MEMÓRIAS DO L. N. E. C.

ÚLTIMOS NÚMEROS PUBLICADOS

499 — RODRIGUES, J. Delgado — *About the quantitative determination of rock weatherability. A case history.* Lisboa, 1978. 18 p. 175 × 250.
500 — MATEUS, Tomás J. E. — *O emprego da madeira de pinho bravo em estruturas.* Lisboa, 1978. 30 p. 210 × 297.
501 — GOMES, Ruy José — *Necessidades humanas e exigências da habitação.* Lisboa, 1978. 32 p. 210 × 297.
502 — LEMOS, Fernando Oliveira; FERREIRA, João P. C. Lobo — *Estruturas compactas para dissipação de energia por ressalto.* Lisboa, 1978, 16 p. 175 × 250.
503 — MORAIS, C. Campos; ABECASIS, Fernando — *Storm surge effects at Leixões.* Lisboa, 1978. 24 p. 175 × 250.
504 — SEABRA, Antera V. de — *O controle de qualidade nos tratamentos térmicos.* Lisboa, 1978. 16 p. 210 × 297.
505 — SANTOS, Pompeu — *As ligações em estruturas prefabricadas de betão.* Lisboa, 1978. 18 p. 210 × 297.
506 — TRIGO, J. Teixeira — *Industrialização e qualidade da construção.* Lisboa, 1978. 12 p. 210 × 297.
507 — TRIGO, J. Teixeira — *Tecnologias da construção de habitação.* Lisboa, 1978. 24 p. 210 × 297.
508 — SILVA, P. Martins — *Ruído urbano — Modelos de previsão.* Lisboa, 1978. 320 p. 175 × 250.
509 — LEMOS, Fernando de Oliveira — *Critérios para o dimensionamento hidráulico de barragens descarregadoras com paramento de montante inclinado a 1:3.* Lisboa, 1978. 16 p. 175 × 250.
510 — CUNHA, João Duarte; CUNHA, Luís Arriaga da — *Implementation of reliable software in assembly language using SIMULA-67.* Lisboa, 1978. 24 p. 175 × 250.
511 — OLIVEIRA, Ricardo — *Teaching and training of engineering geology in Portugal,* Lisboa, 1978. 8 p. 210 × 297.
512 — OLIVEIRA, Ricardo; RODRIGUES, J. Delgado; COELHO, A. Gomes — *Engineering geological studies for the Sines harbour (Portugal).* Lisboa, 1978. 16p. 175 × 250.
513 — AZEVEDO, M. Cruz; FERREIRA, M. J. Esteves — *Two-dimensional coordimeter with electrical strain ganges. Lisboa,* 1979. 24 p. 175 × 250.
514 — GONÇALVES, Fernando — *Plano Director do Município; seu lugar entre os planos de urbanização e os planos de ordenamento do território.* Lisboa, 1979. 26 p. 175 × 250.
515 — MARECOS, José — *The measurement of vertical displacements through water levelling method.* Lisboa, 1979. 26 p. 175 × 250.
516 — CASTRO, Elda de — *Les methodes de succion dans l'étude de l'altération des pierres.* Lisboa, 1979. 30 p. 175 × 250.
517 — RODRIGUES, J. Delgado — *Some problems raised by the study of the whathering of igneous rocks.* Lisboa, 1979. 16 p. 175 × 250°
518 — LEMOS, Fernando Oliveira — *Criteria for the hydraulic design of overflow dams with 2:3 upstream face slope.* Lisboa, 1979. 16 p. 175 × 250.
519 — RODRIGUES, J. Delgado — *L'echantillonage en monuments.* Lisboa, 1979. 10 p. 175 × 250.
520 — MAGALHÃES, A. Pinto de — *Bacias de dissipação de energia divergentes em planta de secção rectangular e com fundo horizontal.* Lisboa, 1979, 28 p. 175 × 250.
521 — MARECOS, José; FERNANDES, João Almeida — *Medium term observation of shiprepair and shipbuilding docks.* Lisboa, 1979. 36 p. 175 × 250.
522 — SEABRA, Antera Valeriana — *Correlação das propriedades mecânicas dos aços com a microestrutura.* Lisboa, 1979. 74 p. 175 × 250.
523 — QUINTELA, António C.; ABECASIS, Fernando M. — *Hysteresis in the transition from supercritical to subcritical flow.* Lisboa, 1979. 32 p. 175 × 250.
524 — AZEVEDO, M. Cruz; FERREIRA, M. J. Esteves; COSTA, C. A. Pereira da — *Influence of the spillways on the safety of arch dams.* Lisboa, 1979. 20 p. 175 × 250.

MINISTÉRIO DA HABITAÇÃO E OBRAS PÚBLICAS
LABORATÓRIO NACIONAL DE ENGENHARIA CIVIL

MEMÓRIA N.º 525

GEOM: A PROLOG GEOMETRY THEOREM PROVER

HELDER COELHO

LUIS MONIZ PEREIRA

LISBOA
1979

MINISTÉRIO DA HABITAÇÃO E OBRAS PÚBLICAS
LABORATÓRIO NACIONAL DE ENGENHARIA CIVIL

C. D. U. 519.682.5

MEMÓRIA N.º 525

GEOM: A PROLOG GEOMETRY THEOREM PROVER

HELDER COELHO

Engenheiro Electrotécnico, Estagiário para Especialista da Divisão de Informática

LUIS MONIZ PEREIRA

Professor Auxiliar, Universidade Nova de Lisboa

LISBOA

1 9 7 9

Work reported herein was conducted during 1975 at the Department of Artificial Intelligence, University of Edinburgh, and suported under grant no. 37/74 by INVOTAN (Lisbon) (H. C.) and Department of A. I. (L. M. P.).

Memória n.º 525
Tema — A1
1200 exemplares

GEOM: A PROLOG GEOMETRY THEOREM PROVER

SYNOPSIS

This report describes progress made in the development of GEOM, a PROLOG geometry theorem-prover, with the objective of achieving a better understanding of the capabilities of PROLOG and of geometry theorem-proving.

The report is divided into 8 sections which cover three main aspects: the definition of the problem domain, the points of friction found in the development of GEOM and the directions suggested by our research.

GEOM: UM DEMONSTRADOR DE TEOREMAS DE GEOMETRIA EM PROLOG

R E S U M O

Esta memória descreve os resultados obtidos com o desenvolvimento de GEOM, um demonstrador de teoremas de geometria em PROLOG, no intuito de obter uma melhor compreensão das capacidades do PROLOG e da demonstração de teoremas de geometria.

A memória está dividida em 8 secções as quais cobrem três aspectos principais a definição do domínio dos problemas, os pontos de fricção revelados pelo desenvolvimento de GEOM e as direcções sugeridas pela nova investigação.

GEOM: A PROLOG GEOMETRY THEOREM PROVER

1 — INTRODUCTION

1.0 — SUMMARY

This report describes progress made in the development of GEOM, a PROLOG geometry theorem-prover, with the objective of achieving a better understanding of the capabilities of PROLOG and of geometry theorem proving.

1.1 — OVERVIEW OF THE REPORT

The report is divided into 8 sections which cover three main aspects: the definition of the problem domain, the points of friction found in the development of GEOM and the directions suggested by our research.

The report begins 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 concludes with directions of research suggested for further work, such as the improvement of the language used itself.

1.2 — DESCRIPTION OF GEOM

GEOM is a PROLOG program for elementary plane geometry theorem proving. It is divided into sections which cover geometric and

arithmetic knowledge, the printing and assertion facilities and the utilities (1).

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).

1.3 — LANGUAGE USED

PROLOG is a programming language based on predicate logic (PL) [11]. The easiness of writing and understanding for the human problem solver support this choice.

1.4 — CONTRIBUTIONS

This report extends previous work done by Gelernter [3,4,5], Rochester [3], Gilmore [6], Reiter [10], Goldstein [7], Nevins [8] and Welham [13,14]. Basically, the following research questions 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;
10) non-proved goals in the context constructions made.

(1) A listing of the program GEOM and a PROLOG user's guide will be sent upon request.

2 — MOTIVATION

2.1 — This research was oriented to achieve a better understanding of the capabilities of PROLOG, a programming language based on first order logic or predicate calculus. To understand this language 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.

Elementary plane geometry theorem-provers have been attempted at times, during the last 16 years, as an exploration field of Artificial Intelligence.

The difficulties which prevent the development and more general use of a geometry theorem-prover are stressed in a report on previous work, by Coelho [1].

2.2 — At the start of our work, some general questions were put forward such as:

(1) to pay attention to the limitations and possible developments of PROLOG while using it for writing a geometry theorem-prover;

(2) how to have in a program a useable map of geometric knowledge;

(3) what geometric knowledge was needed for a collection of problems selected from previous work on geometry theorem-provers.

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 work, defining subgoals which are described in sections 5 and 6.

3 — PROBLEM FORMULATION

3.1 — INTRODUCTION

The type of problems suitable for GEOM, 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 canonical naming method.

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 that it be stored 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 involve combinatorial problems which must be harnessed.

3.2 — 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 present it aids in the proof of the goal. The diagram, however, 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 4.4.

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 music, are

RA(right angles)
RECTANGLE
SQUARE
PARALLELOGRAM
MIDPOINT

There are seven predicates to indicate the 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

3.3 — 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 [5], 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)

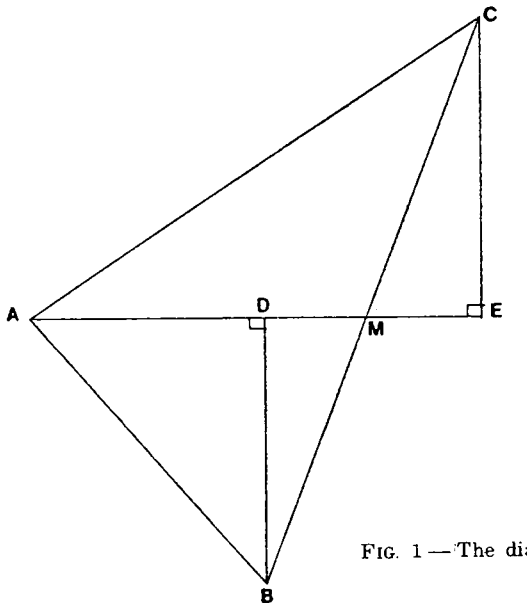


FIG. 1 — The diagram of Problem 3

3.4 — THE PROBLEM COLLECTION

The problem collection was built with problems from previous works:

Problem	1:	Gelernter[5]	problem	1
»	2:	»	»	2
»	3:	»	»	3
»	4:	»	»	4
»	5:	»	»	5
»	6:	Goldstein[7]	»	1
»	7:	»	»	2
»	8:	»	»	3
»	9:	»	»	4
»	10:	»	»	5
»	11:	»	»	6
»	12:	Nevins[8]	»	1
»	13:	»	»	2
»	14:	»	»	4
»	15:	Welham[13]	»	1
»	16:	»	»	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: for each problem only one proof is provided.

3.5 — 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.

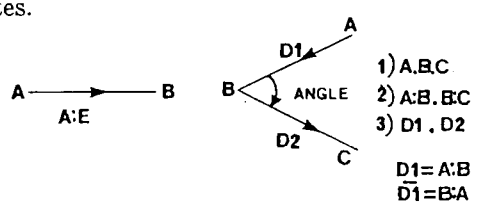


FIG. 2 — The representation of an angle

Each angle segment becomes defined by two points and its direction:

(A:B) means the direction from A to B in segment AB.

One direction can be defined by any pair of points belonging to the same equivalence class.

In figure 2, we can see the four cases for an angle, less than 180 degrees, defined by two directions: a and b.

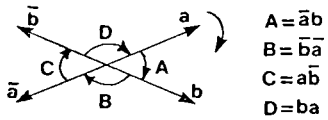


FIG. 3 — The four cases arising in the representation of an angle

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

3.6 — THE CANONICAL NAMING

The canonical naming routines are a set of rewrite rules, applied to an expression, to set it into some standard format. They reduce the ambiguity resulting from the syntactic variations of the thing 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 5.8.

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 becomes 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(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 de 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 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 segment defined by three points (A,B,C) or by two directions. Let us calculate for these two representations

the number of points of an angle segment after which canonical naming becomes more efficient:

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 recommanded for angles 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

4.1 — INTRODUCTION

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 [2]. 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.

4.2 — 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 explain what is required for converting 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 difference of segments is used. To avoid this combinatorial problem, Goldstein [7] and Welham [13, 14] 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 and 14 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 the congruence procedure is then compulsive and a depth-first exploration is done for each possible construction.

4.3 — 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;

(*) In parenthesis we give the name of the corresponding predicate.

(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;
(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.

4.4 — THE USES OF GEOMETRIC DIAGRAM

Two uses of a 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 by Gelernter [5].

The use of the diagram as a guide for helping the search is briefly explained in the following example (see figure 4).

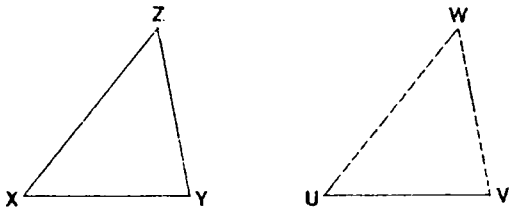


FIG. 4 — Proof of two equal segments, UV = XY, by congruent triangles

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

5.1 — INTRODUCTION

During the development of GEOM several points, called «of friction», recurred as problematic and motivating discussion. 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.

5.2 — 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 synthetize 4 search cases for pairs of triangles:

1) existing-existing
2) existing-generated
3) generated-existing
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 directions 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).

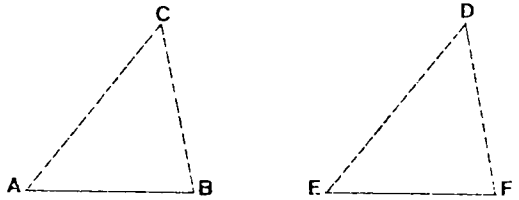


FIG. 5 — Generation of congruent triangles for the proof of two equal segments AB = EF by construent triangles

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.

5.3 — 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.

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

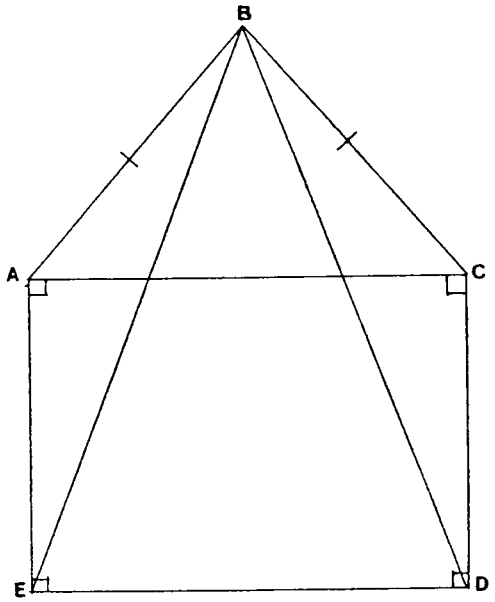


FIG. 6 — The diagram of Problem 10

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 [7], without the introduction of F is too particular a proof 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 non general model can be considered as a general one, if what it particularizes is not used in the proof. Goldstein's error consists of using in his proof facts not in the hypotheses (or not concluded), i.e. he uses the diagram unwarrentedly as a 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 the simplification obtained when a new point is introduced. Some 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 catrastrophic: a combinatorial explosion would occur. For the problem collection a heuristic was devised in order to balance the advantages and disadvantages of constructing: «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 quadrilaterals 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 the test of its existance, and followed by the assertion of equal segments, equal halves and directions for the new constructed segments.

5.4 — 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 [8]: 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 PROLOG. 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 existant, there is a progressive search in depth, the generation of more subgoals and a combinatorial explosion.

5.5 — 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 construction?
(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 figures 19 and 20.

One objective was to implement a construction facility for missing segments when proving that two segments are equal.

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.

The third question concerns the management in the data base of the additional clause, defining a construction. In fact two additional clauses are asserted, because the opposite direction is also stored (just as supplements are also stored for angles). 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).

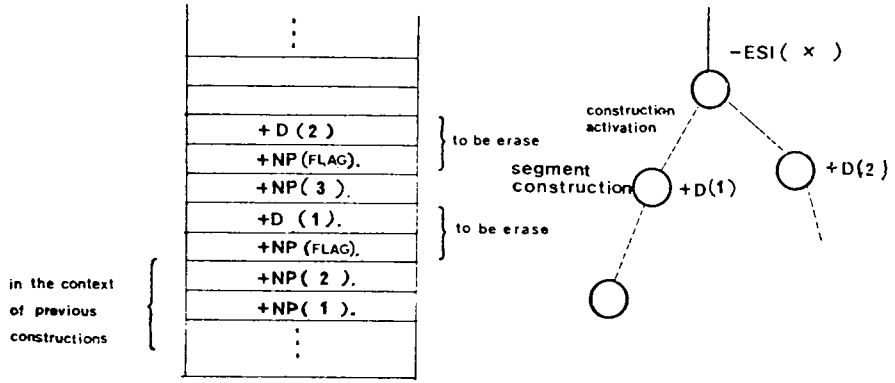


FIG. 7 — Data base of non-provable goals in the context of constructions made

After the two nonprovable subgoals, a construction is done motivated by a goal in the congruence clauses for equal segments or equal angles. A flag and the new unit clause are asserted in the data base. The flag hides the previous nonprovable goals, which may now become provable when the construcution is done. For this construction a new fact is derived and asserted as a lemma. No other lemmas are generated and the goal is not proved. If before a new construction is done the first one is undone (i.e. the flag, the direction unit clauses, but not the nonprovable subgoals are elliminated). The two clauses defining the construction are erased whether the goal be proved or not.

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.

5.6 — 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 shown in figure 19 for BA = NC.

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 is the recognition of a goal which failed before but only if no new construction has been made. The mechanism to implement this objective is composed by two clauses, the nonprovable 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 6.4.

5.7 — 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 [2].

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 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 *S2 and *S4 act as input variables.

The second tyee, to find, is used to find a desirable possibly existing 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 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. *S2 and *S4 act as output variables: 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 procedure for bottom-up is activated.

5.8 — 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 classe concept it 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 relation (although no explicit arc may exist between them).

These trees grow by merging as explained in the example illustrated in figure 8a, 8b and 8c.

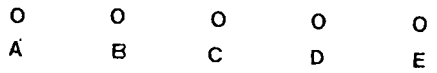


FIG. 8a — The growing of an equivalence class tree

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.

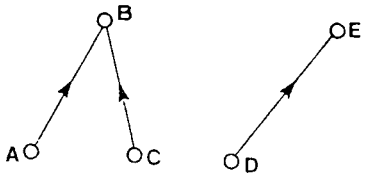


FIG. 8b — The growing of an equivalence classe tree

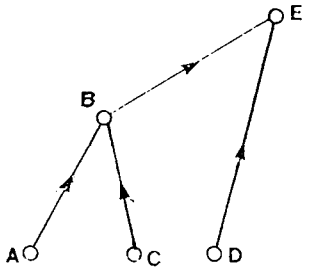


FIG. 8c — The growing of an equivalence class tree

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 grouwth of the tree.

In the data base three relation elements are stored: R(A,B), R(C,B) and R(D,E). One relation element, R(A,C), is implicit. Consider a new 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 fol 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.

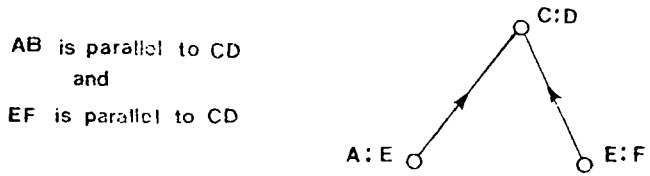
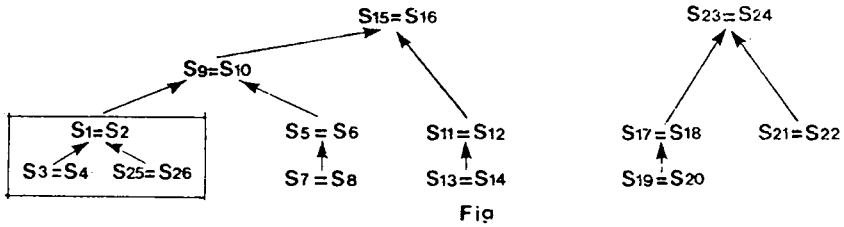


FIG. 9 — A tree of parallel directions

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.



Fig

FIG. 10 — The tree organization of equal sides

This characteristic of the data, the existance 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 occured, on account of the type of questions posed. 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 congruent two 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.

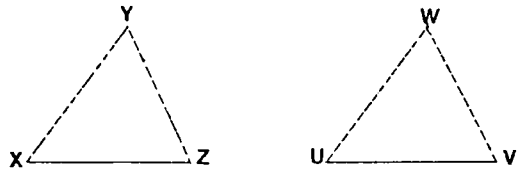


FIG. 11 — The construction of construent triangles on equal sides

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. 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.

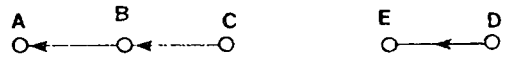


FIG. 12 — List structures representing equivalence classes

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.

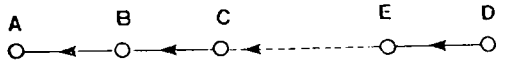


FIG. 13 — The construction of a list structure representing the fusion of two equivalence classes

The witness or head 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.

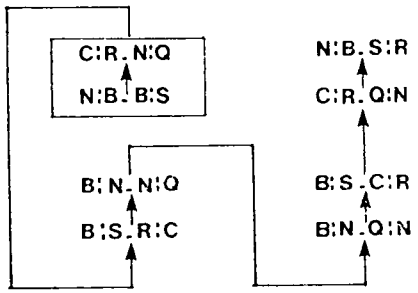


FIG. 14 — The list organization for equal angles

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.
(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.

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

to prove BA is equal to CD
try first to verify if the fact «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, to 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.

5.9 — 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 [5] and Goldstein [7] defended the first

approach. Nevins [8] 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, an 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 question: «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.

Nevins's problems, typicaly bottom-up, are in a way very concocted because they hide given facts behind congruence relations. Welham [13, 14] 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, chosen to fare well with his bottom-up methods, are difficult for our top-down prover.

Analysing the proof tree for one of these problems (PR13), we may observe 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 our 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».

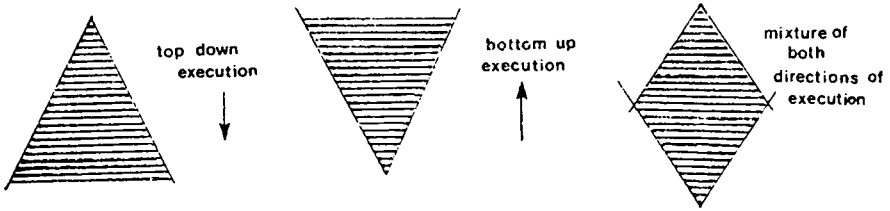


FIG. 15 — Top-down versus bottom-up searches

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.

5.10 — 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).

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

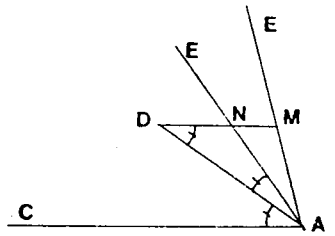


FIG. 16 — A part of the diagram of Problem 14

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 parallel or antiparallel sides
2st. SUG-GOAL: to prove the equality of EAD and NDA.

The motivation for the use of transitivity was spurred by the anlysis 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

6.1 — INTRODUCTION

Efforts were done to separate 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.

6.2 — 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 [2]).

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.

6.2.1 — *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 ration is done by the procedure CON3. For the first clause, 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 (B.*Z = D.*W !*W1 !DBAS)

of the clause

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

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

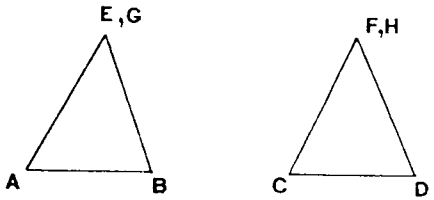


FIG. 17 — The search for a congruence relation

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, controled by clause 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.

6.2.2 — *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, 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.

6.3 — 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;
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 4.5 and proposed by Gelernter [5]. 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 5.5, 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.

6.4 — SUBGOAL CONTROL

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

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

An example illustrates these facilities and its deficiencies, and the need for a better interpreter [9]. 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 extant 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.

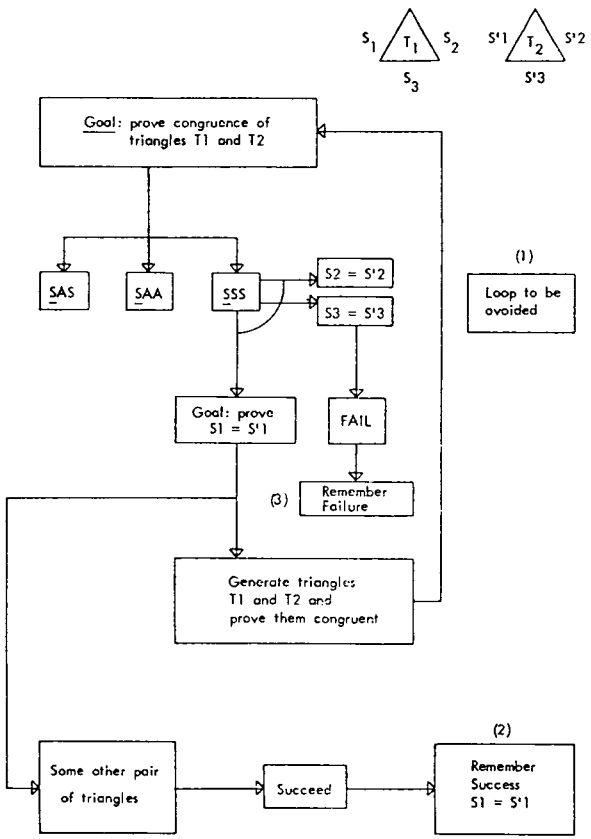


FIG. 18 — Cases of sub-goal control

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.

6.5 — 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 any procedure;
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 [8]; Welham [13, 14]).

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

7.1 — INTRODUCTION

Some aspects are discussed, suggesting further research directions for the development of a geometry theorem prover.

The main aspect is the improvement of PROLOG 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.

7.2 — ALTERNATIVE PROOF 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 [13, 14].

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 buurocrat, proving a unnecessary fact: AP = CP.

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

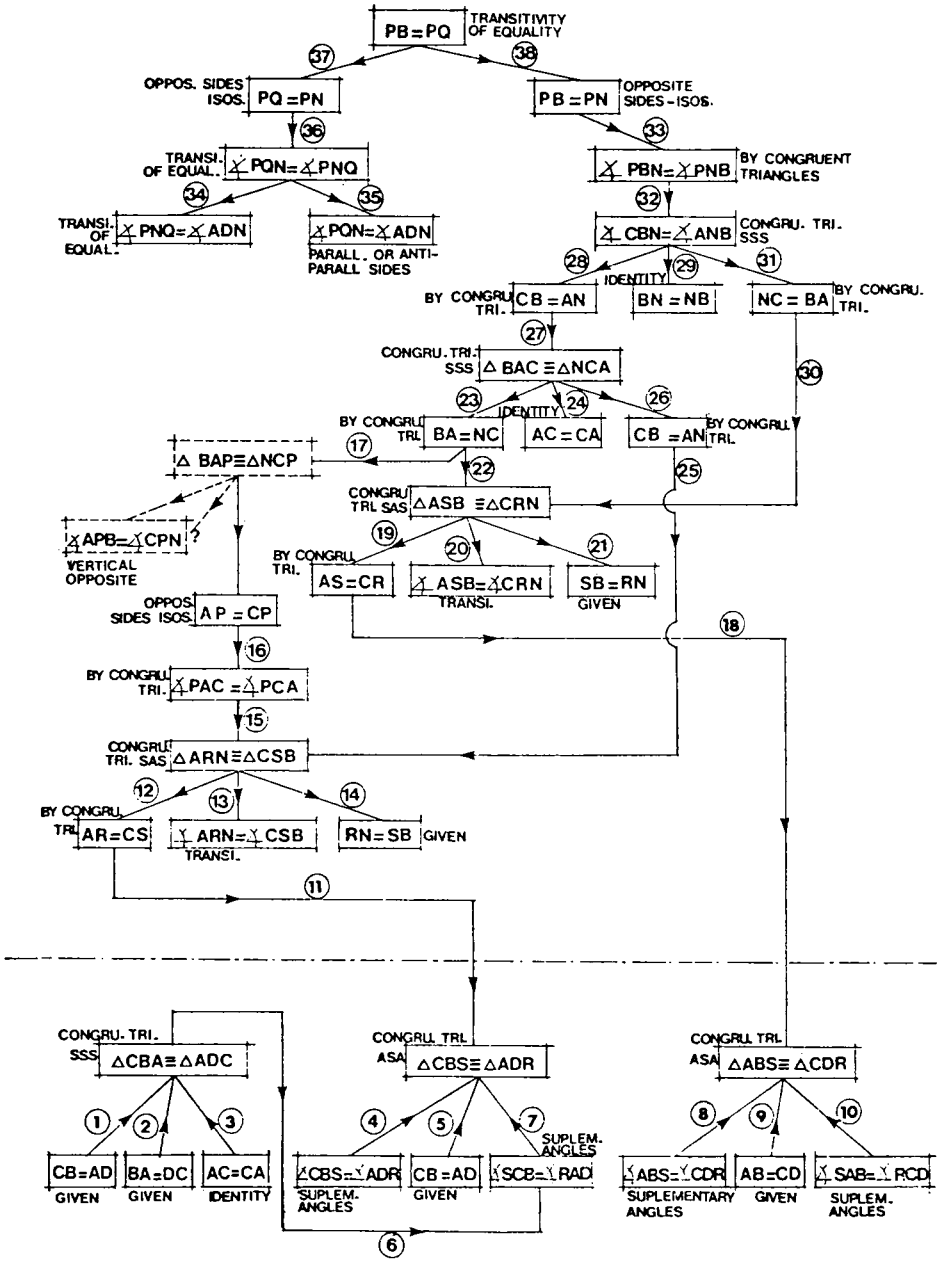


FIG. 19 — The proof tree of Problem 13 used by GEOM

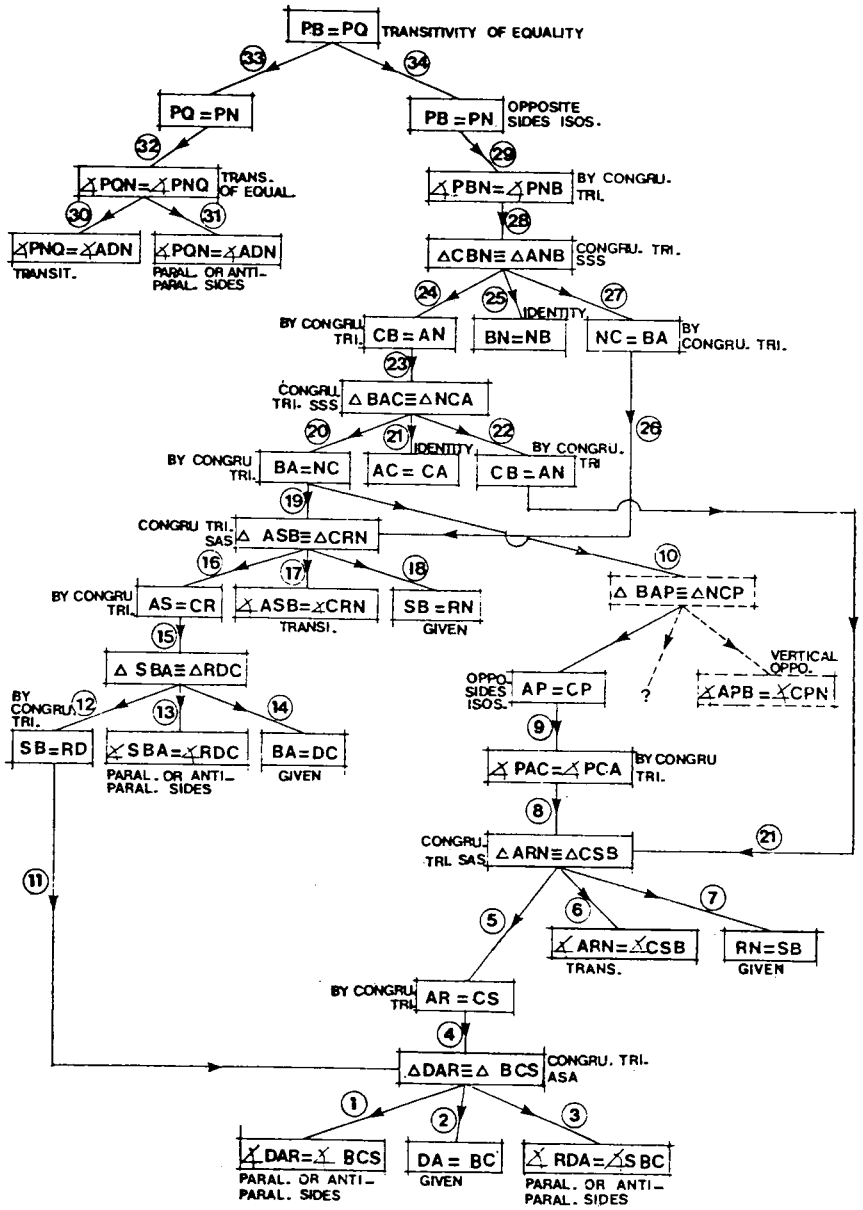


FIG. 20 — The proof tree of Problem 13 used by GEOM 1

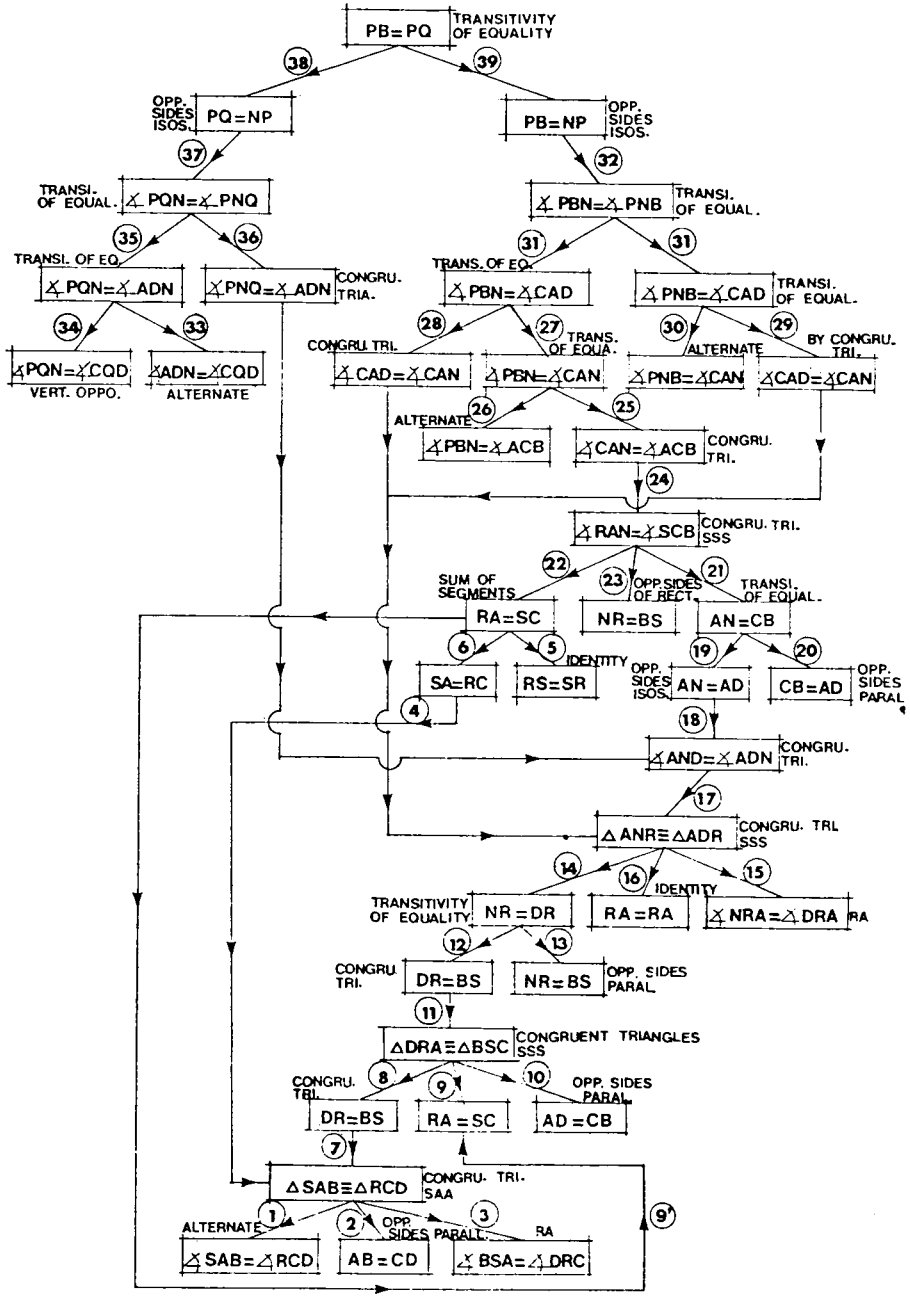


FIG. 21 — The proof tree of Problem 13 used by G

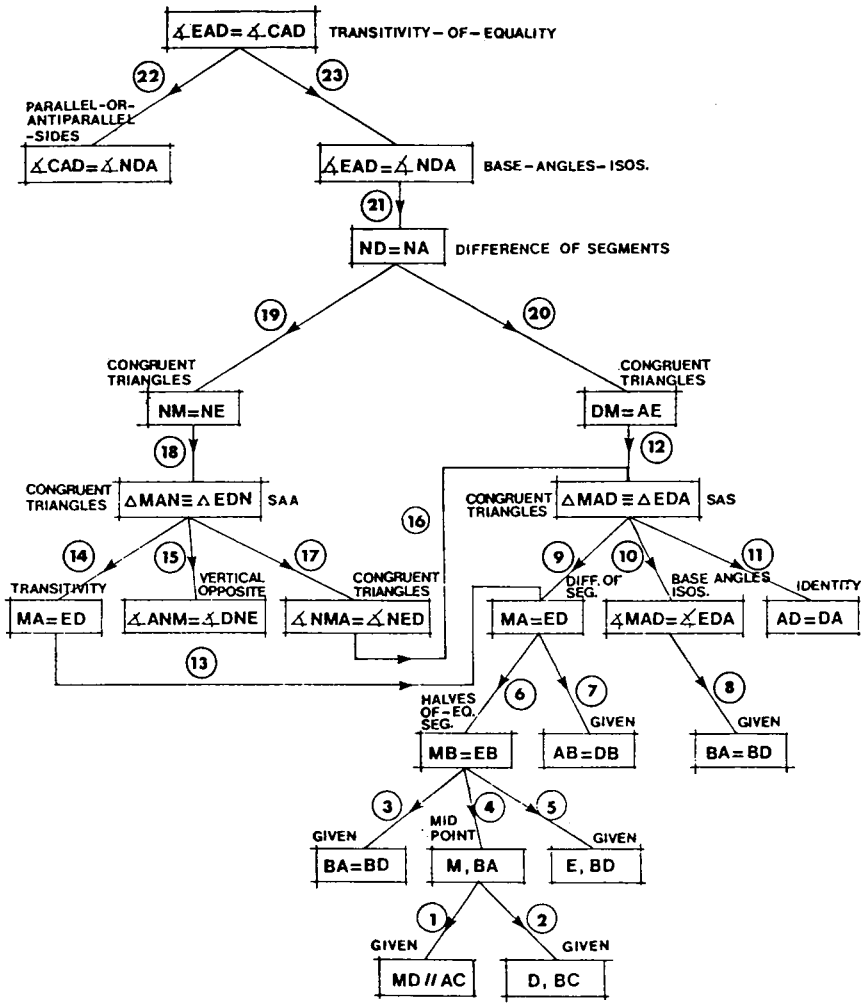


FIG. 22 — The proof tree of Problem 14 used by GEOM

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

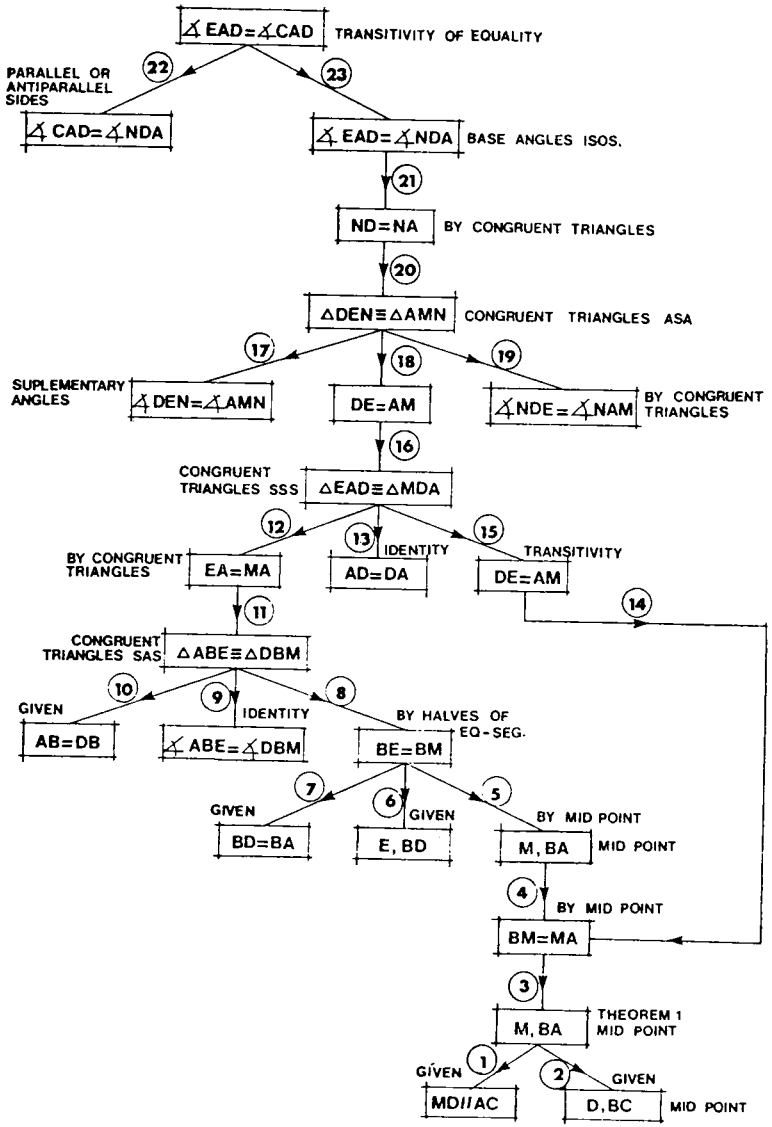


FIG. 23 — The proof tree of Problem 14 used by GEOM 1

Constructions have been discussed as a facility for proving equal segments or equal angles by congruent triangles (see section 5.5). Those constructions, line segments, were done for achieving a certain subgoal, and erased afterwards.

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 induce 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.

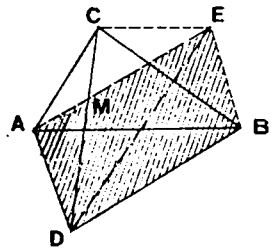


FIG. 24 — A construction pattern: the parallelogram

The second approach, the implementation of a new inference system from scratch proposed by Warren (Pereira and Meltzer [9]) is discussed in section 7.6, concerning the overall improvements of PROLOG.

7.3 — 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 discuussed in section 6.4. 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 or the theorem may be the only component to be stored. New theorems would be used by the already existing control. In section 6.2 the separation between logic and control was discussed, and examples were given on how this problem was tackled in GEOM.

7.4 — THE USES OF SYMMETRY

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

1) syntactic
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 hypotheses 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 hypotheses 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 [3, 4] 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;
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 'similarl').

Gelernter's symmetry is not calculated by his program. On the contrary, it is declared by the user by observing the geometric diagram. A combinatorial 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 3.6.

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 Pereira, on making a PROLOG program, called SYMM ([3]), 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.

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

([3]) A listing of program SYMM will be sent upon request.

RULE 1T:

point U is symmetric relatively to pair XY (X different from Y) if there is a pair X1Y1 symmetric to pair XY, and point X1 is different from point Y1, pair UX1 equals pair UY1 (see figure 25).

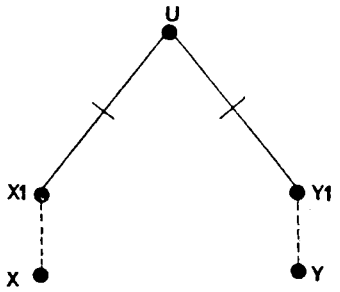


FIG. 25 — Symmetry rule 1T

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, angles < UZX1 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 <VY1X1 are equal, and direction X1 to Y1 is parallel to direction U to V (see figure 26).

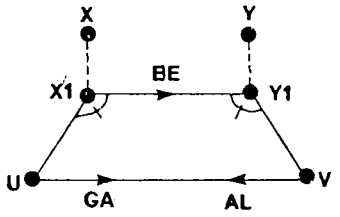


FIG. 26 — Symmetry rule 3T

RULE 4TC:

pair UV is symmetric to pair XY if point P 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, 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).

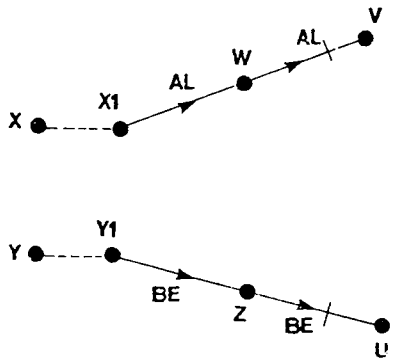


FIG. 27 — Symmetry rule 4TC

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

Futher 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 giver 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».

7.5 — 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 algorithm 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.

7.6 — THE IMPROVEMENT OF PROLOG

Our experience with problematic situations, detailed in sections 6.3 and 6.4, 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 Meltzer [9]), 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 6.3 and 6.4. The example shown in figure 18 is now revisited with the help of figure 28.

In this figure, 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

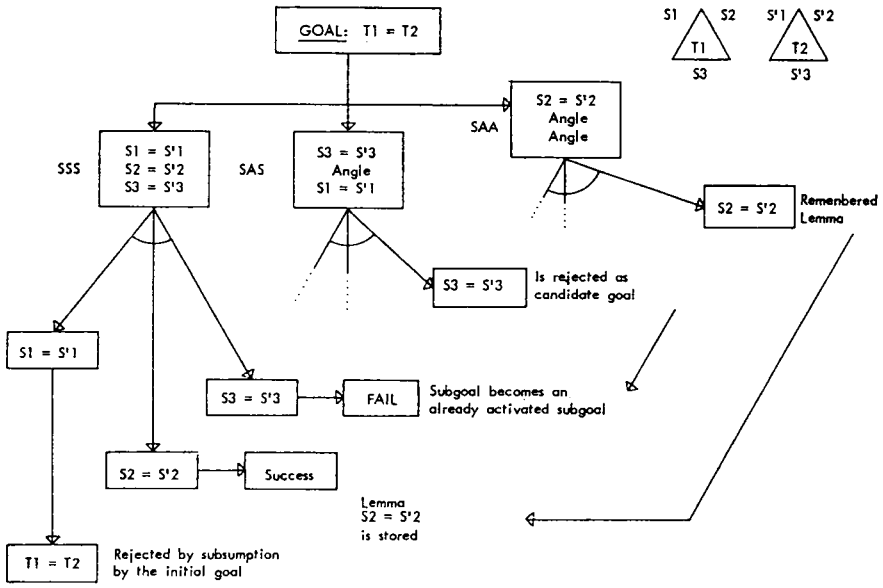


FIG. 28 — The sub-goal control examples reviseted

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 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 reiterate some of the relevant 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.

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 motivated improvements of the used language, PROLOG. The main improvement

of the PROLOG operational semantics will be the implementation of an efficient predicate logic interpreter, based on Earley deduction. This improvement will provide complete satisfaction for three general problematic situations: avoiding loops, storing proved facts, and remembering goals which failed.

12) the enlargement of PROLOG applications

Earley deduction and other improvements over PROLOG, such as data base management, will support the use of a language based on PL in a large and complex domain.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the helpful suggestions of Pavel Brazdil, Bob Welham, for his ideas and comments on how to implement a geometry theorem-prover in PROLOG, Bob Kowalski, who introduced us to predicate logic programming, and Dave Warren, who taught us the power of PROLOG.

REFERENCES

[1] COELHO, H. — *An inquiry on the geometry machine and its extensions with a review of previous work.* Working report, no. 1 for INVOTAN, 1974.

[2] van EMDEN, M. — *Programming with resolution logic.* Research report CS-75-30, University of Waterloo, 1975.

[3] GELERNTER, H. and ROCHESTER, N. — *Intelligent behaviour in problem-solving machines.* «IBM Journal», October, 1958.

[4] GELERNTER, H. — *A note on syntactic symmetry and the manipulation of formal systems by machine.* «Information and Control» n.° 2, 1959.

[5] GELERNTER, H. — *Realization of a geometry theorem proving machine.* «Proc. Int. Conf. on Information Processing», 1959.

[6] GILMORE, P. C. — *An examination of the geometry theorem machine.* «Artificial Intelligence», 1, 171-187, 1970.

[7] GOLDSTEIN, I. — *Elementary geometry theorem proving.* MIT, AIL memo, n.° 280, April 1973.

[8] NEVINS, A. — *Plane geometry theorem proving using forward chaining.* «Artificial Intelligence», vol. 6 n.° 1, Spring 1975.

[9] PEREIRA, L. M. and MELTZER, B. — *Implementation of an efficient predicate logic interpreter based on Earley deduction.* Scientific proposal report for SRC, April 1975.

[10] REITER, R. — *The use of models in automatic theorem-proving.* Univ. of British Columbia, Techn. report 72-04. September 1972.

[11] ROUSSEL, P. — PROLOG — *Manuel de reference et d'utilization.* G.I.A., U.E.R. de Luminy, Univ. d'Aix-Marseille, 1975.

[12] WARREN, D. — Epilog: an user' guide to DEC-10 PROLOG. DAI, report stored in 400,400 area of DEC-10 of the Univ. of Edinburgh, August, 1975.

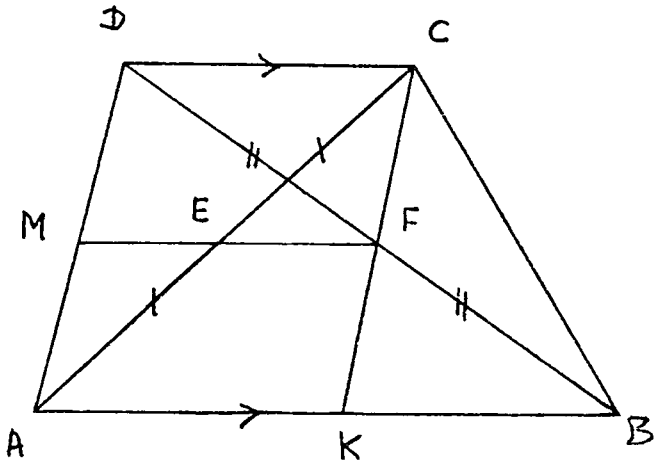
[13] WELHAM, R. — *G, a geometry theorem prover.* (unpublished), March 1975.

[14] WELHAM, R. — *Geometry problem solving.* D.A.I. Research report n.° 14, January 1976.

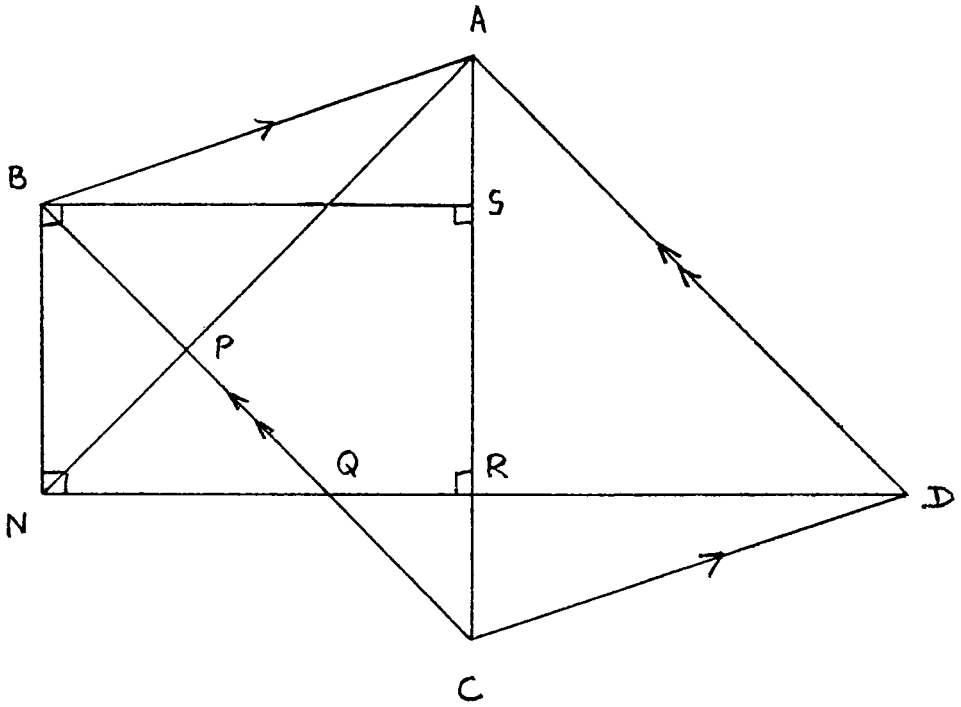
APPENDIX 1

A SAMPLE OF THE PROBLEM COLLECTION

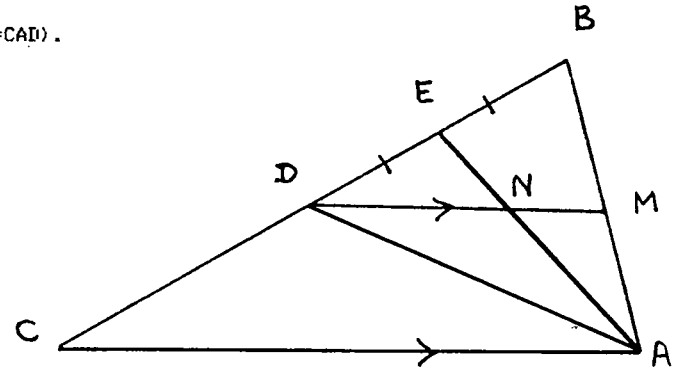
```
-SORCHA("PROBLEM 5     GELERNTER,5").
+DIAGRAM.

-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 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.AN.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 SAMPLE 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 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                          UFFOS - 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.

Composto e impresso por Ramos, Afonso & Moita, Lda. — Lisboa