GEOTHER 1.1: Handling and Proving Geometric Theorems Automatically

Dongming Wang

GEOTHER 1.1: Handling and Proving Geometric Theorems Automatically

Dongming Wang

Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie – CNRS
4 place Jussieu, F-75252 Paris Cedex 05, France

Abstract. GEOTHER provides an environment for handling and proving theorems in geometry automatically. In this environment, geometric theorems are represented by means of predicate specifications. Several functions are implemented that allow one to translate the specification of a geometric theorem into English and Chinese statements, into algebraic expressions, and into logic formulas automatically. Geometric diagrams can also be drawn automatically from the predicate specification, and the drawn diagrams may be modified and animated with mouse click and dragging. Five algebraic provers based on Wu's method of characteristic sets, the Gröbner basis method, and other triangularization techniques are available for proving such theorems in elementary (and differential) geometry. Geometric meanings of the produced algebraic nondegeneracy conditions can be interpreted automatically, in most cases. PostScript and HTML files can be generated, also automatically, to document the manipulation and machine proof of the theorem. This paper presents these capabilities of GEOTHER, addresses some implementation issues, and reports on the performance of GEOTHER's algebraic provers.

1 Introduction

We refer to the proceedings of the first three International Workshops on Automated Deduction in Geometry (published as LNAI 1360, 1669, and 2061 by Springer-Verlag in 1997, 1999, and 2001 respectively) and the Bibliography on Geometric Reasoning (`http://calfor.lip6.fr/~wang/GRBib`) for the current state-of-the-art on automated theorem proving in geometry. The construction of theorem provers has been a common practice along with the development of effective algorithms on the subject. The GEOTHER environment described in this paper is the outcome of the author's practice for more than a decade. An early version of it was ready for demonstration in 1991, and in 1996 was published a short description of the enhanced version GEOTHER 1.0 [8]. The current version GEOTHER 1.1 provides a user-friendly environment for handling and proving theorems in geometry automatically. In this environment, geometric theorems are represented by means of predicate specifications. Several functions are implemented that allow one to translate the specification of a geometric theorem into English and Chinese statements, into algebraic expressions, and into logic formulas automatically. Geometric diagrams can also be drawn automatically from the predicate specification, and the drawn diagrams may be modified and animated with mouse click and dragging. Five algebraic provers based on Wu's method of characteristic sets, the

Gröbner basis method, and other triangularization techniques are available for proving such theorems in elementary (and differential) geometry. Geometric meanings of the produced algebraic nondegeneracy conditions can be interpreted automatically, in most cases. PostScript and HTML files can be generated, also automatically, to document the manipulation and machine proof of the theorem.

The majority of GEOTHER code has been written as Maple programs, and one can use GEOTHER as a standard Maple package. Some of the GEOTHER functions need external programs written in Java (and previously in C) and interact with the operating system. This concerns in particular the functions for automatic generation of diagrams and documents and the graphic interface, which might not work properly under certain operating systems and Java installations. GEOTHER has been included as an application module in the author's Epsilon library [10] which will be made available publicly in early 2003. The reader will find more information about GEOTHER from `http://calfor.lip6.fr/~wang/GEOTHER`.

There are several similar geometric theorem provers implemented on the basis of algebraic methods. We refer to [1–3] for descriptions of such provers. As a distinct feature of it compared to other provers, GEOTHER is designed not only for proving geometric theorems but also for handling such theorems automatically. Our design and full implementation of new algorithms for (irreducible) triangular decomposition make GEOTHER's proof engine also more efficient and complete. In this paper we describe the capabilities of GEOTHER, discuss some implementation strategies, and report experimental data on the performance of GEOTHER's algebraic provers.

2 Specification of Geometric Theorems

We recall the following specification of Simson's theorem:

```
Simson := Theorem(
          [arbitrary(A,B,C), oncircle(A,B,C,D), perpfoot(D,P,A,B,P),
           perpfoot(D,Q,A,C,Q), perpfoot(D,R,B,C,R)],
           collinear(P,Q,R), [x5, x6, x7, x8, x9] );
```

This is a typical example of predicate specification, which will be used throughout the paper for illustration. In general, a geometric theorem specified in GEOTHER has the following form

```
T := Theorem(H,C,X)
```

where `Theorem` is a predicate specially reserved, `T` is the name, `H` the hypothesis and `C` the conclusion of the theorem, and the optional `X` is a list of *dependent* variables. The hypothesis `H` is a list or set of geometric predicates or polynomials, while the conclusion `C` may be a single geometric predicate or polynomial, or a list or set of predicates or polynomials. Geometric predicates are usually of the form

```
Relation(A,B,C, ... )
```

which declares a geometric relation among the objects `A`, `B`, `C`, ... For example, `oncircle(A,B,C,D)` declares that "`the point D is on the circumcircle of the`

`triangle ABC` and `collinear(P,Q,R)` declares that "`the three points P, Q and R are collinear`."

Most of the arguments to the geometric predicates in GEOTHER are points in the plane. In order to perform translation, drawing and proving, coordinates have to be assigned to these points, so that geometric problems may be solved by using algebraic techniques. The assignment of coordinates can be done either manually by the function `Let` or automatically by the function `Coordinate`.

2.1 `Let` and `Coordinate`

– Inputted a sequence `T` of equalities $P_1 = [x_1, y_1], \ldots, P_n = [x_n, y_n]$, `Let(T)` assigns the coordinates (x_i, y_i) to each point P_i for $1 \leq i \leq n$.

For instance, the coordinates of the points in Simson's theorem may be assigned by

```
Let(A = [-x1,0], B = [x1,0], C = [x2,x3], D = [x4,x5], P = [x4,0],
    Q = [x6,x7], R = [x8,x9]):
```

A point may be *free* if both of its coordinates are free parameters, or *semi-free* if one coordinate is free and the other is a dependent variable (constrained by the geometric condition), or *dependent* if both coordinates are dependent variables. All the dependent coordinates listed according to their order of introduction are supplied as the third argument `X` to `Theorem`.

– Inputted the predicate specification `T` of a geometric theorem, `Coordinate(T)` reassigns the coordinates of the points appearing in `T` and returns a (new) specification of the same theorem, in which only the third argument (i.e., the list of variables) is modified.

For example, `Coordinate(Simson)` reassigns the coordinates

```
[u1,0], [u2,0], [0,v1], [x1,y1], [x2,y2], [x3,y3], [x4,y4]
```

respectively to the points `A`, `B`, `C`, `D`, `P`, `Q`, `R` in Simson's theorem. The output specification is the same as `Simson` but the third argument is replaced by `[x1, y1, x2, y2, x3, y3, x4, y4]`.

2.2 Algebraic Specification

There are a (limited) number of geometric predicates available in GEOTHER, which are implemented mainly for specifying theorems of equality type in plane Euclidean geometry. The user may add new predicates to the environment if he or she looks into the GEOTHER code and figures out how such predicates may be implemented (which in fact is rather easy). If the user wishes to specify a geometric theorem but cannot find applicable predicates to express the involved geometric relations, he or she may transform the geometric relations (for the hypothesis and conclusion) into algebraic equations manually and provide the set of hypothesis-polynomials to `H` and the conclusion-polynomial or the set of conclusion-polynomials to `C`. For example, the following is an algebraic specification of a theorem in solid geometry:

```
Solid := Theorem( {2*u1*x7-u1*(u2+u4),
          2*(u2-u1)*x7+2*u3*x8-u4*(u2-u1)-u3*u5,
          2*(u4-u2)*x7+2*(u5-u3)*x8+2*u6*x9-u1*(u4-u2),
          (2*u2-u1)*x11-2*u3*x10+u1*u3,
          (u2-2*u1)*x11-u3*x10+u1*u3,
          4*x12-3*x10-u4, 4*x13-3*x11-u5, 4*x14-u6,
          2*x15-u1, 2*u2*x15+2*u3*x16-u2^2-u3^2,
          2*u4*x15+2*u5*x16+2*u6*x17-u4^2-u5^2-u6^2},
         {x13*x15+x7*x16+x8*x12-x7*x13-x8*x15-x12*x16,
          x9*x12+x14*x15+x7*x17-x7*x14-x9*x15-x12*x17},
         [x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17] );
```

Geometric theorems with algebraic specifications may be proved by any of the algebraic provers as well. However, English and Chinese translation, automated diagram generation, and automated interpretation of nondegeneracy conditions do not work for algebraic specifications.

3 Basic Translations

The predicate specification (together with the optional assignment of coordinates by `Let`) is all what is needed for handling and proving the theorem.

3.1 `English` and `Chinese`

– Inputted the predicate specification `T` of a geometric theorem or a geometric predicate `T` with arguments, `English(T)` translates `T` into an English statement.

For example, `English(Simson)` yields

```
Theorem: If the points A, B, and C are arbitrary, the point D is on
the circumcircle of the triangle ABC, P is the perpendicular foot of
the line DP to the line AB, Q is the perpendicular foot of the line
DQ to the line AC, and R is the perpendicular foot of the line DR to
the line BC, then the three points P, Q and R are collinear.
```

and `English(perpfoot(D, P, A, B, P))` yields:

```
P is the perpendicular foot of the line DP to the line AB
```

– Inputted the predicate specification `T` of a geometric theorem or a geometric predicate `T` with arguments, `Chinese(T)` translates `T` into a Chinese statement.

For example, `Chinese(Simson)` yields

定理：设A、B、C为任意点，D点位于三角形ABC的外接圆上，P点为直线DP对于直线AB的垂足，Q点为直线DQ对于直线AC的垂足，且R点为直线DR对于直线BC的垂足．则P、Q、R三点共线．

and `Chinese(oncircle(A, B, C, D))` yields:

D点位于三角形ABC的外接圆上

3.2 `Algebraic` and `Logic`

– Inputted the predicate specification `T` of a geometric theorem, or a geometric predicate `T` with arguments, or a sequence `T` of points occurring in the current theorem loaded to the GEOTHER session, `Algebraic(T)` translates `T` into an algebraic specification of the theorem or into one or several algebraic expressions, or prints out the coordinates of the points in `T`.

The output algebraic specification has the same form as the predicate specification `T`, but the geometric predicates are replaced by their corresponding polynomials (or polynomial inequations). Usually, a polynomial means a polynomial equation (i.e., "`= 0`" is omitted), and a polynomial inequation is represented by means of "`<> 0`." For example, `Algebraic(Simson)` yields

```
Theorem([2 x1 (x1  x3 - x1  x5 - x5  x3 + x5 x2  - x4  x3 + x5 x3 ),
    x6 x2 + x6 x1 - x4 x2 - x4 x1 + x3 x7 - x3 x5,
    -x3 x6 - x3 x1 + x2 x7 + x1 x7,
    x8 x2 - x8 x1 - x4 x2 + x4 x1 + x3 x9 - x3 x5,
    -x3 x8 + x3 x1 + x2 x9 - x1 x9], -x7 x8 + x7 x4 + x6 x9 - x9 x4,
    [x5, x6, x7, x8, x9])
```

`Algebraic(A,B,C,D,P,Q,R)` prints

```
 [-x1, 0], [x1, 0], [x2, x3], [x4, x5], [x4, 0], [x6, x7], [x8, x9]
```

and `Algebraic(not collinear(P,Q,R))` yields:

```
-x7 x8 + x7 x4 + x6 x9 - x9 x4 <> 0
```

– Inputted the specification `T` of a geometric theorem, `Logic(T)` translates `T` into a logic formula.

For instance, the output of `Logic(Simson)` is:

```
(Any D A R B C P Q) [ oncircle(A,B,C,D) /\ perpfoot(D,P,A,B,P) /\
perpfoot(D,Q,A,C,Q) /\ perpfoot(D,R,B,C,R) ==> collinear(P,Q,R) ]
```

The translations explained in this section are simple and straightforward. A more complicated translation discussed in Sect. 6.1 is to interpret algebraic nondegeneracy conditions geometrically.

4 Proving Geometric Theorems Automatically

The five provers presented in this section are the algebraic proof engine of GEOTHER. They are implemented on the basis of some sophisticated elimination algorithms using characteristic sets, triangular decompositions, and Gröbner bases. In fact, the main subroutines of these functions are from the Epsilon modules `CharSets` and `TriSys` and the Maple built-in `Groebner` package. The reader is pointed to the references given below in which the proving methods underlying our implementation are described.

4.1 Wprover

– Inputted the specification T of a geometric theorem, Wprover(T) proves or disproves T, with subsidiary conditions provided.

Wprover is essentially an implementation of Wu's method as described in [12]. The following is part of a machine proof produced by this prover for the butterfly theorem.

```
GEOTHER> Wprover(Butterfly);

Theorem: If the points A, B, and C are arbitrary, the point O is the
circumcenter of the triangle ABC, the point D is on the circumcircle of
the triangle ABC, the two lines AB and CD intersect at H, the point E
is on the line AC, the line OH is perpendicular to the line EH, and the
two lines EH and BD intersect at F, then H is the midpoint of E and F.

Proof:

char set:   [2 x4 1]   [6 x5 1]   [6 x7 2]   [4 x8 1]   [5 x9 1]
            [2 x10 1]   [8 x11 1]   [4 x12 1]
pseudo-remainder:  [3 x11 1] = -x11 + ... (2 terms)
pseudo-remainder:  [9 x10 1] = x1*x8*x10 + ... (8 terms)
pseudo-remainder:  [9 x9 2] = -x2*x7*x9^2 + ... (8 terms)
pseudo-remainder:  [10 x8 3] = x2^2*x8^3*x7*x5*x3 + ... (9 terms)
pseudo-remainder:  [36 x7 4] = -x7^4*x2^5*x5*x3 + ... (35 terms)
pseudo-remainder:  [22 x5 2] = -2*x3^2*x1*x2^7*x5^2 + ... (21 terms)
pseudo-remainder:  [37 x4 2] = -4*x4^2*x2^9*x1 + ... (36 terms)
pseudo-remainder:  [37 x5 2] = -2*x2^9*x3*x5^2 + ... (36 terms)
pseudo-remainder:  [61 x4 2] = -4*x4^2*x2^11 + ... (60 terms)
pseudo-remainder:  [22 x5 2] = 4*x2^6*x3^3*x5^2 + ... (21 terms)
pseudo-remainder:  [37 x4 2] = 12*x4^2*x2^8*x3 + ... (36 terms)
pseudo-remainder:  [30 x5 2] = 2*x3^2*x2^7*x5^2 + ... (29 terms)
pseudo-remainder:  [62 x4 2] = 4*x4^2*x2^9 + ... (61 terms)

    . . . . . .

pseudo-remainder:  [82 x5 1] = 2*x3^2*x2^11*x5 + ... (81 terms)
pseudo-remainder:  [62 x4 2] = 2*x3*x2^11*x4^2 + ... (61 terms)

  The theorem is true under the following subsidiary conditions:

.   the three points A, B and C are not collinear
.   the line AB is not parallel to the line CD
.   the line EH is not parallel to the line BD
.   the line AC is not perpendicular to the line AB
.   the line DB is not perpendicular to the line AB
.   the line OH is not perpendicular to the line CA

QED.
```

The geometric interpretation of algebraic nondegeneracy conditions is done by using the function `Generic` (see Sect. 6.1).

4.2 `Gprover`

– Inputted the specification `T` of a geometric theorem, `Gprover(T)` or `Gprover(T, Kapur)` proves `T` (without providing subsidiary conditions) or reports that it cannot confirm the theorem.

`Gprover` is an implementation of Kutzler–Stifter's method as described in [5] and Kapur's method as described in [4], both based on Gröbner bases. The former is used if the optional second argument is omitted or given as `KS`, and the latter is used if the second argument is given as `Kapur`.

4.3 `GCprover`

– Inputted the specification `T` of a geometric theorem, `GCprover(T)` proves `T`, with subsidiary conditions provided, or reports that it cannot confirm the theorem.

`GCprover` is an implementation of the method proposed in [9]. It works by first computing a Gröbner basis plus normal-form reduction (over the ground field) according to Kutzler–Stifter's approach [5] and, if this step fails, then taking a quasi-basic set of the computed Gröbner basis and performing pseudo-division. In this way, subsidiary conditions may be provided explicitly.

4.4 `Tprover`

– Inputted the specification `T` of a geometric theorem, `Tprover(T)` proves or disproves `T`, with subsidiary conditions provided.

`Tprover` implements the method using zero decomposition proposed by the author in [6]. The system of hypothesis-polynomials (for both equations and inequations) are fully decomposed into (irreducible) triangular systems. The theorem is proved to be true or false for all the components including degenerate ones, so the provided subsidiary conditions are minimized (see Sect. 7.5).

4.5 `Dprover`

– Inputted the algebraic specification `T` of a (differential geometry) theorem in the local theory of curves, `Dprover(T)` proves or disproves `T`, with algebraic subsidiary conditions provided.

`Dprover` implements the method using ordinary differential zero decomposition proposed in [7]. The system of hypothesis-differential-polynomials (for equations and inequations) are fully decomposed into (irreducible) differential triangular systems. The theorem is proved to be true or false for all the components including degenerate ones, thus allowing the subsidiary conditions to be minimized.

5 Automated Generation of Diagrams and Documents

5.1 `Geometric`

– Inputted the predicate specifi cation `T` of a geometric theorem, `Geometric(T)` generates one or several diagrams for `T`.

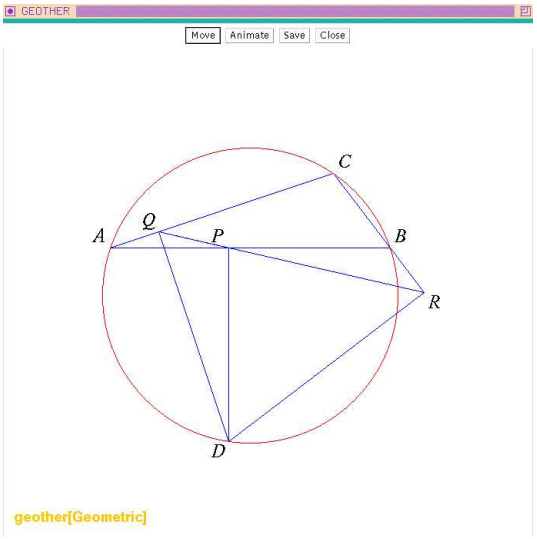

Figure 1. Output of `Geometric`

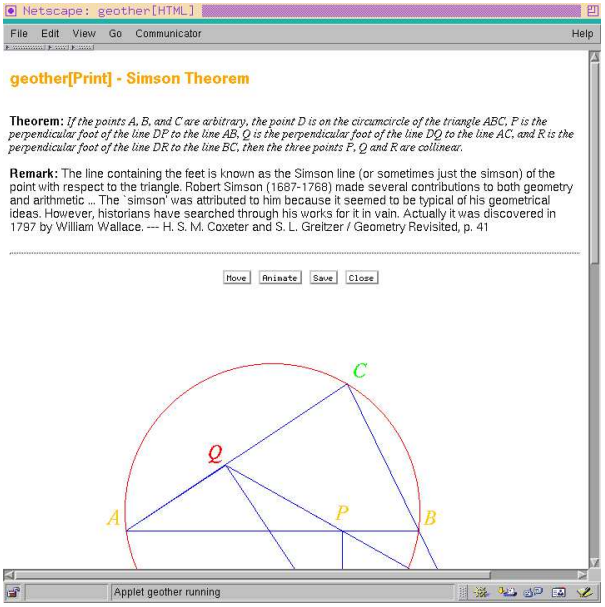

Figure 2. HTTP document generated by `Print`

Each generated diagram is displayed in a new window and can be modified and animated by mouse click and dragging. The Maple function `system` is used for the interaction with the operating system. Some implementation details about this drawing function is described in [11]. The window shown in Fig. 1 is an output of `Geometric(Simson)`.

5.2 `Print`

– Inputted optionally the name `N` of the current geometric theorem `T` in the session, `Print(T, N)` generates an HTML file (with Java applet) documenting the last manipulations and proof of `T` and invokes Netscape to view the file. If the optional third argument is given as `LaTeX`, then `Print(T, N, LaTeX)` generates a PostScript file documenting the theorem and invokes Ghostview to view the file.

The window dumps in Figs. 2 and 3 show parts of the generated documents (with `Print(Simson, 'Simson')` and `Print(Simson, 'Simson', LaTeX)`) viewed by Netscape and Ghostview for Simson's theorem.

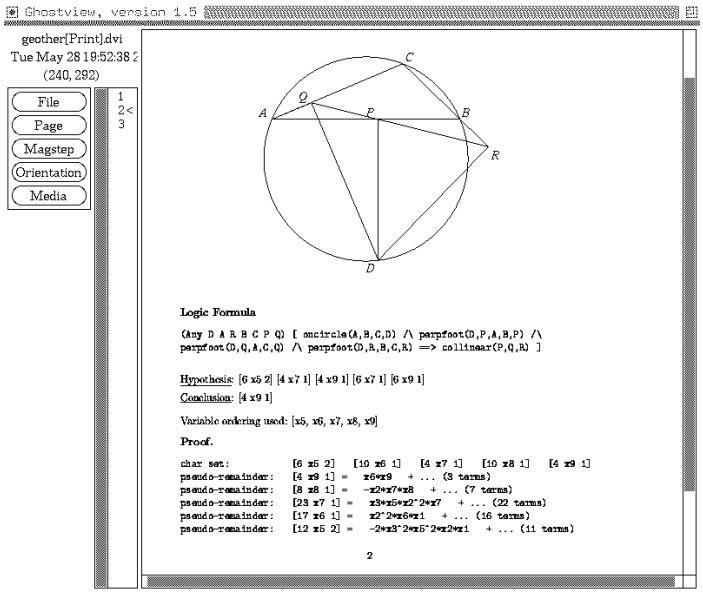

Figure 3. PostScript document generated by `Print`

6 Nondegeneracy Conditions and Miscellaneous Functions

6.1 Interpretation of Nondegeneracy Conditions

– Inputted the predicate specification `T` of a geometric theorem and a (simple) polynomial `p` (or a set `p` of polynomials) in the coordinates of the points occurring in `T`, `Generic(T, p)` translates the algebraic condition `p <> 0` with respect to `T` into geometric/predicate form.

This translation might not work for an arbitrary polynomial `p`. It works only for some simple polynomial whose nonvanishing has a clear geometric meaning with respect to the theorem. When the translation fails, the algebraic condition `p <> 0` is returned. For instance, `Generic(Simson, x1-x2)` results in

```
.   the line AB is not perpendicular to the line BC
                        not perpendicular(A, B, B, C)
```

and `Generic(Simson, x1-2*x2)` returns

```
.   x1-2*x2 <> 0
                                    not x1 - 2 x2
```

The function `Generic` is used within the provers to produce nondegeneracy conditions in geometric form stated in English or Chinese. See Sect. 7.4 for some discussions about its implementation.

6.2 `Load` and `Search`

– Inputted the name `T` of a geometric theorem, `Load(T)` reads the specification of the theorem from the built-in library to the GEOTHER session.

GEOTHER has a small collection of well-known geometric theorems specified by the author. The user may load any of these theorems to the GEOTHER session and write his or her own specifications for interesting geometric theorems (in plane Euclidean geometry). To see which theorems are contained in the library, one may use the following function.

– Inputted a string `S`, `Search(S)` lists all the names of theorems containing `S` in the library.

If `S` is given as `all` or `All`, then all the theorems in the library are listed.

6.3 `Demo` and `Click`

– `Demo()` gives an automated demonstration of GEOTHER.

During the demonstration, GEOTHER commands appear automatically and the user only needs to enter semicolon `;` and then return.

– `Click()` starts a mouse-driving GEOTHER interface.

The windows in Fig. 4 show the graphic interface, with which the user does not need to type in commands except for the name of a theorem, a string, or a polynomial when calling `Load`, `Search`, or `Generic` respectively.

Click on a button causes an application of the indicated function to the current theorem (last loaded) in the session. This interface works only under Unix and Linux, of which the pipe facility is used for data communication.

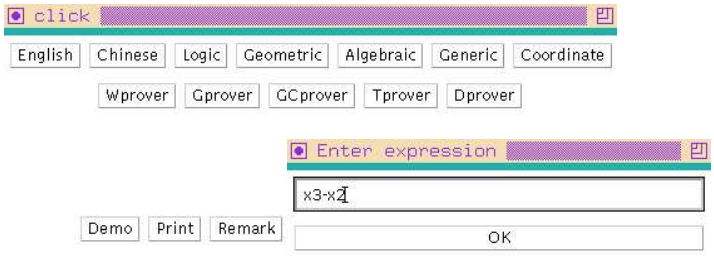

Figure 4. Graphic user interface

6.4 Online Help

GEOTHER provides online help for all its functions with examples. This help facility is used in the usual way as for standard Maple packages. This paper and part of a forthcoming book by the author will serve as an introduction to GEOTHER. Additional documents and future updates of GEOTHER will be made available on the author's web page.

7 Implementation Details

The implementation of most GEOTHER functions is easy and straightforward, but as usual it is a time-consuming task and requires special care to handle all the technical details at the programming level. This section discusses some of the implementation issues; of course we cannot enter into all the details.

7.1 The Geometric Specification Language

As the reader has seen, a geometric theorem in GEOTHER is specifi ed by using a simple predicate language. This specifi cation language provides a basic representation, that is extensible, for a large class of geometric theorems, and with which a theorem may be manipulated, proved, and translated into other representations.

A basic element of the language is predicate. Except for a few specially reserved predicates like `Theorem` and `dTheorem`, the other predicates declaring geometric relations are implemented with a standard list of information entries. A typical example is the routine corresponding to the predicate `online` shown in Fig. 5. Information contained in this routine includes the English and Chinese interpretations of `online` and its negation, a possible degeneracy condition `isotropic(a,b)`, two geometric objects `line(a,b)` and `line(a,c)` involved, and the corresponding algebraic expressions.

It is not diffi cult to imagine how the information contained in the predicate entries may be used naturally for manipulating and translating a geometric theorem involving such predicates at the running time. We shall discuss part of the usage in the following subsections. Composition of the corresponding entries of all the involved predicates using appropriate connectives for the whole theorem is merely an implementation of simple heuristics.

```
@online := proc(a, b, c)
    if args[nargs] = p_eng then
        cat(`the point `, c, ` is on the line `, a, b)
    elif args[nargs] = n_eng then
        cat(`the point `, c, ` is not on the line `, a, b)
    elif args[nargs] = p_chi then cat(c, `µãÔÚÖ±Ïß`, a, b, `ÉÏ`)
    elif args[nargs] = n_chi then cat(c, `µã²»ÔÚÖ±Ïß`, a, b, `ÉÏ`)
    elif args[nargs] = n_deg then isotropic(a, b)
    elif args[nargs] = g_obj then line(a, b), line(a, c)
    elif not (type(a, list) and type(b, list) and type(c, list)) then
        ERROR(`wrong assignment of coordinates of points`)
    elif nops(a) = 2 and nops(b) = 2 and nops(c) = 2 then -b[2]*c[1]
         + b[2]*a[1] + a[2]*c[1] + b[1]*c[2] - b[1]*a[2] - a[1]*c[2]
    elif nops(a) = 3 and nops(b) = 3 and nops(c) = 3 then -b[2]*c[1]
         + b[2]*a[1] + a[2]*c[1] + b[1]*c[2] - b[1]*a[2] - a[1]*c[2],
        -b[3]*c[1] + b[3]*a[1] + a[3]*c[1] + b[1]*c[3] - b[1]*a[3]
         - a[1]*c[3]
    else ERROR(
        `wrong assignment of coordinates or high-dimensional points`)
    end if
end proc
```

Figure 5. Maple routine for predicate `online`

Since predicate routines have a standard form, it is easy to add new predicates to the geometric language; thus the language may be easily extended and made more expressive.

7.2 Automated Assignment of Coordinates

Coordinates of points are assigned by `Let` to a Maple function internally, not to the letters labeling the points. This keeps the letters always as symbols, to which the corresponding coordinates of points may be printed out by `Algebraic`.

The function `Coordinate` for assigning coordinates of points automatically in the plane is implemented according to a simple principle: it checks whether the theorem's specification contains a perpendicularity predicate, and if so, takes the two perpendicular lines (on which there are more points) as the two axes; then the coordinates of their intersection point are both 0. If there is no perpendicularity predicate in the specification, then take a line which contains the maximal number of points possible as the first axis, with the other axis passing through one or more points not on the first axis. For any point other than the origin on the axes, one of the coordinates is assigned 0. Finally, generic coordinates are assigned to the remaining points not on the axes.

The heuristic choice of axes aims at getting more coordinates to 0, but without loss of generality, in order to simplify the involved algebraic computations.

7.3 Automated Generation of Diagrams

Generating geometric diagrams automatically is a more complex process. Our basic idea is to take random values for the free coordinates, solve the geometric constraint equations to get the values for the dependent coordinates, finally check whether there

exist two points which are too close, or too far away (so that the points cannot fit into a fixed window), and if so, then go back to take other random values. When proper values for the coordinates of points are determined, the geometric objects such as lines and circles contained in the `g_obj` entries of the predicates are drawn (and this is rather easy). The letters labeling the points are placed according to the centroid principle explained in [11].

Our drawing program is made somewhat complicated because of the requirement that the drawn diagrams can be modified and animated with mouse click and dragging. We shifted our implementation of the drawing function from C to the Java language. In our current version, not simply the concrete coordinate values but also the triangularized geometric constraint relations computed in Maple are passed onto the Java programs. This is detailed in [11].

7.4 Automated Interpretation of Nondegeneracy Conditions

Interpreting the geometric meanings of algebraic nondegeneracy conditions automatically is mainly a heuristic process. The function `Generic` is implemented for this purpose according to the following two principles. First, in the routine of each predicate an information entry is included that predetermines possible degeneracy conditions associated with this predicate. The reader may recall the degeneracy condition `isotropic(a, b)` associated with `online`. For another example: in the predicate `intersection(A, B, C, D, P)` (meaning "`the two lines AB and CD intersect at P`") the degeneracy condition `parallel(A, B, C, D)` is included. For each given predicate specification `T` of a geometric theorem, there is a finite collection C of possible nondegeneracy conditions in predicate form. For any input `p`, `Generic(T, p)` works by translating all the conditions in C into polynomial inequations and then comparing if any of the inequations is equivalent to `p <> 0`. If so, then `p <> 0` is "translated" into the corresponding nondegeneracy condition of predicate form in C.

The second principle works by fixing a finite set of predicates corresponding to popular degeneracy conditions such as two points coincide, three points are collinear, and two lines are parallel. For a given specification `T`, `Generic` applies each of these predicates to the points occurring in `T` in a combinatoric way, translates the predicate applied to concrete points into a polynomial equation, and verify if the obtained polynomial is equivalent to the input `p`. This is done one by one for different combinations of points and thus involves heuristic search. If an equivalence is discovered, then the translation is done for `p`. This approach is used when the approach according to the first principle fails. It requires extensive search, but the involved computation is not expensive, so it works rather well.

7.5 Implementation of Algebraic Provers

The main routines for polynomial elimination and reduction in GEOTHER provers are called from the two EPSILON modules `CharSets` and `TriSys` and the `Groebner` package provided by Maple. We implemented the proof methods described in [12, 4–7, 9] as indicated in Sect. 4 in a quite straightforward way, so we do not discuss the aspect of algebraic computation.

A major issue is how to produce appropriate subsidiary conditions. Our attention has been paid mainly to `Tprover`, which decomposes the system of hypothesis-polynomials (for equations and inequations) into irreducible triangular systems and check whether the conclusion of the theorem holds true for each triangular system. In other words, it is verified whether the theorem is true in all cases, generic or degenerate. Since a geometric theorem may be false in the degenerate cases and the algebraic formulation using polynomial equations and inequations may not rule out some undesired situations introduced by geometric ambiguities such as internal or external bisection of angles and internal or external contact of circles, the conclusion is true usually only for some of the irreducible triangular systems.

Now the question is how to form the subsidiary conditions to exclude those triangular systems for which the conclusion is false. Let us look at a simple example: we want to prove that

$$(\forall x,y)[x^2 = 1 \wedge y^2 = 1 \Longrightarrow (x+1)(y+1) = 0].$$

In this case, we have four irreducible triangular sets

$$\mathbb{T}_1 = [x+1, y+1], \quad \mathbb{T}_2 = [x+1, y-1], \quad \mathbb{T}_2 = [x-1, y+1], \quad \mathbb{T}_4 = [x-1, y-1]$$

and the conclusion is false for $\mathbb{T}_4$ and true for the other three. The triangular set $\mathbb{T}_4$ may be excluded by the subsidiary condition $x \neq 1 \vee y \neq 1$. This example shows why we need to use disjunction to form subsidiary conditions as proposed in [6].

Nevertheless, subsidiary conditions formed by means of conjunction and disjunction as in [6] may be very tedious. How to simplify the conditions is a question that still remains. We have adopted the following heuristics for `Tprover` and `Dprover`.

Suppose that the conclusion of the theorem is true for $\mathbb{T}_1, \ldots, \mathbb{T}_e$ and false for $\overline{\mathbb{T}}_1, \ldots, \overline{\mathbb{T}}_t$, where the triangular sets $\mathbb{T}_i$ and $\overline{\mathbb{T}}_j$ are all irreducible. For $j = 1, \ldots, t$, let Δ_j be the set of those polynomials in $\overline{\mathbb{T}}_j$ whose pseudo-remainders are not identically equal to 0 with respect to all $\mathbb{T}_i$. If $\Delta_j = \emptyset$, then let $D_j := \bigvee_{T \in \overline{\mathbb{T}}_j} (T \neq 0)$. Otherwise, let E_j be an element of Δ_j that has the maximal number of occurrences in $\Delta_1, \ldots, \Delta_t$ and $D_j := (E_j \neq 0)$. We take $\Omega := \bigwedge_{D \in \{D_1, \ldots, D_t\}} D$ as the subsidiary condition for the theorem to be true.

The generation of Ω requires extra computation. Without this computation, the produced condition may exclude more components for which the theorem is true. This may happen for the subsidiary conditions produced by the other provers in GEOTHER.

7.6 Soundness Remarks

Since a predicate or algebraic specification of a geometric theorem may be a false proposition in the logical sense, proving the theorem is not simply a yes or no confirmation. Often we want and try to produce a proof of the theorem, even if its specification is not correct logically, by imposing certain subsidiary conditions. This makes the proving problem much harder, in particular, when a *bad* nonsensical specification is submitted to the prover. In this case, a nonsensical proof of the "theorem" may be produced. This is not the fault of our provers because determining whether a meaningful theorem can be derived from an arbitrary specification is much beyond the scope of the provers. Therefore, the user must make sure that his or her specification of the geometric theo-

rem is correct, or at least *almost* correct. If a nonsensical proof is produced, we advise the user to check the specification carefully, or try to use a different specification.

Moreover, GEOTHER functions for manipulation and proof are made as automatic as possible. To achieve this, we have adopted a number of heuristics at the implementation level. These heuristics may fail in certain cases, and thus some functions may not work or work unpleasantly for some specifications of theorems. It is completely normal if the user gets an automatically drawn diagram that has a poor look or even looks bizarre. GEOTHER is not yet a perfect piece of software, but it shows how geometric theorems can be handled and proved automatically and nicely.

8 Experiments with Algebraic Provers

Comparing algebraic provers in terms of computing time is not necessarily instructive because the quality of the proofs produced by different provers may differ. For instance, `Wprover` and `GCprover` provide explicit subsidiary conditions, while `Gprover` does not.

Theorem	`Wprover`	`Gprover`		`GCprover`	`Tprover`
		`KS`	`Kapur`		
Butterfly	.411	.639	1.999	>2000	25.109
Ceva	.280	1.409	1.641	.940	1.921
Desargues	.019	.031	.060	.090	.120
Euler Line	.040	.051	.080	.340	.500
Feuerbach	.120	.330	.441	.720	.919
Gauss Line	.040	.080	.110	.070	.101
Gauss Point	.759	.950	2.160	1.711	1.579
Leisenring	.250	.360	.359	>2000	69.459
Menelaus	.230	.089	.180	.170	.220
Miquel	>2000	>2000	>2000	>2000	>2000
Morley	63.950	* 58.920	* 34.411	>2000	279.091
Morley–Wu	1.561	2.420	3.089	>2000	257.900
Orthocenter	.009	.021	.040	.029	.049
Pappus	.060	.100	.150	2.591	1.259
Pascal	.120	1.849	57.201	>2000	264.960
Pascal Conic	2.681	>2000	>2000	>2000	>2000
Poncelet	6.761	* .269	* 2.901	* .650	3.319
Ptolemy	.030	.059	.110	.090	.051
Secant	.010	.069	.090	.180	.091
Simson	.051	.089	.129	3.619	1.119
Simson Line	.039	.080	.110	.649	.709
Steiner	4.429	* .090	* 7.021	* .829	1.510
Steiner–Lehmus	1.380	* 15.991	* 443.689	* 153.630	3.960
Steiner–Wu	1.611	1.910	3.080	>2000	>2000
Thébault	10.501	* .600	* 53.179	>2000	19.829

Table 1. Proving times in Maple 8 on Pentium III 700

`Wprover` does not verify whether the theorem is true in the degenerate cases, while `Tprover` does. The purpose of the experiments reported in this section is to provide the reader with an idea about the magnitude of computing time required by different provers for some well-known geometric theorems. In fact, our main interest is not in the quantitative performance of these provers (because the efficiency of the underlying algebraic methods is quite well known); rather we want to see how much qualitatively we can do with algebraic methods for geometric theorem proving.

Table 1 shows the times for proving 25 theorems in plane Euclidean geometry using different provers. All the computations were performed in Maple 8 under Linux 2.4.7-10 on a laptop Pentium III 700 MHz with 128 MB of memory. The proving time is given in CPU seconds and includes the time for producing subsidiary conditions and for garbage collection. The cases in which the prover fails to prove the theorem after the indicated time are marked with $*$.

As it is well known, Wu's method (implemented as `Wprover` in GEOTHER) is the most efficient for confirming geometric theorems. Methods based on Gröbner bases (computed over the field of rational functions in parametric variables) are slightly slower but also efficient. For `GCprover` Gröbner bases are computed over the ground field (i.e., the field of rational numbers), so the computation is much more expensive. This prover fails to prove some of the theorems within 2000 CPU seconds, while it is capable of deciding whether a geometric theorem is universally true, and if not, providing explicit subsidiary conditions for the theorem to be true. The method based on zero decomposition is also less efficient, but the produced proofs have higher quality: all the degenerate cases are verified and the generated subsidiary conditions exclude fewer degenerate cases where the theorem is true. For some theorems such as Butterfly, Leisenring and Pascal, a considerable amount of time has been spent to generate such conditions.

Acknowledgments. The author wishes to thank Roman Winkler and Xiaofan Jin who have contributed to the implementation of `Coordinate` and `Generic` respectively. Part of this work is supported by the SPACES project (`http://www.spaces-soft.org`) and the Chinese national 973 project NKBRSF G19980306.

References

1. Chou, S.-C., Gao, X.-S., Liu, Z., Wang, D.-K., Wang, D.: Geometric theorem provers and algebraic equation solvers. In: *Mathematics Mechanization and Applications* (X.-S. Gao and D. Wang, eds.), pp. 491–505. Academic Press, London (2000).
2. Gao, X.-S., Zhang, J.-Z., Chou, S.-C.: *Geometry Expert* (in Chinese). Nine Chapters Publ., Taiwan (1998).
3. Hong, H., Wang, D., Winkler, F.: Short description of existing provers. *Ann. Math. Artif. Intell.* **13**: 195–202 (1995).
4. Kapur, D.: Using Gröbner bases to reason about geometry problems. *J. Symb. Comput.* **2**: 399–408 (1986).
5. Kutzler, B., Stifter, S.: On the application of Buchberger's algorithm to automated geometry theorem proving. *J. Symb. Comput.* **2**: 389–397 (1986).
6. Wang, D.: Elimination procedures for mechanical theorem proving in geometry. *Ann. Math. Artif. Intell.* **13**: 1–24 (1995).

7. Wang, D.: A method for proving theorems in differential geometry and mechanics. *J. Univ. Comput. Sci.* **1**: 658–673 (1995).
8. Wang, D.: GEOTHER: A geometry theorem prover. In: *Automated Deduction* (M. A. McRobbie and J. K. Slaney, eds.), LNAI 1104, pp. 166–170. Springer-Verlag, Berlin Heidelberg (1996).
9. Wang, D.: Gröbner bases applied to geometric theorem proving and discovering. In: *Gröbner Bases and Applications* (B. Buchberger and F. Winkler, eds.), pp. 281–301. Cambridge University Press, Cambridge (1998).
10. Wang, D.: Epsilon: A library of software tools for polynomial elimination. In: *Mathematical Software — Proceedings ICMS 2002* (A. M. Cohen, X.-S. Gao, and N. Takayama, eds.), pp. 379–389. World Scientific, Singapore New Jersey (2002).
11. Wang, D.: Automated generation of diagrams with Maple and Java. In: *Algebra, Geometry, and Software Systems* (M. Joswig and N. Takayama, eds.), pp. 277–287. Springer-Verlag, Berlin Heidelberg (2003).
12. Wu, W.-t.: *Mechanical Theorem Proving in Geometries: Basic Principles* (translated from the Chinese by X. Jin and D. Wang). Springer-Verlag, Wien New York (1994).