. A relaxation approach to splitting in an automatic theorem prover, J. Artificial Intelligence (forthcoming).

8. Newell, A., Shaw, J.C. and Simon, H.A. Report on a general problem solving program, Proc. Int. Conf. on Information Processing, Paris: UNESCO House, 1959, pp. 256-264.

9. Reiter, R. A semantically guided deductive system for automatic theorem proving, Proc. Third Int. Joint. Conf. on Artificial Intelligence, Stanford, California, August 1973, pp. 41-46.

10. Slagle, J.R. and Bursky, P. Experiments with a multipurpose, theorem-proving heuristic program, J. Assoc. Computing Machinery, Vol. 15, January 1968, pp. 85-99.

11. Wong, R. Construction heuristics for geometry and a vector algebra representation of geometry, technical memorandum 28, Project MAC, Massachusetts Institute of Technology, June 1972