The *SymbolicData* Proof Scheme Collection as a Multiplatform Project

Hans-Gert Gräbe, Univ. Leipzig, Germany

June 1, 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 up to the International Mathematics Olympiad (IMO) [9].

The great range of ideas involved in 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 their 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 (*explicitly constructive* in [23]), where the geometric configuration can be described 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 to 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 simplification problem is well understood and admits an efficient solution that is implemented in the core of all major (and minor) Computer Algebra Systems. 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. Note that new approaches as, e.g., the Clifford algebra method discussed in [25], also address the class of constructive problems.

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 these *proof schemes* or *coordinatizations*, i.e., descriptions of geometric configurations of points, lines, and circles in a syntactically strong way, are invented by the author 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 titles of [21] and [23]) 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, see, e.g., [3].

Mechanization of geometry theorem proving hence requires the creativity of diligent "proof writers" that carry out this translation process to eliminate all informal elements in a geometric statement and fix the result in a (geometric) proof scheme. The (formal) target language to be used for these efforts is inherently of geometric nature. The resulting geometric proof schemes are the starting point for further automated translation to, e.g., algebraic statements.

Even though different authors use different languages and different commands to write down their proof schemes the interface parts of these languages are very similar not only if applications of Descartes-Wu's algebraization method are considered as, e.g., in [2], but also in applications of Clifford algebra techniques as, e.g., in [17] or even in non-Euclidean geometry as, e.g., in [24]. For intercommunication purposes and to store proof schemes in a common publicly available repository it is necessary to make these similarities more explicit and to invent a *generic proof scheme standard* as a common interface that can be implemented with appropriate tools by all interested parties.

In this paper I describe a first approach to such a generic proof scheme standard, the *GeoCode*. It is used to store electronically 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 Chou's book [2]. Special *SymbolicData* tools are designed to support the syntactical translation of *GeoCode* into special proof scheme languages that support this common interface. At the moment – as a first reference application – this interface is implemented in the author's *GeoProver* packages [6], that provide tools to run translated proof schemes on one of the major CAS Maple, Mathematica, MuPAD, or Reduce.

This paper starts with some background on geometry theorem proving and a short overview about the functionality of the *GeoProver* packages (sections 2 and 3). Then I describe the design of the GEO records and the syntax of the *GeoCode* standard (section 4). With these prerequisites at hand it is easy to compile new proof schemes, to translate them into their *GeoProver* algebraic counterparts for different CAS and experiment with these algebraic problems (section 5).

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 should easily to be implemented. The problem will be discussed in more detail in section 6 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].

During our experiments with the *GeoProver* interface the *GeoCode* standard did already evolve and change. Of course, it needs further experiments to fix the standard in such a way that it can reliably be implemented on different platforms. Since the *GeoCode* description is fixed in the same *SymbolicData* format as the GEO records themselves it can as easily be revised and extended with *SymbolicData* tools. This also will be discussed in section 6.

Similar questions to formalize rather informal geometric statements arise in the design of dynamical geometry software (DGS). To store and draw geometric configurations requires their formal description as *construction schemes*. At a first glance, the design solutions of different DGS (e.g., Cinderella [12] or Geonext [5]) – they use strong XML syntax – look very different from *GeoCode* proof schemes. Construction schemes and proof schemes differ since the former handle and store also intermediate construction steps. Construction schemes are close to human readable *construction descriptions* and Cinderella and Geonext provide tools to extract such a description from a geometric configuration that was compiled by 'click and drop'. Note that this is not a step towards 'intelligent computers' since 'click and drop' already requires some formalization of geometric statements: a construction plan must be set up ahead.

Despite of this difference there is much similarity between proof and construction schemes. D.Wang implemented within his GEOTHER project an interface that extracts human readable descriptions of geometric configurations and generates plots (i.e., construction schemes) from his proof schemes. The same way it should be possible at least for constructive proof schemes to translate their *GeoCode* representation into construction schemes and construction descriptions. A first (Perl based) prototype interface to Geonext is already running.

It is a task of future research to clarify the relation between (*GeoCode*) proof schemes, construction schemes and construction descriptions in more detail.

2 Geometry Theorem Proving

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 algebraic translation of geometric statements can be produced cascading only a small number of elementary functions and data types.

We write $P = \mathtt{Point}(x, y)$ for a point with coordinates (x, y), $g = \mathtt{Line}(g_1, g_2, g_3)$ for the line

$$\{(x, y)\ :\ g_1\, x + g_2\, y + g_3 = 0\}$$

and $c = \mathtt{Circle}(c_1, c_2, c_3, c_4)$ for the circle

$$\{(x, y)\ :\ c_1\,(x^2 + y^2) + c_2\, x + c_3\, y + c_4 = 0\}.$$

Note that the coordinates of lines and circles are *homogeneous* and defined by the corresponding geometric objects only up to a scalar factor.

For example, to prove the centroid intersection theorem choose points

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

with generic (i.e., symbolic) coordinates, compute

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

and evaluate the statement

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

where $\mathtt{midpoint}(X, Y)$ returns (a formula for) the midpoint of the line XY, $\mathtt{pp\_line}(X, Y)$ computes the homogeneous coordinates of the line through X and Y and $\mathtt{is\_concurrent}(a, b, c)$ returns a polynomial in the coordinates of the lines a, b, c (in fact, a determinantal expression) that vanishes iff these lines 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.

To prove a geometry theorem of this type means to compose the nested rational expression (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) since their coordinates usually cannot be rationally expressed in the basic coordinates. 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 line 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 $g = \texttt{Line}(g_1, g_2, g_3)$ and $h = \texttt{Line}(h_1, h_2, h_3)$ that can be computed as

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

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 for P to be on either the inner or the outer bisector of $\angle ABC$. Note that in our algebraization of 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 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:={ is_point_on_bisector(P,A,B,C), is_point_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});
```

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

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

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, lets ask if the conclusion polynomial

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

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

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 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_α contains 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 [10] or the monographs [18, 23] for a more complete survey.

3 The *GeoProver* Packages

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

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

(2) We need tools to translate geometric statements into their algebraic counterparts.

(3) We need a "proof writer" that combines these tools and tries to write (realistic) proof schemes for given geometry theorems.

(4) The CAS should be able 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 (1) 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 of the same problem can yield algebraic formulations of very different run time also within the same CAS.

3.1 About the *GeoProver*

The *GeoProver* (formerly *Geometry*) provides tools for topic (2). It is a small package for mechanized translation of statements of (plane) geometry to algebraic statements with non degeneracy tracing, available for different CAS platforms (Maple, MuPAD, Mathematica, and Reduce) that provides a set of functions to cope with generic and special geometric configurations containing points, lines and circles as introduced above. For details I refer to the sample calculations in the previous section and the documentation [6] of the package. For some target systems there exists also a plot extension that allows to draw graphics from scenes, i.e., (of course special) geometric configurations.

Altogether the package provides the casual user with a couple of procedures that allow him / her to mechanize his / her own geometry proofs. A first prototype grew out from a course of lectures for students of computer science on this topic held 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.

For version 1.2 I prepared a scheme that uses the *SymbolicData* tools to handle the versions for different platforms in a unique way. This generic management of the source code uses many ideas approved during the compilation of the *SymbolicData* GEO records.

As of today, also the *GeoCode* description was separated from the *GeoProver* implementation, fixed in *SymbolicData* format as the `GeoCode` table and added to the *SymbolicData* Project. Some *GeoCode* functions ('macros') have default *GeoCode* implementations that are translated into platform-specific code if the *GeoProver* packages are created. The other functions of the interface are implemented 'inline' in different platform-specific code packages.

Note that for version 1.2 not only the package name changed (to avoid name clashes with another Maple package called 'geometry') but also the *GeoCode* specification was revised.

3.2 Writing Mechanized Geometry Proofs

In most cases topic (3) is straightforward, in particular if the 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 (forthcomig) 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 intersect at a point C such that d and e are the bisectors of ABC through A and B.
- Compute `is_point_on_bisector`(P, B, C, A), i.e., prove that P is also on the third bisector.

Here is the proof scheme in a form that can be run with the MuPAD *GeoProver* version.

```
A:=Point(0,0); B:=Point(1,0); P:=Point(u1,u2);
c:=pp_line(A,B); d:=pp_line(A,P); e:=pp_line(B,P);
u:=sym_line(c,d); v:=sym_line(c,e);
C:=intersection_point(u,v);
is_point_on_bisector(P,B,C,A);
```

Letter by letter the same proof scheme works also for Maple. Reduce (in the default settings) does not distinguish between up-case and down-case letters and reports e to be protected. Mathematica requires square brackets and does not accept underscores as valid letters for function names. Generic code handling has to clear all these obstacles. Our solution will be described below.

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) more than 200 of them. We collected also solutions of geometry problems from other sources, e.g., the IMO contests, see [9]. Much of this work was done by my "proof writers", i.e., the students Malte Witte and Ben Friedrich who compiled first electronic versions for many of these examples.

4 *GeoCode* and GEO Records

4.1 The *SymbolicData* Project

The *SymbolicData* project was set up to create and manage a publicly available repository of digital 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. I refer to [1, 7, 8] and the *SymbolicData* documentation for more details.

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. I refer to that source for more information about the *SymbolicData* project. We kindly acknowledge support also from UMS MEDICIS of CNR/École Polytechnique (France) who provides us with the needed hard- and software for establishing and running this web site.

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 (via HTML download from our Web site) under the terms of the GNU Public License.

4.2 *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 proof scheme for the geometric statement that 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 and give a first flavor about *GeoCode* syntax.

Note that the description of the internal structure of these attributes is given in the same sd-file format in a `META/GEO` table. Hence new attributes can be added and existing attributes can be modified in the same way as for data records.

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 unspecific programming overhead to a minimum. Note that 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.

4.3 *SymbolicData* Proof Schemes

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

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 following attributes (with *GeoCode* values) are mandatory:

`coordinates`	assignments that construct step by step the generic geometric configuration of the proof scheme
`conclusion`	the conclusion of the proof scheme (optional if proof type is `deduction`)

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

<Id>         GEO/Parallelogram_2            </Id>
<Type>       GEO                            </Type>
<Key>        Parallelogram_2                </Key>
<prooftype> constructive                    </prooftype>
<parameters> [u1, u2, u3]                   </parameters>
<coordinates>
$A:=Point[0,0]; $B:=Point[u1,0]; $D:=Point[u2,u3];
$C:=intersection_point[par_line[$D,pp_line[$A,$B]],
  par_line[$B,pp_line[$A,$D]]];
$P:=intersection_point[pp_line[$A,$C],pp_line[$B,$D]];
</coordinates>
<conclusion>
$result:=sqrdist[$A,$P]-sqrdist[$C,$P];
</conclusion>
<CRef>
PROBLEMS/Geometry/Parallelogram => problem description
</CRef>
<Comment>
Feb 10 2002 graebe: translated to GeoProver 1.2 syntax
</Comment>
<Version> ... </Version>
<PERSON>     graebe                         </PERSON>
<Date>       Nov 1 1999                     </Date>

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

The GEO record 'Parallelogram_2'

This already completes the data required for a constructive proof scheme. For equational proof schemes the following additional (optional) attributes with *GeoCode* values are defined:

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

The proof idea can be sketched within the `ProofIdea` attribute as plain text if not yet evident from the code.

I refer to [8] for a detailed description of the GEO record structure and concentrate in the following on the *GeoCode* part.

4.4 The *GeoCode* Syntax

The invention of a generic *GeoCode* language is mainly motivated by our aim to run geometry theorem proof schemes on different target CAS. 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 could avoid these efforts so far and define the generic language in such a way, that it can be cross compiled using only regular patterns. Due to its elaborated pattern matching facilities Perl is best suited to realize this approach.

Since proof schemes are composed by a sequence of assignments with nested function calls as right hand sides that refer to previously defined geometric objects and rational expressions as arguments the *GeoCode* language should meet the following requirements:

(1) Due to different naming conventions of the target CAS it should be possible to identify (and substitute) variable, symbol and function names.

(2) The generic *GeoCode* should easily be mapped to the syntax of the target CAS without name clashes.

(3) Optionally, it should extend to a concept not only to translate the proof scheme, but also to run and evaluate it on the target CAS. For equational proof schemes this requires additional efforts to give a "generic" solution for the algebraic part of the problem.

For (1) note that in this context the words 'variable' and 'symbol' are used in the following meaning: 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 in function calls. Since this cannot be distinguished within a regular language we use the Mathematica convention (i.e., brackets) for function call notation.

For (3) note that GEO records use an extension of the *GeoCode* language to fix solutions of such algebraic tasks in a way that can easily be translated to different target CAS. Interface packages for Maple, MuPAD, Mathematica, and Reduce that map these generic commands to appropriate constructs are part of the *SymbolicData* distribution. For details see [8].

This gives the main outline of the *GeoCode* syntax. The names and signatures of the functions that are available within the *GeoCode* standard are stored in the *SymbolicData* `GeoCode` table and hence can be extracted, extended and modified in the same way as other sd-records. Two such `GeoCode` records are reproduced on page 13. The first one corresponds to an 'inline' function that requires a special implementation, the second one to a 'macro' where a definition is given in *GeoCode* syntax as value of a special attribute `code` that can be used to create an implementation automatically. For a complete description of all functions I refer to the *SymbolicData* `GeoCode` documentation.

5 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.

1. The "cathedral example", [10, 5.3]

We use the problem formulation and notational conventions from [10] 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 consists of 6 conditions that arize from two tangency conditions of circles and yield 3 polynomials each: 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'.

In *GeoCode* syntax this may be written in the following way:

```
<vars>       [x1,y1,x2,y2,y3,r]                             </vars>
<coordinates>
$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];
$c1:=pc_circle[$A,$M]; $c2:=pc_circle[$Q,$A];
</coordinates>
<polynomials>
$polys:=List[
is_point_on_circle[$T1,$c1], is_collinear[$A,$T1,$O], r^2-sqrdist[$O,$T1],
is_point_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 line and circle objects since they are widely used, e.g., by most DGS systems. Note that `is_point_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.

```
#########################################################
# 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'

The solution of the algebraic problem (four different values of r) 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:

```
<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 corresponds to rational parametrizations of such points:

```
<vars>      [x1,x2,y3,r]                          </vars>
<coordinates>
$O:=Point[0,y3]; $A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
$T1:=circle_slider[$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 `polynomials` 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 `solve($polys,$vars)` with MuPAD 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. This approach 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 x_1 and x_2

```
$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 20.

To translate and run that code with MuPAD we call the *SymbolicData* `MuPADCode` action. It mainly maps *GeoCode* syntax to *GeoProver* MuPAD syntax, resolves name clashes and yields (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 the `MuPADCode` 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 [10]. 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 [10] but not discussed there. It corresponds to imaginary coordinates of O and hence is virtual.

A similar solution admits the question for the length of the radius of the circle in the top region of the picture in [10, 5.3]. Take the origin at D and the center O and a circumfere point X of that circle on the y-axis. For the other points see [10]. We get the proof scheme

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

that may be translated and solved in a similar way. 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]

The statement is the following:

> Take three points A_2, B_2, C_2 respectively on the three midpoint perpendiculars of BC, AC, AB of any triangle ABC such that
>
> $$d(A_2, BC) = t \cdot |BC|, d(B_2, AC) = t \cdot |AC|, d(C2, AB) = t \cdot |AB|,$$
>
> 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.

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 are cross referenced in `GEO` records since several proof schemes may refer to the same problem.

Wang proves the generalized Steiner theorem 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>
<coordinates>
$A:=Point[0,0]; $B:=Point[0,1]; $C:=Point[u1,u2];
$A1:=midpoint[$B,$C]; $B1:=midpoint[$A,$C]; $C1:=midpoint[$A,$B];
$A2:=line_slider[midpoint_perpendicular[$B,$C],x1];
$B2:=line_slider[midpoint_perpendicular[$A,$C],x2];
$C2:=line_slider[midpoint_perpendicular[$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.

The proof scheme is not at the best for the given statement since one has to expect that the theorem is true only for two of the 8 possible combinations. I checked this for Maple (7.0), MuPAD (2.0), and Reduce (3.7) with the solution

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

It calls the `solve` command of the underlying CAS, substitutes the different solutions into the conclusion expression and tries to simplify that result. The solution 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 created expressions with several root symbols that afterwards could not be completely simplified.

3. The Miquel Circle, [11, Ex. 5]

The statement is the following:

> If four circles are arranged in a (cyclic) sequence, each two successive circles intersecting, and another circle passes through one of each such intersection points, then the remaining intersection points lie on another circle.

[11] quotes the problem to be 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). Here is the *GeoCode* formulation:

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

$M12:=line_slider[midpoint_perpendicular[$A1,$A2],v1];
$M23:=line_slider[midpoint_perpendicular[$A2,$A3],v2];
$M34:=line_slider[midpoint_perpendicular[$A3,$A4],v3];
$M41:=line_slider[midpoint_perpendicular[$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>
$result:=is_p4_circle[$B1, $B2, $B3, $B4];
</conclusion>
```

The simplification of the resulting rational expression in seven parameters is indeed very hard and no one of the benchmarked 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 different settings always yield zero for the remaining simplification problem of a nested rational expression in the three variables u_1, u_2, u_3 thus "proving" the theorem by a stochastic approach. Note that Maple simplified the new expressions much faster than MuPAD or Reduce.

6 The *GeoCode* Standard Revised and Implemented

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. This can be seen, e.g., for the proof scheme commands used in [17] and, in a slightly revised form, also in D.Wang's GEOTHER project [14, 16]. We take this prover in the following considerations as a prototypical target.

Here is a table that collects a couple of commands from [17] and their *GeoCode* translations:

```
on_line(P1,P2)             line_slider[pp_line[P1,P2],u]
par(P1,P2,P3)              line_slider[par_line[P3,pp_line[P1,P2]],u]
per(P1,P2,P3)              line_slider[ortho_line[P3,pp_line[P1,P2]],u]
on_cir(P1,P2,P3)           circle_slider[circle_center[
                           p3_circle[P1,P2,P3]], P1, u]
midp(P1,P2)                midpoint[P1,P2]
per_ft(P1,P2,P3)           pedalpoint[P3,pp_line[P1,P2]]
cir_ctr(P1,P2,P3)          circle_center[p3_circle[P1,P2,P3]]
par_int(P1,P2,P3,P4,P5)    intersection_point[pp_line[P3,P4],
                           par_line[P5, pp_line[P1,P2]]]
per_int(P1,P2,P3,P4,P5)    intersection_point[pp_line[P3,P4],
                           ortho_line[P5, pp_line[P1,P2]]]
```

Hence proof schemes compiled for Wang's prover could easily be rewritten in *GeoCode* syntax.

The other direction, i.e., to implement the full *GeoCode* standard for Wang's prover and thus to run within that prover (translated) proof schemes from the *SymbolicData* GEO collection, requires some more efforts.

First, the prover should provide also line and circle objects. It was a special design decision of Wang's prover not to introduce such objects as basics 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. The same concept is present in most of the DGS systems and hence worth to be respected more clearly within the *GeoCode* standard.

On the other hand, such an implementational detail poses some restrictions on the *GeoCode* standard itself: Direct constructors for lines and circles from their homogeneous coordinates (i.e., the constructors Line$[a_0, a_1, a_2]$ and Circle$[a_0, a_1, a_2, a_3]$) should be removed from the standard in favour of the point-point (pp_line) and point-center (pc_circle) constructors for lines and circles. Note that based on our experiments with Geonext [5] the command pc_circle (with the current syntax) was added to the *GeoCode* standard during the revision process along the compilation of version 1.2 of the *GeoProver*. This gives already some experience with such adjustments. Of course, all GEO records should be scanned for obsolete proof scheme commands and fixed accordingly.

Second, the prover should implement the complete *GeoCode* interface. The methods in this interface can be divided into two types of functions, inline functions and macros. Inline functions require a special implementation that depends upon the target system and the approach. Macros come with a *GeoCode* default (functional) implementation that can be compiled to the target system with appropriate tools. The structure of the *GeoCode* storage system as GeoCode table of the *SymbolicData* project can support this implementation process.

It allows also to add new macros in an easy way. For example, since the sentence "construct the circumcenter of the triangle ABC" can be decoded as "construct the intersection point of the midpoint perpendiculars of AB and AC", a macro circumcenter corresponds to the nested *GeoCode* call

```
circumcenter[$A,$B,$C] =
   intersection_point[midpoint_perpendicular[$A,$B],
     midpoint_perpendicular[$A,$C]]
```

Hence the *GeoCode* standard can be (and was already) extended with that function merely adding a new sd-record circumcenter (see page 20) to the GeoCode table in the *SymbolicData* data base. Sinve the *GeoProver* packages are compiled from these sources a recompilation automatically adds the new functionality to that prover for each of the supported platform CAS.

In the same way the *GeoCode* syntax could be extended with all original commands from Wang's prover. This, of course, is not a good idea, since syntactically different functions with similar semantics blow up the interface unnecessarily and don't allow to identify semantic proof ideas by syntactical search. The latter question, i.e., (re)definition of the *GeoCode* standard in such a way that it supports also complete navigation in the proof scheme collection to find proofs that address specific geometric configurations, needs further investigation.

```
##########################################################
<Type>      GEO                              </Type>
<Key>       Cathedral_1                      </Key>
<prooftype> equational, deduction            </prooftype>
<vars>      [x1,x2,r]                        </vars>
<coordinates>
$O:=Point[0,x1]; $X:=Point[0,x2];
$A:=Point[-1/4,0]; $Q:=Point[7/12,0]; $M:=Point[0,0];
$c1:=pc_circle[$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 [10, 5.3] as GEO record

```
###########################################################
# Record 'GeoCode/circumcenter'

<Id>        GeoCode/circumcenter          </Id>
<Type>      GeoCode                       </Type>
<Key>       circumcenter                  </Key>
<call>      circumcenter[$A::Point,$B::Point,$C::Point]::Point </call>
<verbose>   circumcenter of the triangle ABC </verbose>
<code>
intersection_point[midpoint_perpendicular[$A,$B,$C],
midpoint_perpendicular[$B,$C,$A]]
</code>
<description>
The circumcenter of the triangle <math>ABC</math>.
</description>
<Version>   ...                           </Version>
<PERSON>    graebe                        </PERSON>
<Date>      May 28 2002                   </Date>

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

A new record 'GeoCode/circumcenter' for the `GeoCode` data base

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] K. Fuchs-Kittowski. Wissens-Ko-Produktion. Verarbeitung, Verteilung und Entstehung von Informationen in kreativ-lernenden Organisationen. In C. Floyd and W. Hofkirchner, editors, *Stufen zur Informationsgesellschaft für alle. Festschrift zum 65. Geburtstag Klaus Fuchs-Kittowskis.* Peter Lang-Verlag, Frankfurt, 2001.

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

[6] 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`.

[7] 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. Publ. by IAS Karlsruhe, see also `http://www.informatik.uni-leipzig.de/~graebe/publications`.

[8] 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. Publ. by Univ. Mannheim, see also `http://www.informatik.uni-leipzig.de/~graebe/publications`.

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

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

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

[12] J. Richter-Gebert and U. Kortenkamp. Cinderella – the interactive geometry software, 1997–2002. See `http://www.cinderella.de`.

[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 Mechanisation*, volume 489 of *Mathematica and Its Applications.* Science Press, Beijing, and Kluwer Academic Publishers, Dordrecht, 2000.

[24] L. Yang, X.-S. Gao, S.-C. Chou, and J.-Z. Zhang. Automated production of readable proofs for theorems in non-euclidean geometries. In Wang [15], pages 171 – 188.

[25] L. Yang, X. Hou, and B. Xia. Automated discovering and proving for geometric inequalities. In Gao et al. [4], pages 30 – 46.