Journal of Automated Reasoning **25:** 219–246, 2000.

A Deductive Database Approach to Automated Geometry Theorem Proving and Discovering ⋆

SHANG-CHING CHOU
Dept. of Computer Science, Wichita State University, Wichita, KS 67260, USA

XIAO-SHAN GAO
Inst. of Systems Science, Academia Sinica, Beijing 100080, China

JING-ZHONG ZHANG
Inst. of Computer App., Academia Sinica, ChengDu 610015, China

(Received: 26 September 1996)

Abstract. We report our effort to build a geometry deductive database, which can be used to find the *fixpoint* for a geometric configuration. The system can find all the properties of the configuration that can be deduced using a fixed set of geometric rules. To control the size of the database, we propose the idea of a *structured deductive database*. Our experiments show that this technique could reduce the size of the database by one hundred times. We propose the data-based search strategy to improve the efficiency of forward chaining. We also make clear progress in the problems of how to select good geometric rules, how to add auxiliary points, and how to construct numerical diagrams as models automatically. The program is tested with 160 nontrivial geometry configurations. For these geometric configurations, the program not only finds most of their well-known properties but also often gives unexpected results, some of which are possibly new. Also, the proofs generated by the program are generally short and totally geometric.

Key words: deductive database, automated geometry theorem proving and discovering, search strategies, redundant deduction, Skolemization, structured database.

1. Introduction

1.1. INTRODUCTION

The aim of this paper is to develop a geometry deductive database that can be used to prove or discover nontrivial geometry theorems by exploring the full strength of forward chaining. For a given geometric configuration, the program can find its *fixpoint* with respect to a fixed set of geometric rules or axioms; in other words, it can find all the properties of the configuration that can be deduced using these axioms.

The basic ideas behind the program, such as fixpoints and the treatment of negative clauses, come from deductive database theory [10]. Our contributions are as

⋆ This work was supported in part by the NSF Grant CCR-9420857 and the Chinese NSF.

follows: (1) in the general setting, we propose the idea of the *structured deductive database* and the *data-based search strategy* to improve the search efficiency; and (2) in the geometry reasoning setting, we show how to select a good set of rules, how to add auxiliary points, and how to construct numerical diagrams as models automatically.

We tested the program with 160 geometry configurations ranging from well-known geometry theorems such as the centroid theorem, the orthocenter theorem, and Simson's theorem to problems recently proposed in the problem section of the *American Mathematical Monthly*. The program not only finds most of the well-known properties of these configurations but also often gives many unexpected results, some of which are possibly new (see Example 6.2). As we know, most of the 160 theorems cannot be proved by the previous programs based on synthetic approaches. The strength of our program is mainly based on the following improvements.

In traditional deductive databases, each n-ary predicate is associated with an n-dimensional relation. Since most of the geometric predicates satisfy special properties such as transitivity and symmetry, the traditional way of representing a database is not suitable for building a geometry deductive database for two reasons: the excessively large database and repetitive representation of information. On average, the databases for the 160 tested geometry configurations would be of size 242,117 if using the traditional representation. We solve this problem by using some simple mathematical structures such as sequences and equivalent classes to represent facts in the database. The average size of the structured databases for the 160 configurations is 221. So the size of the structured database is one hundred times smaller.

We propose the *data-based search strategy* as opposed to the previous *rule-based search strategy*. In the data-based search, we keep a list of 'new data' and for each new data the system searches the rule set (or intensional database) to find and apply the rules using this data. If using the data-based strategy, the redundant deductions caused by repeated application of a rule to the same facts will be automatically eliminated. In this aspect, the data-based strategy is similar to the *semi-naive strategy* [1]. However, the data-based strategy also uses the *combined rules* and *dynamic database update* strategies to further improve efficiency. Also, the data-based search strategy is particularly efficient in the case of a structured database, since in such a database a fact could represent a large amount of information. For instance, a fact could be 'ten triangles are similar to each other'.

All the previous work based on the synthetic approach [8, 11, 15, 17] uses geometric rules about congruent triangles as its basic geometric rules. Without adding techniques about auxiliary points, these rules can be used to prove a limited number of high-school-level theorems involving straight lines only. Most of the basic results in geometry such as the centroid theorem, the orthocenter theorem, and Simson's theorem are beyond the scope of these rules. In Section 2, we will discuss three concerns about selecting a set of geometric rules: the number of proof

steps, the auxiliary points, and the order relation. We also show how to solve some of the related problems by selecting good rules.

We discuss how to make the adding of auxiliary points or Skolemization in geometry reasoning practically possible. About twenty rules of adding auxiliary points are implemented, and 39 of the 160 configurations solved by our program need auxiliary points. This is the first implementation of a nontrivial set of rules of adding auxiliary points.

Numerical diagrams of geometry statements are used as models to deal with the negative information in the rules. Our program can construct the diagram automatically for a class of linear constructive geometry statements (see Section 2.3). In all the previous work using diagrams as models [8, 11, 15], the diagrams are constructed by the user.

1.2. RELATED WORK

There are mainly three approaches to automated geometry reasoning: the approach based on *synthetic deduction* [8, 11, 15, 17, 19], the approach based on *algebraic computation* (mainly Wu's method and the Gröbner basis method) [3, 14, 26], and the *geometric invariant approach*, like the area method [5] and the methods given in [12, 24]. Other interesting work can be found in [9, 16, 23].

We will discuss the synthetic approaches below because they are more closely related to this paper. Most of the synthetic approaches to automated geometry theorem proving use backward chaining [8, 11, 15]. In [17], Nevins used a combination of forward chaining and backward chaining with emphasis on the forward chaining. But the fixpoint is not reached. All the synthetic approaches except Nevins's use numerical diagrams as models. The idea of adding auxiliary points is discussed in [19, 21] but not implemented. More about this topic can be found in Section 5. All previous synthetic work deals with theorems involving straight lines only. The problem of including circles into the program is discussed but not implemented in [17]. Our program, using geometric rules with the *full-angle congruence* as the central concept, can deal with theorems involving circles naturally.

Generally speaking, the algebraic approaches are decision procedures and are more powerful. The synthetic approaches, including the one in this paper, are not decision procedures. In theory, all theorems that can be proved with the method reported in this paper can also be proved with algebraic methods such as Wu's method. Despite its 'weakness,' it is still worth improving the synthetic approach because this may lead to techniques useful to automated reasoning in the general case. Even for automated geometry reasoning alone, improving the synthetic method has the following positive aspects. (1) Proofs produced by synthetic methods are generally easier to understand than proofs given by algebraic computations. (2) Using predicates only (no algebraic computation) makes the reaching of fixpoints possible. As a result, new theorems may be discovered. Using algebraic methods, one can also discover new geometric facts [4, 18], but in a quite differ-

ent manner. (3) Although algebraic methods can prove a much greater number of theorems, there still exist theorems (Example 6.2) that can be solved by the synthetic approaches elegantly but have not been solved with the algebraic approaches because to prove them we need excessively large computer memory.

General-purpose deductive database systems fail to reach fixpoints for most of our examples within reasonable time because of the large database and the weak strategies for the redundancy control. Some useful general methods of controlling redundancies [1, 2, 13] and their usage in our case are discussed in Section 4.2.

Finally, we mention that the method in this paper benefits from many ideas developed in the algebraic approaches. The idea of using geometric predicates not involving the order relation, such as the full-angle congruent, is initiated by Wu in his algebraic method [26]. The concept of nondegenerate conditions and statements of constructive type is also from algebraic methods.

The rest of the paper is organized as follows. The selection of geometric rules is discussed in Section 2. The database organization is discussed in Section 3.1. The search strategies are discussed in Section 4. Adding auxiliary points is discussed in Section 5. Concluding remarks are given in Section 6.

2. The Geometric Rules

To select a set of 'better' geometric (inference) rules, we need to consider the following issues.

Auxiliary Points

Since most synthetic geometry reasoning systems use Horn clauses as rules, the systems have no ability to generate auxiliary points. Thus it is important that we can prove at least a large portion of the geometry theorems with the chosen geometric rules. The rules about congruent triangles, used by most of the previous work on synthetic automated geometry theorem proving, do not have this property. Most of the commonly used theorems including basic results such as the orthocenter theorem and the centroid theorem cannot be proved using these rules without adding auxiliary points. The rules used by us improve a lot in this aspect: most of the 160 geometry theorems proved by our program without adding auxiliary points cannot be proved with the previous programs based on congruent triangles.

Order Relations

The validity of most elementary geometry theorems involving only equalities is independent of the relative order positions of the points involved. Such geometry theorems belong to the so-called unordered geometry. This idea originated from Wu's algebraic method of automated reasoning [26]. In unordered geometry, the proofs of these theorems can be very simple. However, the ordinary proofs of these theorems involve the order relation (or inequalities); hence they are not only

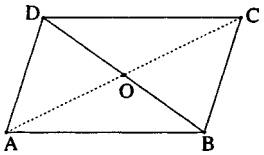

Figure 1.

complicated but also not strict. The method based on congruent triangles is not for unordered geometry. In the work using this method, the order relations needed in the deduction are either derived from a numerical diagram or given in the input. As a result, these programs can be used only to prove particular cases of a geometry theorem. On the other hand, the rules used by us are for unordered geometry.

For instance, one proof of the theorem 'the two diagonals of a parallelogram bisect each other' in Figure 1 is based on the congruence of triangles ABO and CDO which is in turn based on the fact that angles $\angle OAB$ and $\angle OCD$ are alternative angles of the parallel lines AB and CD. In this statement, we implicitly assume that points B and D are on different sides of line AC. But this fact is not proved logically in most machine produced proofs. If a set of rules for unordered geometry, like the one in this paper, is used, a proof of this theorem does not need the fact that points B and D are on different sides of the line AC.

2.1. THE GEOMETRIC RULES

A rule is called a *definite Horn clause* if it has the following form:

$$Q(x) :- \quad P_1(x), \ldots, P_k(x) \quad \text{meaning}$$
$$\forall x[(P_1(x) \wedge \cdots \wedge P_k(x)) \Rightarrow Q(x)]$$

where the x are the points occurring in the geometry predicates $P_1, \ldots, P_k$, and Q. Also $(P_1(x) \wedge \cdots \wedge P_k(x))$ and $Q(x)$ are called the *body* and *head* of the rule, respectively. As in the field of deductive bases [10], only definite Horn clauses without function symbols are allowed in our program. As a consequence, no algebraic computations are allowed because algebraic computations are actually function symbols and may easily lead to infinite inference sequences and hence make the reaching of fixpoints impossible.

We use the following predicates: points, coll (collinear), para (parallel), perp (perpendicular), midp (midpoint), cyclic, circle, eqangle, cong (congruent of segment), eqratio, simtri (similar triangle), and contri (congruent triangle).

The central concept is eqangle. Here the angle is not the ordinary angle but the full-angle. Intuitively, a *full-angle* $\angle[u, v]$ is the angle from line u to line v. Note that u and v are not rays as in the definition for the ordinary angles. Two full-angles $\angle[l, m]$ and $\angle[u, v]$ are equal if there exists a rotation K such that $K(l) \parallel u$ and $K(m) \parallel v$. If A, B and C, D are distinct points on l and m, respectively, then $\angle[l, m]$ is also denoted by $\angle[AB, CD]$, $\angle[BA, CD]$, $\angle[AB, DC]$, and $\angle[BA, DC]$.

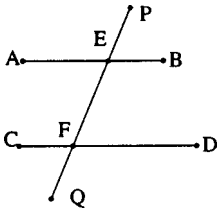

Figure 2.

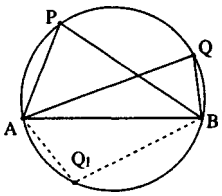

Figure 3.

The introduction of full-angles greatly simplifies the predicate of the angle congruence. For instance, we have the following rule about parallel lines and angles.

R1. $AB \parallel CD$ if and only if $\angle[AB, PQ] = \angle[CD, PQ]$ (Figure 2).

If using ordinary angles, we need to specify the relations among eight angles and we need to use order relations (inequalities) to distinguish the cases. For instance, we have 'if points B, D are on the same side of line PQ and point P, C are on the different sides of line AB (the order relations), then $AB \parallel CD \Leftrightarrow \angle PEB = \angle PFD$.' This rule is very difficult to use and may lead to branchings during the deduction. The following two rules also show why full-angle is crucial to our approach.

R2. $\angle[PA, PB] = \angle[QA, QB] :-$ cyclic(A, B, P, Q) (Figure 3).

R3. cyclic$(A, B, P, Q) :- \angle[PA, PB] = \angle[QA, QB]$, $\neg$ coll(P, Q, A, B) (Figure 3).

In rule R2, if using the ordinary angle, we need two conditions (Figure 3): $\angle APB = \angle AQB$ or $\angle APB + \angle AQ_1B = 180°$ and to distinguish these two cases, we need to know 'points P and Q are on the same or different sides of line AB.' This kind of order relations is difficult to deal with, because for different diagrams of the same theorem, the answer could be different. Using full-angles, the two cases can be treated uniformly. Also note that the use of full-angles here is quite different from our previous full-angle method in [7], which uses algebraic computation and backward chaining as the main deduction tools, while here only the concept of angle congruence is used.

The program uses about seventy rules (see the appendix). Some of the rules describe basic properties of the geometry predicates, such as $AB \parallel CD :- AB \parallel PQ, CD \parallel PQ$. Some of the important rules are listed below.

R4. para$(E, F, B, C) :-$ midp(E, A, B), midp(F, A, C).
R5. midp$(F, A, C) :-$ midp(E, A, B), para(E, F, B, C), coll(F, A, C).
R6. $\angle[OA, AB] = \angle[AB, OB] :-$ cong(O, A, O, B).
R7. cong$(O, A, O, B) :- \angle[OA, AB] = \angle[AB, OB]$, $\neg$ coll(O, A, B).
R8. $\angle[AX, AB] = \angle[CA, CB] :-$ circle(O, A, B, C), perp(O, A, A, X).
R9. $\angle[AB, AC] = \angle[OB, OM] :-$ circle(O, A, B, C), midp(M, B, C).
R10. perp$(A, B, P, Q) :-$ cong(A, P, B, P), cong(A, Q, B, Q).
R11. perp$(P, A, A, Q) :-$ cong(A, P, B, P), cong(A, Q, B, Q),
cyclic(A, B, P, Q).
R12. simtri$(A, B, C, P, Q, R) :- \angle[AB, BC] = \angle[PQ, QR]$,
$\angle[AC, BC] = \angle[PR, QR]$, $\neg$coll(A, B, C).
R13. eqratio$(A, B, A, C, P, Q, P, R) :-$ simtri(A, B, C, P, Q, R).
R14. contri$(A, B, C, P, Q, R) :-$ simtri(A, B, C, P, Q, R), cong(A, B, P, Q).

Notice that we also use $\angle[AB, CD] = \angle[PQ, UV]$ and $AB = CD$ to represent full-angle and segment congruences. Though not axioms in common textbooks, these rules are basic geometric facts that can be proved without difficulty.

In common textbooks and previous synthetic approaches, the three theorems about triangle congruence, s.s.s, s.a.s, and a.a.s, are the key deduction rules. In our program, only the a.a.s (Rule R14) is used. The s.a.s rule is not correct if full-angle is used. For instance, triangles AOB and AOD in Figure 1 satisfy $BO = DO$, $AO = AO$, $\angle[BO, OA] = \angle[DO, OA]$, but they are not congruent. The s.s.s rule is correct but cannot generate new properties about full-angles in the general case.

The rules are highly complicated from the viewpoint of a deductive database: all the predicates are mutually recursive, and most of the rules are not linear [1]. We mention that the rules are not complete in the sense that a valid geometry theorem described using our predicates may not be proved using our rules.

2.2. NONDEGENERATE CONDITIONS

In some rules such as Rule R3, there are negations in their bodies. Strictly speaking, these rules are not Horn clauses. We solve this problem using the *negation by failure* criteria proposed in [20]; that is, we assume $\neg P$ to be valid if P cannot be deduced by the program. This may lead to inconsistency, since the rules used by us are not complete and hence failure to prove P does not mean that $\neg P$ is valid. We solve this problem in two steps.

First, whenever used, a negative condition will be added to the hypotheses of the statement and called the *nondegenerate* (ndg) *condition* of the geometry statement. Adding the ndg conditions to the geometry statement is acceptable for the following reasons. The original description for most of the geometry theorems in textbooks implicitly assumes some necessary conditions, which are part of the ndg conditions found by our method. For some theorems proved by previous synthetic

provers, the necessary ndg conditions are missing. More about ndg conditions can be found in [3, 9, 26].

Second, suppose that we have an exact numerical diagram for the statement. When encountering a negative predicate $\neg P$ in a rule, the prover checks whether P is true in the diagram. This rule can be used only if P is false in the numerical diagram. As a consequence, we avoid the possible proof of 'trivially true' statements, namely, statements whose hypotheses are inconsistent.

2.3. CONSTRUCTING EXACT NUMERICAL MODELS

The use of exact numerical diagrams as semantic models has been the cornerstone of most of the previous efforts of synthetic mechanical geometry theorem proving [8, 11, 15]. There are two benefits the early provers derive from a numerical diagram. (1) The diagram is used as a filter to reject goals occurring during a backward chaining that are not consistent with its numerical representation. (2) More important, the numerical diagram is used to determine order relations among points and lines. The first benefit is very important to the backward search and is not useful in the forward chaining. The second benefit is related to the problem of producing diagram-independent proofs [6]. In this paper, the diagrams are used to treat negative information in the Horn clauses.

In previous work, the diagram was prepared by the user. In our program, the diagram is generated automatically for a class of *linear constructive geometry statements*. A geometry statement is said to be linear constructive if the points in the statement can be listed in a sequence such that each point in the sequence can be uniquely constructed from the previous points in the sequence. More precisely, a geometry statement is linear constructive if the points in it can be described by the following constructions:

- Take a free point.
- Take an arbitrary point on a line.
- Take the intersection of two lines.
- Take the intersection of a line and a circle or two circles when the other intersection point of them is already constructed.

For instance, the statement in Figure 1 is a linear constructive statement. Its points can be introduced as follows: take three free points A, B, and C; take the intersection D of the line passing through point A and parallel to BC and the line passing through point C and parallel to AB; take the intersection O of lines AC and BD. More details can be found in [5]. Most of the commonly used geometry statements are linear constructive ones. For example, 80 percent of the 512 geometry theorems in [3] are in this class.

For a linear constructive geometry statement, the coordinates of each point can be represented as rational expressions in the coordinates of the previous points.

Thus, by assigning random numbers to the coordinates of the free points in the statement, we can easily compute the coordinates of all the points as rational numbers and hence an exact numerical model for the statement.

If a geometry statement is not linear, then to construct its diagram we generally need algebraic numbers to represent the coordinates. Also, if the statement is reducible (see [3] for the definition) more than one diagram is needed for the statement. Our program cannot construct an exact numerical model for such statements.

3. Structured Database

3.1. STRUCTURE OF THE DATABASE

In the traditional way of representing a relational database, each n-ary predicate is associated with an n-dimensional array. Since most geometric predicates used by us satisfy some special properties, building databases according to the traditional way will lead to very large databases. The main aim of this section is to design a structured database whose size can be effectively controlled. The main idea is to represent the properties involving a predicate alone using the structure of the database. In other words, we build some rules about predicates into the structure of the database. The following are three such principles:

(1) *Use canonical form for predicates*. One geometric property can be represented as many predicate forms. For instance, the predicate coll satisfies the following rules:

$$\text{coll}(A, B, C) :- \text{coll}(A, C, B),$$

$$\text{coll}(A, B, C) :- \text{coll}(B, A, C).$$

Thus, from coll(A, B, C), we can obtain five 'new' facts: coll(A, C, B), coll (B, A, C), coll(B, C, A), coll(C, A, B), coll(C, B, A). In order to save space and enhance search speed, we represent predicates as canonical forms by assigning an order to the points in a geometry statement. With such an order, predicates can be represented uniquely. Furthermore, by using the canonical form, the above rules are not needed explicitly in the deduction steps. This will reduce the number of rules used in the deduction process.

(2) *Use equivalent classes to represent some predicates*. We may use sequences to represent certain transitive geometric predicates. For instance, the fact that points $A_1, A_2, \ldots, A_n$ are on the same line can be represented by a sequence of points. In predicate form, we need $n(n-1)(n-2)$ different forms like coll(A_i, A_j, A_k) to represent this fact.

(3) *Use representative elements for equivalent classes*. Some predicates are essentially geometric relations about equivalent classes. In that case, we will use a representative element to represent the equivalent class. For instance, the fact

that the line containing points $A_1, \ldots, A_n$ is parallel to the line containing points $B_1, \ldots, B_k$ can be represented by $l_1 \parallel l_2$ if using l_1 and l_2 to represent the two lines. If using predicates para(A_i, A_j, B_k, B_l) to represent this fact, we need $2n(n-1)k(k-1)$ predicates.

We will now give the structure of the database. At the top level, the facts satisfying each predicate are represented by a list of structures defined below.

coll. The structure is a list of points on the same line. Let n be the number of points on the line. We need $n(n-1)(n-2)$ predicate forms to represent this fact.

para. The structure is a pair of line pointers l_1 and l_2, meaning that $l_1 \parallel l_2$. Let l_1 and l_2 contain n_1 and n_2 points, respectively. Then we need $2n_1(n_1-1)n_2(n_2-1)$ predicate forms.

perp. The structure is a pair of line pointers l_1 and l_2, meaning that $l_1 \perp l_2$. Let l_1 and l_2 contain n_1 and n_2 points, respectively. Then we need $2n_1(n_1-1)n_2(n_2-1)$ predicate forms.

eqangle. The structure is a four tuple of line pointers $[l_1, l_2, l_3, l_4]$, meaning that $\angle[l_1, l_2] = \angle[l_3, l_4]$. Let l_i contain n_i points. Then we need $8\prod_{i=1}^{4} n_i(n_i-1)$ predicate forms.

cong. The structure is a list of pairs of points. If the list contains n pairs of points, we need $4n(n-1)$ predicate forms.

eqratio. The structure is a four tuple of cong pointers $[c_1, c_2, c_3, c_4]$, meaning that $c_1/c_2 = c_3/c_4$. Let c_i contain n_i points. Then we need $16\prod_{i=1}^{4} n_i(n_i-1)$ predicate forms.

midp. The structure is a three tuple of points $[M, A, B]$, meaning that M is the midpoint of AB. We need two predicate forms to represent this fact.

circle. The structure is a list of points $[O, P_1, \ldots, P_n]$, meaning that points $P_1, \ldots, P_n$ are on a circle with center O. We need $n(n-1)(n-2)(n-3)$ predicate (cyclic) forms.

simtri (contri). The structure is a list of three-tuple of points. If the list contains n tuples, we need $6n(n-1)$ predicate forms.

DEFINITION 3.1. An element in the database described above is called a *data* or a *fact* of the corresponding predicate. If d is a fact of predicate P, then we also call P the predicate of d. Facts in the traditional sense, such as coll(A, B, C) and para(A, B, C, D) for concrete points A, B, C, and D, are called simple facts.

We also need to know how to trace the deduction route to give a proof for a given fact in the database. For this reason, when updating a fact in the database we first copy the old fact to a new position and update the new one, leaving the old fact intact. For each fact, we also need to save the information such as which lemma and what conditions are used to get this fact. The real structure of a fact in

the structured database is as follows:

[TYPE, LEMMA, COND, DATA, LINK]

where TYPE tells whether this fact is an 'old' one in the sense that it has been updated; LEMMA contains the rules or axioms used to obtain this fact; COND contains the facts used to obtain this fact; DATA is the fact in structured form; LINK points to the next data in the database.

The above structure is not optimized for size control. We could further reduce the size of the database by introducing more complicated structures. But the more complicated the database structure, the more complicated the program needed to handle it. Here we need to consider a tradeoff between the size of the database and the difficulty of implementation. From the following facts, we know that the above structure already reduces the database size significantly.

For the 160 geometry configurations solved by our prover, the average size of the databases is 221 if the above structure is used. With the predicate form, the average size would be 242,117, or one hundred times larger. For many configurations such as Example 6.3, we cannot reach the fixpoint within reasonable time if the above structure is not used.

3.2. GENERATING PROOFS

After a fixpoint is reached, we can generate a proof for each fact in the database. This task is not trivial mainly because we use a database with complicated structures. A straightforward print of the deduction steps is not easy to understand.

We have mentioned in Section 3.1.1 that for each fact in the database, we save the name of the lemma and the conditions used to derive this fact. So a proof step has the following form:

$$(R): \quad C :- P_1, \ldots, P_k,$$

where C is a simple fact (see Definition 3.1) and the P_i are either facts or simple facts. C is always a simple fact for the following reasons: at first step, C is the geometry statement the user want to prove, and at the following steps this is ensured by our tracing procedure described below.

To print the deduction R above, we first print the simple fact C. For each P_i, if P_i is a simple fact, we print it and find the first fact (there might be more than one fact) in the database that implies P_i; if P_i is not a simple fact, we need to find a simple fact that can be deduced from P_i and is relevant to deduction R (see (2) below). After printing deduction R, we repeat the process for each P_i until P_i is in the hypotheses of the geometry statement.

The following three strategies are used in the above tracing process.

(1) Finding missing conditions. Because of the structured database, some conditions are implicitly assumed and hence missed in the database. In the tracing process, we need to bring them back to make the proofs easy to understand. The

most often missed condition is collinear. For instance, in the following deduction in Example 6.1

$$\text{perp}[AB, CG] :- \text{perp}[BC, AF], \angle[BC, AF] = \angle[AB, CH].$$

a condition coll$[GCH]$ is missed. This condition will be added to the proof in this step.

(2) Finding proper conditions. If a predicate is represented by a sequence, its facts can be very large. In the tracing process, we need to find the proper simple facts from them. For instance, the following deduction

$$AB = PQ :- AB = CD, CD = EF = XY = PQ$$

can be simplified to

$$AB = PQ :- AB = CD, PQ = CD.$$

(3) Avoiding redundancies. Before printing a fact P_i in a deduction, we first check whether it has been printed before. If so, we need only to point to the fact printed before. In the machine proof for Example 6.4, Step 4 is used by Steps 2 and 7.

4. Search and Control Strategies

4.1. DATA-BASED SEARCH

Since we want to reach the fixpoint, breadth-first forward chaining is the natural choice for us. The geometric rules are highly complicated: all the predicates are mutually recursive, and most of the rules are not linear. Hence, it is impossible to use some of the more specialized search strategies in [1] that are designed for certain special situations.

Basically speaking, the breadth-first forward chaining search works as follows:

$$\boxed{D_0} \overset{R}{\subset} \boxed{D_1} \overset{R}{\subset} \cdots \overset{R}{\subset} \boxed{D_k} \quad \text{(Fixpoint)}$$

where D_0 is the hypotheses of the geometry statement and R is the rule set. For each rule r in R, apply it to D_0 to obtain new facts. Let D_1 be the union of D_0 and the set of new facts obtained. Repeat the above process for D_1 to obtain D_2, and so on. If at certain step $D_k = D_{k+1}$, we say that a *fixpoint* for D_0 and R is reached. In the case of a deductive database [10] (i.e., when we assume that the rules are Horn clauses without function symbols) the fixpoint can always be reached if the rule set R (i.e., the intensional database) and the initial fact set D_0 (i.e., the extensional database) are finite.

Since D_1 contains D_0 as a subset, the derivation of D_2 from D_1 clearly repeats all the previous deductions used to derive D_1 from D_0. The *semi-naive evaluation* is proposed to solve this problem [1]. The basic idea is that the input fact for at least one of the predicates in the body of the rule must be a new one, namely, a fact

in $D_1 - D_0$. Note that in the semi-naive and all the forward chaining searches, the main loop of process is to search the rule set R. We call such search strategies *rule-based search strategies*. In what follows, we present a *data-based search strategy*, in which we keep a new-fact-list and for each fact d in the list we find and apply all the rules whose bodies contain the predicate of d.

The Data-Based Search Algorithm

Step 1. Set the hypotheses of the statement to be initial new-fact-list and the initial database. While the new-fact-list is not empty, do Step 2.

Step 2. Let d be the first new fact in the list. Delete it from the list, add it to the database, and do Step 3.

Step 3. Let r be a rule whose body contains a predicate P_0 of the fact d. To apply the rule r, we need to instantiate other predicates in r. Since predicate P_0 will be instantiated as the new fact d, other predicates in r need to be instantiated for all the facts in the database. For all the predicate forms of fact d (notice that a fact could have many predicate forms) and for all the facts of the other predicates in r, do Step 4.

Step 4. Apply rule r to obtain a fact d'. If d' is in the database, do nothing. Otherwise, add it to the end of the new-fact-list.

Since the hypothesis set of a geometry statement is finite and we use a finite rule set without function symbols, a fixpoint will always be reached. It is clear that the fixpoint is unique and does not depend on the search strategies and the order of the applications of the rules.

Generally speaking, the breadth-first search is extremely expensive. However, our implementation of it for geometry reasoning performs quite well. This is due to several factors. We use the structured database to reduce the size of the database dramatically and to reduce the number of rules that are built into the structure of the database. Also, our implementation using the C language is specially targeted toward the geometry case.

An interesting fact about the data-based search is that the order of application of the rules is determined by the order of the facts in the new-fact-list: rules are selected according to the type of the fact on the top of the new-fact-list. The user may assign an order of importance to the predicates. Those predicates used more frequently, such as middle points, parallel, and perpendicular, will have higher orders. During the search, the program will sort the new fact list according to this order so that the fact with the highest order will be the first data of the list. In this way, the order of application of rules can be determined automatically.

Another related improvement of the data-based search is to form *combined rules*. Let d be the new fact on the top of the new-fact-list. There are rules like

$$P \wedge f_1 \Rightarrow Q_1, \ldots, P \wedge f_s \Rightarrow Q_s,$$

where P is the predicate of fact d and f_i are conjunctions of other predicates. Then we can form a new combined rule

$$P \wedge (f_1 \Rightarrow Q_1, \vee \cdots \vee, f_s \Rightarrow Q_s).$$

In the new rule we need only search fact d once. Since a fact in our database could be very large (e.g., it might be a sequence of similar triangles), the new rule will clearly save time. We can go further to combine the rules $f_1 \Rightarrow Q_1, \ldots, f_s \Rightarrow Q_s$ recursively to obtain *multiple combined rules*. For instance, the following multiple combined rule is formed from rules R6, R10, and R11:

$$\begin{aligned} \text{cong}(O, A, O, B) \ &\wedge [\Rightarrow \ \angle[OA, AB] = \angle[AB, OB], \\ \text{cong}(U, A, U, B) \ &\wedge [\Rightarrow \ \text{perp}(A, B, O, U), \\ \text{cyclic}(A, B, O, U) \ &\Rightarrow \ \text{perp}(O, A, A, U)]]. \end{aligned}$$

To further improve the search efficiency, we can update and use the database dynamically. In the traditional approaches, the facts are clearly divided into levels, and to obtain a fact at k-level, only the facts at $(k-1)$-level will be used. A better way seems to be that new facts will be stored into the database and used immediately. This makes the program using a combination of the breadth-first search and the depth-first search.

4.2. AVOIDING REDUNDANT DEDUCTIONS

A *deduction is redundant* if it generates a fact that is already in the database. The redundant deduction is a major hurdle for speeding up the search, and eliminating redundancies is proposed as a basic research problem by L. Wos [25]. It is proved that in general cases, the problem of eliminating all redundancies is undecidable [13]. So the best we can do is to design strategies reducing the redundancies.

There are three kinds of redundant deductions. First, repeated use of the same rule to the same fact will generate the same result. These kind of redundancies can be solved by the semi-naive search or our data-based search.

Second, some redundant deductions are logically guaranteed. For example, if a rule r in a rule set R is a logical consequence of $R - \{r\}$, then each fact deduced using rule r will also be deduced by other rules in R. For a given rule set, the problem of obtaining a minimal and logically equivalent rule set is undecidable [22]. Many useful partial methods are given in [2, 13, 22]. Since the rule set used by us is fixed, we try to remove the redundant rules based on our geometric intuition. The *diagram-based standard rules* explained below can be considered special cases of these methods.

Third, two logically irrelevant deductions may give the same fact if the input facts satisfy certain conditions. We call these kind of redundancies the *conditional redundancies*. There seems no general method to control conditional redundancies. In our program, the heuristic (1) explained below is designed for this purpose.

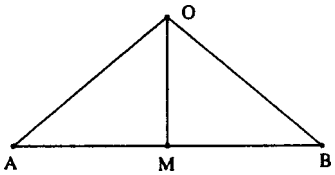

Figure 4.

The following heuristics are used to control redundant deductions in our program:

(1) *Checking Results before Searching*. During the execution of a rule

$$Q :- P_1, \ldots, P_s,$$

if all the variables in Q are instantiated and predicates $P_1, \ldots, P_k$ $(k < s)$ are valid, we have two ways of finishing the deduction. First, we can go on to search for facts of predicates $P_{k+1}, \ldots, P_s$. If they are instantiated and Q is not in the database, we obtain a new fact Q. Second, we can check Q first. If Q is already in the database, nothing is needed to be done. Otherwise we need to check $P_{k+1}, \ldots, P_s$. If the head Q of the rule is in the database, the second way is always faster. Otherwise the step of checking for Q is a waste of time. In our program, we use the second way if the fact for P_1, (i.e., the new fact) is a derived fact, because in that case the chance of redundant deduction is fairly high.

(2) *Avoiding Tautologies.* If the composition of rules is an identity, the successive deductions by these rules will give the same fact as the input. A special but often-encountered case is the composition of a rule and its reversed rule, such as rules R2 and R3. For each deduced fact in our database, the program always remembers the lemma used to deduce it (Section 3.1). Thus it is easy to detect and delete these kinds of redundancy.

(3) *Avoiding Empty Fact*. We call a fact *empty* if no new information can be derived from it. An example of empty fact is contri(O, A, B, O, B, A) if $OA = OB$, because all the information we can get from this fact is already known. We could have many such congruent triangles if there are circles in the geometry statement. For instance, in a circle with center O and passing through points $P_1, \ldots, P_n$, there exist $\frac{n(n-1)}{2}$ empty facts of the form contri$(O, P_i, P_j, O, P_j, P_i)$. In our program, we do not store empty facts in the database to reduce the database size and to reduce redundancies in later steps.

(4) *Diagram Based Standard Forms*. Many rules generating redundancies are related to a diagram. In this case, we can eliminate some rules to avoid redundancies. We use the following example to illustrate how to do this.

In Figure 4, we have five properties: midp(M, A, B), perp(O, M, A, B), cong(O, A, O, B), eqangle(A, O, M, M, O, B), and eqangle(O, A, B, A, B, O).

By rules R6 and R7, cong(O, A, O, B) and eqangle(O, A, B, A, B, O) are equivalent, so we need consider only one of them. The interesting fact about this figure is that any two of the four statements implies the other three. We thus could have twelve rules. These rules clearly lead to many redundant deductions. To avoid this, we choose a set of *standard rules*, say,

S1. cong$(O, A, O, B) :-$ midp(M, A, B), perp(O, M, A, B).

S2. eqangle$(A, O, M, M, O, B) :-$ midp(M, A, B), perp(O, M, A, B).

For the ten rules left, we keep only those whose head is either midp(M, A, B) or perp(O, M, A, B). As a result, only six rules are left.

5. Constructing Auxiliary Points and Skolemization

In logic, constructing new points corresponds to the Skolemization of the existential quantifiers. Even before the first geometry theorem prover was developed, A. Robinson suggested that the auxiliary points and lines needed in a proof can be constructed as elements of the Herbrand universe for the problem [21]. Based on similar ideas, Reiter presented a deductive method that can generate new points [19]. But both of the above ideas are not implemented. It is clear that constructing auxiliary points may lead to infinite geometric objects and prevent the reaching of fixpoints. Also, introducing new points may drastically increase the size of the database. We use two strategies to control the adding of new points to achieve effectiveness.

Our first strategy is to separate the process of adding new points from the process of reaching the fixpoints. The program works precisely as follows. For a geometry theorem, we first find a fixpoint for the corresponding configuration of the theorem without adding auxiliary points. If the conclusion of the statement is already in the database, the program terminates. Otherwise, the program will try to construct an auxiliary point, add the facts related to the auxiliary point to the new-fact-list, and find a new fixpoint. The program will repeat this process until either the conclusion is in the new database or there exist no new auxiliary points.

If the above strategy is not used, we may encounter the unpleasant situation that the conclusion of the statement can be deduced without adding auxiliary points, but in the process of finding the first fixpoint, many auxiliary points will still be constructed. Construction of many irrelevant points may dramatically drag down the searching speed.

Our second strategy is to use two heuristics to control the constructing of too many auxiliary points. (1) After an auxiliary point is added and a new fixpoint is reached, we will check whether new properties about the original diagram are found. If they are, we will keep this auxiliary point; otherwise, the auxiliary point will be deleted. (2) No recursive auxiliary points are allowed. In other words, to construct an auxiliary point, we can use only points occurring in the original statement. The second condition guarantees that the program will terminate.

A rule of constructing auxiliary points is actually the modification of a rule given in Section 2 in which a universal quantifier is changed to an existential quantifier. The following are four rules of constructing auxiliary points. Rules A1 and A2 are related to Rules S1 and R3.

$A1$: [perp(O, M, M, A) and $\angle[XO, MO] = \angle[MO, AO]] \Rightarrow$
$\exists B$[coll(B, A, M) and coll(B, O, X) and cong(O, B, O, A)
and midp(M, A, B)].
$A2$: [$\angle[AP, BP] = \angle[AX, BY]$ and $\neg$coll$(A, B, P)] \Rightarrow$
$\exists Q[\angle[AP, BP] = \angle[AQ, BQ]$ and cyclic$[A, B, P, Q]$.]
$A3$: [midp(M, A, B) and midp$(N, C, D)] \Rightarrow$
$\exists P$[midp(P, A, D), para(P, M, B, D), and para$[P, N, A, C]$.]
$A4$: [cong(O, C, O, D) and perp$(A, B, B, O)] \Rightarrow$
$\exists P$[cong(O, C, O, P), para(P, C, A, B), cong$[B, C, B, P]$.]

The negative statement used in rule A2 is treated similarly to the negative information in Section 2.3. In rules A1 and A2, the new point is introduced as the intersection of two non-parallel lines. In rule A3, the new point is introduced as the midpoint of a line segment. In rule A4, the new point is introduced as the intersection of a line and a circle.

Totally, we have about twenty rules of constructing auxiliary points. The most often used experimental 'methods' of adding auxiliary points are selected as rules. These rules are not complete. Of the 160 geometry configurations solved by our program, 39 of them need the construction of auxiliary points. Two of them are Examples 6.4 and 6.5.

In [8, 11], two constructions are considered: construction of new points and construction of new lines between two existing points. In our program, lines connecting all possible pairs of points will be stored into the database if more than two points are on them or they are used in one of the following predicates: para, perp, and eqangle. In other words, lines connecting any pair of points in the diagram are considered to exist in our program and do not need to be added.

6. Conclusion

6.1. IMPLEMENTATION

The method reported in this paper is actually one of the proving methods implemented in our geometry reasoning system, *Geometry Expert*, which is available via ftp at emcity.cs.twsu.edu: pub/geometry/software/gex_sparc.tar.Z. First, you need to select the 'Deductive Database Method' item in the 'Parameter' menu to use the fixpoint approach. The default proving method of the program is the area method [5]. For a given geometry statement, the program works as follows:

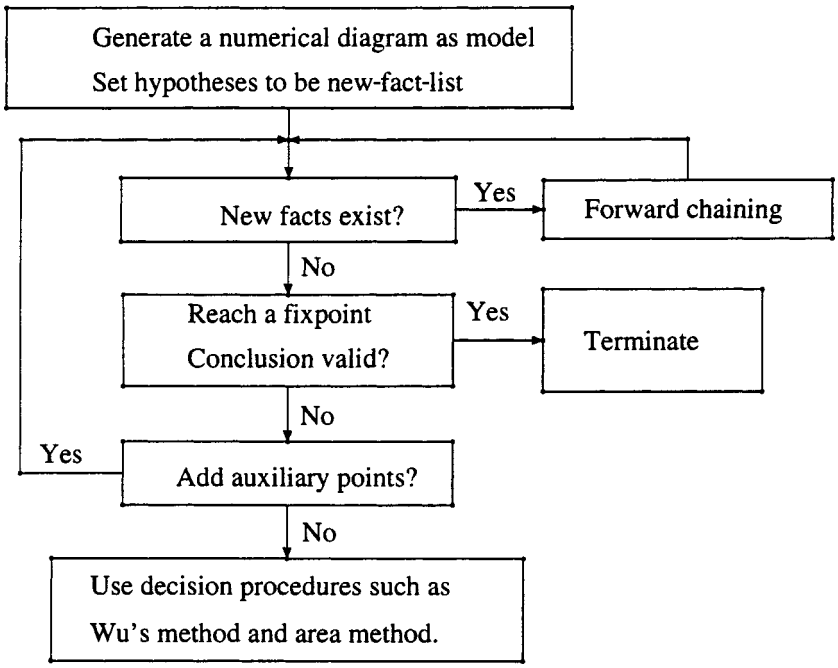

Figure 5.

6.2. APPLICATIONS

The current program has the following applications.

(1) *Automated Theorem Proving and Discovering*. It is clear that the program can be used to prove a given geometry theorem. A more interesting application is to discover 'new' facts about the geometric configuration. Anything obtained in the forward chaining may be looked at as a 'new' result. Our experiments show that our program can discover most of the well-known results and often some unexpected ones. Take the simple configuration (Figure 6) related to the ortho-center theorem as an example. Our program discovered the most often mentioned properties about this configuration: (1) the three altitudes are concurrent and (2) $\angle[EG, CG] = \angle[CG, FG]$. The program also finds 105 essentially different ratios in such a simple configuration! In Example 6.2, the program even discovered some new results.

(2) *Basis for Other Reasoning Methods*. We plan to develop a geometry reasoning system having the merits of both the synthetic and algebraic approaches. For a geometry theorem, the system first forms a database using the synthetic approach. If the conclusion is in the database, then a synthetic proof for it can be produced. Otherwise, the system proves the theorem using the decision procedures such as

the area method [5], and the database may help these decision methods to produce more elegant proofs.

(3) *Geometry Education*. Starting from the hypotheses of a geometry statement, reasoning forward to prove the conclusion is one of the basic proving techniques in geometry. Our program mechanizes this approach and gives it more power. By adding a nice interface, the program can teach a student how to prove a specific theorem with forward chaining or can how to explore interesting properties of a given configuration. For another application, since the current program can generate almost all the properties for a configuration within seconds, a geometry teacher can use it to select exercises for the students or examples for a geometry textbook.

6.3. EXPERIMENT RESULTS AND EXAMPLES

The 160 geometry configurations are mainly from [3, 5]. All the theorems in [17] and the nontrivial theorems in [8, 11] are also included. There are about 600 theorems in [3, 5] that are solved with Wu's method or the area method. We can find fixpoints for each of the 600 theorems, but only about 160 of them can be proved in this way. Some well-known theorems such as Pappus's theorem and Pascal's theorem are beyond the scope of our program, although the fixpoints can be reached for both of them. For Pappus's theorem, the fixpoint is the set of original hypotheses. For Pascal's theorem, the fixpoint contains 89 facts. Therefore, although it might be the most powerful synthetic approach, our deductive database approach is still much less powerful in scope than the algebraic approaches. On the other hand, our approach has the following advantages: (1) it can be used to discover theorems automatically; (2) the proofs produced are more intelligible; and (3) for some theorems (we have two such examples), the deductive database method can produce elegant proofs while no algebraic method can even prove them because exceedingly large amounts of computer memory and computing time are required. Using the geometric rules adopted by us, one can find short proofs for these theorems.

Table I contains the timing and database size statistics for the 160 geometry theorems solved by the program. The timing is collected on a NeXT workstation.

The predicate that has the largest database is eqratio. Without this predicate, the size of the database will reduce significantly. So a heuristic might be: First obtain a fixpoint without considering predicate eqratio; if the conclusion is not in the database, then obtain a new fixpoint including eqratio.

EXAMPLE 6.1 (The Orthocenter Theorem). *Show that the three altitudes of a triangle are concurrent* (*Figure* 6).

The hypotheses (extensional database) are points(A, B, C), coll(E, A, C), perp (B, E, A, C), coll(F, B, C), perp(A, F, B, C) coll(H, A, F), coll(H, B, E), coll (G, A, B), coll(G, C, H).

Table I. Structures for 160 Geometry Theorems

Proving Time (seconds)		Size of Structured DB		Size of Predicate DB	
Time	Theorems	Size	Theorems	Size	Theorems
≤ 0.1	30%	≤ 50	16%	$\leq 10,000$	11%
≤ 1	69%	≤ 100	42%	$\leq 50,000$	43%
≤ 10	94%	≤ 200	66%	$\leq 100,000$	59%
≤ 60	98%	≤ 500	91%	$\leq 1,000,000$	95%
≤ 650	100%	≤ 4021	100%	$\leq 5,041,102$	100%
8.37 (average)		221 (average)		242,117 (average)	

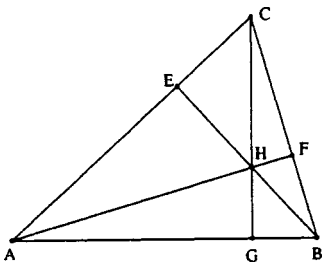

Figure 6.

Reaching the fixpoint costs the program 0.75 second. The size of the fixpoint is 146 if the structured database is used. In predicate form, the size of the fixpoint would be 56,940. The following is the breakdown of the fixpoint into predicates: coll 9 (36 in predicate form), perp 3 (216), cyclic 6 (144), eqangle 19 (29376), simtri 4 (288), eqratio 105 (26880).

The fixpoint contains two of the most often encountered properties of this configuration: perp(C, G, A, B) (the conclusion) and $\angle[GF, GC] = \angle[GC, GE]$. Another amazing fact is that this simple configuration contains 105 nontrivial ratios! We list those ratios involving segment HC below.

$$HC * BE = EC * BA, HC * EA = BA * HE,$$
$$HC * HG = FH * HA = HE * HB,$$
$$HC * AF = EF * AC = BA * CF,$$
$$HC * FB = HB * FE = FH * BA,$$
$$HC * FG = CF * HB = FH * AC,$$
$$HC * BG = EC * HB = FH * BC,$$
$$HC * CG = BC * FC = EC * CA,$$

$$HC * AG = HE * AC = HA * CF,$$
$$HC * EG = HE * BC = HA * EC.$$

The following is the proof for the fact perp(C, G, A, B), which is *automatically* generated by our prover. In the proof (hyp) means that the corresponding fact is from the hypotheses.

The Machine Proof

1. perp[CG, AB] :− (hyp)perp[BC, AF], (hyp)coll[GCH], (2)[BC, AF] = [AB, CH].
2. $\angle[BC, AF] = \angle[AB, CH]$:− (3)$\angle[BC, AB] = \angle[AF, CH]$.
3. $\angle[BC, AB] = \angle[AF, CH]$:− (4)$\angle[BC, AB] = \angle[FE, AC]$, (5)$\angle[AF, CH] = \angle[FE, AC]$.
4. $\angle[BC, AB] = \angle[FE, AC]$:− (hyp)coll[CBF], (hyp)coll[CEA], (6)$\angle[BF, BA] = \angle[EF, EA]$.
5. $\angle[AF, CH] = \angle[FE, AC]$:− (hyp)coll[AHF], (hyp)coll[AEC], (7)$\angle[HF, HC] = \angle[EF, EC]$.
6. $\angle[BF, BA] = \angle[EF, EA]$:− (8)cyclic[$AFBE$].
7. $\angle[HF, HC] = \angle[EF, EC]$:− (9)cyclic[$CFEH$].
8. cyclic[$AFBE$] :− (hyp)perp[FB, FA], (hyp)perp[EB, EA].
9. cyclic[$CFEH$] :− (hyp)perp[FH, FC], (hyp)perp[EH, EC].

EXAMPLE 6.2 (The Five Circle Theorem⋆). *As in Figure* 7, $P_0P_1P_2P_3P_4$ *is a pentagon.* $Q_i = P_{i-1}P_i \cap P_{i+1}P_{i+2}$, $M_i = \text{circle}(Q_{i-1}P_{i-1}P_i) \cap \text{circle}(Q_iP_iP_{i+1})$ (*the subscripts are understood to be* mod 5). *Show that points* M_0, M_1, M_2, M_3, M_4 *are cyclic.*

The fixpoint is reached in 3.89 seconds and contains 541 (220,680 in predicate form) facts. Besides the fact that M_0, M_1, M_2, M_3, and M_4 are cyclic, our program finds the following new result: The following ten groups of lines

$$\{P_{i+1}M_{i+1}, Q_{i-1}M_{i-1}, Q_{i+2}M_{i-2}\}, \{P_{i-1}M_{i-2}, P_iM_{i+1}, Q_{i-1}M_{i+2}\},$$
$$i = 0, 1, 2, 3, 4,$$

are concurrent, and the ten intersection points of them are on the circle determined by M_0, M_1, M_2, M_3, and M_4. In other words, this circle contains 15 points. The three dotted lines in Figure 7 represent one group of concurrent lines.

EXAMPLE 6.3. *In the right triangle ABC,* $\angle A = 90°$; $AH \perp BC$; *S is the midpoint of* AH; $KN \parallel AB$; $PL \parallel AC$; $QM \parallel BC$. *Show that six points* P, Q, K, L, M, N *are on the same circle* (*Figure* 8).

⋆ This problem was proposed in the news group sci.math by Noam D. Elkies of Harvard University in 1992. The theorem was proved by Gerald A. Edgar of Ohio State University with Maple. However, for the general-purpose geometry theorem provers based on the algebraic methods, the proofs require exceedingly large amounts of computer memory, which are currently not available on most computer systems. Wen-Tsün Wu was later able to give a simple synthetic proof. Our program can discover his result totally automatically.

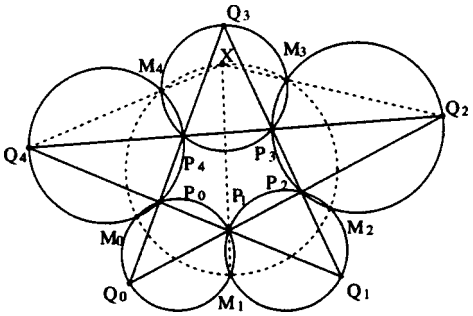

Figure 7.

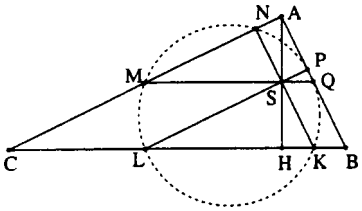

Figure 8.

It takes the program 461.06 seconds to reach the fixpoint which contains 1326 (5,041,102 in predicate form) facts. Without using the structured database, the fixpoint is too big to be reached within reasonable time.

EXAMPLE 6.4. *$ABCD$ is a trapezoid such that $AB \parallel CD$. M and N are the midpoints of AC and BD. Let E be the intersection of MN and BC. Show that E is the midpoint of BC* (*Figure* 9).

This example is originally given in [11] and used in [8, 19]. In [8, 11], an auxiliary point $K = CN \cap AB$ is added by the user. In [19], it is reported that the same auxiliary point will be added based on Skolemization. But the method has not been implemented. Our program solves this problem by adding a different auxiliary point using rule A3: A_0 is the midpoint of AD. With this auxiliary point, the proof of the conclusion seems easier using rules R4 and R5.

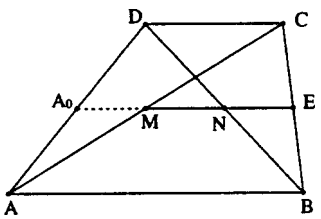

Figure 9.

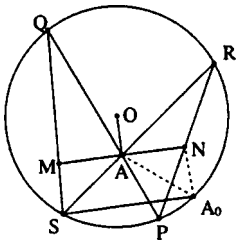

Figure 10.

The Machine Proof
1. midp[E, BC] :– (2)para[CD, EN], (hyp)midp[N, BD].
2. para[CD, EN] :– (3)coll[$ENMA_0$], (4)para[CD, MA_0].
3. coll[$ENMA_0$] :– (hyp)line[MNE], (5)line[MNA_0].
4. para[CD, MA_0] :– (hyp)midp[M, AC], (hyp)midp[A_0, DA].
5. line[MNA_0] :– (6)para[A_0M, A_0N].
6. para[A_0M, A_0N] :– (7)para[A_0M, AB], (8)para[A_0N, AB].
7. para[A_0M, AB] :– (4)para[CD, MA_0], (hyp)para[AB, CD].
8. para[A_0N, AB] :– (hyp)midp[N, BD], (hyp)midp[A_0, DA].

EXAMPLE 6.5 (The Butterfly Theorem). *P, Q, R, and S are on the same circle with center O. A is the intersection of PQ and SR. The line passing through A and perpendicular to OA meets PR and QS in N and M, respectively. Show that A is the midpoint of NM* (*Figure* 10).

The conclusion is not in the first fixpoint. The program automatically adds an auxiliary point A_0 (using rule A4), that is the intersection of the line passing through S and parallel to AN and the circle O. With the point A_0, it takes the program 0.4 second to reach the fixpoint which contains the conclusion. The key steps in the proof are (1) $\angle NAA_0 = \angle SA_0A$, $\angle SA_0A = \angle ASA_0 = \angle RSA_0$, and $\angle RSA_0 = \angle RPA_0$ imply $\angle NAA_0 = \angle NPA_0$; (2) $\angle NAA_0 = \angle NPA_0$ implies cyclic(A, N, P, A_0); (3) cyclic(A, N, P, A_0) implies $\angle AA_0N = \angle APN = \angle QPR = \angle QSR = \angle MSA$; (4) $\angle MSA = \angle AA_0N$, $\angle MAS = \angle NAA_0$, and $AS = AA_0$ imply $\triangle AMS \cong \triangle ANA_0$; (5) $\triangle AMS \cong \triangle ANA_0$ implies $AM = AN$; and (6) $AM = AN$ implies midp(A, M, N). Note that this step uses the technique mentioned in Section 2.2. We need to check whether $M = N$ numerically. Since this is not true, we can deduce midp(A, M, N) from $AM = AN$.

Acknowledgment

We thank Prof. D. Kapur for providing many valuable suggestions on this paper.

Appendix

This appendix contains the geometric deductive rules and the rules for adding auxiliary points used in our program. The meaning of the geometric predicates can be found in Section 2.1. Rules D1–D38 describe basic properties of the geometry predicates, and have been built into the structure of the deductive database (see Section 3.1). We implicitly assume that all points in a rule are different.

D1. coll$(A, C, B) :-$ coll(A, B, C).
D2. coll$(B, A, C) :-$ coll(A, B, C).
D3. coll$(C, D, A) :-$ coll(A, B, C), coll(A, B, D).
D4. para$(A, B, D, C) :-$ para(A, B, C, D).
D5. para$(C, D, A, B) :-$ para(A, B, C, D).
D6. para$(A, B, E, F) :-$ para(A, B, C, D), para(C, D, E, F).
D7. perp$(A, B, D, C) :-$ perp(A, B, C, D).
D8. perp$(C, D, A, B) :-$ perp(A, B, C, D).
D9. para$(A, B, E, F) :-$ perp(A, B, C, D), perp(C, D, E, F).
D10. perp$(A, B, E, F) :-$ para(A, B, C, D), perp(C, D, E, F).
D11. midp$(M, A, B) :-$ midp(M, B, A).
D12. circle$(O, A, B, C) :-$ cong(O, A, O, B), cong(O, A, O, C).
D13. cyclic$(A, B, C, D) :-$
cong(O, A, O, B), cong(O, A, O, C), cong(O, A, O, D).
D14. cyclic$(A, B, D, C) :-$ cyclic(A, B, C, D).
D15. cyclic$(A, C, B, D) :-$ cyclic(A, B, C, D).
D16. cyclic$(B, A, C, D) :-$ cyclic(A, B, C, D).
D17. cyclic$(B, C, D, E) :-$ cyclic(A, B, C, D), cyclic(A, B, C, E).
D18. eqangle$(B, A, C, D, P, Q, U, V) :-$ eqangle(A, B, C, D, P, Q, U, V).
D19. eqangle$(C, D, A, B, U, V, P, Q) :-$ eqangle(A, B, C, D, P, Q, U, V).
D20. eqangle$(P, Q, U, V, A, B, C, D) :-$ eqangle(A, B, C, D, P, Q, U, V).
D21. eqangle$(A, B, P, Q, C, D, U, V) :-$ eqangle(A, B, C, D, P, Q, U, V).
D22. eqangle$(A, B, C, D, E, F, G, H) :-$
eqangle(A, B, C, D, P, Q, U, V), eqangle(P, Q, U, V, E, F, G, H).
D23. cong$(A, B, D, C) :-$ cong(A, B, C, D).
D24. cong$(C, D, A, B) :-$ cong(A, B, C, D).
D25. cong$(A, B, E, F) :-$ cong(A, B, C, D), cong(C, D, E, F).
D26. eqratio$(B, A, C, D, P, Q, U, V) :-$ eqratio(A, B, C, D, P, Q, U, V).
D27. eqratio$(C, D, A, B, U, V, P, Q) :-$ eqratio(A, B, C, D, P, Q, U, V).
D28. eqratio$(P, Q, U, V, A, B, C, D) :-$ eqratio(A, B, C, D, P, Q, U, V).
D29. eqratio$(A, B, P, Q, C, D, U, V) :-$ eqratio(A, B, C, D, P, Q, U, V).
D30. eqratio$(A, B, C, D, E, F, G, H) :-$
eqratio(A, B, C, D, P, Q, U, V), eqratio(P, Q, U, V, E, F, G, H).
D31. simtri$(A, B, C, P, Q, R) :-$ simtri(A, C, B, P, R, Q).
D32. simtri$(A, B, C, P, Q, R) :-$ simtri(B, A, C, Q, P, R).
D33. simtri$(A, B, C, P, Q, R) :-$ simtri(P, Q, R, A, B, C).

D34. simtri(A, B, C, P, Q, R) :–
simtri(A, B, C, E, F, G), simtri(E, F, G, P, Q, R).
D35. contri(A, B, C, P, Q, R) :– contri(A, C, B, P, R, Q).
D36. contri(A, B, C, P, Q, R) :– contri(B, A, C, Q, P, R).
D37. contri(A, B, C, P, Q, R) :– contri(P, Q, R, A, B, C).
D38. contri(A, B, C, P, Q, R) :–
contri(A, B, C, E, F, G), contri(E, F, G, P, Q, R).
D39. para(A, B, C, D) :– eqangle(A, B, P, Q, C, D, P, Q).
D40. eqangle(A, B, P, Q, C, D, P, Q) :– para(A, B, C, D).
D41. eqangle(P, A, P, B, Q, A, Q, B) :– cyclic(A, B, P, Q).
D42. cyclic(A, B, P, Q) :–
eqangle(P, A, P, B, Q, A, Q, B), ¬ coll(P, Q, A, B).
D43. cong(A, B, P, Q) :–
cyclic(A, B, C, P, Q, R), eqangle(C, A, C, B, R, P, R, Q).
D44. para(E, F, B, C) :– midp(E, A, B), midp(F, A, C).
D45. midp(F, A, C) :– midp(E, A, B), para(E, F, B, C), coll(F, A, C).
D46. eqangle(O, A, A, B, A, B, O, B) :– cong(O, A, O, B).
D47. cong(O, A, O, B) :– eqangle(O, A, A, B, A, B, O, B), ¬ coll(O, A, B).
D48. eqangle(A, X, A, B, C, A, C, B) :– circle(O, A, B, C), perp(O, A, A, X).
D49. perp(O, A, A, X) :– circle(O, A, B, C), eqangle(A, X, A, B, C, A, C, B).
D50. eqangle(A, B, A, C, O, B, O, M) :– circle(O, A, B, C), midp(M, B, C).
D51. midp(M, B, C) :–
circle(O, A, B, C), coll(M, B, C), eqangle(A, B, A, C, O, B, O, M).
D52. cong(A, M, B, M) :– perp(A, B, B, C), midp(M, A, C).
D53. perp(A, B, B, C) :– circle(O, A, B, C), coll(O, A, C).
D54. eqangle(A, D, C, D, C, D, C, B) :– cyclic(A, B, C, D), para(A, B, C, D).
D55. cong(O, A, O, B) :– midp(M, A, B), perp(O, M, A, B).
D56. perp(A, B, P, Q) :– cong(A, P, B, P), cong(A, Q, B, Q).
D57. perp(P, A, A, Q) :–
cong(A, P, B, P), cong(A, Q, B, Q), cyclic(A, B, P, Q).
D58. simtri(A, B, C, P, Q, R) :–
eqangle[A, B, B, C, P, Q, Q, R], eqangle[A, C, B, C, P, R, Q, R],
¬ coll(A, B, C).
D59. eqratio(A, B, A, C, P, Q, P, R) :– simtri(A, B, C, P, Q, R).
D60. eqangle(A, B, B, C, P, Q, Q, R) :– simtri(A, B, C, P, Q, R).
D61. contri(A, B, C, P, Q, R) :– simtri(A, B, C, P, Q, R), $AB = PQ$.
D62. cong(A, B, P, Q) :– contri(A, B, C, P, Q, R).
D63. para(A, C, B, D) :– midp(M, A, B), midp(M, C, D).
D64. midp(M, C, D) :– midp(M, A, B), para(A, C, B, D), para(A, D, B, C).
D65. eqratio(O, A, A, C, O, B, B, D) :–
para(A, B, C, D), coll(O, A, C), coll(O, B, D).
D66. coll(A, B, C) :– para(A, B, A, C).
D67. midp(A, B, C) :– cong(A, B, A, C), coll(A, B, C).

D68. $\text{cong}(A, B, A, C) :- \text{midp}(A, B, C)$.
D69. $\text{coll}(A, B, C) :- \text{midp}(A, B, C)$.
D70. $\text{eqratio}(M, A, A, B, N, C, C, D) :- \text{midp}(M, A, B), \text{midp}(N, C, D)$.
D71. $\text{perp}(A, B, C, D) :- \text{eqangle}(A, B, C, D, C, D, A, B), \neg\, \text{para}(A, B, C, D)$.
D72. $\text{para}(A, B, C, D) :- \text{eqangle}(A, B, C, D, C, D, A, B), \neg\, \text{perp}(A, B, C, D)$.
D73. $\text{para}(A, B, C, D) :- \text{eqangle}(A, B, C, D, P, Q, U, V), \text{para}(P, Q, U, V)$.
D74. $\text{perp}(A, B, C, D) :- \text{eqangle}(A, B, C, D, P, Q, U, V), \text{perp}(P, Q, U, V)$.
D75. $\text{cong}(A, B, C, D) :- \text{eqratio}(A, B, C, D, P, Q, U, V), \text{cong}(P, Q, U, V)$.

The following rules are used to add auxiliary points in our program.

X1. $[\text{perp}(O, M, M, A), \text{eqangle}(X, O, M, O, M, O, A, O)] \Rightarrow$
$\exists B\, [\text{coll}(B, A, M), \text{coll}(B, O, X)]$.
X2. $[\text{cong}(O, A, O, B), \text{eqangle}(A, O, O, X, O, X, O, B)] \Rightarrow$
$\exists M\, [\text{coll}(B, A, M), \text{coll}(M, O, X)]$.
X3. $[\text{perp}(O, X, A, B), \text{eqangle}(A, O, O, X, O, X, O, B)] \Rightarrow$
$\exists M\, [\text{coll}(B, A, M), \text{coll}(M, O, X)]$.
X4. $[\text{perp}(O, X, A, B), \text{cong}(O, A, O, B)] \Rightarrow$
$\exists M\, [\text{coll}(B, A, M), \text{coll}(M, O, X)]$.
X5. $[\text{eqangle}(A, P, B, P, A, X, B, Y), \neg\text{coll}(A, B, P)] \Rightarrow$
$\exists Q\, [\text{eqangle}(A, P, B, P, A, Q, B, Q), \text{cyclic}[X, B, P, Q]]$.
X6. $[\text{midp}(M, A, B), \text{midp}(N, C, D)] \Rightarrow$
$\exists P\, [\text{midp}(P, A, D), \text{para}(P, M, B, D), \text{para}[P, N, A, C]]$.
X7. $[\text{midp}(M, A, B), \text{midp}(N, C, D), \text{coll}(C, A, B), \text{coll}(D, A, B), \text{point}(Q)]$
$\Rightarrow \exists P\, [\text{midp}(P, A, Q)]$.
X8. $[\text{midp}(M, A, B), \text{para}(A, P, R, M), \text{para}(A, P, B, Q), \text{coll}(P, Q, R)] \Rightarrow$
$\exists X\, [\text{coll}(X, A, Q), \text{coll}(X, M, R)]$.
X9. $[\text{cong}(O, C, O, D), \text{perp}(A, B, B, O)] \Rightarrow$
$\exists P\, [\text{cong}(O, C, O, P), \text{para}(P, C, A, B), \text{cong}[B, C, B, P]]$.
X10. $[\text{perp}(A, H, B, C), \text{perp}(B, H, A, C)] \Rightarrow$
$\exists P, Q\, [\text{coll}(P, C, B), \text{perp}(A, P, C, B), \text{coll}(Q, C, A), \text{perp}(B, Q, C, A)]$.
X11. $[\text{circle}(O, A, B, C)] \Rightarrow \exists P\, [\text{perp}(P, A, A, O)]$.
X12. $[\text{circle}(M, A, B, C), \text{cong}(M, A, M, D), \text{cong}(N, A, N, B)], M \neq N] \Rightarrow$
$\exists P, Q\, [\text{coll}(P, A, C), \text{cong}(P, N, N, A), \text{coll}(Q, B, D), \text{cong}(Q, N, N, A)]$.
X13. $[\text{cyclic}(A, B, C, D), \text{para}(A, B, C, D)], \text{midp}(M, A, B) \Rightarrow$
$\exists O\, [\text{circle}(O, A, B, C)]$.
X14. $[\text{perp}(A, C, C, B), \text{cyclic}(A, B, C, D)] \Rightarrow \exists O\, [\text{circle}(O, A, B, C)]$.
X15. $[\text{perp}(A, C, C, B), \text{coll}(B, E, F)] \Rightarrow \exists P\, [\text{coll}(P, E, F), \text{perp}(P, A, E, F)]$.
X16. $[\text{perp}(A, B, A, C), \text{perp}(C, A, C, D), \text{midp}(M, B, D)] \Rightarrow$
$\exists P\, [\text{midp}(P, A, C)]$.
X17. $[\text{cong}(O, A, O, B), \text{perp}(A, O, O, B)] \Rightarrow$
$\exists C\, [\text{coll}(A, O, C), \text{cong}(O, A, O, C)]$.
X18. $[\text{para}(A, B, C, D), \text{coll}(P, A, C)], \text{coll}(P, B, D), \text{coll}(Q, A, B)] \Rightarrow$
$\exists R\, [\text{coll}(P, Q, R), \text{coll}(R, C, D)]$.

References

1. Bancilhon, F. and Ramakrishnan, R.: An amateur's introduction to recursive query processing strategies, in C. Zanioilo (ed.), *Proc. of ACM SIGMOD Conference*, 1986, pp. 16–52.
2. Buntine, W.: Generalized subsumption and its application to induction and redundancy, *Artif. Intell.* **36**(2) (1988), 149–179.
3. Chou, S. C.: *Mechanical Geometry Theorem Proving*, D. Reidel Publishing Company, Dordrecht, Netherlands, 1988.
4. Chou, S. C. and Gao, X. S.: Mechanical formula derivation in elementary geometries, in *Proc. ISSAC-90*, ACM, New York, 1990, pp. 265–270.
5. Chou, S. C., Gao, X. S. and Zhang, J. Z.: *Machine Proofs in Geometry*, World Scientific, 1994.
6. Chou, S. C., Gao, X. S. and Zhang, J. Z.: Automated generation of readable proofs with geometric invariants, I: Multiple and shortest proof generation, *J. Automated Reasoning* **17**(3) (1996), 325–347.
7. Chou, S. C., Gao, X. S. and Zhang, J. Z.: Automated generation of readable proofs with geometric invariants, II: Proving theorems with full-angles, WSUCS-94-3, CS Dept, Wichita State University, March 1994, *J. Automated Reasoning* **17**(3) (1996), 349–370.
8. Coelho, H. and Pereira, L. M.: Automated reasoning in geometry theorem proving with prolog, *J. Automated Reasoning* **2** (1986), 329–390.
9. Bulmer, M. and Fearnley-Sander, D.: The kinds of truth of geometry theorems, Preprint.
10. Gallaire, H., Minker, J. and Nicola, J. M.: Logic and databases: A deductive approach, *ACM Comput. Surveys* **16**(2) (1984), 153–185.
11. Gerlentner, H., Hanson, J. R. and Loveland, D. W.: Empirical explorations of the geometry-theorem proving machine, in *Proc. West. Joint Computer Conf.*, 1960, pp. 143–147.
12. Havel, T.: The use of distance as coordinates in computer-aided proofs of theorems in Euclidean geometry, IMA Preprint, No. 389, University of Minnesota, 1988.
13. Helm, R.: On the elimination of redundant derivations during execution, in *Proc. of 1990 North American Conference on Logic Programming*, The MIT Press, 1990, pp. 551–568.
14. Kapur, D.: Geometry theorem proving using Hilbert's nullstellensatz, in *Proc. SYMSAC'86*, Waterloo, 1986, pp. 202–208.
15. Koedinger, K. R. and Anderson, J. R.: Abstract planning and perceptual chunks: Elements of expertise in geometry, *Cognitive Science* **14** (1990), 511–550.
16. Hong-Bo, Li and Minteh, Cheng: Clifford algebraic reduction method for automated theorem proving in differential geometry, *J. Automated Reasoning* **21**(1) (1998), 1–21.
17. Nevins, A. J.: Plane geometry theorem proving using forward chaining, *Artif. Intell.* **6** (1975), 1–23.
18. Recio, T. and Velex-Melon, M. P.: Automatic discovery of theorems in elementary geometry, Preprint, 1997.
19. Reiter, R.: A semantically guided deductive system for automatic theorem proving, *IEEE Trans. on Computers* **C-25**(4) (1976), 328–334.
20. Reiter, R.: On the closed world data bases, in H. Gallaire and J. Minker (eds), *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 55–76.
21. Robinson, A.: Proving a theorem (as done by Man, Logician, or Machine), in J. Siekmann and G. Wrightson (eds), *Automation of Reasoning*, Springer-Verlag, 1983, pp. 74–78.
22. Sagiv, Y.: Optimizing datalog programs, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kauffmann, 1988, pp. 659–698.
23. Wang, D. M.: Reasoning about geometric problems using an elimination method, in J. Pfalzgraf and D. M. Wang (eds), *Automated Practical Reasoning*, Springer-Verlag, 1995, pp. 148–185.
24. White, N. L. and Mcmillan, T.: Cayley factorization, in *Proc. ISSAC'88*, ACM Press, 1988, pp. 4–8.

25. Wos, L.: *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
26. Wu Wen-tsün: *Basic Principles of Mechanical Theorem Proving in Geometries, Volume I: Part of Elementary Geometries*, Science Press, Beijing (in Chinese), 1984. English version, Springer-Verlag, 1993.