Journal of Automated Reasoning **2** (1986) 329–390.
© 1986 *by D. Reidel Publishing Company.*

Automated Reasoning in Geometry Theorem Proving with Prolog

HELDER COELHO
Laboratório Nacional de Engenharia Civil, 101, Av. do Brasil, 1799 Lisboa Codex, Portugal

and

LUIS MONIZ PEREIRA
Departamento de Informática, Universidade Nova de Lisboa 2825 Monte de Caparica, Portugal

(Received: 7 April 1985)

Abstract. This paper describes automated reasoning in a PROLOG Euclidean geometry theorem-prover. It brings into focus general topics in automated reasoning and the ability of Prolog in coping with them.

Key words. Automated reasoning, logic programming, theorem proving, knowledge representation, heuristics.

1. Introduction

We describe the development of GEOM, a geometry theorem-prover, with the objective of *achieving a better understanding* of the *capabilities of Prolog* as a tool for automated reasoning in this 'classical' AI domain.

The paper is divided into 8 sections which cover three main aspects: the definition of the problem domain, the points of friction found, the directions suggested for further research, and conclusions. We add a comparative survey of some previous work in geometry theorem proving.

We begin with the motivation inherent to our work. For each section, an introduction covers briefly the topics discussed. Section 3 presents the problem collection and the representation chosen for the basic geometric primitives. Section 4 details the program's knowledge. Section 5 discusses difficulties encountered and how they were coped with. Section 6 introduces the advantages of the separation between logic and control and how this distinction was partly implemented in GEOM. Section 7 contains directions of research suggested for further work, such as the improvement of the programming language itself. Section 8 puts forth our conclusions.

GEOM is a Prolog program that generates proofs for problems in high school plane geometry. It is divided into modules which cover geometric and arithmetic knowledge, the printing and assertion facilities and the utilities.★

★ A listing of the program GEOM written in the Edinburgh standard notation will be sent upon request. However, along this paper we used the old syntaxe available with the Prolog written in FORTRAN IV developed at Marseille, where literals have '+' or '−' according to whether they are positive or negative. Infix operators are used, and variables are preceded by '*'

This organization allows easy reading, understanding and fast updating of the program. A user presents problems to GEOM by declaring the hypotheses, the optional diagram and the goal (see Appendix 1). GEOM starts from the goal, top-down and with a depth-first strategy, outputing its deductions and reasons for each step of the proof (see Appendix 2).

This work extends previous work done by Gelernter [8, 9, 10], Rochester [8], Gilmore [11], Reiter [15], Goldstein [12], Nevins [13], and Welham [19]. Recently other work has been presented by Anderson [1, 2], Fearnley-Sandor [7], and Wen-Tsun [20, 21]. Basically, the following research question were examined:

(1) the mixture of chaining backward (top-down) and forward (bottom-up);
(2) the separation between logic and control;
(3) the introduction of new points;
(4) the introduction of line segments (constructions);
(5) the uses of a diagram;
(6) the use of geometrical symmetry;
(7) the implicit use of transitivity;
(8) the way of handling congruence relations (equivalence classes);
(9) the use of a language based on predicate calculus in a large and complex domain; and,
(10) non-proved goals in the context of constructions made.

2. Motivation

This research was mainly oriented to achieve a better understanding of the capabilities of Prolog, a programming language based on first order logic or predicate calculus, as a tool for automated reasoning. To do so so we chose a specific domain, elementary plane geometry, and we analysed how Prolog could cope with the construction of a geometry theorem-prover. Some deficiencies and limitations were found, suggesting improvements of Prolog.

Geometry theorem-provers have been attempted at times, as an exploration field during the first 16 years of Artificial Intelligence. Later on [1, 2], proving a theorem in geometry was used also to develop intelligent tutoring systems, capable to communicate to the student the logical structure of a proof and the structure of the problem solving process by which a proof is generated.

The difficulties which prevent the development and more general use of a geometry theorem-prover were stressed in a report on previous work by Coelho [3]. In Appendix 3 we present a practical guide on the analysis of early studies.

Some general questions were put forward, from the start, such as:

(1) to be attentive to the limitations and possible developments of Prolog while using it for writing a geometry theorem-prover;
(2) what geometric knowledge was needed for a collection of problems selected from previous work on geometry theorem-provers;

(3) how to have in a program a useable map of geometric knowledge.

Later on, other questions were added:

(4) how to formulate the problem: the choice of representation and canonical naming;

(5) how to identify construction strategies from known geometric constructions used in text books.

Besides these starting questions, other ones were introduced during the research, defining subgoals which are described in Sections 5 and 6.

3. Problem Formulation

The types of problem suitable for our geometry theorem-prover are here presented, by means of the description of the statement of a problem and by an example. Details are given of the representation chosen for the basic geometric primitives and the choice of a canonical naming method, i.e. how to define a unique fixed name for the geometric objects.

The representation and canonical naming have a particular influence on computation time. Moreover, further developments of GEOM will also depend importantly on how basic geometric primitives are manipulated if each time a new fact (or lemma) is derived it is desired to stored it in the data base. Also, during the proof it is often necessary to retrieve an already proved fact from the data base. Storing and retrieving huge amounts of derived facts, not always useful, may lead to combinatorial problems which must be harnessed.

3 1. THE STATING OF PROBLEMS IN GEOMETRY

The statement of a problem in geometry is done by its (optional) diagram, the hypotheses and the goal.

The *geometric diagram* is a set of points, defined by their cartesian coordinates. Its declaration is optional for the user of GEOM with minor changes to the program; when it is given it aids in the proof of the goal. However, the diagram is only a particular case of a whole class of geometric figures for which the problem (theorem) in question must be true. The diagram works mostly as a source of counter-examples for pruning unprovable goals, and so proofs need not depend on it: a proof of a theorem can be carried out without the use of a diagram. However, the diagram may also be used in a positive guiding way as described in Section 4.3

There are nine predicates with which to express the *hypotheses* of a geometry problem:

(1) the basic ones are
LINES
PR(parallel segments)
ES(equal segments)
EA(equal angles);

(2) the convenient high-order ones, definable in terms of the basic ones above, are
RA(right angles)
RECTANGLE
SQUARE
PARALLELOGRAM
MIDPOINT

There are seven predicates to indicate the top possible *goals*:

(1) the basic ones are
PR, EA, ES

(2) the convenient high-order ones, definable in terms of the basic are
RA, CONGRUENT, PARALLELOGRAM, MIDPOINT

3.2. EXAMPLE OF A PROBLEM SPECIFICATION

A geometric problem becomes defined by the optional diagram (cartesian coordinates), the hypotheses and the goal.

Let us take an example from Gelernter [10], used as input to GEOM:

DIAGRAM:
A(0,4) B(2,0) C(8,8) D(2,4) E(8,4) M(5,4)

HYPOTHESES:
LINES(AB,BMC,CA,ADME,BD,CE)
MIDPOINT(M,BC)
RA(ADB) RA(AEC)

GOAL:
ES(BD = EC)

3.3. THE PROBLEM COLLECTION

The problem collection was built with problems from previous work:

problem 1:	Gelernter [10] problem 1
problem 2:	Gelernter [10] problem 2
problem 3:	Gelernter [10] problem 3
problem 4:	Gelernter [10] problem 4
problem 5:	Gelernter [10] problem 5
problem 6:	Goldstein [12] problem 1
problem 7:	Goldstein [12] problem 2
problem 8:	Goldstein [12] problem 3
problem 9:	Goldstein [12] problem 4
problem 10:	Goldstein [12] problem 5

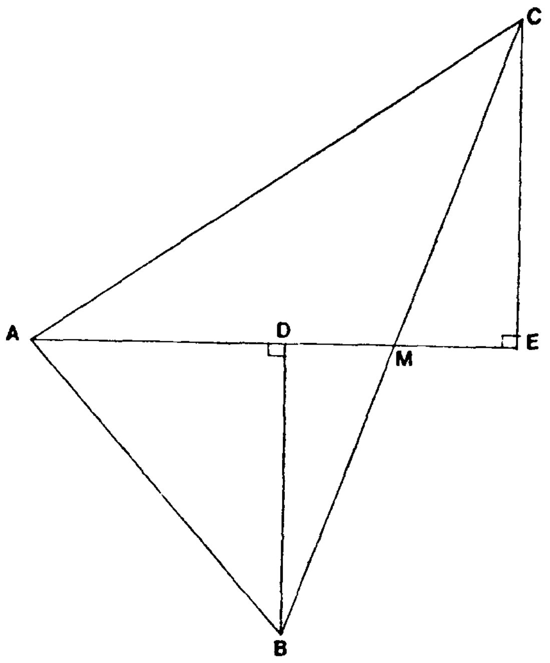


Fig. 1 The diagram of problem 3.

problem 11:	Goldstein	[12]	problem 6
problem 12:	Nevins	[13]	problem 1
problem 13:	Nevins	[13]	problem 2
problem 14:	Nevins	[13]	problem 4
problem 15:	Welham	[18]	problem 1
problem 16:	Welham	[18]	problem 2

The selection of these problems was based upon the 'degree of difficulty' of their proofs, and in order to permit comparison between program characteristics. In Appendix 1 a sample of this collection is presented. Note that for each problem only one proof is provided.

3.4. CHOICE OF A REPRESENTATION FOR THE BASIC GEOMETRIC PRIMITIVES

A general and flexible representation for the three basic geometric primitives used – *segments, directions and angles* – is needed to encode them, since these primitives form the basis of any geometric knowledge to be added to the data base. The flexibility and generality are achieved by the concept of *equivalence class*, which allows, for example, that one direction be represented by any other element of its equivalence class. An angle can be defined by three points or by two directions, as Figure 2 illustrates. A direction is represented by two points, e.g. D1 = A : B, where

$\overline{D1}$ = B : A is the opposite direction

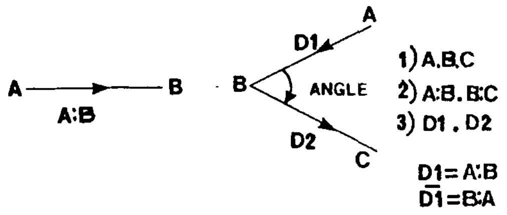


Fig. 2. The representation of an angle.

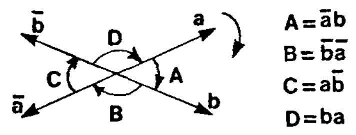


Fig. 3. The four cases arising in the representation of an angle.

Each angle segment becomes defined by two points of its direction. One direction can be defined by any pair of points belonging to the same equivalence class of pairs. In Figure 3 we can see the four cases for an angle, less than 180 degrees, defined by two directions: a and b.

Once we impose a reading direction, e.g. the clockwise direction, we may have only two cases: angles A and B, i.e. acute and obtuse. In our problem collection we have only angles less than 180 degrees and no other angles are considered.

3.5. THE CANONICAL NAMING

The *canonical naming* routines are a set of rewrite rules, applied to an expression, to get it into some standard format. They reduce the ambiguity resulting from the syntactic variations of the thin named. In fact, *canonical naming* is a technique for overcoming combinatorial problems and to make the data base inquiry easy and fast. This elimination of redundant searching is also achieved by the data base organization, as it is explained further on in Section 5.7.

In geometry, combinatorial problems are very common, partly because of transitivity, when equality and congruence relations are involved. For example, if triangle ABC is congruent to triangle DEF one can store this fact in 72 variations. When during the proof it become necessary to retrieve that triangle EFD is congruent to triangle BCA, the fact can be as just one of the possibilities of that set of variations. This is done with the use of canonical names in the geometric primitives for segments, directions and angles, i.e. a standard variation representing the thing named.

For *segments*, the endpoints are ordered alphabetically. In the following example, segment CA would be represented as AC. For segment AC, the representation would be AC itself. The case of an *angle*, DEH for example, is dealt with in the following way.

As D < H the canonical name of angle DEH (i.e. D : E . E : H) is D : E . E : H.

For angle HED, as H > D, the canonical name is not H : E . E : D, but it is D : E . E : H (there is an inversion of pairs and an inversion of each pair). In fact angles DEH and HED are the same and, thus, they have the same canonical name.

The case of an *angle* defined by two directions, D1 . D2 (e.g. A : B . C : D) expressed by different points is dealt in another way. As A is the alphabetical least of four points A, B, C, D, i.e. A < B, A < C and A < D, the canonical name of angle A : B . C : D is A : B . C : D.

Now, consider an angle defined by three points (A, B, C) or by two directions. Let us calculate, for each of these two representations, the number of alternative names for the angle when there are additional points in the two angle segments:

No. of points	RP	RD
3	2	4
4	4	4
5	8	4

RP – representation with points.
RD – representation with directions.

The representation with directions is thus recommended for angles that can be defined by more than for 4 points.

Canonical naming permits also to assert equal supplement angles when equal angles are proved. This is a consequence of the chosen representation (*directions instead of points*). The representation by points does not allow this.

4. Problem Specification

A brief description of GEOM's knowledge domain is presented: the geometric (axioms and theorems for elementary plane geometry) and arithmetic (needed for using the diagram of points with coordinates) knowledges, the utilities (the procedures available for special purposes, e.g. procedures to find points or directions) and the uses of a geometric diagram. This knowledge is sufficient to deal with the geometry problem domain covered in the previous section.

In GEOM there is a clear distinction between two components of an algorithm specification, the logic component (what it is required to be solved) and the control component (how the problem is to be solved). This separation is facilitated in PROLOG, a more descriptive or high level language than the conventional procedure oriented ones [6], PROLOG allows the programmer to explain what is the case, knowing at the same time that he is implicitly specifying to PROLOG how the case is to be searched for.

4.1. GEOMETRIC KNOWLEDGE

The geometric knowledge of GEOM, i.e. some of the axioms and theorems of elementary plane geometry, is embodied in nine procedures. They are: equal angles (EAI),★ right angles (RAI), equal magnitude (EM, EM1), equal segments (ESI), midpoints (MP), parallel segments (PRI), parallelogram (PG), congruence (DIRCON) and diagram routines.

The equal magnitude procedures convert angles to their internal representation before testing for an immediate equality or a data base equality. The midpoint procedures are of two sorts: the first is dedicated to storing a useful theorem, relating midpoints and parallels in a triangle; the second is able to generate new points.

Each procedure is organized to allow for a first look into the data base before any attempt to prove is made. Thus, for each one, the first clause provides access to the data base. To facilitate the access to a specific clause (e.g. the case for congruence routines), each of the clauses of equal angles and equal segments procedures are given a name, specified as one of the arguments of that clause.

Because each procedure may call itself through others, the search space can grow quite large, in particular when the clause for differences of segments is used. To avoid this combinatorial problem, Golstein [12] and Welham [18, 19] have not adopted the method of difference of segments.

Again, when constructions are introduced through congruence procedures, extra clauses are added to the data base and the explosive situation is aggravated. This is particularly visible for problems 13 (PR13) and 14 (PR14) of our collection on account of their large space of derivations. However, a combinatorial explosion occurs for PR14, even when the difference of segments method is not considered. The use of this congruence procedure is then compulsive and a depth-first exploration is done for each possible construction.

4.2. THE UTILITIES

The utilities are special purpose and data management procedures. As examples we mention:

(1) a counter of the number of points in the diagram;
(2) the procedure for trying congruences using bottom-up inference making on the data base;
(3) procedures for finding points and directions using diagram knowledge;
(4) a clause to get three points for defining an angle given two directions;
(5) procedure to operate on lists;
(6) clauses to verify identities (points, angles);
(7) clauses to identify opposite, same and distinct directions;

★ In parenthesis we give the name of the corresponding predicate.

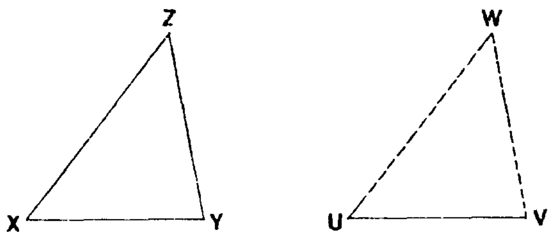


Fig. 4. Proof of two equal segments (UV = XY) by congruent triangles.

(8) procedures to verify point collinearity;
(9) clause for verifying a triangle equilateral;
(10) procedures to pick up a third side, a third equal side or a third equal angle, given the other two;
(11) procedure for the management of unit clauses in the data base;
(12) procedures for generating permutations.

4.3. THE USES OF A GEOMETRIC DIAGRAM

Two uses of the geometric diagram as a model are made:

(1) the diagram as a filter (it acts as a *counter-example*);
(2) the diagram as a guide (it acts as an *example* suggesting eventual conclusions).

As a *filter* the diagram permits to test the nonprovability of a candidate subgoal, by doing calculations with the coordinates given by the diagram. This way of rejecting goals was proposed for the first time by Gelernter [10].

The use of the diagram as a *guide* for helping the search is briefly explained in the following example (see Figure 4). We want to prove two equal segments UV = XY, by congruent triangles. Suppose triangle XYZ exists, and our purpose is to find a triangle UVW on UV to compare to triangle XYZ. We need to search for existing or generated triangles on UV. The first thing is to find a convenient third point W, which must be different from U and V. The possible coordinates of the sought point W are computed from the coordinates of X, Y, Z, U and V, and a check is made in the diagram to see if a point with such coordinates exists. The diagram is used in a *positive* way for computing the possible coordinates for W.

5. Points of Friction

During the development of GEOM, several problematic points ocurred which motivate the discussion about the arquitecture of the whole program. These points were largely suggested by an analysis of the geometry problems collection. In this section we state these points and we consider the methods used to solve them.

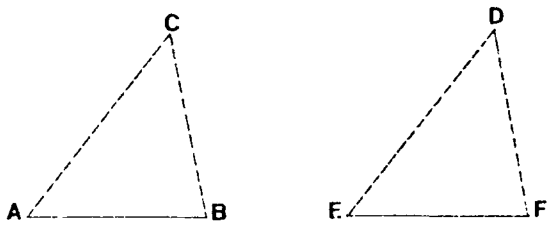


Fig. 5. Generation of congruent triangles for the proof of two equal segments (AB = EF) by congruent triangles.

5.1. THE GENERATION OF TRIANGLES

The proof of two equal angles or two equal sides can be done by congruent triangles. Before the use of the congruence procedures it is necessary to have (either by generating them or by checking for their existance) triangles containing the angles or the segments.

The equal angles and equal segments procedures of GEOM have two clauses, under the heading 'indirect strategies', which make use of the congruence procedures. These two clauses synthestize four search cases for pairs of triangles:

(1) existing-existing,
(2) existing-generated,
(3) generated-existing, and
(4) generated-generated.

The above order is motivated by the need to use first what is already known and stored in the data base – what we call existing triangles. For example, if it is required to prove AB = EF by congruent triangles, GEOM checks its data base for points C and D, and directions C : A, C : B, D : E and D : F. If these direction exist, we have triangles CAB on AB and DEF on EF. If not, it is necessary to add such directions to the data base as a means to construct segments A . C, B . C, E . D and F . D (see Figure 5).

Only the first case corresponds to the situation where the triangles already exist on the given segment or contain the given angle. The other cases refer to the generation of triangles and the possibility of making constructions as they are needed.

5.2. THE INTRODUCTION OF NEW POINTS

The introduction of new points can be envisaged as a means to make *explicit* more information in the model (diagram), which is not contradictory with the hypotheses. This introduction does not reduce the search space but for certain cases it may create short cuts or new paths, which diminish the steps of a proof. The introduction of new points was motivated by the analysis of problem 10 (see Figure 6).

F is the point to be introduced. As point F is the intersection of the diagonals of a rectangle, its existance is known for any model. So, during the input of rectangle ACDE, F is introduced and its consequences, new equal angles and sides (e.g. diagonals equality), are asserted in the data base.

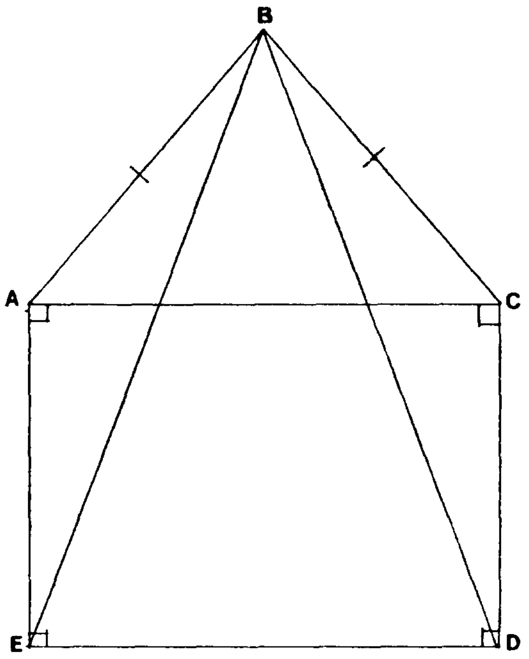


Fig. 6. The diagram of problem 10.

The existance of point F allows the construction of segment BF, and thus the congruence of triangles BAF and BCF to be proved. As a consequence, a new fact is asserted, the equality of angles BAF and BCF, and another congruence of triangles, between BCE and BAD, becomes possible.

This proof is general, as it utilizes a theorem (congruence of triangles) independent of point B's position. On the other hand, Goldstein's proof [12], without the introduction of F is too particular, because it depends on the position of B:

(1) for B out of the rectangle it uses sum of angles theorem;
(2) for B inside the rectangle it uses the difference of angles theorem.

Both theorems particularize the model, i.e. they are falsely used for some interpretation. But in first order logic theorems must be valid for all interpretations. However, a specific model can be considered as a general or categorical one, if what it particularizes is not used in the proof. *Goldstein's error* consisted in using facts not in the hypotheses (or not concluded), i.e. he used a specific diagram *unwarrentedly* as a general positive example. However, the diagram's role in this case is to act as a counter-example.

But a diagram for this problem is not unique (B can be anywhere on the perpendicular bissector of AC). For GEOM, there is no difficulty at all if the new element F is coincident with B, since the congruence of two degenerate triangles is still a congruence. Thus, no use is made of what particularizes the diagram.

For this example, *it is clear that simplification is obtained when a new point is introduced.* A combinatorial explosion would occur if F was not created, because a case analysis, involving a sum of angles clause would be required. On the other hand, for problems with a large search space, the introduction of a new point would be relatively catreastrophic: a combinatorial explosion would occur. For our problem collection a heuristic was devised in order to balance the advantages and disadvantages of doing this construction: "only for diagrams with less than 8 points is the introduction of a new point for quadrilaterals permitted".

This facility, of introducing a new point, is available for quadrilaterials only. The new point is the intersection of the two diagonals, and the midpoint of each one. The coordinates of the midpoint are calculated using the diagram, and its name is chosen from an alphabetical list, from which the characters of the existing points are taken out. The generation of midpoints is preceeded by testing for then existence, and followed by the assertion of equal segments, equal halves and directions for the new constructed segments.

5.3. BREADTH-FIRST VERSUS DEPTH-FIRST SEARCH OF THE CONGRUENCE PROCEDURE

The congruence procedure allows *two kinds of search*:

(1) a shallow breadth-first search in the data base;
(2) a general depth-first search.

The *first kind* is done beforehand for each of the five methods of triangle congruence (Side-Side-Side, Side-Angle-Side, Angle-Side-Angle, Side-Angle-Angle, Rightangle-Side), when looking for known facts.

The *second kind* is only attempted if the first fails to find a triangle congruence. This was also done in Nevins's program [13]: it tries to narrow down the selection of new subgoals on the basis of information already present in the data base.

The motivation for this sequence of attempts was suggested by the analysis of problem 1 (PR1) and by the behaviour of the PROLOG system. A quick look at PR1 revealed that the facts necessary for the proof were already available in the data base. No depth-first search (imposed by PROLOG strategy) was required. However, if a shallow breadth-first is not in force, there is a progressive search in depth, the generation of more subgoals and a combinatorial chain reaction.

5.4. DOING, NOT DOING AND UNDOING CONSTRUCTIONS

When a human being does a proof he sometimes introduces new relations, by making constructions which fill a gap in the chain of reasoning.

In automatic theorem-proving it is also advisable to explore this mechanism of doing constructions. To discuss its implementation in GEOM, let us consider three questions:

(1) what are the objectives of this mechanism? When should it be used?
(2) what are the required kinds of constructions?
(3) should constructions remain in the data base after they have been used?

The *motivation* for discussing these questions was raised by problem 8 (PR8) where two constructions, lines SU and TR, made the proof possible. Let us see the construction process for this example. In order to prove the equality of angles STU and RUT, by congruence, we need to construct the missing parts of triangles STU and RUT: lines SU and TR. These lines are also necessary for proving the equality of sides US and TR by congruence, which are in fact the missing parts of triangles SRU and RST. This last equality is motivated by the first congruence. Another motivation came from PR13, where a construction, line NC, explores a new pathway to the goal, as it is depicted in shown in Figures 19 and 20.

This *kind of construction* only requires additional line segments, constructed between points already present in the diagram. Each segment is defined by a direction, i.e. a unit clause. In the search tree, each construction is a new terminal node, and a new link is made when a unit clause corresponding to a construction is used.

A further motivation was to implement a construction facility for missing segments when proving that two segments are equal.

In particular, the *third question* concerns the data base management of the additional clause, defining a construction. In fact two additional clauses are asserted, because the opposite direction is also stored (similarly, the supplement angle is also stored with each angle).

Consider the example of part of the structured data base for a proof, as illustrated in Figure 7.

For this example two subgoals were tried without success. These failures are stored as nonprovable goals (NP).

After the failure of the two non-provable subgoals, a construction is made on demand of a goal in the congruence clauses for equal sides. A flag and the new unit clause are asserted in the data base. The flag hides the previous nonprovable goals, which may become provable now that the construction was done. In the context of this construction a third new provable goal fact is derived and asserted as a lemma.

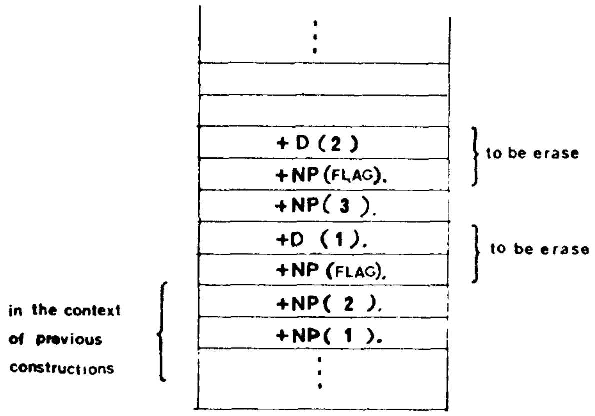


Fig 7 Data base of non-provable goals in the context of constructions made

No other lemmas are generated and still the goal which motivates the construction, is not proved.

The undoing of a construction is motivated by the need of avoiding a combinatorial explosion, more likely if the unit clause was kept forever in the data base. The nonprovable subgoals and lemmas generated after a construction process, stay available a fortiori for the continuation of the proof, even after that construction is undone. If and when a construction is undone, the flag, the direction unit clauses, but not the nonprovable subgoals are eliminated.

5.5. THE NEED FOR THE USE OF CONTEXT

Paths enabled by constructions may not enable the proof of the goal clause. In spite of this failure, some facts may be proved and asserted in the data base. However, *no context distinction* is done between these facts and the facts proved during a successful path. For example, to prove equal sides by congruence, additional line segments are required for building triangles. During the exploration of each construction some proved lemmas are asserted and may be used later on, as it is shown for BA = NC along the proof of Figure 19.

One *use of context* was implemented in GEOM, concerning the recording of failure goals (nonprovable goals). Consider Figure 7 where a construction occurs after two nonprovable goals. A flag makes invisible these nonprovable goals to the exploration subsequent to that construction. Thus, these nonprovable goals remain only in the context of previous constructions.

The objective of using context information is the recognition of a goal which failed before, but only if no new construction has been made since. The mechanism to implement this objective is composed of two clauses, the non provable filter and the record failure ones, respectively at the top and at the bottom of the equal angles and equal sides procedures. The first clause recognizes failed goals in the context of constructions made and the second one stores them. A further discussion of this point is done in Section 6.3.

5.6. TWO TYPES OF CALL OF A CLAUSE

PROLOG has only *one kind of variable* – the logical variable – which may be either an input or output variable. This distinction depends on the mode of use of the clause containing the variable. We distinguish *two modes* or types of call of a clause:

(1) all variables are instantiated – to verify;
(2) at least one variable is not instantiated – to find.

The *first type*, *to verify*, is used for example for verifying the existance in the data base of a certain fact. It corresponds to checking, the first of four tasks in automatic theorem-proving discussed in van Emden [6].

Consider a theorem to be proved of the following form: R(A, B). This form determines the task of checking, with two possible answers: yes or no. An example from geometry illustrates this task:

Question: is segment AB equal to segment CD?

The translation of this question into PROLOG is the single argument procedure call:

```
- ESI(A . B = C . D!*X1!*X2)
```

which activates the equal segment procedure of GEOM:

```
+ ESI(S1 . *S2 = *S3 . *S4!WHY!DBAS)
```

where *S1, *S2, *S3 and *S4 act as *input variables*. The atoms WHY and DBAS that are used to denote look up the reason for the truth of the goal.

The *second type*, *to find*, is adopted to search for a desired fact in the data base. It corresponds to simulation, another task of automatic theorem-proving.

Consider a theorem to be proved of the form: ∃X R(A, *X). This form determines the task of simulation, with two possible answers: yes *X = B, or no. An other example from geometry illustrates this task:

Question: is there any segment with extremity A
equal to any segment with extremity C?

The translation of this question into PROLOG is the procedure call:

```
- ESI(A . *X = C . *Y!*X1!*X2)
```

which activates the equal segments procedure (ESI) of GEOM. *S3 and *S4 act as *output variables*. The result is: *X = B and *Y = D

The first type of call of a clause, to verify, is the most common in GEOM. The second type, to find, is used for instance when the bottom-up procedures are activated.

5 7. THE DATA BASE AND ITS ACCESS

The ultimate objective of a data base is updating and retrieving facts. The addition of new facts and its retrieval depend on the structuring and the searching of the data base. Two concepts, *equivalence classes* and *canonical naming*, help to structure and access the data. The equivalence class concept is particularly important in geometry since we are dealing with equivalence (or rather congruence) relations, such as side and angle equality or parallelism.

The data base we have used stores each relation between two elements in the equivalence class of all other elements known to be in the same class. Each equivalence class is represented by an oriented tree in which the arcs stand for individual relations between elements (the nodes) and in which the root is taken as the representative element (or witness) of the class. With this representation transitivity is obtained for free since any two elements with the same witness are implicitly in

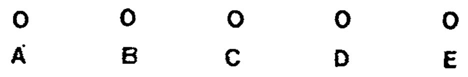


Fig. 8a. The growing of an equivalence classe tree.

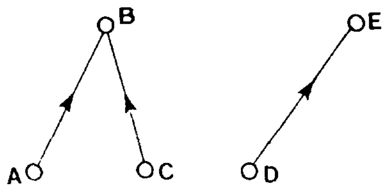


Fig. 8b The growing of an equivalence classe tree.

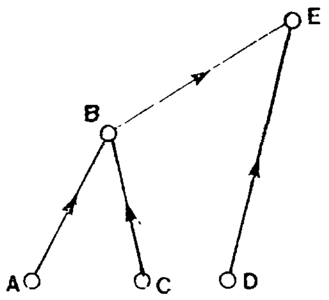


Fig. 8c. The growing of an equivalence class tree.

relation (although no explicit arc may exist between them). These trees grow by merging as explained in the example illustrated in Figures 8a, b, and c.

Five facts (A, B, C, D and E) and three (equivalence) relations (R(A, B), R(B, C) and R(D, E)) are given. When these relations are stored, the resulting trees are sketched as follows in Figure 8b.

The trees pictured above are composed of two disjoint equivalent classes. Elements B and E are chosen arbitrarily to be the witness in each class. The arrows on the arcs show the direction of growth of thre tree. In the data base three relation elements are stored: R(A, B), R(C, B) and R(D, E). One relation element, R(C, D), is added. Both classes are merged and one of the witness, E, is chosen to be the witness of the enlarged class. The tree at this stage is illustrated in Figure 8c. This kind of organization was firstly suggested and implemented for all three sets of assertions (equal angles, equal sides and same direction). An example of a tree of parallel directions is presented in Figure 9.

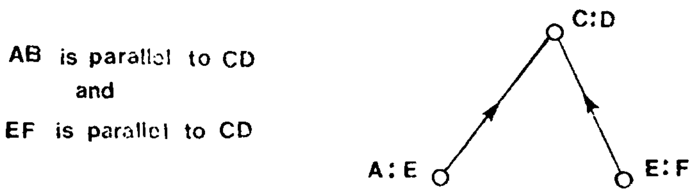


Fig. 9. A tree of parallel directions

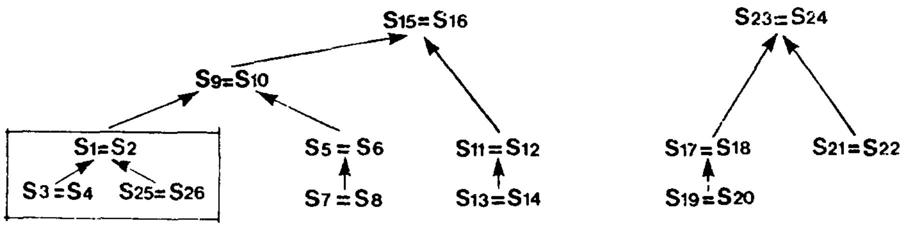


Fig. 10. The tree organization of equal sides

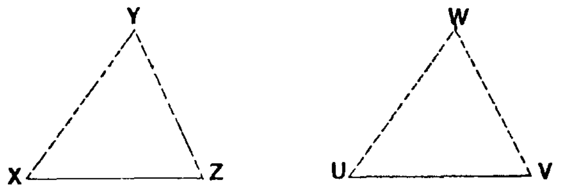


Fig 11 The construction of congruent triangles on equal sides.

The direction of CD (C : D) is the witness of these three directions. This tree represents an equivalence class, and it is easy to conclude that AB is parallel to EF, i.e. we get transitivity for free. We say (C : D) is the witness of (A : B) if there is another direction (E : F) related to (A : B), i.e. with the same direction. Another example, for equal sides, is depicted in Figure 10.

This characteristic of the data, the existence of relations with the three properties symmetry, reflexivity and transitivity, suggests an appropriate data structure, *the tree*, for the equivalence classes, which aids in deductions based on the set of properties. Transitivity implications from sets of facts are automatically available in the structure, with no additional memory, and with an appropriate access scheme. Symmetry is dealt with by canonical naming.

This data structure, the tree, copes well with one sort of query. The following question is an example: "is AB parallel to CD?" However, when a bottom-up procedure was introduced, to generate more facts from the existing and given facts, some difficulties occurred, on account of the type of questions stated. The following question is an example: "is there any segment with extremity A equal to any segment with extremity C, AX = CY, where the variables X and Y are not instantiated?"

Let us see, by means of an example, the kind of *difficulties* that occur if we use the *tree structure.*

The bottom-up procedure consists in trying to find and to prove two congruent triangles, given the facts in the data base. The new facts infered and asserted may be useful in the top-down search. In order to find two congruent triangles, two equal segments are picked up from one tree, starting at the top witness. For each segment, one point is picked up, Y and W, such that XY = UW and YZ = WV.

To retrieve such pairs of equal sides, from each tree, takes too much time, on account of the necessity of scanning the whole tree, although it is fast to find a witness.

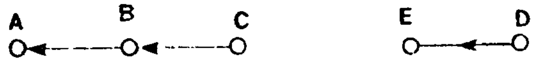


Fig. 12. List structures representing equivalence classes.

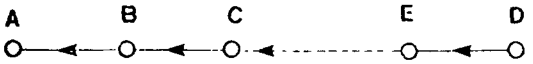


Fig. 13. The construction of a list structure.

Instead, if a *list structure* is chosen, picking up a pair is easier and faster, as we shall show next. An advantage is that one needs only one access clause for both types of access.

The list structure is obtained by a different way of linking the equivalence class witnesses: the witness of an equivalence class points to the tail of the related equivalence class. Consider the previous example, employed to explain the growing tree structure. For the list structure, Figure 12 below sketches the stage of adding three new relations to five old facts.

The lists represent two disjoint equivalent classes. Elements A and E, respectively, are chosen to be the witness of each class. The arrows on the arcs show the building direction of the lists and point towards the head or witness of the lists. If a new relation, R(C, D), is added, both lists are linked as shown in Figure 13.

The witness or hear of one equivalence class is connected to the tail of the other related class. If a question occurs about the equality of AB and GH, the answer is quickly retrieved. A practical example for equal angles is depicted in Figure 14.

The *final* adopted *solution* contains both data structures discussed, in order to balance the requirements imposed by the two sorts of questions. Both data base structures are devoted to relate the elements of each one of the sets of relations of assertions:

(1) a tree for directions, and
(2) lists for equal segments and equal angles.

There are two *data base axioms*, one for equal segments and another for equal angles. These axioms allow two sorts of retrieving, which correspond to the two questions pointed above: to verify or to find a relation in the data base. The axioms are founded on the concepts of witness and equivalence class, and use canonical naming.

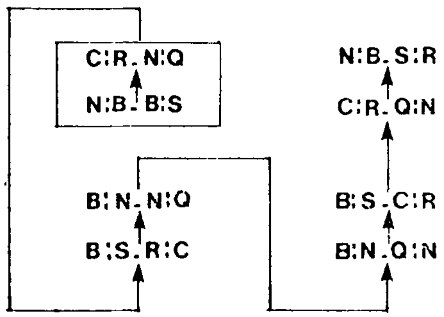


Fig 14 The list organization for equal angles.

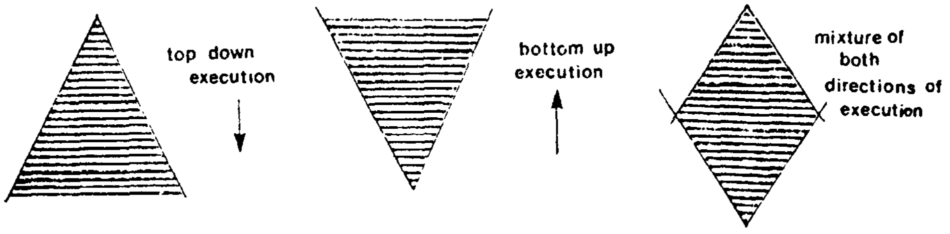


Fig. 15 Top-down versus bottom-up searches.

Let us for the *first type* – to verify – present an example:

to prove BA is equal to CD
try first to verify if the face 'BA = CD' is in the data base.

As this pair of sides is not an identity, each element is put in the canonical format (AB and CD); and if both sides have the same witness, they are found to be equal.

In the *second type*, so find, the access axioms are able to discover if the relation required is in the data base. Consider the following example to explain how this discovery is carried out:

to generate more facts based on the given fact AB = CD, try to prove triangle ABX congruent to triangle CDY; to achieve this goal; find AX is equal to CY (X and Y are variables).

The objective is to find in the data base a pair of equal sides with only two known points A and C. The equal segment procedure is activated by a procedure call with two variables not instantiated and acting as output variables. The data base access axioms are also activated with the two variables not instantiated. For this case, the identity and canonical naming clauses stay inactive and it is only found whether there is any pair of equal sides on A and on C.

5.8. TOP-DOWN VERSUS BOTTOM-UP SEARCHES

The controversy between the adoption of top-down or bottom-up directions of execution is also present in making a geometry theorem-prover. While Gelernter [10] and Goldstein [12] defended the first approach, Nevins [13] argued in favour of the second. However, it is quite clear that the set of problems chosen by each researcher was primarily linked to their point of view, and each problem was selected to adjust to it: Gelernter's and Goldstein's problems were suitable for top-down analysis, and Nevin's problems for bottom-up analysis. Indeed, the efficiency of each geometry prover was doped by the direction of analysis of the problem sample, and no one clarifies this. A general prover should be able to mix both directions of execution.

This controversy would be more relevant if we had a precise answer to the questions: "how do we define a typical bottom-up or top-down problem in geometry?". We propose the following definitions.

Typical top-down problems are those for which there is only very few consequences when the congruence relation is applied on the given facts.

Typical bottom-up problems are those for which there is a lot of consequences when the congruence relation is applied on the given facts

Nevin's problems, typically bottom-up, are in a way very connected because they hide given facts behind congruence relations. Welham [18, 19] showed that when the problem is adequate for a bottom-up analysis a typical top-down prover does a lot of search to find the halt clause. Similarly all of Nevins's examples, which do very well with this bottom-up methods, are difficult for our top-down prover. Analysing the proof tree for one of these problems (PR13), we may observed the following:

(1) the clauses' ordering for a predicate defines the search sequence. An ordering adequate for bottom-up problems is not suitable for top-down problems. The position of the transitivity clause is especially critical. In our program, during top-down, transitivity transforms one given problem into two similar problems. However, used bottom-up, it transforms two known facts into three known facts. That is why we structured out data base to get the third fact for free, by using the equivalence class plus witness structure. We are in fact doing an implicit (shallow) bottom-up when we assert facts into the data base.
(2) generally, in bottom-up problems we find a gap when we go top-down. This gap may be easier to fill if we generate some more facts at the bottom.

The bottom-up direction of execution is independent of the goal clause. Top down direction of execution is independent of the hypotheses. A mixture of both is desirable to control 'dispersion', but the two "cones" may miss each other completely.

Deciding upon one, it is advisable to compensate by doing a bit of the other. We note in passing that Nevins also does top-down search, when proving equal angles or equal segments by triangle congruence. With this view in mind, and for bottom-up problems (e.g. PR13), we introduced a shallow bottom up search right at the start. The bottom procedure generates more facts upon the given hypotheses, trying to find implicit triangle congruences in the problem data. For each pair of equal sides it looks for existing triangles, with at least three already known facts, and asserts its deductions in the data base.

5.9. THE HEURISTIC USE OF TRANSITIVITY

Transitivity relations may play an important role in proofs, because they guide the search for the proof, improving the efficiency of GEOM. This role is particularly relevant for typical bottom-up problems, on account of the top-down character of GEOM. For this kind of problems we usually find a gap, when we do top-down. One way to fill this gap is the generation of more facts at the bottom, through the bottom procedure discussed in the previous section. An alternative way is the use of transitivity at the top, to reduce one given problem to two dependent sub-problems.

As an example consider problem 14, sketched briefly in Figure 16 (see its proof in Figure 22).

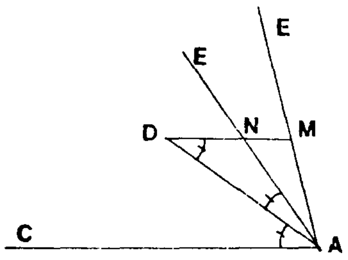


Fig 16. A part of the diagram of problem 14.

The goal is to prove two equal angles EAD and CAD. A simple use of transitivity consists in replacing the goal by two sub-goals, through the introduction of a third element. One way to find this element is to recognize a third equal angle by parallel sides. Thus, the first sub-goal shown below is proved, and a proof is tried for the second sub-goal.

GOAL: $<$EAD$=$ $<$CAD third equal angle: $<$NDA

1st. SUB-GOAL: $<$CAD$=$ $<$NDA by paralell or antiparallel sides
2st. SUB-GOAL: to prove the equality of EAD and NDA.

The motivation for the use of transitivity was spurred by the analysis of PR13 and PR14. In both cases there was an immediate need for the use of transitivity to link proof steps. This need was uncovered by the top-down character of GEOM. In Nevins's work this need was taken care of by the use of bottom-up search.

A pertinent question is the selection of the best place in the program for the transitivity clauses for equal angles, equal segments and parallel lines procedures. In GEOM the transitivity clauses were inserted at the bottom of each group of clauses, as the last thing to try. However, would this property be explored at its maximum, if it was embedded in each clause?

6. Computational Control

Efforts were made to sepaerate logic from control and to make explicit pieces of control, because it becomes easier to understand and modify a program, and it makes possible the use of the same logical clauses with different controls. In fact, we experimented and concluded that we could separate them in GEOM.

6.1. THE SEPARATION BETWEEN LOGIC AND CONTROL

A separation between logic and control is possible and desirable in PROLOG programs, as GEOM. The distinction between the two components of the specification of an algorithm A:

$$A = L + C$$

the 'logic' component L, which expresses what is to be done, and the 'control' component C, which expresses how it is to be done, makes it easier to write and to understand a program (see van Emden [6]).

Prolog, a programming language based on predicate calculus, is not a purely descriptive language as predicate logic. It requires one to express additional information on how to do it. This separation was implemented in three procedures: equal angles, equal sides and congruence routines. It was motivated by the need for more control because of the use of a bottom up procedure concurrently with the top-down ones. This implies two types of call of a clause and data base handling operations with variables not instantiated, already discussed. Moreover, the use of a shallow bottom-up requires a discussion on the modification of the congruence routines.

6.1.1. *In Equal Segments and in Equal Angles*

The *first point*, similar in both the equal segments and equal angles procedures, is related to the two types of call of the clause giving access to the data base, clause DBAS.

For the first type, to verify, all variables are instantiated (or ground) and the clause is required to be used only once. For the second type, to find, at least one variable is not ground and it is necessary to access the data base several times, in order to retrieve all possible instantiations, and get a fully exploration by the congruence procedure. This control is accomplished by the clause CHECKS (and by CHECKA for the equal angles procedure).

Let us see an example concerning the use of a congruence clause, CON3 (a congruence theorem), when it is called by BOTTOM, with two variables not instantiated.

This is the case which arises in the derivation of the possible consequences from the following fact: AB = CD. The objective is the exploration of all existing triangles on AB and on CD. This exploration is done by the procedure CON3. For the first clause, the congruence theorem regarding equality of sides (S-S-S), is required to find one or more pairs of sides, on points B and D. This is done by the first call

```
–ESI(8.*Z=D.*W!*W1!DBAS)
```

of the clause with head

```
+CON3(A.B.*Z=C.D.*W,DBAS.DBAS, GIVEN).
```

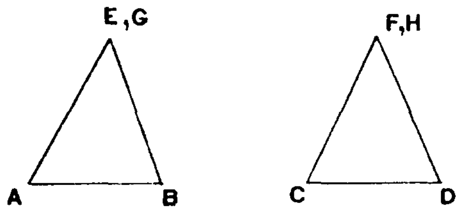


Fig. 17. The search for a congruence relation.

The variable *W1 acts as a slot for explanation information. The atom DBAS is used to force data base look up first.

A pair is picked up, for instance BE = DF, and it is checked if AE = CF (see Figure 17).

As this fact is not true, another pair is picked up, for instance BG = DH, and it is checked if AG = CH. The search goes on, controlled by claus CS till all possible pairs are searched for.

The *second point*, for equal angles, is related to the use of canonical naming routines, when at least one variable is not ground. This point is not yet solved, and a by-pass solution was adopted: the angles are generated and filtered by the diagram (EAFILTER); afterwards, they are checked either as immediate equalities or as existing already in the data base. The second point, similar to the one for equal segments, is easily solved and direct retrieval is possible either with all variables instantiated or not.

6.1.2. *In the Congruence Procedure*

The introduction of a shallow bottom-up search leads to two types of call of a clause, and to a slight modification of the control for the congruence routines.

A separation between the logic and the control was done by eliminating extra control evaluable predicates in the clauses of the congruence procedure, and constructing clauses to describe the behaviour of the inference process (how to do it).

Let us remark in passing that in general, and so for the whole of GEOM program, one can isolate the control in control clauses which access the appropriate logic clauses as long as they are identified by an extra argument giving them a name.

One of the calls of each congruence clauses (CON3) is a filter (CONFILTER) able to control the search. It is only activated for the bottom-up search, simultaneously with the clause EAFILTER, responsible for the generation of angles.

During bottom-up search, each congruence clause (congruence method) is used at least once, to retrieve all possible sides from the data base which are necessary for constructing triangles on the pair of sides chosen by the bottom procedure.

The CONFILTER, composed by a diagram filter and a switch, is able to reject pairs of triangles for the following cases: collinear points, same triangle and already proved congruent triangles. The re-use of each congruence clause for finding another pair of congruent triangles is effected by a simple switch clause.

6.2. MAIN CONTROL DEVICES

Control devices are special clauses able to guide the search and avoid unproductive search (e.g. impossible goals, loops). In GEOM, there are *two types* of control devices:

(1) model as a filter, and
(2) filters.

In the *first type*, a model (i.e. a particular interpretation of a general logical statement) is used as a filter. It prevents irrelevant goals from being pursued. As an example, we have the geometric diagram filter, discussed in Section 4.4 and proposed by Gelernter [10]. It uses analytical geometry to reject false goals through simple numerical computations.

In the *second type* the filter is not a model. As examples we have the uniqueness filter, the nonprovable filter and the DC filter.

The uniqueness filter prevents looping by allowing each subgoal to be attempted only once in a branch. It is inserted in the following procedures: equal angles, equal segments, parallel lines and congruence routines, and only in these since all other procedures necessarily use them.

The nonprovable filter, discussed in Section 5.4, recognizes nonprovable goals, previously stored by clauses which record failures. It is inserted in the equal angles and equal segments procedures.

The DC (direct congruence) filter rejects undesirable pairs of congruent triangles, not caught by the diagram filter: isoscelles triangles, already proved congruent triangles, and identities. It is therefore inserted in the direct congruence routines after the diagram filter. The DC filter is a part of a more sophisticated one, the CONFILTER, used when the bottom procedure is activated. The CONFILTER is also able to avoid the generation of collinear points for a triangle.

6.3. SUBGOAL CONTROL

GEOM has two facilities for subgoal control, which are summarized by *two situations*:

(1) remembering proved subgoals, and
(2) remembering subgoals which failed.

An example illustrates these facilities and its deficiencies, and the *need* for a *better interpreter* [14]. In this example, the subgoal is to establish two congruent triangles, and three methods are available (Side-Side-Side, Side-Angle-Side, Side-Angle-Angle) (see Figure 18).

The *first situation* arises when goal S1 = S′1 is finally proved, in the context of the SSS strategy. One would like the established fact S1 = S′1 to be stored so that it may be used in the context of the two other strategies (SAS and SAA) for proving triangle congruence. Briefly, we would like information to be passed from one branch of the search tree to another. The mechanism for doing this has to be made explicit by the user in his program, as it is done in GEOM.

The *second situation* arises when the attempt to prove a subgoal fails, as with the subgoal S3 = S′3. One would like to have this information available in the other two branches (SAS and SAA) of the search tree, so that no further attempt to prove it need be made. It is up to the Prolog user, however, to provide the mechanism for storing and retrieving such information. This mechanism is implemented by two devices: the nonprovable filter and the record failure clauses, placed at the top and at the bottom of equal angles and equal segments procedure, respectively.

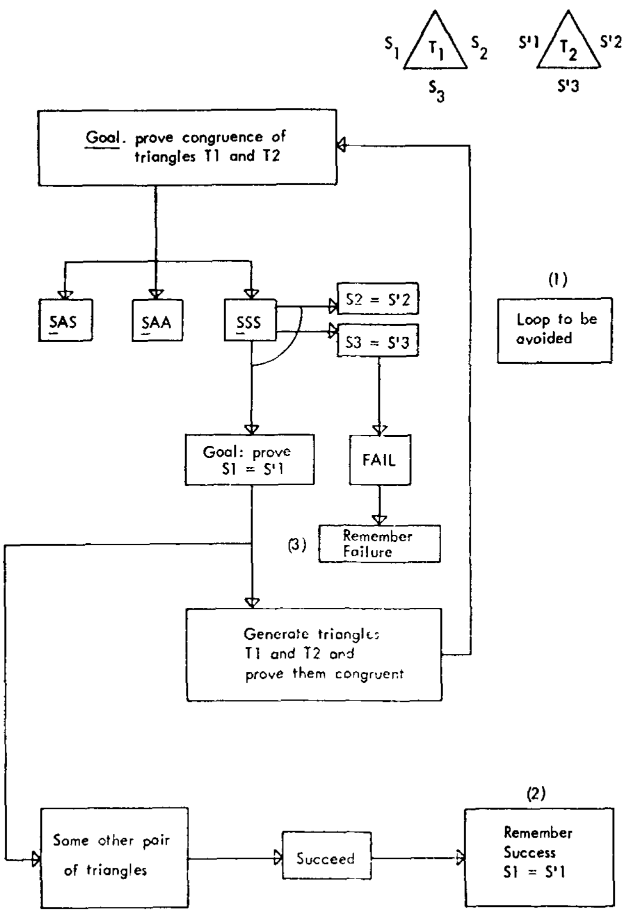

S1 T1 S2 S3
S'1 T2 S'2 S'3
Goal. prove congruence of triangles T1 and T2
SAS
SAA
SSS
S2 = S'2
S3 = S'3
(1)
Loop to be avoided
Goal: prove S1 = S'1
FAIL
(3)
Remember Failure
Generate triangles T1 and T2 and prove them congruent
(2)
Some other pair of triangles
Succeed
Remember Success S1 = S'1

Fig. 18 Cases of sub-goal control.

6 4. GOAL CONTROL

Each time a lemma is asserted in the data base, the program may ask whether it is identical to the goal it is trying to prove. If the answer is yes, the program stops with a successful proof. This goal may be the top goal or any other subgoal, and it may be possible to infer it by transitivity performed on the data base.

Two *situations*, though not considered in GEOM, regarding the control of goal statement generation are discussed:

(1) the goal statement is generated by an procedure; and,
(2) the goal can be infered by transitivity from the data base.

The discussion is motivated by two deficiencies of GEOM.

The *first situation* concerns the possibility of a program recognizing that a derivation has proved its goal statement and stoping execution. Each time a new fact is proved, by say the congruence procedure, it would only be asserted and added to the data base if it is not the goal statement (Nevins [13]; Welham [18, 19]).

The *second situation* concerns data base management and relational inference. Consider the example:

GIVEN FACT: AB = CD

GOAL: CD = EF

When the fact AB = EF is proved and asserted in the data base, the structure of related equal segments (RESO) contains the information that the goal statement is proved by transitivity:

AB = CD = EF

and no further search is necessary. However, the GEOM data base has no capability to relate the fact CD = EF, to the goal CD = EF, and the search goes on.

7. Suggested Clues for Further Work

Some aspects are discussed, suggesting further research directions for the development of a geometry theorem prover. The *main aspect* is the improvement of Prolog system itself. *Other aspects* focussed are alternative proofs of a theorem, storing and using proved theorems, the uses of symmetry and the automatic generation of diagrams. These accessory aims are intended as a sophistication of the program for geometry theorem-proving, and not as an enlargement of its geometry domain. Enlargement of this domain would be done for instance by the addition of knowledge about proportions.

7.1. ALTERNATIVE PROOFS OF A THEOREM

Two points arise when discussing the generation of alternative proofs of a theorem:

(1) the organization of geometric knowledge, and
(2) the difficulty in using diverse overall strategies.

The *first point* is concerned with the organization of the geometric knowledge (axioms and theorems) in the program: the ordering of the clauses for each procedure. The ordering for equal segments is particularly critical and determines not only different searches but combinatorial explosions, mainly because GEOM has a construction facility. Figures 19, 20, 22, and 23 picture the close dependence between the proofs of PR13 and PR14 and GEOM organization (GEOM1 is a version of GEOM with no difference of segments clause in the equal segment procedure). Figure 21 shows another proof for PR13 done by program G [18, 19].

For PR13 we only need three clauses for equal segments procedure (S4, S8, S10), but GEOM has no possibility to detect it. Instead, it uses every clause in a depth-first way. Figures 19 and 20 illustrate one disadvantage of the depth-first search, as the methodic pleasure of a burocrat proving a unnecessary fact: AP = CP.

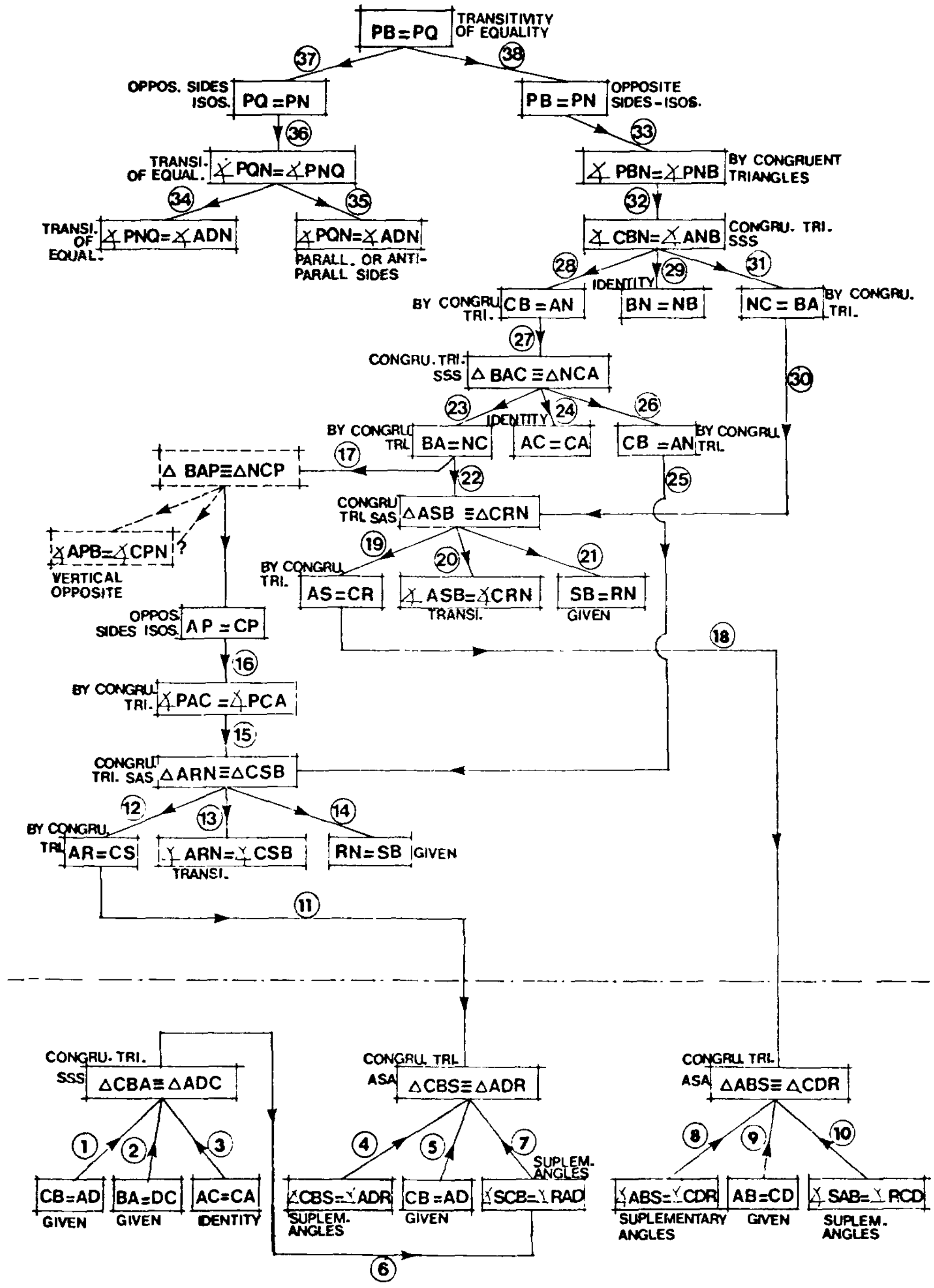


Fig 19. The proof tree of problem 13 generated by GEOM

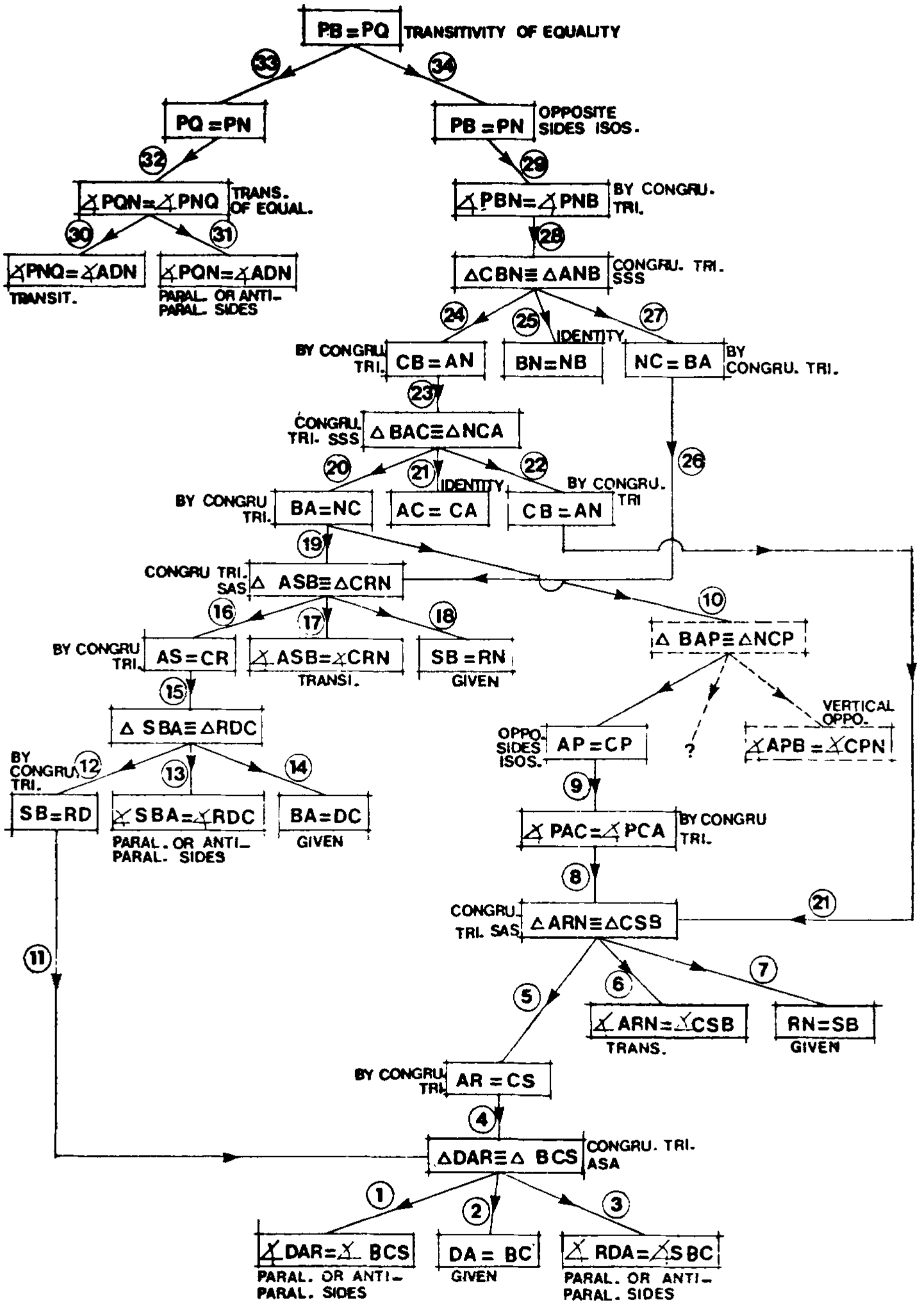


Fig. 20. The proof tree of problem 13 generated by GEOM 1.

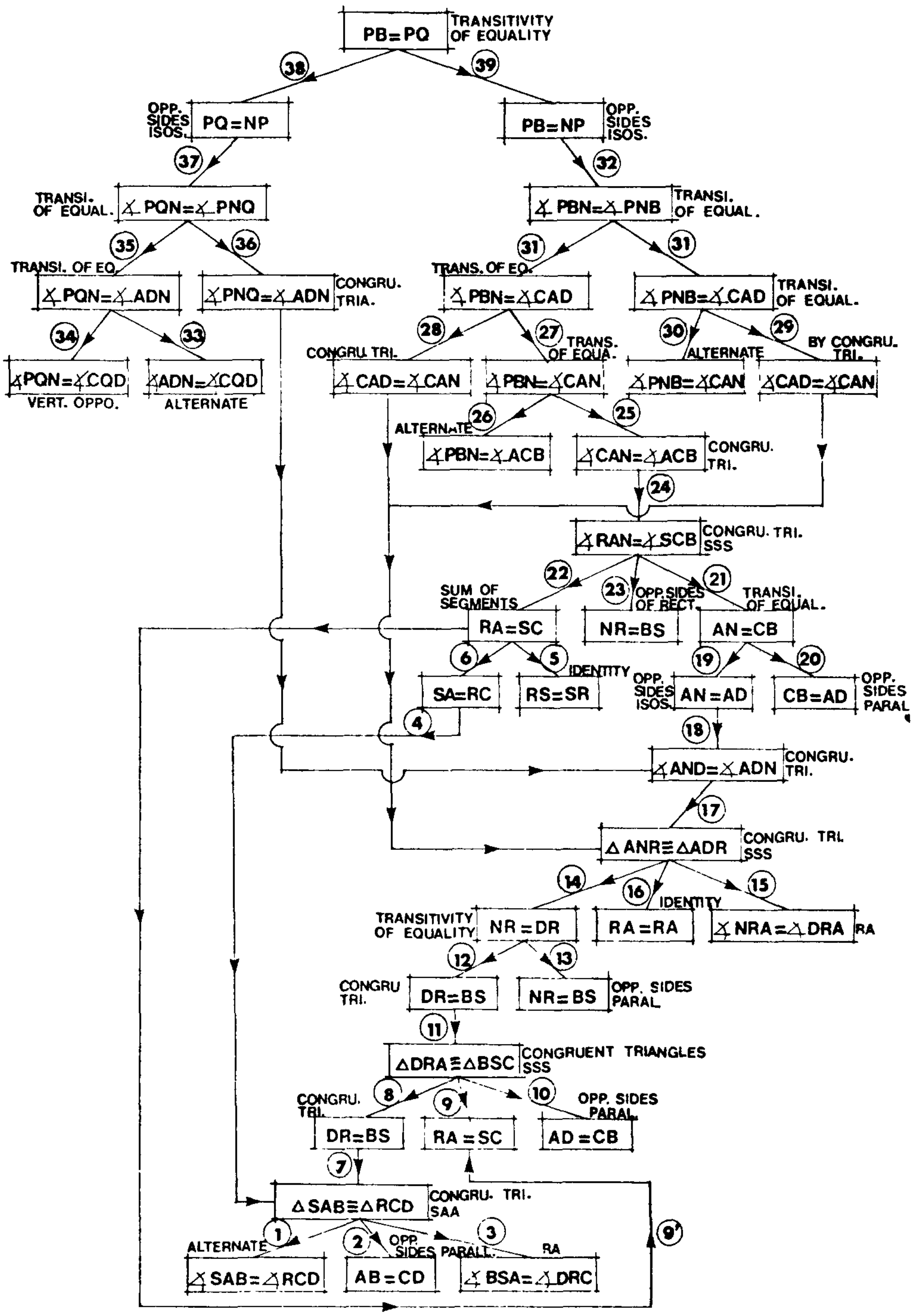


Fig. 21. The proof tree of problem 13 generated by G.

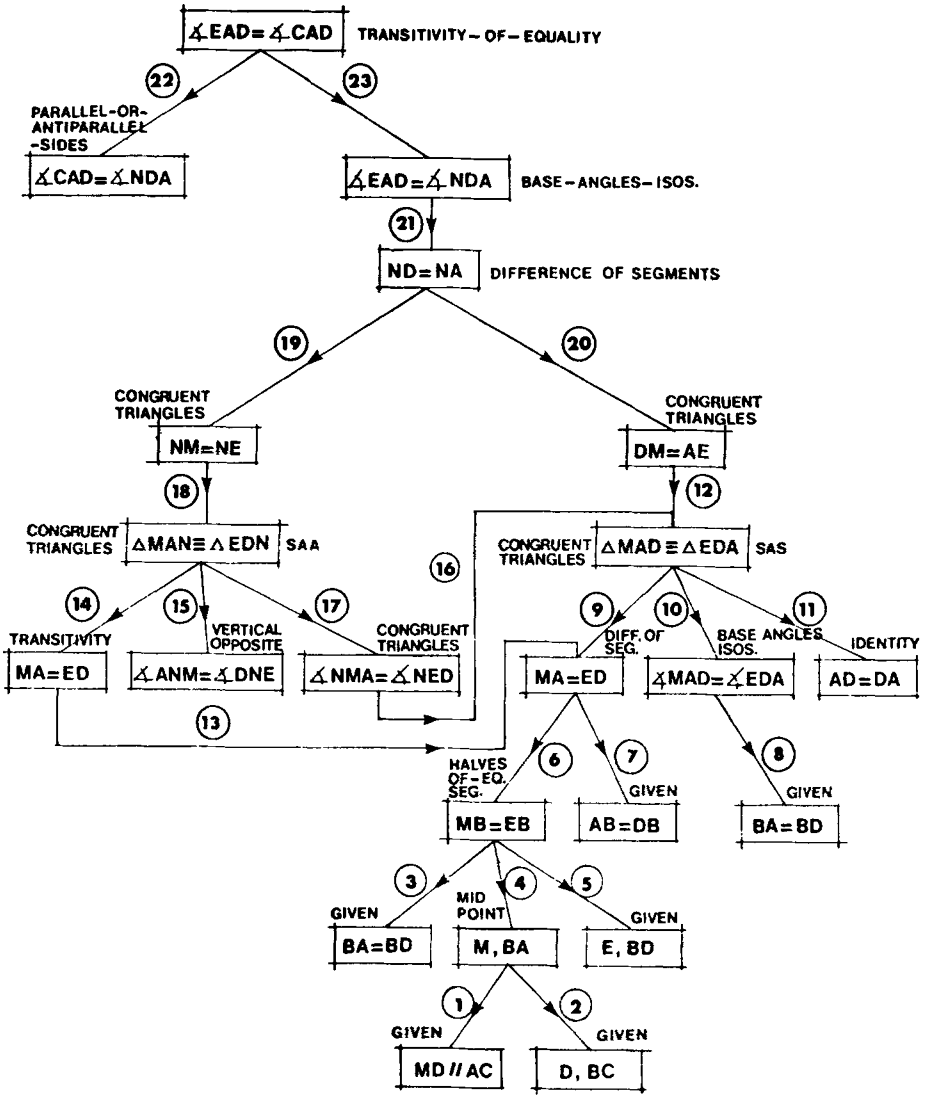


Fig. 22 The proof tree of problem 14 generated by GEOM.

The *second point* is concerned with the lack of different overall strategies, either in GEOM or in Prolog, inbuilt but selectable. This point may be solved by two approaches:

(1) the addition of more theorems to GEOM; and

(2) the implementation of a new inference system, Earley deduction, in order to explore differently the body of geometric knowledge of GEOM.

For the *first approach* it would be more interesting to have specific knowledge on how to use the existing theorems than to add new geometry theorems, i.e. more control

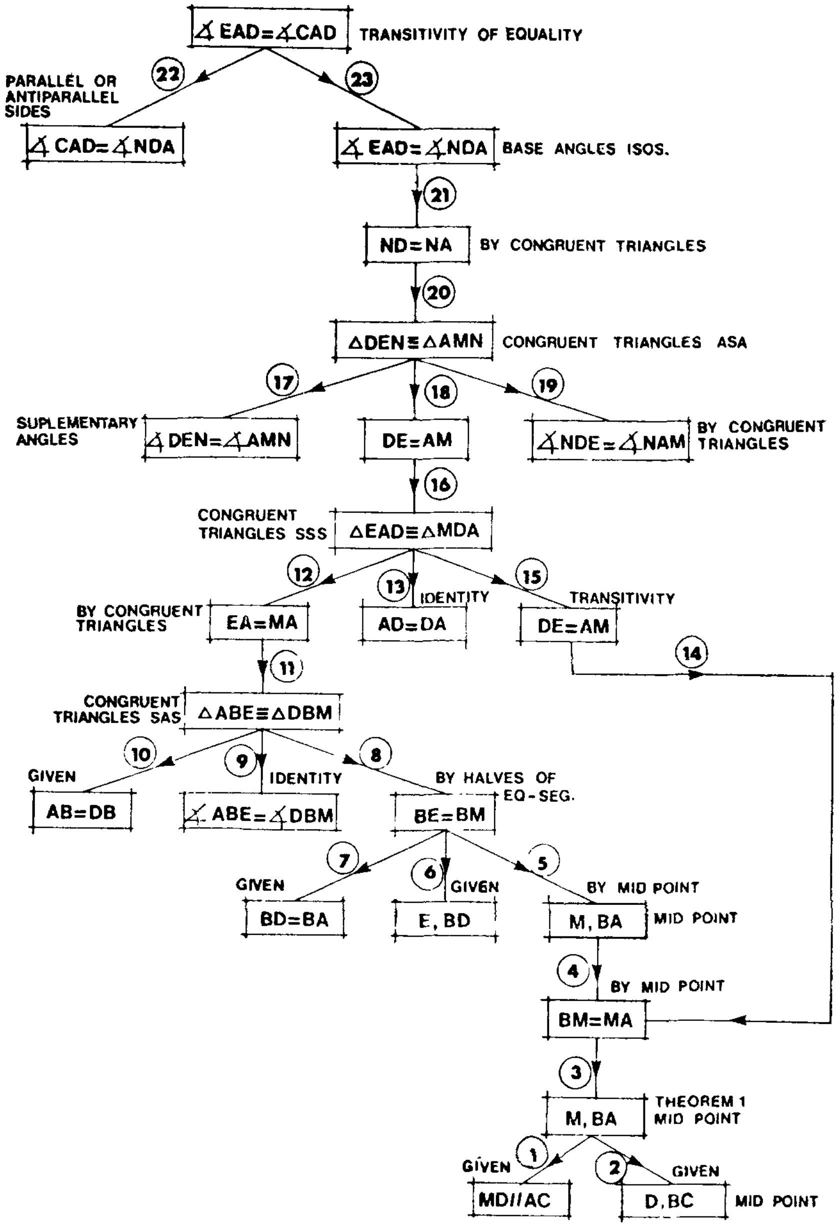


Fig 23 The proof tree of problem 14 generated GEOM 1

clauses. One approach is the identification of patterns which call for special constructions.

Constructions have been discussed as a facility for proving equal segments or equal angles by congruent triangles (see Section 5.4.). Those constructions, line segments, where done for achieving a certain subgoal, and erased afterwards.

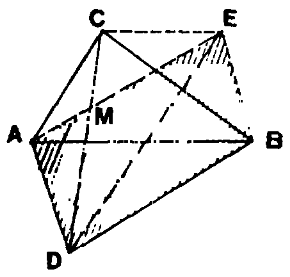


Fig. 24. A construction pattern, the parallelogram.

Let us consider an example to illustrate what we mean by a construction pattern.

GIVEN: triangles ACB and ADB are equal and on opposite sides of AB;
TO PROVE: M is the midpoint of CD.
CONSTRUCTIONS: draw AE parallel to BD;
BE parallel to AD;
join EC, ED, CD.
PATTERN: parallelogram AEBD

For this example, the pattern is a parallelogram, created in the diagram by means of constructions. The new elements make explicit more information (relations in the diagram (model), which are not contradictory with the hypotheses. The new relations induced short cuts in the search space of the problem and in some cases help to fill in gaps in the proof. The pattern may be viewed as a tactic to help a strategy.

The *second approach*, the implementation of a new inference system from scratch proposed by Warren (Pereira and Meltzer [14]) is discussed in Section 7.5, concerning the overall improvement of Prolog.

7.2. STORING AND USING PROVED THEOREMS

Storing and using proved theorems is a matter of interest for further developments of a geometry theorem-prover. This interest suggests one of the reasons for having a more complete separation between logic and control, because we would like to add the new theorem without having to specify within it any control.

We may consider this problem as more complex than the mechanism for remembering proved subgoals (lemmas), already implemented in GEOM, and discussed in Section 6.3. Now, we are interested in storing clauses instead of unit clauses. The complexity arises when clauses describing theorems contain the information on how to use them, i.e. logic and control are mixed. On the contrary, the problem becomes simpler when a complete separation is made. Thus, the logic of the theorem may be the only component to be stored. New theorems would be used by the already existing control. In Section 6.1 the separation between logic and control was discussed, and examples were given on how this problem was tackled in GEOM.

7.3 THE USES OF SYMMETRY

The uses of symmetry in geometry theorem-proving are based upon two fundamental *types of symmetry*:

(1) syntactic, and
(2) geometrical.

Syntactic symmetry is general and it may be applied to any formal system. Consider the formal system of plane geometry and the set of hypothese of problem 6:

ES(AB = AD)

ES(BC = DC)

If we exchange B with D in each unit clause, we get the same set of clauses:

ES(AD = AB)   ES(DC = BC)

and we may say that problem 6 has a syntactic symmetry, which makes the set of hypothese invariant under the syntactic transformation: B<->D.

The predicates of geometry exhibit a high degree of symmetry and by discovering syntactic transformations one can manage to reduce the computing effort of a geometry theorem-prover. Gelernter [8, 9] was the first researcher to recognize the power of syntactic symmetry and he proposes *two uses*:

(1) a *negative* one – for pruning subgoals which are syntactic variants of subgoals already tried without success; and
(2) a *positive* one – by establishing subgoals which are syntactic variants of other already proved subgoals, since their proof would simply be a syntactic variant of an already existing proof (mathematician's do it by saying 'similarly').

Gelernter's symmetry is not calculated by his program. On the contrary, it is declared by the user by observing the geometric diagram. A combinatoral problem was thus avoided for geometric problems with a large number of points (e.g. 10). A fortiori, a dynamic use of this symmetry is not explored in his program. In GEOM a single use of syntactic symmetry is implemented, through canonical naming. It handles the permutation on the names of the syntactic variables in unit clauses, as discussed in Section 3.5.

Geometrical symmetry is the arrangement of the points of a figure into pairs of points (where a point may pair with itself). It is also a syntactic symmetry.

A study of this symmetry was developed by L. M. Pereira, on making a Prolog program, called SYMM,★ which is capable of finding partial and complete line symmetries of a geometry problem.

SYMM may calculate this symmetry for problems defined either with a diagram or without it. SYMM may be inserted in GEOM and be a good guider of the search.

★ A listing of program SYMM will be sent upon request.

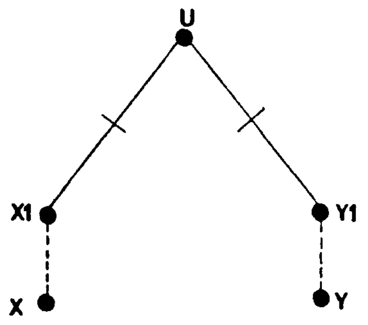


Fig. 25. Symmetry rule 1T.

SYMM has 8 rules of line symmetry, implemented in procedure LSYMM. We only consider the following ones, as examples:

RULE 1T:
point U is symmetric relatively to pair XY (X different from Y) if
there is some known a pair X1Y1 symmetric relative to pair XY, and point X1 is different relative from point Y1, pair UX1 equals pair UY1 (see Figure 25).

RULE 8T:
point U is symmetric to pair XY if
there is a pair X1Y1 symmetric to pair XY, there is a point Z symmetric to XY, point Z is different from point X1 or Y1, and angles < UZY1 and < UZY1 are equal.

RULE 3T:
pair UV is symmetric to pair XY if
point U is different from point V, there is a pair X1Y1 symmetric to XY, point X1 is different from point Y1, direction from X1 to Y1 is different from directions U to V and V to U, angles < UX1Y1 and < UY1X1 are equal, and direction X1 to Y1 is parallel to direction U to V (see figure 26).

RULE 4TC:
pair UV is symmetric to pair XY if
point U is different from point V, there are two pairs ZW and X1Y1 symmetric to pair XY, point X1 is different from point Z, directions X1 to W and W to V are the same,

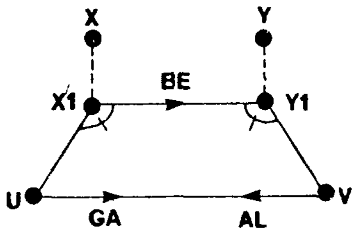


Fig. 26. Symmetry rule 3T.

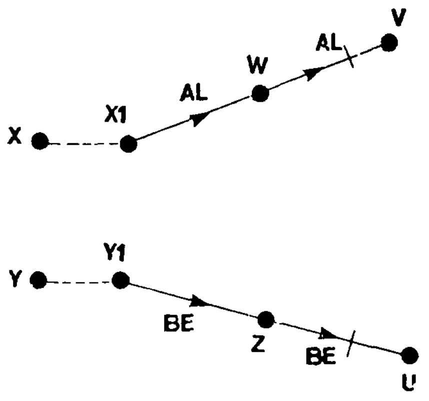


Fig 27. Symmetry rule 4TC

directions Y1 to Z and Z to U are the same, and pair UZ equals pair VW or pair UY1 equals pair VX1 (see figure 27).

Now let us reconsider problem 6 to show how SYMM works. SYMM gives a partial symmetry through the application of rule 1T, to points A and C:

AC is a line of symmetry

D is symmetric to B

Further on, it finds a complete symmetry through the application of rule 8T to point E:

ACE is a line of symmetry

This symmetry could only have been proven after the previous one. Similarly, any symmetry proof depends on the availability of needed facts. Thus, a certain symmetry may not be possible to prove at a given stage but may be possible to prove later on. This points to a dynamic use of symmetry.

Following the proof of problem 6, we envisage the use of geometrical symmetry for guiding the search by providing the appropriate congruent triangles even when there is no diagram for helping with the coordinates.

When we look at a diagram, the recognition of geometrical symmetries helps us to sketch a plan and to direct our proof of a theorem: to structure the proof and to re-arrange its pieces of reasoning. In a way, geometrical symmetry is viewed as a higher level concept giving global information, which allows the increase of directionality and the decrease of search.

But how can this be used in GEOM? For PR6 the existance of line symmetries gives us a straightforward way for concluding more facts about the equality of angles and sides: the goal ES(BE = DE) may be immediately asserted by symmetry. However, this is a unfair proof!

Of course, one could use SYMM as a device for pruning, but we would like to use it in a more positive way. We envisage SYMM as a hunch given and not as the basis for a proof. Thus, what is desired is to find its function as a control device, a kind of 'strategist'.

7.4. THE GENERATION OF A DIAGRAM

With the given hypotheses how is it possible to generate a diagram?

A diagram (model) is one particular interpretation of the given hypotheses (the hypotheses define a family of diagrams), which can be used as a counter-example during the proof.

One may consider the automatization of the task of generating a diagram, instead of having to give a set of points and its coordinates. This generation may be viewed either as static, done before the proof, or as dynamic, done during the proof as needed.

Let us consider only the static generation of a diagram, analysing, as an example, the protocol of a subject for problem 6:

(1) pick up the first element of LINES, BE;
(2) check if BE (or EB) is present in any given relation;
(3) generate a value for B;
(4) generate a value for E;
(5) pick up a second element of LINES, DE;
(6) check if DE (or ED) is present in any given relation;
(7) generate a value for D;
(8) do not generate a value for E, because it exists already;
(9) pick up the third element of LINES, ACE;
(10) check if AC (or CA) is present in any given relation;
(11) check if CE (or EC) . . . ;
(12) check if EA (or AE) . . . ;
(13) generate a value for A . . . ;
(14) generate a value for C, noting that A, C and E must be on the same straight line;
(15) pick up the fourth element of LINES, AB;
(16) check if AB (or BA) . . . ;
(17) as AB = AD, A must be located on a straight line crossing the midpoint of BD;
(18) re-check the values generated for A, B and D in order to be adjusted to 17);
(19) generate a new value for A;
(20) generate a new value for C, noting 14);
(21) pick up the fifth element of LINES, AD;
(22) check if AD (or DA) is present in any given relation, and if AD (or DA) was already used jump to the following pick up;
(23) pick up the sixth element of LINES, CB;
(24) check if CB (or BC) . . . ;
(25) as BC = DC, C must be located on a straight line crossing the midpoint of BD;
(26) re-check the values generated for B, C and D in order to adjust to 25);

(27) pick up the seventh element of LINES, CD;
(28) check if CD (or DC) . . . and if CD (or DC) was already used jump to the following pick up, if it exists;
(29) no more elements in LINES: STOP.

With this sequence of tasks one may derive an experimental algorithm, and refine it with more protocols. For the programming of this we may consider *two points*: the construction of lists of points (during the process abstract values are generated for the coordinates) and the use of symmetry to guide the generation of the diagram.

7.5. THE IMPROVEMENT OF *PROLOG*

Our experience with problematic situations, detailed in Sections 6.2 and 6.3, motivated the need for a more sophisticated predicate logic interpreter. This sophistication will be achieved mainly through the improvement of Prolog's operational semantics (proof procedure), which will provide more powerful facilities. These facilities will free Prolog users from having to provide special 'mechanisms'. Better and clearer programs will be written.

A proposal for a new inference system was done by Warren (Pereira and Meltza [14]), on the implementation of an efficient predicate logic interpreter based on Earley deduction (ED). The ED is a top-down proof procedure, analogous to Earley's algorithm for parsing context-free languages, and it uses simple instantiation as a rule of inference in addition to resolution. This improvement will provide complete satisfaction for the *three problematic situations* already discussed: automatically avoiding loops, storing proved facts, and remembering goals which failed.

Let us examine how ED deals with the three problems, discussed in sections 6.2 and 6.3. The example shown in Figure 18 is now revisited with the help of Figure 28.

In Figure 28, we have the same top goal as before (to prove T1 and T2 congruent). However, when the SSS method is tried out, subgoal T1 = T2 is rejected since it is subsumed by the top goal, subgoal S2 = S′2 is stored as a lemma after being proved, and subgoal S3 = S′3 (which could not be proved) becomes an already activated subgoal. Thus, when trying the other two methods (SAS and SAA), enough information is available for rejecting S3 = S′3 as a subgoal to be pursued, and for solving S2 = S′2 straightaway since it has been previously stored as a lemma.

Other improvements over Prolog system are envisaged, such as associative memory, space saving and compilation instead of interpretation. Earley deduction will require large data bases and further studies on their management will be carried out. This line of research will no doubt enlarge the increasing number of Prolog applications.

8. Conclusions

In this section we shall briefly review some of the *issues* that have been explored:

(1) The representation of geometric primitives.

Flexibility and generality are achieved by the concepts of equivalence class and canonical naming. Representation of angles by directions is prefered.

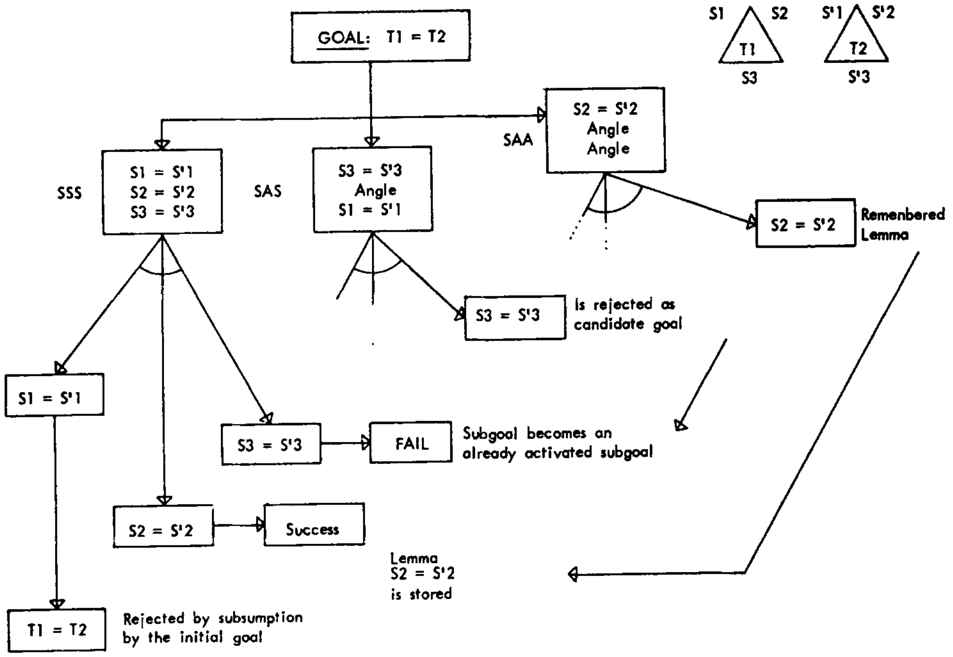


Fig. 28. The sub-goal control example revisited.

(2) The uses of a geometric diagram as a model.

Two uses of a diagram are made: as a filter (it acts as a counterexample); as a guide (it acts as an example).

(3) The introduction of new points.

The introduction of new points to make explicit more information in the diagram and to create, for certain cases, shortcuts or new paths, which diminish the steps of a proof.

(4) Shallow top-down breadth search versus depth-first search for the congruence procedures.

To suspend the selection of new subgoals until more information, perhaps already present in the data base, is searched for, by means of a shallow breadth search for each of the congruence procedures.

(5) The introduction of constructions.

A construction facility (doing, not doing and undoing) for missing segments, when proving that two segments are equal, as corresponding parts of congruent triangles.

(6) The data base and its access.

Structuring and accessing the data is achieved through two concepts: the equivalence class and canonical naming.

(7) Top-down versus bottom-up.

The mixture of top-down with a shallow bottom-up which only explores consequences of the given data, in what concerns the triangle congruence relation.

(8) The uses of transitivity.

Two uses of transitivity are implemented: the implicit one – where transitivity is obtained for free, on account of the use made of equivalence classes – and the usual explicit one.

(9) The separation between logic and control.

Separation between logic and control is possible and desirable in Prolog programs, as in GEOM. Two immediate advantages: 1) the storing of proved theorems independently of how they will be used; 2) the use of logic clauses in different ways, according to the control clause.

(10) The use of geometrical symmetry.

Geometrical symmetry can give global information about the problem or its symmetric parts. It can be used for pruning and directing the search, especially if no diagram is provided. Other uses of symmetry have yet to be explored.

(11) The need for a more sophisticated predicate logic interpreter.

The work on geometry theorem-proving points to an improvement of the used language, Prolog. The main improvement would be the implementation of an efficient predicate logic interpreter, based on Earley deduction. This improvement would provide complete satisfaction for three general problematic situations: avoiding loops, storing proved facts, and remembering goals which failed.

(12) The widening of Prolog applications.

Earley deduction and other improvements on Prolog, such as data base management, will help the use of a language based on FOL in a large and complex domain.

Acknowledgements

The authors gratefully acknowledge the helpful suggestions of Pavel Brazdil, Robert Welham, for his ideas and comments on how to implement a geometry theorem-prover in Prolog, Robert Kowalski who seduced us to predicate logic programming, David Warren who taught us the power of Prolog and devised Early deduction, and finally the referee, who made a number of useful criticisms that led the authors to revise their first draft.

References

[1] Anderson, J R., -'Turning of search of the problem space for geometry proofs' *Proc of the 7th IJCAI* (1981).

[2] Anderson, J R., Boyle, C F., Yost, G., *The Geometry Tutor*, Proc of the 9th IJCAI (1985).

[3] Coelho, H , 'An inquiry on the geometry machine and its extensions with a review of previous work,' *Working report No. 1 for INVOTAN* (1974).

[4] Coelho, H., Cotta, J C., and Pereira, L M., 'How to solve it with Prolog,' LNEC, 2nd. edition (1982)

[5] Clocksin, W and Mellish, C., *Programming in Prolog* Springer-Verlag (1981).

[6] van Emden, M , *Programming with Resolution Logic*, Research report CS-75-30, University of Waterloo (1975)

[7] Fearnley-Sander, D. 'Using and computing symmetry in geometry proofs,' in (ed. Ross, P.) *Proc. of AISB-84* (1985).

[8] Gelernter, H. and Rochester, N., 'Intelligent behaviour in problem-solving machines,' *IBM Journal*, October (1958).

[9] Gelernter, H., 'A note on syntactic symmetry and the manipulation of formal systems by machine,' *Information and Control* No. 2 (1959).

[10] Gelernter, H., 'Realization of a geometry theorem proving machine,' *Proc. Int Conf. on Information Processing* (1959).

[11] Gilmore, P. C., 'An examination of the geometry theorem machine.' *Artificial Intelligence* 1 (1970), 171-187.

[12] Goldstein, I., *Elementary Geometry Theorem Proving*, MIT, AI Lab memo no. 280 (April 1973).

[13] Nevins, A., 'Plane geometry theorem proving using forward chaining,' Vrtificial Intelligence No. 1 (Spring 1975).

[14] Pereira, L. M. and Meltzer, B., 'Implementation of an efficient predicate logic interpreter based on Earley deduction,' *Scientific proposal report for the SRC* (April 1975).

[15] Reiter, R., 'The use of models in automatic theorem-proving,' Univ. of British Columbia, Techn. report 72-04 (Sept. 1972).

[16] Roussel, P., 'Prolog – Manuel de reference et d'utilization,' G.I.A., U.E R. de Luminy, Univ. d'Aix-Marseille, 1975.

[17] Warren, D., 'Epilog: an users' guide to DEC-10 Prolog,' DAI report, Univ. of Edinburgh (August 1975)

[18] Welham, R., 'G,a geometry theorem prover,' (unpublished) (March 1975).

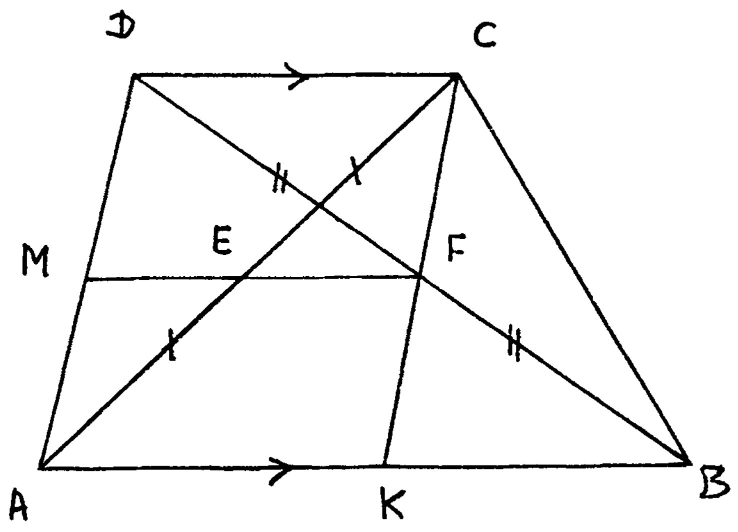
[19] Welham, R., 'Geometry problem solving,' D.A.I. Research Report No. 14 (January 1976).

[20] Wen-Tsun, W , 'Some recent advances in mechanical theorem-proving of geometries,' in *Contemporary Mathematics· Automated Theorem Proving after 25 Years*, American Mathematical Society, Volume 29, 1984.

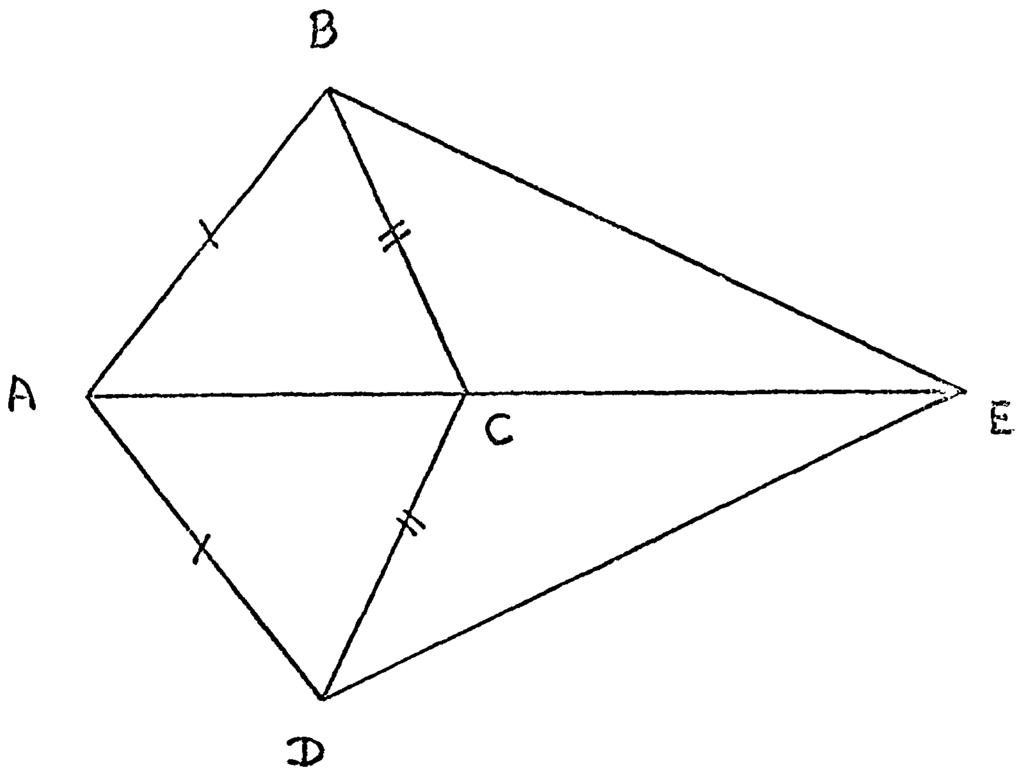
[21] Wen-Tsun, W., 'On the decision problem and the mechanization of theorem-proving in elementary geometry,' in *Contemporany Mathematics: Automated Theorem Proving After 25 Years*, American Mathematical Society, Vol. 29, 1984

Appendix 1. A Sample of the Problem Collection

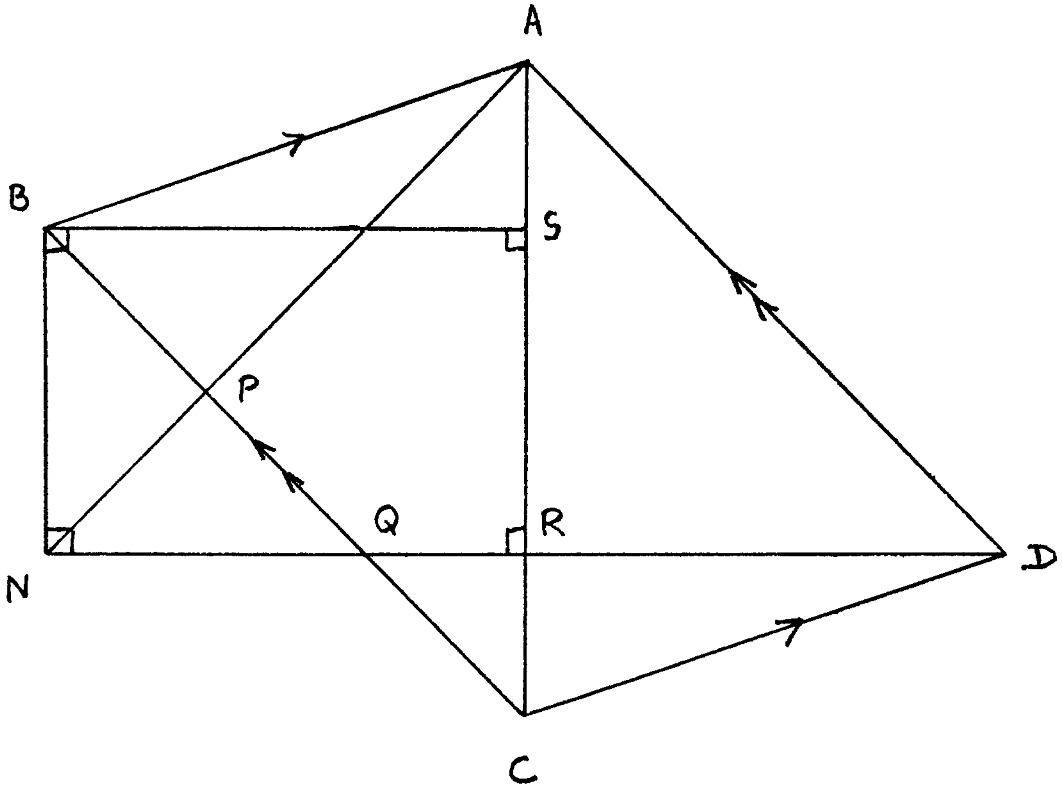
```
-SORCHA("PROBLEM 5    GELERNTER,5").
+DIAGRAM.

-A(0,0).
-B(12,0).
-C(4,2).
-D(2,2).
-E(2,1).
-F(7,1).
-K(10,0).
-M(1,1).

+FIN.

+LINES(AKB.BC.CD.DMA.AEC.DFB.MEF.CFK).

+PR(AB.DC).
+MIDPOINT(E,AC).
+MIDPOINT(F,BD).

+GOAL.
-MIDPOINT(M,AD).
```



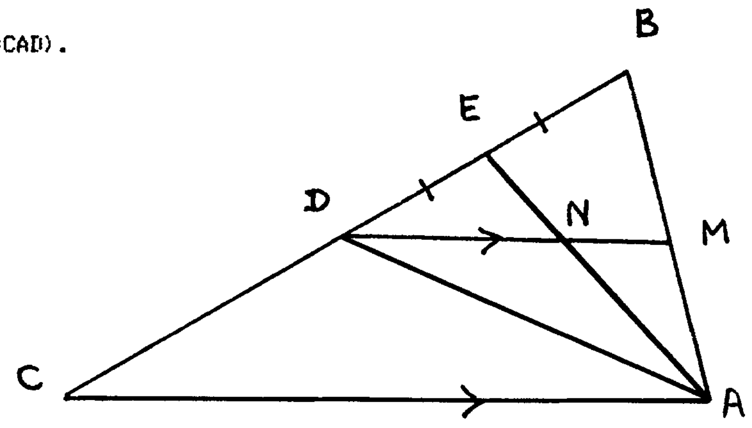
```
-SORCHA("PROBLEM 6    GOLDSTEIN,1").
+DIAGRAM.

-A(0,4).
-B(3,8).
-C(4,4).
-D(3,0).
-E(10,4).

+FIN.

+LINES(BE.DE.ACE.AB.AD.CB.CD).

+ES(AD=AB).
+ES(BC=DC).

+GOAL.
-ES(BE=DE).
```



```
-SORCHA("PROBLEM 13   NEVINS,2").

+DIAGRAM.

-N(0,2).
-Q(2,2).
-R(3,2).
-D(6,2).
-A(3,8).
-B(0,6).
-C(3,0).
-P(1,4).
-S(3,6).

+FIN.

+LINES(AB.BN.APN.NQRD.BPQC.CRSA.SB.DA.CD).

+RECTANGLE(NBSR).
+PARALLELOGRAM(ABCD).

+GOAL.
-ES(PB=PQ).
```



```
-SORCHA("PROBLEM 14   NEVINS,4").

+DIAGRAM.

-C(0,0).
-A(30,0).
-B(24,12).
-M(27,6).
-N(22,6).
-D(12,6).
-E(18,9).

+FIN.

+LINES(BEDC.CA.AMB.ANE.DNM.AD).

+MIDPOINT(D,BC).
+MIDPOINT(E,BD).
+ES(BD=BA).
+PR(DM.CA).

+GOAL.
-EA(EAD=CAD).
```



Appendix 2. Solutions Given by GEOM for the Same Problem Collection

```
PROBLEM 5     GELERNTER,5

GIVEN LINES:   AKB,BC,CD,DMA,AEC,DFB,MEF,CFK
GIVEN AB PARALLEL TO DC
GIVEN E MIDPOINT OF AC
GIVEN F MIDPOINT OF BD
READY



  M  IS MIDPOINT OF AD          TO - BE - PROVED


PROOF:

TOP-DOWN SEARCH:

  <DFC = <BFK                   IDENTITY

  DF=BF                         BY - MID - POINT

  <CDF = <KBF                   PARALLEL - OR - ANTIPARALLEL - SIDES

THEREFORE: TRIANGLE DFC CONGRUENT TO TRIANGLE BFK          ASA

THEREFORE:
  FC=FK                         BY - CONGRUENT - TRIANGLES


  F  IS MIDPOINT OF CK          EQ - SIDES
  E  IS MIDPOINT OF CA          DATABASE

THEREFORE:   AK PARALLEL TO EF  BY TWO MIDPOINTS IN TRIANGLE AKC


  AK\\DC                        DATABASE

  AK\\ME                        PROVED

THEREFORE:
  ME\\DC                        BY - TRANSITIVITY - OF - PARALLELISM



  ME\\DC                        PROVED
  E  IS MIDPOINT OF AC          DATABASE

THEREFORE:   M IS THE MIDPOINT OF AD BY BISECTION OF THIRD SIDE
          IN TRIANGLE ADC
THEREFORE:
  AM=MD                         BY - MID - POINT
                              Q.E.D.
```

```
PROBLEM 6     GOLDSTEIN,1

GIVEN LINES:   BE,DE,ACE,AB,AD,CB,CD

  AB=AD                         GIVEN


  BC=DC                         GIVEN

READY



  BC=DE                         TO - BE - PROVED


PROOF:

TOP-DOWN SEARCH:

  CA=CA                         IDENTITY

  AB=AD                         GIVEN

  BC=DC                         GIVEN

THEREFORE: TRIANGLE CAB CONGRUENT TO TRIANGLE CAD          SSS

THEREFORE:
  <EAB = <EAD                   BY - CONGRUENT - TRIANGLES

  EA=EA                         IDENTITY

  <EAB = <EAD                   PROVED

  AB=AD                         GIVEN

THEREFORE: TRIANGLE EAB CONGRUENT TO TRIANGLE EAD          SAS

THEREFORE:
  BE=DE                         BY - CONGRUENT - TRIANGLES
                              Q.E.D.
```

```
PROBLEM 13   NEVINS,2

GIVEN LINES:  AB,BN,APN,NQRD,BPQC,CRSA,SB,DA,CD
GIVEN RECTANGLE:  NBSR
GIVEN PARALLELOGRAM:  ABCD
READY



  PB=PQ                          TO - BE - PROVED


PROOF:

BOTTOM-UP SEARCH FOR BC-AD

  CB=AD                          SIDES - OF - GIVEN - PARALLELOGRAM

  BA=DC                          SIDES - OF - GIVEN - PARALLELOGRAM

  AC=CA                          IDENTITY

THEREFORE: TRIANGLE CBA CONGRUENT TO TRIANGLE ADC         SSS

  <CBS = <ADR                    SUPPLEMENTARY - ANGLES

  CB=AD                          SIDES - OF - GIVEN - PARALLELOGRAM

  <SCB = <RAD                    CONGRUENT - TRIANGLES

THEREFORE: TRIANGLE CBS CONGRUENT TO TRIANGLE ADR         ASA

BOTTOM-UP SEARCH FOR AB-CD

  <ABS = <CDR                    SUPPLEMENTARY - ANGLES

  AB=CD                          SIDES - OF - GIVEN - PARALLELOGRAM

  <SAB = <RCD                    SUPPLEMENTARY - ANGLES

THEREFORE: TRIANGLE ABS CONGRUENT TO TRIANGLE CDR         ASA

BOTTOM-UP SEARCH FOR BS-NR

BOTTOM-UP SEARCH FOR BN-RS

TOP-DOWN SEARCH:

  AR=CS                          CONGRUENT - TRIANGLES

  <ARN = <CSB                    TRANSITIVITY

  RN=SB                          SIDES - OF - GIVEN - RECTANGLE

THEREFORE: TRIANGLE ARN CONGRUENT TO TRIANGLE CSB         SAS
```

```
THEREFORE:
  <PAC = <PCA                    BY - CONGRUENT - TRIANGLES



  <PAC = <PCA                    PROVED

THEREFORE:
  AP=CP                          OPPOS - SIDES - ISOS

  AS=CR                          CONGRUENT - TRIANGLES

  <ASB = <CRN                    TRANSITIVITY

  SB=RN                          SIDES - OF - GIVEN - RECTANGLE

THEREFORE: TRIANGLE ASB CONGRUENT TO TRIANGLE CRN         SAS

THEREFORE:
  BA=NC                          BY - CONGRUENT - TRIANGLES

  BA=NC                          PROVED

  AC=CA                          IDENTITY

  CB=AN                          CONGRUENT - TRIANGLES

THEREFORE: TRIANGLE BAC CONGRUENT TO TRIANGLE NCA         SSS

THEREFORE:
  CB=AN                          BY - CONGRUENT - TRIANGLES

  CB=AN                          PROVED

  BN=NB                          IDENTITY

  NC=BA                          CONGRUENT - TRIANGLES

THEREFORE: TRIANGLE CBN CONGRUENT TO TRIANGLE ANB         SSS

THEREFORE:
  <PBN = <PNB                    BY - CONGRUENT - TRIANGLES



  <PBN = <PNB                    PROVED

THEREFORE:
  PB=PN                          OPPOS - SIDES - ISOS



  <PNQ = <ADN                    TRANSITIVITY

  <PQN = <ADN                    PARALLEL - OR - ANTIPARALLEL - SIDES
```

```
THEREFORE:
   <PQN = <PNQ                          TRANSITIVITY - OF - EQUALITY



   <PQN = <PNQ                          PROVED

THEREFORE:
   PQ=PN                                OPPOS - SIDES - ISOS



   PQ=PN                                PROVED

   PB=PN                                PROVED

THEREFORE:
   PB=PQ                                TRANSITIVITY - OF - EQUALITY
                                       Q.E.D.
```

```
PROBLEM 14   NEVINS,4

GIVEN LINES:  BEDC,CA,AMB,ANE,DNM,AD
GIVEN D MIDPOINT OF BC
GIVEN E MIDPOINT OF BD

  BD=BA                          GIVEN

GIVEN DM PARALLEL TO CA
READY



  <EAD = <CAD                    TO - BE - PROVED


PROOF:

TOP-DOWN SEARCH:



  MD\\AC                         DATABASE
  D  IS MIDPOINT OF BC           DATABASE

THEREFORE:    M IS THE MIDPOINT OF BA BY BISECTION OF THIRD SIDE
             IN TRIANGLE BAC
THEREFORE:
  BM=MA                          BY - MID - POINT



  BA=BD                          GIVEN
  M  IS MIDPOINT OF BA           PROVED
  E  IS MIDPOINT OF BD           DATABASE

THEREFORE:
  MB=EB                          HALVES - OF - EQ - SEGMENTS



  MB=EB                          PROVED

  AB=DB                          GIVEN

THEREFORE:
  MA=ED                          DIFFERENCE - OF - SEGMENTS



  BA=BD                          GIVEN
```

```
THEREFORE:
  <MAD = <EDA                     BASE - ANGLES - ISOS

  MA=ED                           PROVED

  <MAD = <EDA                     PROVED

  AD=DA                           IDENTITY

THEREFORE: TRIANGLE MAD CONGRUENT TO TRIANGLE EDA          SAS

THEREFORE:
  DM=AE                           BY - CONGRUENT - TRIANGLES

  MA=ED                           TRANSITIVITY _
                                                 _
  <ANM = <DNE                     VERTICAL - OPPOSITE - ANGLES

  <NMA = <NED                     CONGRUENT - TRIANGLES

THEREFORE: TRIANGLE MAN CONGRUENT TO TRIANGLE EDN          SAA

THEREFORE:
  NM=NE                           BY - CONGRUENT - TRIANGLES


  NM=NE                           PROVED

  DM=AE                           PROVED

THEREFORE:
  ND=NA                           DIFFERENCE - OF - SEGMENTS


  ND=NA                           PROVED

THEREFORE:
  <NDA = <EAD                     BASE - ANGLES - ISOS


  <CAD = <NDA                     PARALLEL - OR - ANTIPARALLEL - SIDES

  <EAD = <NDA                     PROVED

THEREFORE:
  <EAD = <CAD                     TRANSITIVITY - OF - EQUALITY
                                   Q.E.D.
```

Practical Guide. For an analysis of previous works in geometry theorem-proving.

	1950 Gelernter 1969	1972 Reiter	1973 Goldstein	1974 Mevins
P1 Mathematical knowledge embedded in the programs	- subset of elementary euclidean plane geometry: theorems on parallel lines, congruence and equality and inequality of segments and angles, two classes of axioms underlying the geometry machine. class 1 – universally quantified axioms class 2 · existencially quantified axioms - cartesian geometry to construct the diagram.	– subset of elementary euclidean geometry; note the finite model for full plane geometry did not exist. However if we consider a fragment of plane geometry (Gelernter's) we have an useable model.	– subset of elementary euclideen plane geometry. triangles congruent, segments equal, lines parallel, angles equal and quadrilaterals to be parallograms; note. the theorem prover was not simply a copy of the standard axiomatization for euclidean geometry and the objective as to find the most powerful representation for doing actual problem solving. – cartesian geometry used by the diagram filter	– subset of elementary euclidean plane geometry; no curved lines, nor introduction on new points.

P2 Problem solving concepts arisen	transformation from suggestion space to problem space;
	· heuristic devices· diagrams, models
		– the knowledge about the problem domain must be used in logic; two approaches semantics as domain dependent heuristics and semantics as the representation of knowledge/the last one is defeneded by the author/.	– domain;
			– procedural representations· consequent knowledge;
			antecedent knowledge: elimination of synonyms, derivation of corollaries, experts, naming: canonical names, cycling; controling computation: diagram and uniqueness filters; failure; planning depth-first search plausible move generator (PMG), incomplete bindings and constructions
				– the ability to structure a problem space as a consequence of a *meaningful* use of forward chaining;
				– the representation of the problem domain: *an ordered sequence* of points. recognition of six predicates· LN, PR, PRP, RT. ES, EA;
				– the addition of new facts to the data base, as provided by forward chaining, can be a source of great strength if these factors are relevant to the goals of the problem solver.

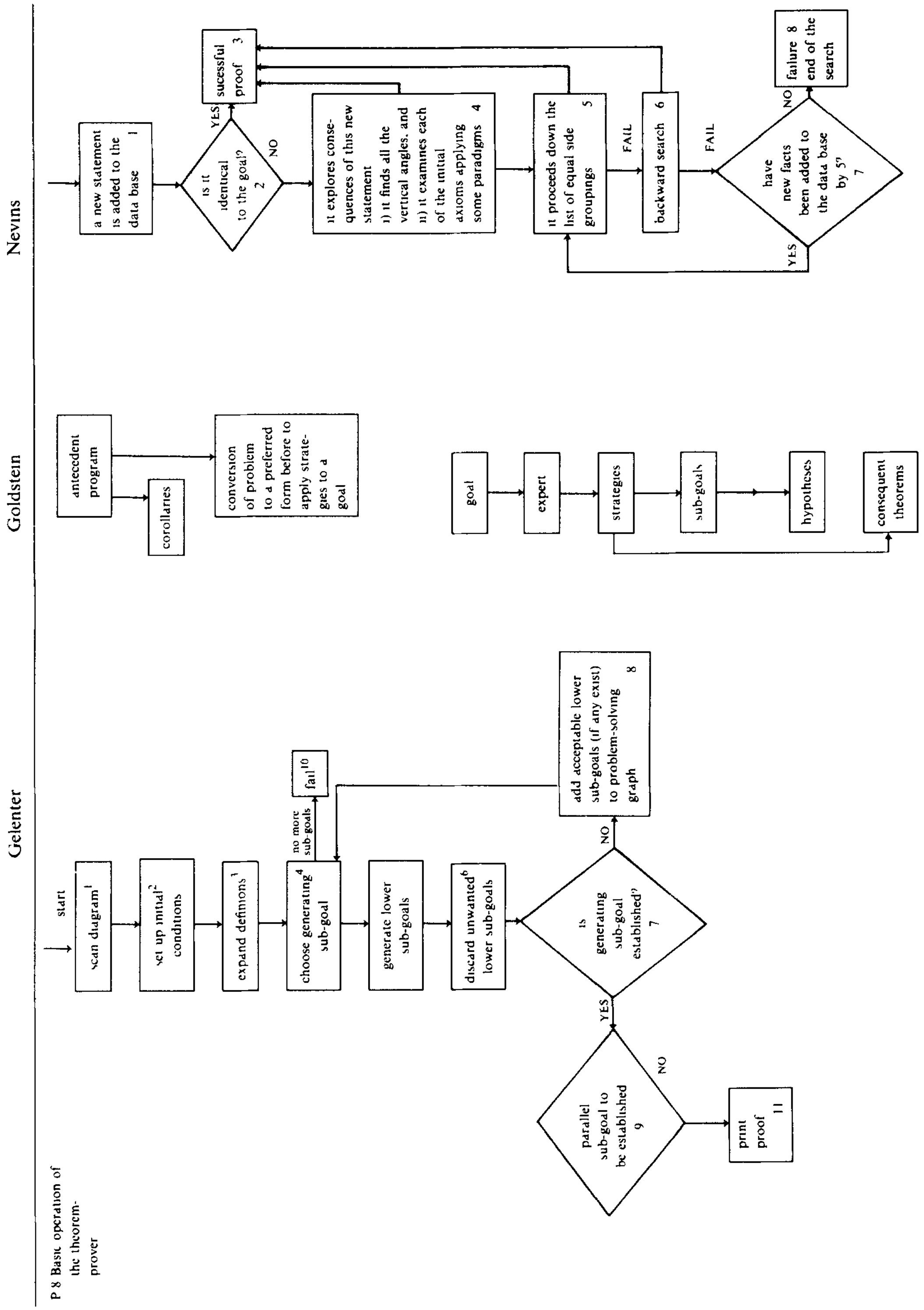
P3 Languages used	special purpose list processing language compiled by Fortran		– microplanner [BTP]; conniver [PTP] B – basic – P – plausible TP – theorem prover	– Lisp (PDP-10)

Practical Guide For an analysis of previous works in geometry theorem-proving

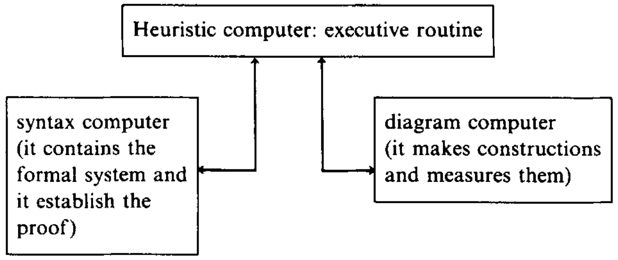
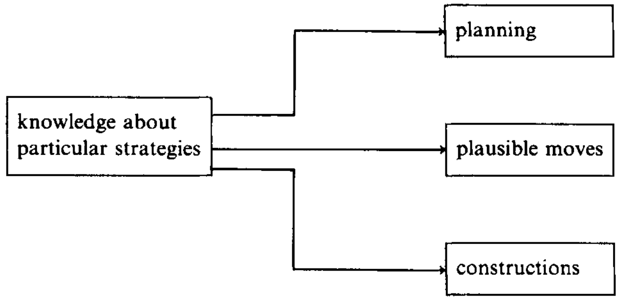
	1950 Gelernter	1969 1972 Reiter	1973 Goldstein	1974 Mevins
P4 Motivation	- the fact Euclid's elements are common knowledge of our society, - the illustration of very general theorem proving ideas, top-analysis and syntactic symmetry. - the application of computers to problems that require more than minimum of ingenuity or resourcefulness, - the suggestion space offered by the diagram on plane geometry and the ease of transforming proof indications into problem space	- geometry seems to be the best example of a mathematical theory for which we have a vast store of "real world" knowledge about its principal models As human provers we have very good visual and analytic procedures for testing the satisfiability of effs in these models	- geometry reveals the nature of axiomatic systems and logic rules of inference, - the understanding of the reasoning and the knowledge of mathematics, - euclidean geometry is an excellent subject for introducing rigorous thinking, - the existence of new goal oriented languages such as planner and corniver	- the test of the sucess of an automatic theorem prover programmed in Lisp, which takes advantage of the procedural power that Lisp provides
P5 Start question	- how to get the machine to check and to discover a proof. - how to use symmetry in the search for a proof, - the problem of strategy and tactics in choosing methods. - how can we understand the precise meaning and use of heuristic methods by machine, - how can the diagram model be served as a heuristic device, - which are the heuristic that can be obtained by observing the diagram formation of the best set of heuristic rules by experiment		- how can geometry be used to solve problems?	which of the uses of a diagram, as a semantic model, are applicable to a moder computer? - is it possible that the filter mechanics may not be essential even for humans?

Practical Guide. For an analysis of previous works in geometry theorem-proving

	1950 Gelernter 1969	1972 Reiter	1973 Goldstein	1974 Mevins
P6 Goals	– the design of a machine whose behavior exhibits more of the characteristics of human intelligence, – to have the machine relyed upon heuristic methods; – an efficient theorem prover in some undecidable system; – the proof of theorems in the manner of a high school sophomore	– the abstraction and the generalization of the underlying semantic and syntactic formal system of Gelernter's machine; – the presentation of a semantic and deductive formal system for automatic theorem-proving	– to building a procedural model for geometry as a first step in understanding the reasoning and knowledge of mathematics; – to represent geometric knowledge in procedural terms, – to illustrate important problem-solving concepts that arise *in building procedural* models for mathematics, – to become *theorem not simples* declative statements but strategies for solving problems, – to cope the theorem prover with analysis of signficant and breadth in the mathematical domain; – to bring into focus planning methods for finding a proof amidst a subset of geometric knowledge; – simulation of PTP in microplanner.	– to obtain a program which compares favorably with the best human theorem-provers, – to prove the use of the diagram is not essential to mechanize plane geometry theorem proving; – to emphasize the importance of forward chaining as opposed to backward chaining, closed connected with the diagram filter (the size of the data base probably is far less important than the manner by which the data base is organized and the heuristics employed in its management).
P7 Type of method and of exploration	– analytic method. working backwards; top-down derivation		– analytic method working backwards top-down derivation, – switch between depth-first and breadth first	– combination of forward and backward chaining, with the forward component playing the more important role (dispensation of the diagram filter)


Gelenter
start
scan diagram[1]
set up initial[2] conditions
expand definitions[3]
choose generating[4] sub-goal
no more sub-goals
fail[10]
generate lower sub-goals
discard unwanted[6] lower sub-goals
is generating sub-goal established? 7
NO
add acceptable lower sub-goals (if any exist) to problem-solving graph 8
YES
parallel sub-goal to be established 9
NO
print proof 11
Goldstein
antecedent program
corollaries
conversion of problem to a preferred form before to apply strategies to a goal
goal
expert
strategies
sub-goals
hypotheses
consequent theorems
Nevins
a new statement is added to the data base 1
is it identical to the goal? 2
YES
sucessful proof 3
NO
it explores consequences of this new statement
i) it finds all the vertical angles, and
ii) it examines each of the initial axioms applying some paradigms 4
it proceeds down the list of equal side groupings 5
FAIL
backward search 6
FAIL
have new facts been added to the data base by 5? 7
YES
NO
failure 8 end of the search

P 8 Basic operation of the theorem-prover

	Gelernter	Goldstein
P9. Elementary model: internal structure of the theorem-prover	 Heuristic computer: executive routine syntax computer (it contains the formal system and it establish the proof) diagram computer (it makes constructions and measures them) note: heuristic computer: contains the heuristic rules it decides what to do next and organizes the proof search process and the recognition of syntactic symmetric	 knowledge about particular strategies planning plausible moves constructions
P10. Computational control		– discoveries in one strategy allows erasing of another strategy. i.e., to prune subsequent computation (this is important when the model is capable of supporting real problem solving) – strategies with a common goal are organized into experts – two heuristics: diagram filter (to reject false goals); uniqueness filter; do not try an unsucessful goal a second time (the ability to back-up and try alternative strategies when a failure occurs).

P11 Own contributions	– major ones along the different configurations. i) the system was equipped with a single major semantic heuristic (based on an interpretation of the formal system rather than on the structure of the strings within that system). ii) the introduction of selection heuristics for both sub-goals and sub-goals generating 4theorems, iii) the addition of a simple construction routine to expand the theorems proving power of the machine to include an entirely new class of problems, hitherto logical unattainable, iv) the use of the diagram to solve the angle problem; the machine is able to find proofs for theorems of greater order of difficulty. – the theorem proving algorithm is also a decision algorithm, – the procedures are clearly independent of geometry and are applicable to any formal system with its corresponding interpretation; – heuristic: diagram (filter interposed between the solution generator and the solution evaluator) to reject false goas, the use of diagram suggested some modifications of the algorithms in order to affect the process of constructing proofs and provided a partical ordering of the possible substitution instances of some axioms, the ordering in which new line segments should be considered.	– the generalization of the Gelernter's formal system will provide i) dynamic modifications of models as the proof unfolds, ii) arbitrary effs as axioms and theorems, iii) negation and disjunction, iv) infinite models, v) the special treatment of the equality predicate; vi) the use of the model to make inferences; and has also the following features i) the purely logical component resembles natural deduction rather than resolution; ii) a very smooth and natural interface between the semantics and the deductive syntax (a model should be used to guide the proof and conversely the current state of the incomplet proof should suggest a model), – the deductive system L (generalization of LG) has twelve rules of inference, which are applied in order to any current goal (L is incomplet); – the system L provides for the use of counter examples to prune the search tree. the model is used as a "semantic seive" for trapping out incorrect substitutions	– a high level, natural formalism for representing semantical knowledge as programs (possible by new languages), – a canonical name scheme to allow the program to identify the many synonyms for a given gometry entity, - a plausible move generator (PMG) for guiding the search for a proof, it is coupled with knowledge for adding constructions, each expert can have a PMG to choose which strategies to apply first; – constructions are generated for the specific purpose of making a desired strategy applicable to the current diagram (and not randomly made between unconnected points when all else fails, - strategies with a common goal are organized into experts (conceptual clarity and computational efficiency), - each time an expert suceeds the result is remembered as a lemma (set of assertions; data base), - orderings on points and set of points plus some additional cartesian knowledge is used to define a canonical format for geometric statements, – combination of euclidean and cartesian geometries with planning metaknowledge.	– the efficient representation of data base, – the confination of the normal use of forward chaining to those points and line segments that were implicit in the statement of the problem: - the limited use of backward chaining does not generate an and/or goal trees and so no need for a diagram filter, – techniques which make important and extensive use of paradigms (based upon a mental picture of some possible situation from the plane geometry world), provide the motivation for different routines used in program; – the way the theorem prover handles the equality relation, – the use of forward chaining is not coupled with a representation that generated a host of redundant statements for each new statement added to the data base.

	– theory machine. to bring theory (the syntactic computer) and experiment (the diagram computer) into harmony and thereby discover what kind of a world it lives in, it is a device that conjectures about its environment and tests its conjectures, – the use of a special purpose list processing language; – a finite model (and usable) for the fragment of plane geometry			
P12 Resistante centres: limitations and difficulties	– no interplay between the syntax and the semantics; – constructions are randomly made between unconnected points when all else fails; – the machine exhibit intelligent behavior but does not improve its technique, except for the annexation of previously proved theorems to its axiom list, its structure is static, – the machine in incapable of developing its own structure and is limited in the class of problems it can solve by the initial intent of its designer; – the semantic heuristics used by the heuristic computer are dependent of geometry; – semantics is used only to prune the search tree for the syntactic proof; – the problem of speed and memory of the computer used; no decision algorithm;	L is incomplete.	– no complete solution was given to the construction problem; – the construction does not involve any global planning; – no connection to include symmetry in construction process.	– the introduction of new points into the diagram: most of the described paradigms find points, although these points already were present in the diagram.

	– with a linear increase in the number of individual points mentioned in the premises, the rate of growth of the problem solving graph increases exponentially and the time required to explore the graph increases correspondingly, – well formed formulas are defined by scheme rather than recursively, – the formal system defines a negationless logic since the negation sign does not appear explicitly in any of its rules of inference; similarly there is no rule for handling disjunction			
P13 Clues suggested for further work	– to become the machine more selective in its choice of theorem for permanent storage, rejecting those which do not seem to be sufficiently interesting or general to be useful later on; – the machine might select the especially lemmas for its list of established theorems (instead of forgetting all ones it might have established as intermediate steps in the proof of the theorems offered to it, to be rederived when needed), – the next level of learning (involved when the machine uses results on one problem to improve its guesses about similar problems) 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 with a given character,	– a strong semantic theory for quantifier-free effs would be welcome, – a theory of counter examples (semantic notions) it is that mathematics combines the search for counter examples, to current sub-goals, with the search for a proof/in geometry counter examples are analytic techniques/, – a priori concern to recognize and to describe the relationships among the various components of the diagrams	– to check if the translation of figures is a technique that the constructor should consider; – to consider geometry as a game, – a more intelligent planning, when the problems become more complex; – more knowledge for the PMG evidence for a strategy and for a particular choice of points, handling constructive strategies; – the ability to move between contexts, – the implementation of the PTP (BTP + PMG) in connіver, – the use of geometric symmetries (reflections across lines and points) to help plan the proof.	– to introduce new predicates so as to provide the program with a richer representation of knowledge, – to construct similar paradigms, as those described, which deduce points that had not been present in the initial diagram

	– the use of the geometry machine to do research, – a program of the same theorem-proving power as the euclidean theorem prover should be sufficient to prove a large class of non-obvious theorems in non euclidean geometry; a machine furnished with a non euclidean diagram encounters none of the assault on rationality experienced by a human mathematician searching from some heuristic insight into a theorem by considering a non euclidean diagram, – the search for algorithms (instead of methods) using first order predicate calculus (for more complex formal systems, heuristics sufficiently selective to produce, within acceptable bounds on space and time, proofs of any great interest).		
P14. General directions for further research	– one could give the machine a formal system and ask it to see what it could find, instead of providing a machine with a formal system and a sequence of propositions to prove; – another approach for theorem-proving by machine semantic tableaux (Beth, 1957)/enumeration procedure/, method for systematically constructing a counter example for a proposed theorem if there is one, or else establishing the fact that none exists; – the requirement of a digital computer system designed for optimum execution of list processes	– the development of specialized dossia dependent techniques for finding counter examples; – the representation of new knowledge; – the problem of generalizing observations in a model, – the descriptive problem, – the automatic generation of models, appropriate to the special hypothese of the theorem to be proved; – the extension of the system L (generalization of LG)	– global planning line of attack from global considerations; adding constructions on the basis of symmetry considerations, considering different proofs skeletons, proof by contradiction, hypotheticals, symmetry or inequalities; using new sophisticated models for relevance, cost and provability, which might include using the diagram to help plan the proof· a model based on the diagram, as a physical structure, might be useful, – diagram generation. in connection with the basic theorem prover (BTP),

			- learning· to consider the process by which a model for geometry is built; analyze the type of learning when the theorem-prover finds a proof, - to extend Winograd's language system to process english statements of geometry problem, - an alternative approach to mathematical reasoning, resolution, - to attempt a Newell-Simon style of psychological research into mathematical creativity, used the model presented as opposed to a production system; - education. approach to learning mathematics
P15 New starting questions	- how can a theorem machine be sole to observed that a certain sequence of methods in effective in certain circumstances and stream line the sequence into a single method and in this way devise a new method?		about quantitative increase of mathematical knowledge Q1 – would the fact that the hypotheses imply a mechanical linkage between parts of the diagram be evidence for a geometric relationship being provable of those parts? Q2 – how can the basic techniques discussed so far be extended? Q3 – can the same basic structure be used to represent euclidean knowledge about circles? Q4 – what new issues arise when the amount of knowledge in the theorem-prove grows large?

Q5 – must it resort to a big switch structure or can all of its consequent theorems be active at the same time by virtue of the difference in their calling patterns?

Q6 – is the knowledge grouped into a tree of miniworlds, with well defined entry conditions?

about qualitative increase in mathematical knowledge:

Q7 – can the basic structure, used so far, be extended to other areas of mathematics, such as three dimensional geometry, topology, non-euclidean geometry, and compass and straight-edge constructions?

about the program and related with learning:

Q8 – should a new theorem be added as an antecedent or a consequent theorem?