The *SymbolicData* GEO Records – A Public Repository of Geometry Theorem Proof Schemes

Hans-Gert Gräbe, Univ. Leipzig, Germany

November 26, 2002

1 Introduction

Geometry is not only a part of mathematics with ancient roots but also a vivid area of modern research. Especially the field of geometry called by some negligence "elementary" continues to attract the attention also of the great community of leisure mathematicians. This is probably due to the small set of prerequisites necessary to formulate the problems posed in this area and the erudition and non formal approaches ubiquitously needed to solve them. Examples from this area are also an indispensable component of high school mathematical competitions of different levels upto the International Mathematics Olympiad (IMO) [10].

The great range of ideas being involved with elementary geometry theorem proving inspired mathematicians to search for a common framework that allows to discover such geometric statements or, at least, to prove them in a more unified way. These attempts may be traced back until ancient times, e.g., to Euclid and his axiomatic approach to geometry.

A special common framework for geometry theorem proving was known at least since Descartes and his coordinate method: Translate geometric configurations into algebraic relations between coordinates and try to solve the algebraic counterpart of the geometric problem by algebraic methods. It was this framework that inspired the young Gauss for his famous solution to construct a regular 17-gon by ruler and compass.

With the increasing capabilities of modern computer equipment to do term rewriting and symbolic algebraic manipulations this approach obtained new power. The surprising observation that tedious but mostly straightforward algebraic manipulations allow to derive (mathematically strong!) proofs for many theorems in geometry with even ingenious "true geometric" proofs led many researchers to focus anew on questions of automated deduction of geometric statements.

The attempts to algorithmize this part of mathematics found their first culmination in the 80's in the work of W.-T. Wu on "the Chinese Prover", see, e.g., [20, 19] and the surveys in [21] or [22]. In the following these ideas were largely extended by different people, among them the "Chinese provers" in Wu's school at MMRC. Let's mention only the remarkable book [2] of S.-C. Chou who proved 512 geometry theorems with this mechanization method.

There are two conclusions to be drawn from Chou's book. First, the applicability of algebraic methods to geometry theorem proving is really convincing. A surprisingly great number of examples fall into the class of *constructive* problems (*explicitely constructive* in [23]), where the geometric configuration can be constructed step by step in such a way that new coordinates depend rationally on (free parameters and) coordinates of already constructed objects

and the geometric conclusion translates into a rational expression in these coordinates that should vanish. In this situation the solution of the algebraic problem reduces to a zero simplification problem of a rational expression in several (algebraically independent) variables. This problem is well understood and admits an efficient solution that is implemented in the core of all major (and minor) Computer Algebra Systems (CAS). Nevertheless real such simplifications may be very time and memory consuming, so that in some cases a non-constructive algebraic translation has to be preferred.

The coordinate method yields mathematically strong proofs for geometric statements with a serious drawback: Due to the algebraic nature of the intermediate steps these proofs cannot be retranslated to geometric reasoning but for a small number of cases. Often the algebraic statements "fit together" but the underlying geometry remains "invisible". More geometric approaches are discussed, e.g., in [3]. They still use polynomial computations, but take geometric invariants like areas and Pythagoras differences instead of coordinates of points as the basic quantities. Thus the geometric meaning for each step of the proof is clear.

Second, the proofs are not "automated" but "mechanized" in the following sense: A partly informal human readable geometric statement requires a translation into a strong computer readable syntax. In Chou's book [2] these *proof schemes* or *coordinatizations*, i.e., descriptions of geometric configurations of points, lines, and circles in a syntactically strong way, are composed by the author in a LISP like language and afterwards translated to their algebraic counterparts by the computer. For some theorems the given coordinatization is quite tricky, since the algebraic translation of the ad hoc solution is too hard to be handled by the computer. Even though speaking about "automatic geometry problem-solving" ([22]), also Wu emphasizes (on p. 12 of that paper and even in the title of [21]) on "mechanization methods" rather than "automated deduction" in full accordance with modern concepts of human-computer interaction. Such concepts consider computers not as automata but as tools in a more complex human-computer environment that combines precision and speed of computer equipment with human creativity and (informal) experience.

Mechanization of geometry theorem proving hence requires the creativity of diligent "proof writers" to eliminate all informal elements from a geometric statement and to fix the result in a (completely formalized) geometric proof scheme. These proof schemes are the starting point for further automated translation to, e.g., algebraic statements.

For intercommunication purposes and to store proof schemes in a common publicly available repository it is necessary to develop a *generic proof scheme language standard* that can be implemented with appropriate tools by all interested parties. In this paper we describe a first approach to such a standard, the *GeoCode*. It is used to store proof schemes as GEO records in a repository that is publicly available as part of the *SymbolicData* Project [13]. At the moment the *SymbolicData* GEO collection contains more than 250 such proof schemes, mainly from [2]. Special *SymbolicData* tools are designed to support the syntactical translation of *GeoCode* into proof scheme languages of special geometry theorem provers that support this common interface. At the moment – as a first reference application – this interface is implemented in the author's *GeoProver* packages [7], that provide tools to run proof scheme translations based on the coordinate method on one of the major CAS (Maple, Mathematica, MuPAD, Reduce).

The *GeoCode* standard evolved in a tight interplay between the collection of proof schemes and their evaluation with different versions of the *GeoProver* on different platforms. As a result of the discussions at the conference ADG-02 the standard was revised once more in the following directions:

1. The proof schemes collected so far used algebraic expressions to detect equality of angles, segments, triangle areas etc. To serve also more geometric approaches (e.g., the area method) these algebraic expressions were substituted by function calls with clear geometric meaning like `Equal`, `eqdist` etc.

2. The *GeoCode* standard uses `Point`, `Line` and `Circle` as geometric types. Since most of the geometry theorem provers take points as basic primitives and consider lines and circles as derived objects, the constructors `Line`$[a_1, a_2, a_3]$ and `Circle`$[c_0, c_1, c_2, c_3]$ were removed from the standard.

   Moreover, the definition of free points was singled out into a special attribute `Points` of the GEO proof scheme record to separate them from true geometric construction steps.

3. The names of the *GeoCode* functions were adjusted once more.

Since the *GeoCode* description is fixed in the same *SymbolicData* format as the GEO records themselves such adjustments are well supported by the *SymbolicData* action concept and the Perl string manipulation facilities. This allows to write compact Perl scripts to execute the required changes in the GEO proof scheme records.

This paper starts with some background on geometry theorem proving (section 2). Then we describe the design of the GEO records, the syntax of the *GeoCode* standard and the *GeoProver* packages as an implementation of that standard (section 3). In section 4 we discuss by means of examples how to compile new (generic) proof schemes, to translate them into *GeoProver* notion, to run this code on different CAS and to experiment with the resulting algebraic problems.

For real usability of the *GeoCode* concept one has to estimate the efforts required to implement this standard for other provers. Even though there is not yet practical experience with already existing geometry theorem provers the semantic similarity of "foreign" special geometric proof schemes suggests that such an interface should easily to be implemented. The problem is discussed in more detail in section 5 on the special geometric language used in [17] and, in a slightly revised form, also in D.Wang's GEOTHER project [14], see also [16].

A main problem to translate GEO records into geometric proof schemes for special geometry theorem provers is posed by different programming paradigms followed by the underlying geometry theorem provers. The *GeoCode* standard supports a functional programming style and the GEO record attribute values heavily use nested function calls. Several provers and dynamical geometry softwares (DGS) do not support nested function calls since they create and address geometric objects through identifiers.

We studied that problem within the task to translate our (constructive) proof schemes also to *construction schemes* that can be interpreted by DGS to draw a picture of the given geometric configuration or to generate a human readable *construction plan*. Such facilities are part of integrated geometry theorem provers as, e.g., D. Wang's GEOTHER prover [14, 16] or the *Geometry Expert*, [5]. We developed a first (Perl based) prototype interface of constructive GEO proof schemes to the GEONE$_X$T system [6] that really denests nested function calls.

2 Geometry Theorem Proving and the Coordinate Method

As already described in the introduction the main approach to mechanized geometry theorem proving considered so far depends on Descartes-Wu's coordinate method, translates geometric

statements into their algebraic counterparts, i.e., statements about systems of polynomial or rational functions, and tries to solve these algebraic problems by algebraic methods.

2.1 Geometry Theorems of Constructive Type

Usually geometric constructions can be compiled from a small number of elementary constructions, e.g., drawing a line through given points, constructing intersection points, circles with given parameters etc. In the same way also the coordinate representation of geometric statements can be produced cascading only a small number of elementary functions and data types. Hence interpreting the function calls in a geometric proof scheme in such an algebraic manner yields its algebraic translation as the starting point for the application of algebraic methods.

Note that the same proof scheme can be interpreted in a completely different way, e.g., by a drawing tool or geometry theorem prover based on a different method. This aspect will be discussed below. In this section we identify proof schemes and their algebraic translations.

We use points, lines and circles as basic objects with symbolic or numerical coordinates:

(x, y)	the point (x, y),
(g_1, g_2, g_3)	the line $\{(x, y) : g_1\, x + g_2\, y + g_3 = 0\}$, and
(c_1, c_2, c_3, c_4)	the circle $\{(x, y) : c_1\, (x^2 + y^2) + c_2\, x + c_3\, y + c_4 = 0\}$.

Let $\texttt{midpoint}(X, Y)$ be the midpoint of the segment XY, $\texttt{pp\_line}(X, Y)$ the line through X and Y and $\texttt{is\_concurrent}(a, b, c)$ a polynomial condition (in fact, a determinantal expression) that vanishes iff the lines a, b, c pass through a common point. The return values of all these functions are (sequences of) rational expressions in the coordinates of the formal input parameters.

With these functions at hand, e.g., the centroid intersection theorem can be proved in the following way: Choose generic points

$$A := \texttt{Point}(u_1, u_2); \quad B := \texttt{Point}(u_3, u_4); \quad C := \texttt{Point}(u_5, u_6);$$

compute coordinates for

$$A_1 := \texttt{midpoint}(B, C); \quad B_1 := \texttt{midpoint}(A, C); \quad C_1 := \texttt{midpoint}(A, B);$$

and evaluate the statement

$$\texttt{is\_concurrent}(\texttt{pp\_line}(A, A_1), \texttt{pp\_line}(B, B_1), \texttt{pp\_line}(C, C_1)) \tag{1}$$

To prove this theorem (and other theorems of this type) means to compose a nested rational expression like (1) and to check if it simplifies to zero. If it does, it will simplify to zero also for (almost) all *special* geometric configurations obtained from the *generic* configuration plugging in special numerical values for $u_1, \ldots, u_6$.

In general, we say that a geometric configuration is of *constructive type*, if its generic configuration can be constructed step by step in such a way, that the coordinates of each successive geometric object can be expressed as rational functions in the coordinates of already constructed objects and algebraically independent variables, and the conclusion can be expressed as vanishing of a rational function in these coordinates.

Such a theorem is generically true if and only if its configuration is not contradictory and the conclusion expression simplifies to zero.

Note that due to Euclidean symmetry even for generic configurations some of the coordinates can be chosen in a special way.

2.2 Geometry Theorems of Equational Type

Surprisingly many geometry theorems can be translated into statements of constructive type. Problems cause geometric objects derived from non-linear geometric conditions (angles, circles) if they are not uniquely defined or their coordinates cannot be rationally expressed in the given indeterminates. Geometric configurations with such objects require other proof techniques.

For example, given generic points $A = \texttt{Point}(a_1, a_2), B = \texttt{Point}(b_1, b_2), C = \texttt{Point}(c_1, c_2)$, a point $P = \texttt{Point}(x_1, x_2)$ is on the bisector of the angle $\angle\, ABC$ iff $\angle\, ABP = \angle\, PBC$, or, in *GeoProver* notation, iff

$$\texttt{l2\_angle}(\texttt{pp\_line}(A, B), \texttt{pp\_line}(P, B)) = \texttt{l2\_angle}(\texttt{pp\_line}(P, B), \texttt{pp\_line}(C, B))$$

In this formula $\texttt{l2\_angle}(g, h)$ denotes the tangens of the angle between the lines $g = (g_1, g_2, g_3)$ and $h = (h_1, h_2, h_3)$ that can be computed as

$$\frac{g_2\, h_1 - g_1\, h_2}{g_1\, h_1 + g_2\, h_2}.$$

Clearing denominators this condition on P translates into a polynomial of (total) degree 4 in the generic coordinates and quadratic in the coordinates of P. It describes the condition `on_bisector(P,A,B,C)` for P to be on either the inner or the outer bisector of $\angle\, ABC$. Note that in unordered geometry there is no way to distinguish between the inner and outer bisectors.

To prove the bisector intersection theorem lets "compute" the coordinates of the intersection points P of the bisectors through A and B and show that they belong to (one of) the bisectors through C. Due to Euclidean symmetry we can choose special coordinates for A and B to simplify calculations.

```
A:=Point(0,0);  B:=Point(1,0);  C:=Point(u1,u2); P:=Point(x1,x2);

polys:={on_bisector(P,A,B,C), on_bisector(P,C,A,B)};
```

$$\{-2\,x_2 + 2\,u_1\,x_2 + 2\,x_2\,x_1 - 2\,x_2\,u_1\,x_1 - u_2\,{x_2}^2 + u_2 - 2\,u_2\,x_1 + u_2\,{x_1}^2,$$
$$2\,x_2\,u_1\,x_1 - u_2\,{x_1}^2 + u_2\,{x_2}^2\}$$

`polys` is a system of two polynomial equations of degree 2 in (x_1, x_2) with coefficients in $\mathbf{Q}(u_1, u_2)$. It has 4 solutions that correspond to the 4 intersection points of the bisector pairs through A and B. They can be computed, e.g., with Maple:

```
solve(polys,{x1,x2});
```

$$\left\{x_2 = \%1, x_1 = 1/2\,\frac{u_2 - 2\,\%1 + 2\,u_1\,\%1}{u_2 - \%1}\right\}$$

$$\%1 = \mathit{RootOf}\left(4\,u_2\,\_Z^4 + \left(-8\,{u_1}^2 - 8\,{u_2}^2 + 8\,u_1\right)\_Z^3\right.$$
$$\left. + \left(-4\,u_1\,u_2 + 4\,{u_1}^2 u_2 - 4\,u_2 + 4\,{u_2}^3\right)\_Z^2 + 4\,{u_2}^2\,\_Z - {u_2}^3\right)$$

The solution involves algebraic *RootOf*-expressions that require a powerful algebraic engine to cope with.

Another approach uses direct reformulation of the geometry theorem as a vanishing problem of the polynomial conclusion on the zero set of the system of polynomials that describe the given geometric configuration.

For our example, consider the conclusion polynomial

```
con:=on_bisector(P,B,C,A);
```

$$2\,{u_1}^2x_2\,x_1+2\,u_2\,{x_2}^2u_1-2\,u_2\,{x_1}^2u_1-u_2\,{x_2}^2+u_2\,{x_1}^2+2\,u_2\,x_1\,{u_1}^2-2\,{u_2}^2x_1\,x_2-2\,x_2\,u_1\,x_1-{u_1}^2u_2+2\,{u_2}^2x_2-{u_2}^3+2\,x_2\,{u_1}^2-2\,{u_1}^3x_2+2\,{u_2}^3x_1-2\,u_1\,x_2\,{u_2}^2$$

and check if it vanishes on the variety of zeroes of `polys` regarded as zero dimensional polynomial system in $\mathbf{Q}(u_1,u_2)[x_1,x_2]$. This follows if the normal form of `con` with respect to a Gröbner basis of `polys` vanishes. Hence the following Maple computation verifies the theorem:

```
with(Groebner):
TO:=plex(x1,x2): gb:=gbasis(polys,TO):
normalf(con,gb,TO);
```

$$0$$

In general, this kind of algebraization of geometry theorems by the coordinate method yields a polynomial ring $S = k[\mathbf{v}]$ with variables $\mathbf{v} = (v_1, \ldots, v_n)$, a polynomial system $F \subset S$ that describes algebraic dependency relations in the given geometric configuration, a subdivision $\mathbf{v} = \mathbf{x} \cup \mathbf{u}$ of the variables into dependent and independent ones, and the conclusion polynomial $g(\mathbf{x}, \mathbf{u}) \in S$.

A set of variables $\mathbf{u}$ is *independent* wrt. an ideal $I = I(F)$ iff $k[\mathbf{u}] \cap I = (0)$, i.e., if $\mathbf{u}$ is algebraically independent on the variety $Z(F)$ of zeroes of F. In most practical applications such a subdivision is obvious. A strong verification can be derived from a Gröbner basis of F wrt. an appropriate term order.

$Z(F)$ may be decomposed into irreducible components that correspond to prime components P_α of the ideal $I = I(F)$ generated by F over the ring $S = k[\mathbf{x}, \mathbf{u}]$. Since $P_\alpha \supseteq I$ the variables $\mathbf{u}$ may become dependent wrt. P_α. Prime components where $\mathbf{u}$ remains independent are called *generic*, the other components are called *special*. By definition, every special component contains a non zero polynomial in the independent variables $\mathbf{u}$. Multiplying them all together yields a non degeneracy condition $h = h(\mathbf{u}) \in k[\mathbf{u}]$ on the independent variables such that a zero $\mathbf{c} \in Z(F)$ with $h(\mathbf{c}) \neq 0$ necessarily belongs to one of the generic components. Hence they are the "essential" components and we say that *the geometry theorem is generically true*, when the conclusion polynomial g vanishes on all these generic components.

If we compute in the ring $S_0 = k(\mathbf{u})[\mathbf{x}]$ as we did in the above example, i.e., consider the independent variables as parameters, exactly the generic components of I remain visible. Hence if the normal form of g wrt. a Gröbner basis G of F computed in S_0 vanishes the geometry theorem is generically true. More subtle examples can be analyzed with the Gröbner factorizer or more advanced techniques.

There are other algebraic techniques to analyze such polynomial systems, e.g., based on pseudo division and triangular sets. See [11] or the monograph [18] for a more complete survey.

2.3 Mechanized Geometry Theorem Proving

To really run mechanized geometry theorem proofs as described in the previous subsection requires a target CAS and several ingredients:

(1) We need a "proof writer" that writes (realistic) proof schemes for given informal statements of geometry theorems.

(2) We need tools to translate geometric statements into their algebraic counterparts as input for the target CAS.

(3) The CAS should be capable of the required algebraic manipulations.

(4) The CAS should provide tools to analyze the algebraic situation (e.g., to solve systems of equations, to compute Gröbner bases and normal forms etc.)

For Descartes-Wu's approach, topic (3) requires only facilities to compute with rational expressions and is usually not the bottleneck for geometry theorem proving. For some proofs topic (4) may be really challenging since it exploits the full compute power of the algebraic engine of the target CAS. On the other hand different proof schemes for the same problem can yield algebraic formulations of very different run time also within the same CAS.

In most cases topic (1) is straightforward, in particular if the informal geometric statement is already highly constructive. But in some applications the "proof writers" had to develop really ingenious and non trivial ideas to write reliable proofs that can be run automatically. For example, Wu proposed in [21] the following constructive proof for the bisector intersection theorem:

- Start with the vertices A, B and the (future) intersection point P of the bisectors through A and B.
- Draw the lines c through AB, d through AP and e through BP.
- Draw lines u, v derived from c by reflection wrt. to the axes d, e.

  These lines will meet in a point C such that d and e are the bisectors of ABC through A and B.
- Prove that P is also on the third bisector.

Geometry theorem prover usually provide tools for steps (2–4) to run proof schemes written in a prover specific language automatically. With a general purpose CAS at hand it is enough to have tools for (2). Such translation tools for Maple, MuPAD, Mathematica, and Reduce are provided by the author's *GeoProver* packages, [7], see below.

The main drawback of all these systems is the restricted interoperability of proof schemes. To fix a proof scheme for automated processing by different provers requires a generic language that can be mapped to all target systems. Below we report about our experience with the *GeoCode* language that was invented to store generic proof schemes in the *SymbolicData* GEO record collection.

Here is the notion of Wu's constructive proof scheme of the bisector intersection theorem in *GeoCode* notation:

```
<Points>
   $A:=Point[0,0]; $B:=Point[1,0]; $P:=Point[u1,u2];
</Points>
<coordinates>
   $l1:=pp_line[$A,$B];
   $l2:=sym_line[$l1,pp_line[$A,$P]];
   $l3:=sym_line[$l1,pp_line[$B,$P]];
```

```
</coordinates>
<conclusion>
   $result:=on_bisector[$P,$A,$B,intersection_point[$l2,$l3]];
</conclusion>
```

S.-C. Chou is probably one of the most diligent "proof writers" who collected in [2] more than 500 examples of geometric statements and appropriate algebraic translations.

During our work on the *SymbolicData* GEO collection we stored (and partly modified and adapted) about 200 of them. We collected also solutions of geometry problems from other sources, e.g., the IMO contests, see [10]. Much of this work was done by my "proof writers", the students Malte Witte and Ben Friedrich, who compiled first electronic versions for many of these examples.

3 *GeoCode* and GEO Records

3.1 The *SymbolicData* Project

The *SymbolicData* project was set up to create and manage a publicly available repository of digital test and benchmark data from different areas of symbolic computation and to develop tools and concepts to manage such data both in the repository and at a local site. In a first stage we concentrated on the development of practical concepts for a convenient data exchange format, the collection of existing benchmark data from two main areas, polynomial system solving and geometry theorem proving, and the development of appropriate tools to process this data. A tight interplay between conceptual work, data collection, and tools (re)engineering allowed continuously to evaluate the usefulness of each of the components.

For easy reuse we concentrated on free software tools and concepts. The data is stored in a XML like ASCII format that can be edited with your favorite text editor. The tools are completely written in Perl using Perl 5 modular technology.

Some of our ad hoc concepts of data representation changed several times and (although meanwhile being quite elaborated) surely will partly change in the future (e.g., lists and hashes will probably be stored in a more XML compliant form). Having data available in electronic form (so far) it was very easy to translate it into the revised formats. Hence the most complicated part of the project is the collection of benchmark data and its translation from the foreign to the current format of the repository. Note that our concept of data representation is very flexible. The data format can be specified by the user in an easy manner and very broad range. I refer to [1, 8, 9] and the *SymbolicData* documentation for more details.

The project is organized as a free software project. The CVS repository is equally open to people joining the *SymbolicData* project Group. Tools and data are freely available also as tar-files from our Web site under the terms of the GNU Public License.

The *SymbolicData* project is part of the benchmark activities of the German "Fachgruppe Computeralgebra" who also sponsored the web site [13] as a host for presentation and download of the tools and data developed and collected so far. We kindly acknowledge support also from UMS MEDICIS of CNR/École Polytechnique (France) who provides us with the needed hard- and software to run this web site.

3.2 *SymbolicData* Files and *SymbolicData* Records

Records in the *SymbolicData* data base are stored as ASCII files (**sd-files**) in a (flat) XML like syntax. A typical example of such a record, the record `Parallelogram_2` in the GEO table, is given on page 10. It contains information and a mechanized proof scheme for the following geometry theorem:

> *The intersection point of the diagonals of a parallelogram is the midpoint of each of the diagonals.*

Some of the attributes of that record (`Type`, `Key`, `CRef`, ...) serve for identification or store relational information. The other fields store the different parts of the proof scheme in *GeoCode* syntax.

Note that the description of the internal structure of these attributes is given in the same format in a `META/GEO` table. Hence new attributes can be added and existing attributes can be modified in an easy manner.

The sd-files are tight to Perl hashes (**sd-records**) by the *SymbolicData* tools in a transparent way. The *SymbolicData* tools deliver such a hash object to the application programmer for further Perl manipulation. An elaborated **actions** concept reduces common programming overhead to a minimum. Since the detailed requirements of different user driven tasks are not known in advance to the *SymbolicData* developers it is difficult (and probably even not worth) to design a more reliable interface.

3.3 *SymbolicData* Proof Schemes

SymbolicData GEO proof schemes are divided (roughly) into two types according to their `prooftype` attribute: constructive and equational.

The generic variables are provided as values of two attributes:

`parameters`	a list $\mathbf{u}$ of independent parameters
`vars`	a list $\mathbf{x}$ of dependent variables (equational proofs only)

For equational proofs the variable lists $\mathbf{x}$ and $\mathbf{u}$ are chosen in such a way that $\mathbf{u}$ is a maximal independent set of variables for the given algebraic variety over $k[\mathbf{x}, \mathbf{u}]$ as defined above.

The basic attributes (with *GeoCode* values) are:

`Points`	the free points of the proof scheme
`coordinates`	assignments that compose step by step the generic geometric configuration of the proof scheme
`conclusion`	the conclusion of the proof scheme

This already completes the data required for a constructive proof scheme. For equational proof schemes the following additional attributes are defined:

`polynomials`	a list of *GeoCode* predicates that correspond to polynomial or rational conditions describing algebraic dependency relations in the given geometric configuration
`constraints`	a list of *GeoCode* predicates that correspond to polynomial non degeneracy conditions
`solution`	a way to solve the algebraic problem (given in extended *GeoCode* syntax)

```
#########################################################
# Record 'GEO/Parallelogram_2'

<Id>        GEO/Parallelogram_2           </Id>
<Type>      GEO                           </Type>
<Key>       Parallelogram_2               </Key>
<prooftype> constructive                  </prooftype>
<parameters> [u1, u2, u3]                 </parameters>
<Points>
$A:=Point[0,0]; $B:=Point[u1,0]; $D:=Point[u2,u3];
</Points>
<coordinates>
$C:=par_point[$D,$A,$B];
$P:=intersection_point[pp_line[$A,$C],pp_line[$B,$D]];
</coordinates>
<conclusion> $result:=eqdist[$A,$P,$C,$P]; </conclusion>
<CRef>
PROBLEMS/Geometry/Parallelogram => problem description
</CRef>
<Version> ... </Version>
<ChangeLog>
Sep 7 2002 graebe: Translated to GeoCode 1.3
Sep 6 2002 graebe: new tag 'Points' created
Sep 2 2002 graebe: $C=par_point[..]
Feb 10 2002 graebe: translated to GeoProver 1.2 syntax
</ChangeLog>
<PERSON>    graebe                        </PERSON>
<Date>      Nov 1 1999                    </Date>

# End of record 'GEO/Parallelogram_2'
#########################################################
```

The GEO record 'Parallelogram_2'

The proof idea can be sketched within the `ProofIdea` attribute as plain text if not yet evident from the code. See [9] for a detailed description of the GEO record structure. Below we concentrate on the *GeoCode* part.

3.4 The *GeoCode* Syntax

The design of the generic *GeoCode* language is mainly motivated by our aim to fix geometry theorem proof schemes in such a way that they can easily be translated to different target systems. A good but expensive idea would be to define an appropriate (context free) programming language and to write cross compilers or to invent a reliable (full) XML markup and to use style sheet translations. Since the syntaxes of the target languages are very similar we decided to avoid these efforts and defined a generic language that can be cross compiled using only regular patterns. Due to its elaborated pattern matching facilities Perl is best suited to realize this approach.

We assume proof schemes to be composed by a sequence of assignments with nested function calls as right hand sides that refer to previously defined geometric objects and scalars as arguments. To be mapped to different target systems the *GeoCode* language should meet the following requirements:

(1) Variable, symbol and function names can be identified.

(2) The generic *GeoCode* can be mapped to the syntax of the target system without name clashes.

In this context the words 'variable' and 'symbol' are used in the following sense: the former are 'symbols with values' (e.g., names for points, lines, circles), the latter 'symbols without values' (i.e., names for parameters and variables in the previous sense). It is a special peculiarity of symbolic computations that these name spaces usually overlap. For geometry theorem proof schemes this overlap can be avoided. We use Perl like syntax (i.e., `\$[a-zA-z][a-zA-z0-9]*` in Perl regexp notation) for variable names and small letter / digit combinations (i.e., `[a-z][a-z0-9]*` in Perl regexp notation – we don't allow capital letters to avoid name clashes both in Reduce and Mathematica) for symbol names.

Most CAS use parentheses both to group arithmetic expressions and argument lists in function calls. Since this cannot be distinguished within a regular language we use the Mathematica convention (i.e., brackets) for function call notation[1].

Equational GEO records usually contain also a `solution` tag with a description how the algebraic task can be solved. This description is fixed in an extended *GeoCode* syntax. Interface packages for Maple, MuPAD, Mathematica, and Reduce to map these generic commands to appropriate constructs are part of the *SymbolicData* distribution. For details see [9].

The names and signatures of all the *GeoCode* functions are stored in the *SymbolicData* `GeoCode` table and can be extracted, extended and modified in the same way as other sd-records. Two such `GeoCode` records are reproduced below. The first one corresponds to an 'inline' function that requires a special implementation, the second one to a 'macro' with a generic definition in *GeoCode* syntax as value of the `code` attribute that can be used to

[1] Note that most of the arithmetic expressions were replaced by new geometric predicates `Equal`, `eqdist`, `Normal` etc. in the GEO records during preparation of version 1.3 (finished after the ADG-02 conference) to emphasize the geometric nature of the proof schemes.

create an implementation automatically. For a complete description of all functions see the *SymbolicData* `GeoCode` documentation.

3.5 The *GeoProver* Packages

Really to run proof schemes written in *GeoCode* syntax with a geometry theorem prover requires to translate the *GeoCode* to equivalent code in the special language of the target prover. We propose the following approach: First, write Perl tools to translate the generic proof scheme into a syntactic form that is more appropriate for the target system (e.g., change square bracket, fix variable and function names, etc.). This is well supported by the *SymbolicData* actions concept and sample implementations for such translators in the `bin/GEO` directory of the *SymbolicData* distribution.

Second, write an interface package in the language of the target prover that maps the *GeoCode* functions to the prover specific functions. The author's *GeoProver* packages implement such interfaces for Maple, MuPAD, Mathematica, and Reduce.

For each of these CAS the *GeoProver* (formerly *Geometry*) provides a small package for mechanized (plane) geometry manipulations with non degeneracy tracing and a set of functions to handle generic and special geometric configurations containing points, lines and circles.

For a flavor of the usage resp. a formal description of all functions see the sample calculations in the previous section and the documentation [7]. For some target systems there is also a plot extension that allows to draw graphics from scenes, i.e., (of course special) geometric configurations.

A first prototype of the *GeoProver* grew out from a course of lectures for students of computer science on this topic held by the author at the Univ. of Leipzig in fall 1996. It was updated and completed to version 1.1 of a Reduce package after a similar lecture in spring 1998. Later on in cooperation with Malte Witte, at those times one of my students, the package was translated to the other target systems.

Since version 1.2 there is a separate description of the *GeoCode* language that was fixed in *SymbolicData* format and added as the `GeoCode` table to the *SymbolicData* Project later on. Now the complete *GeoProver* source code is generated from a platform-specific 'inline' code part for the basic functions and generic *GeoCode* `code` values for advanced functions using special *SymbolicData* tools. This facilitates a concise code management of the *GeoProver* source code if the *GeoCode* standard changes during development.

4 Some Examples

To get a look and feel about the efforts required to compile new *GeoCode* proof schemes, to translate them with *SymbolicData* tools to *GeoProver* applications and to run them on different target CAS let's consider some examples.

```
############################################################
# Record 'GeoCode/pp_line'

<Id>        GeoCode/pp_line                </Id>
<Type>      GeoCode                        </Type>
<Key>       pp_line                        </Key>
<call>      pp_line[$A::Point,$B::Point]::Line </call>
<verbose>   line through A and B           </verbose>
<description>
The line through <math>A</math> and <math>B</math>.
</description>
...

# End of record 'GeoCode/pp_line'
############################################################

############################################################
# Record 'GeoCode/altitude'

<Id>        GeoCode/altitude               </Id>
<Type>      GeoCode                        </Type>
<Key>       altitude                       </Key>
<call>      altitude[$A::Point,$B::Point,$C::Point]::Line </call>
<verbose>   altitude from A onto g(BC)     </verbose>
<code>      ortho_line[$A,pp_line[$B,$C]]  </code>
<description>
The altitude from <math>A</math> onto <math>g(BC)</math>.
</description>
...

# End of record 'GeoCode/altitude'
############################################################
```

The `GeoCode` records 'pp_line' and 'altidute'

1. The "cathedral example", [11, 5.3]

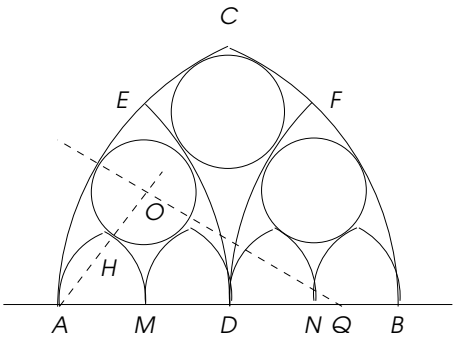

Figure 1: The "cathedral example"

P, Q are centers of the arcs CB, AC, respectively; arcs DE, DF are drawn with R, S as centers, respectively, and AQ as the radius. Further, $AQ = BP = \frac{5}{6}AB$. The goal is to find the radius of the circle tangent to arcs EA, ED, HM (with center A) and KM (with center D) as a function of AB.

We take the problem formulation and notational conventions from [11] with $s = |\overline{AB}| = 1$. In that paper line AB is taken as x-axis with origin at M (written as `$M` in *GeoCode* syntax) and the following coordinates are assigned to points (using the *GeoCode* point constructor):

```
$O:=Point[0,y3]; $A:=Point[-3/12,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
```

Kapur's original proof scheme contains 6 algebraic conditions that arize from the tangency conditions of two circle pairs. A circle tangency condition yields 3 polynomials: If $T_1 = (x_1, y_1)$ is the point of tangency of the circles around A and O these conditions are 'T_1 on the circle around A', 'T_1 on the circle around O', and 'T_1, A, O are collinear'.

Here is that statement in the formal *GeoCode* syntax:

```
<vars>      [x1,y1,x2,y2,y3,r]                          </vars>
<Points>
   $O:=Point[0,y3]; $A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
   $T1:=Point[x1,y1]; $T2:=Point[x2,y2];
</Points>
<coordinates>
   $c1:=pc_circle[$A,$M]; $c2:=pc_circle[$Q,$A];
</coordinates>
<polynomials>
   $polys:=List[
      on_circle[$T1,$c1], is_collinear[$A,$T1,$O], r^2-sqrdist[$O,$T1],
      on_circle[$T2,$c2], is_collinear[$Q,$T2,$O], r^2-sqrdist[$O,$T2]];
</polynomials>
```

`pc_circle` is the point-center circle constructor that returns a circle object. *GeoCode* supports `Point`, `Line` and `Circle` as geometric types. Note that `on_circle[$T1,$c1]` and `sqrdist[$T1,$A]-sqrdist[$M,$A]` yield the same algebraic translation. Hence a similar proof scheme may be composed without references to circle objects[2].

The solution of the algebraic problem may be obtained if all variables but r are eliminated from `$polys` and the remaining equation is solved for r. This can be stored in the `solution` attribute of the GEO record in extended *GeoCode* syntax in the following way:

[2]With *GeoCode* version 1.3 better use the geometric predicate `eqdist[$T1,$A,$M,$A]` instead of the algebraic expression `sqrdist[$T1,$A]-sqrdist[$M,$A]`

```
<solution>
   $result:=geo_solve[geo_eliminate[$polys,$vars,List[x1,y1,x2,y2,y3]],r];
</solution>
```

In the spirit of dynamical geometry software another proof scheme with less variables can be given if T_1 and T_2 are taken as 'circle sliders' that correspond to rational parameterizations of such points:

```
<vars>      [x1,x2,y3,r]                          </vars>
<Points>
   $O:=Point[0,y3]; $A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
</Points>
<coordinates>
   $T1:=circle_slider[$A,$M,x1]; $T2:=circle_slider[$Q,$A,x2];
</coordinates>
<polynomials>
   $polys:=List[
      is_collinear[$A,$T1,$O], r^2-sqrdist[$O,$T1],
      is_collinear[$Q,$T2,$O], r^2-sqrdist[$O,$T2]];
</polynomials>
```

Note that in this case the algebraic translations of the geometric conditions yields in fact rational functions since the coordinates of T_1 and T_2 are not polynomial but rational. A special algebraic approach is required for such proof schemes. A brute force call to the MuPAD command `solve($polys,$vars)` produces 24 solutions with 12 different values of r.

A third approach uses the explicit circle tangency condition, that corresponds to a polynomial condition on the circle parameters. It requires only 3 variables and translates to a polynomial system. Take center O and a circumfere point X on the y-axis with generic y-coordinates

```
$O:=Point[0,x1]; $X:=Point[0,x2];
$A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
```

add variables for the arcs (circles) that should be tangent

```
$c1:=pc_circle[$A,$M]; $c2:=pc_circle[$Q,$A]; $c3:=pc_circle[$O,$X];
```

and fix the tangency conditions and another one for the radius $r = x_1 - x_2$ of the circle c_3

```
$polys:=List[is_cc_tangent[$c1,$c3], is_cc_tangent[$c2,$c3], r-(x1-x2)];
```

The problem is of equational type and poses a deduction task. There are no independent variables and an algebraic solution can be obtained if x_1, x_2 are eliminated from the polynomials and the remaining equation is solved for r. The corresponding GEO record is given on page 22.

To translate and run that code with MuPAD we call the *SymbolicData* `MuPADCode` action. It maps *GeoCode* syntax to *GeoProver* MuPAD syntax, resolves name clashes and yields (*GeoProver* package loading omitted)

```
//==> Example Cathedral_1
clear_ndg():
delete 'x1','x2','s';
_vars:=geoList(x1,x2,s);
//coordinates
_O:=Point(0,x1); _X:=Point(0,x2);
_A:=Point(-3,0); _Q:=Point(7,0); _M:=Point(0,0);
_c1:=pc_circle(_A,_M); _c2:=pc_circle(_Q,_A); _c3:=pc_circle(_O,_X);
//polynomials
_polys:=geoList(is_cc_tangent(_c1,_c3),is_cc_tangent(_c2,_c3),12*s-(x1-x2));
//solution
_result:=geo_solve(geo_eliminate(_polys,_vars,geoList(x1,x2)),s);
quit;
```

The core of that action is a 3-line Perl script

```
sub MuPAD
{
  local $_=shift;
  s/List\[/geoList\[/gs; # since List is now a key word
  tr/\[\]/\(\)/;
  s/\$(\w+)/_$1/gs;
  return $_;
}
```

Since the circle tangency condition is implemented in the *GeoProver* we can run that script with MuPAD to get the solution

$$\left[\left[r=-\frac{17}{56}\right],\left[r=-\frac{17}{104}\right],\left[r=\frac{17}{56}\right],\left[r=\frac{17}{104}\right]\right]$$

in good accordance with [11]. Note that $r=-\frac{17}{104}$ corresponds to the position of X on the 'top' of O since $r=x_1-x_2$. $r=\pm\frac{17}{56}$ is also a common solution of the system given in [11] but not discussed there. It corresponds to imaginary coordinates of O and hence is virtual.

A similar computation yields the length of the radius of the circle in the top region of figure 1: With origin at D, the center O_1 and a circumfere point X of that circle on the y-axis we get the proof scheme

```
<vars>      [x1,x2,r]                          </vars>
<Points>
   $D:=Point[0,0]; $O1:=Point[0,x1]; $X:=Point[0,x2];
   $S:=Point[5/6,0]; $P:=Point[-1/3,0]; $B:=Point[1/2,0];
</Points>
<coordinates>
   $c1:=pc_circle[$S,$D]; $c2:=pc_circle[$P,$B]; $c3:=pc_circle[$O1,$X];
</coordinates>
<polynomials>
   $polys:=List[is_cc_tangent[$c1,$c3], is_cc_tangent[$c2,$c3], r-(x1-x2)];
</polynomials>
```

Running the corresponding computation with MuPAD yields the result

$$\left[\left[r = -\frac{7}{40}\right], \left[r = \frac{7}{40}\right]\right].$$

2. The Generalized Steiner Theorem, [17, Ex. 7]

Take three points A_2, B_2, C_2 respectively on the three perpendicular bisectors of BC, AC, AB of any triangle ABC such that

$$d(A_2, BC) = t \cdot |BC|,\ d(B_2, AC) = t \cdot |AC|,\ d(C_2, AB) = t \cdot |AB|,$$

where $d(P, QR)$ denotes the distance of the point P from the line QR and t is an arbitrary non-negative number.

Then the three lines AA_2, BB_2, CC_2 are concurrent.

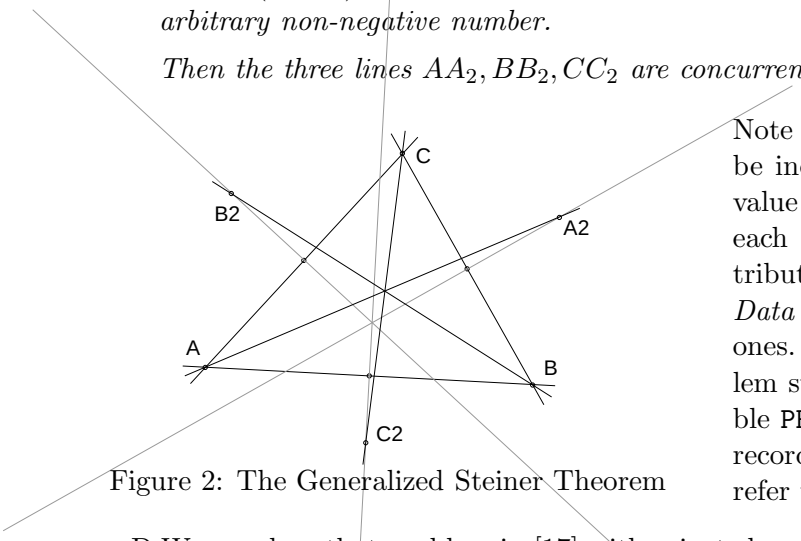

Figure 2: The Generalized Steiner Theorem

Note that the statement of a problem may be included in a *SymbolicData* record as value of a new attribute, say `Text`, since each such record admits "undefined" attributes that are handled by the *SymbolicData* tools in the same way as "defined" ones. In the *SymbolicData* data base problem statements are stored in a special table `PROBLEMS` and cross referenced in `GEO` records since several proof schemes may refer to the same problem.

D.Wang solves that problem in [17] with oriented areas using a Clifford algebra approach that avoids the introduction of virtual solutions. A straightforward coordinatization with independent variables u_1, u_2, t and dependent variables x_1, x_2, x_3 goes as follows:

```
<vars>      [x1,x2,x3]                          </vars>
<parameters> [u1, u2, t]                        </parameters>
<Points>
   $A:=Point[0,0]; $B:=Point[0,1]; $C:=Point[u1,u2];
</Points>
<coordinates>
   $A1:=midpoint[$B,$C]; $B1:=midpoint[$A,$C]; $C1:=midpoint[$A,$B];
   $A2:=line_slider[p_bisector[$B,$C],x1];
   $B2:=line_slider[p_bisector[$A,$C],x2];
   $C2:=line_slider[p_bisector[$A,$B],x3];
</coordinates>
<polynomials>
   $polys:=List[sqrdist[$A1,$A2]-t^2*sqrdist[$B,$C],
                sqrdist[$B1,$B2]-t^2*sqrdist[$A,$C],
                sqrdist[$C1,$C2]-t^2*sqrdist[$A,$B]];
</polynomials>
<conclusion>
```

```
   $con:=is_concurrent[pp_line[$A,$A2], pp_line[$B,$B2], pp_line[$C,$C2]];
</conclusion>
```

It yields a polynomial system with 8 solutions in (x_1, x_2, x_3) in the rational function field $k(u_1, u_2, t)$ that correspond to the different combinations of orientations of the triangles ABC_2, BCA_2, CAB_2. I checked this for Maple 7, MuPAD 2.0, and Reduce 3.7 with the solution

```
<solution>
   $sol:=geo_solve[$polys,$vars];
   $result:=geo_simplify[geo_eval[$con,$sol]];
</solution>
```

that was translated with *SymbolicData* tools to *GeoProver* code for the different CAS. All three CAS found that for exactly two of the 8 solutions the theorem is valid (i.e., the `$result` simplifies to zero). Note that with t^2 replaced by t only the Reduce `solve` command found the same answer. Maple and MuPAD[3] created expressions with several root symbols that could not be completely simplified in the following computation.

3. The Miquel Circle, [12, Ex. 5]

If four circles are arranged in sequence, each two succesive circles intersecting, and a circle pass through one pair of each such pair of intersections, then the remaining intersections lie on another circle.

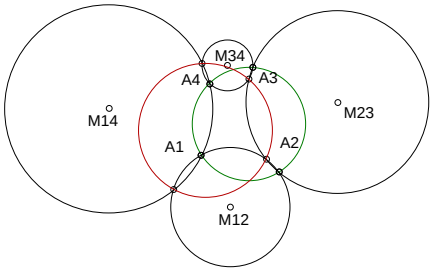

Figure 3: The Miquel Circle

[12] quotes the problem as hard for the Descartes-Wu approach and proposes a more geometric solution within the geometric framework developed in that paper. The problem has even a constructive solution: Take four points $A_1, \ldots, A_4$ on a circle around the origin O, centers $M_{12}, \ldots, M_{41}$ for circles passing through $(A_1, A_2), \ldots, (A_4, A_1)$ and compute for each consecutive pair of such circles the second intersection point (it has rational coordinates in the parameters $u_1, \ldots, v_4$).

Here is the *GeoCode* formulation:

```
<prooftype> constructive </prooftype>
<parameters> [u1, u2, u3, v1, v2, v3, v4]   </parameters>
<Points>    $O:=Point[0,0]; $A1:=Point[1,0];    </Points>
<coordinates>
$A2:=circle_slider[$O,$A1,u1];
$A3:=circle_slider[$O,$A1,u2];
$A4:=circle_slider[$O,$A1,u3];

$M12:=line_slider[p_bisector[$A1,$A2],v1];
```

[3] Our Linux version of Maple 8 yields an empty solution set for `solve`$(x_3^2 - t^2, x_3)$ if the *GeoProver* package is loaded.

```
$M23:=line_slider[p_bisector[$A2,$A3],v2];
$M34:=line_slider[p_bisector[$A3,$A4],v3];
$M41:=line_slider[p_bisector[$A4,$A1],v4];

$c12:=pc_circle[$M12,$A1];
$c23:=pc_circle[$M23,$A2];
$c34:=pc_circle[$M34,$A3];
$c41:=pc_circle[$M41,$A4];

$B1:=other_cc_point[$A1,$c12,$c41];
$B2:=other_cc_point[$A2,$c12,$c23];
$B3:=other_cc_point[$A3,$c23,$c34];
$B4:=other_cc_point[$A4,$c34,$c41];
</coordinates>
<conclusion>
$result:=is_concyclic[$B1, $B2, $B3, $B4];
</conclusion>
```

The simplification of the resulting rational expression in seven parameters indeed turned out to be very hard and no one of the CAS (Maple, MuPAD, Reduce) mastered the task. The situation completely changes if $v_1, \ldots, v_4$ are assigned random integers (not too big, < 100). Several runs with differrent settings always yield 0 (where Maple simplified the new expression much faster than MuPAD or Reduce).

5 Implementing the *GeoCode* Standard

For real usability of the *GeoCode* concept one has to estimate the efforts required to implement this standard. Even though there is not yet practical experience with already existing geometry theorem provers the semantic similarity of "foreign" special geometric proof schemes suggests that such an interface could easily be implemented. We suggest to divide that implementation in two parts as described for the *GeoProver* packages for different target CAS: The first part provides (e.g., Perl based) tools to translate GEO proof schemes into a CAS specific form that fixes requirements of naming and syntax conventions. In a second part these translated proof schemes are passed to a special interface that maps the *GeoCode* functionality to the target CAS.

In general, for the second part one has to implement the *GeoCode* functionality in the target prover language. This requires extensibility of that language and access to the source code or interaction with the system developers if such an interface cannot be added as a supplementary package.

Note that the *GeoCode* syntax provides not only points but also line and circle objects as geometric primitives. It is a special design decision of many geometry theorem provers, e.g., D.Wang's GEOTHER [14], not to introduce the latter objects as basic but to take only point objects as primitives. A prover extension that respects this spirit can implement lines and circles as derived objects and represent lines by two base points and circles by point-center pairs. This is conceptually already present in Wang's prover and merely should be made explicit. To support such an approach we removed direct constructors for lines

and circles from their homogeneous coordinates (i.e., the constructors `Line`$[a_1, a_2, a_3]$ and `Circle`$[a_0, a_1, a_2, a_3]$) from the *GeoCode* standard in version 1.2.

A more serious problem arises with geometry theorem provers that do not support nested function calls. This is typical for systems that contain a drawing tool, since all intermediate construction steps leave their trace in a picture. Usually such systems represent and address geometric objects through explicit identifiers. To fit generic *GeoCode* proof schemes with such a prover nested function calls should be denested. We experimented with a *GeoCode* interface to dynamical geometry software (DGS) that uses the Perl 'eval' mechanism to evaluate nested *GeoCode* function calls and collects these calls as a list of construction steps in a generic format. Later on these construction steps can be mapped to a special DGS, e.g., the system GEONE$_X$T, [6], developed at the University of Bayreuth.

Experiments with different kinds of tools that support implementations entailed changes of the *GeoCode* standard in the past and will entail new requirements and changes also in the future. So far the power of the Perl string manipulation facilities and the *SymbolicData* actions concept were well suited to support such changes, to scan the GEO records for obsolete proof scheme commands and to fix them accordingly.

References

[1] O. Bachmann and H.-G. Gräbe. The *SymbolicData* Project: Towards an electronic repository of tools and data for benchmarks of computer algebra software. Reports on Computer Algebra 27, Jan 2000. Centre for Computer Algebra, University of Kaiserslautern. See `http://www.mathematik.uni-kl.de/~zca`.

[2] S.-C. Chou. *Mechanical Geometry Theorem Proving*. Reidel, Dortrecht, 1988.

[3] S.-C. Chou, X.-S. Gao, and J.-Z. Zhang. *Machine proofs in geometry*, volume 6 of *Series on Applied Mathematics*. World Scientific Singapore, 1994.

[4] X.-S. Gao, D. Wang, and L. Yang, editors. *Automated Deduction in Geometry, Beijing 1998*, volume 1669 of *Lect. Notes Comp. Sci.* Springer, 1999.

[5] X.-S. Gao et al. *Geometry Expert* - a software for dynamic diagram drawing and automated geometry theorem proving and discovering, 2002. See `http://www.mmrc.iss.ac.cn/~xgao/gex.html`.

[6] GEONE$_X$T – a dynamical geometry software, 1998–2002. Lehrstuhl für Mathematik und ihre Didaktik, Univ. Bayreuth. See `http://www.geonext.de`.

[7] H.-G. Gräbe. *GeoProver* - a small package for mechanized plane geometry, 1998–2002. With versions for Reduce, Maple, MuPAD and Mathematica. Some prototypes were compiled in cooperation with M. Witte. See `http://www.informatik.uni-leipzig.de/~compalg/software`.

[8] H.-G. Gräbe. The *SymbolicData* benchmark problems collection of polynomial systems. In *Proceedings of the Workshop on Under- and Overdetermined Systems of Algebraic or Differential Equations, Karlsruhe 2002*, pages 57 – 75, 2002. Publ. by IAS Karlsruhe. See also `http://www.informatik.uni-leipzig.de/~graebe/publications`.

[9] H.-G. Gräbe. The *SymbolicData* geometry collection and the *GeoProver* packages. In *Proceedings "8th Rhine Workshop on Computer Algebra" (RWCA-02), Mannheim 2002*, pages 173 – 194, 2002. Publ. by Univ. Mannheim. See also `http://www.informatik.uni-leipzig.de/~graebe/publications`.

[10] The International Mathematical Olympiads, since 1959. See, e.g., `http://www.kalva.demon.co.uk/imo.html`.

[11] D. Kapur. Automated geometric reasoning: Dixon resultants, Gröbner bases, and characteristic sets. In Wang [15], pages 1 – 36.

[12] C.-Z. Li and J.-Z. Zhang. Readable machine solving in geometry and ICAI software MSG. In Gao et al. [4], pages 67 – 85.

[13] The *SymbolicData* Project, 2000–2002. See `http://www.SymbolicData.org` or the mirror at `http://symbolicdata.uni-leipzig.de`.

[14] D. Wang. *GEOTHER* - geometry theorem prover, 1990–2001. See `http://calfor.lip6.fr/~wang/GEOTHER`.

[15] D. Wang, editor. *Automated Deduction in Geometry, Toulouse 1996*, volume 1360 of *Lect. Notes Comp. Sci.* Springer, 1996.

[16] D. Wang. GEOTHER: A geometry theorem prover. In M.A. McRobbie and J.K. Slaney, editors, *Automated deduction – CADE-13*, volume 1104 of *LNCS*, pages 166 – 170, 1996.

[17] D. Wang. Clifford algebraic calculus for geometric reasoning with applications to computer vision. In Wang [15], pages 115 – 140.

[18] D. Wang. *Elimination Methods*. Texts and Monographs in Symbolic Computation. Springer, Wien, 2001.

[19] W.-T. Wu. On the decision problem and the mechanization of theorem-proving in elementary geometry. In *Contemp. Math.*, volume 19, pages 213 – 234. AMS, Providence, Rhode Island, 1984.

[20] W.-T. Wu. Some recent advances in mechanical theorem proving of geometry. In *Contemp. Math.*, volume 19, pages 235 – 241. AMS, Providence, Rhode Island, 1984.

[21] W.-T. Wu. *Mechanical Theorem Proving in Geometries*. Texts and Monographs in Symbolic Computation. Springer, Wien, 1994.

[22] W.-T. Wu. Automatic geometry theorem-proving and automatic geometry problem solving. In Gao et al. [4], pages 1 – 13.

[23] W.-T. Wu. *Mathematics Mechanization*, volume 489 of *Mathematics and its Applications*. Science Press, Beijing, and Kluwer Acad. Publ., Dordrecht, 2000.

```
########################################################################
<Type>      GEO                              </Type>
<Key>       Cathedral_1                      </Key>
<prooftype> equational, deduction            </prooftype>
<vars>      [x1,x2,r]                        </vars>
<Points>
$O:=Point[0,x1]; $X:=Point[0,x2];
$A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
</Points>
<coordinates>
$c1:=pc_circle[$A,$M]; $c2:=pc_circle[$Q,$A]; $c3:=pc_circle[$O,$X];
</coordinates>
<polynomials>
$polys:=List[is_cc_tangent[$c1,$c3],is_cc_tangent[$c2,$c3],r-(x1-x2)];
</polynomials>

<solution>
$result:=geo_solve[geo_eliminate[$polys,$vars,List[x1,x2]],r];
</solution>
########################################################################
```

A proof scheme for the Cathedral example [11, 5.3] as GEO record