The *SymbolicData* Geometry Collection and the GeoProver Packages

http://www.symbolicdata.org

Hans-Gert Gräbe
Department of Computer Science
University of Leipzig, Germany
graebe@informatik.uni-leipzig.de

February 22, 2002

In: *Proceedings "8th Rhine Workshop on Computer Algebra"* (RWCA-02), March 2002 in Mannheim, pp. 173 - 194, Publ. by Univ. Mannheim

1 Introduction

1.1 Benchmarking Symbolic Computations

After the discussions in a section at ISSAC'98 in Rostock about symbolic computations benchmarking activities we started to collect and evaluate benchmark material from several sources and became aware of some problems related with the state of the art of such benchmarks at those times. Two of them are central:

1. Most of the benchmark data exists only in printed form, often with misprints, and is available to the public only with difficulties (often enough since the author changed her/his place of work or field of interests) and in very different formats.
2. There is no agreed upon form, how to set up and evaluate such benchmarks: What is a fair measure of computing time? How to compare the quality of the answers beyond CPU time? etc.

A reliable answer to the first problem would be a widely accessible central digital repository where people can store and publish (at least) their benchmark input data for comparison and reusage by other research groups. This requires not only to set up such a repository but also to agree upon a common data exchange format and to develop (and provide) tools to store and retrieve data into/from that format. The main part of such tools can be developed once and reused by interested parties since it requires (general) text processing and translating facilities rather than (special) symbolic computational power.

For the second question note that similar problems arise if software for symbolic computations is tested. Tests of symbolic software (beyond early stage 'copy and paste') require to screen large sets of data using batch processing and special test beds. The test bed environment should prepare data for input to the tested software, start and monitor its run, and store and evaluate the output of the computation, i.e., has to provide tools with similar functionality as required for the repository.

Note that the same tools are useful also for the collection and presentation of benchmark output data.

1.2 The *SymbolicData* Project

The *SymbolicData* project was set up to unify the efforts of interested people in that direction. 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 diligent 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, but that topic will not be discussed in this paper. We refer to [1, 6] and the *SymbolicData* documentation for more information.

The *SymbolicData* project is part of the benchmark activities of the German "Fachgruppe Computeralgebra" who also sponsored the web site [8] as a host for presentation and download of the tools and data developed and collected so far. We 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.

1.3 This Paper

This paper gives an introduction into the *SymbolicData* collection of examples from geometry theorem proving (GEO records) and the GeoProver packages [5] that provide software capable really to run (with one of the major computer algebra systems Maple, MuPAD, Mathematica, or Reduce) these geometry theorem proofs.

We give some background on geometry theorem proving and a short overview about the functionality of the GeoProver packages (sections 2 and 3). Then we describe the design of the GEO records and the syntax of the generic GEO code that can be translated to the different target languages (section 4).

The *SymbolicData* tools can be used not only to manage symbolic benchmark data but also to manage source code and documentation of a project like the GeoProver. This non standard application of the *SymbolicData* tools (with an alternative data base) will be discussed in section 5.

We conclude (section 6) with some remarks about the efforts required to really set up benchmark computations with the GeoProver on different platforms (Maple, MuPAD, Mathematica, Reduce) and this GEO data.

2 Geometry Theorem Proving

Many hard problems in polynomial system solving arise from automated proofs of (elementary plane) geometric problems. Synthetic geometry proofs usually involve tricky arguments that require a lot of experience and creativity to be found. It was an old dream to mechanize such proofs, and already Fermat knew a general approach: introduce coordinates, translate all statements into algebraic formulas and try to solve the corresponding algebraic problem by algebraic methods.

The attempts to algorithmize this part of mathematics found their culmination in the 80's in the work of W.-T. Wu [11] on "the Chinese Prover" and the fundamental book [3] of S.-C. Chou who proved 512 geometry theorems with this mechanized method, see also [2, 4, 9, 10].

It is a surprising fact that tedious but mostly straightforward manipulations of the algebraic counterparts of geometric statements allow to prove many theorems in geometry with even ingenious "true geometric" proofs. Supported by a Computer Algebra System (CAS) for the algebraic manipulations part this approach obtains new power. The method is not automatic, since one often needs a good feeling how to encode a problem efficiently, but mechanized in the sense that one can develop a tool box to support this encoding and some very standard tools to derive a (mathematically strong!) proof from these encoded data.

Such a tool box is provided by the GeoProver packages [5] that are available for Reduce, Maple, MuPAD and Mathematica. A generalized syntax that can be mapped to each of the target languages is used to store the proof schemes in the *SymbolicData* GEO table.

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

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

and $c = \texttt{Circle}(c_1, c_2, c_3, c_4)$ 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 upto a scalar factor.

For example, to prove the centroid intersection theorem choose points

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

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

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

and evaluate the statement

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

where $\texttt{midpoint}(X, Y)$ returns (a formula for) the midpoint of the line XY, $\texttt{pp\_line}(X, Y)$ computes the homogeneous coordinates of the line through X and Y and $\texttt{is\_concurrent}(a, b, c)$ returns a polynomial in the coordinates of the lines a, b, c (in fact, a determinantal expression) that vanishes iff these lines meet at 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*[1], 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 of the coordinates of objects already available or algebraically independent variables, and the conclusion can be expressed as vanishing of a rational function in the coordinates of the available geometric objects.

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

[1]This notion is different from [3].

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

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

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 we "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)};
```

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

`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});
```

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

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

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, we 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 $Z(F)$ defined by 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.

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.)

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 (plane) geometry manipulations 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.

We don't give here a formal description of all these functions but refer the interested reader to the sample calculations in the previous section and the documentation [5] of the package. For some target systems there is 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 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.

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. I come back to that topic below.

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 names of the procedures were completely 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 [11] 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.

- 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 written down in the MuPAD version of the GeoProver language.

```
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 [3] 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 [7]. 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 GEO Records and GEO Code

4.1 About the *SymbolicData* Data Base Structure

We mentioned already in the introduction that 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 11. It contains information and a mechanized proof scheme for the following geometry theorem:

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

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

The sd-files are tight to Perl hashes (**sd-records**) by the *SymbolicData* tools in a transparent way. Hence additional Perl programming required for benchmark activities (the *SymbolicData* Compute environment is still under development) can easily access the values of the different attributes of a record. In section 6 we describe a benchmark computation on GEO records that gives a real estimation of additional Perl programming efforts required to set up such a computation. Note that since the detailed requirements of such a computation are almost unknown to the *SymbolicData* developers it is difficult (and probably even not worth) to design a reliable interface.

```
#######################################################
# Record 'META/Key'

<Id>        META/Key                      </Id>
<Type>      META                          </Type>
<Key>       Key                           </Key>
<Syntax>    KeyName                       </Syntax>
<description>
Identifying key of the record. Must be unique within its Type
</description>
<help>
Identifying key of the record. Must be unique within its Type
</help>
<htm>       ignore                        </htm>
<level>     0                             </level>
<order>     3                             </order>
<Version>   ...                           </Version>
<PERSON>    graebe                        </PERSON>
<Date>      Jul 6 2000                    </Date>

# End of record 'META/Key'
#######################################################
```

The META record 'Key'

Similar records share a common structure and are collected into **tables**. The *SymbolicData* geometry theorem proof schemes are collected in the GEO table. The corresponding sd-files are physically stored in a subdirectory `GEO` of the *SymbolicData* data directory (`$SD_HOME/Data` by default).

The common structure of the records within such a table is reflected in a common XML tag structure (**attributes**) that is fixed in another table – the corresponding META table. This allows for flexible extension not only of data but also of data structures and tag syntax restrictions. We refer to [1, 6] and the *SymbolicData* documentation for more details.

The content of a typical META record (the description of the attribute `Key`) is shown on page 12. It is stored in the same format as a data record and contains information about importance (`level`), output order (`order`), HTML handling (`htm`), verbose and detailed descriptions (`help` and `description`) of the attribute to be defined. Note that most of these tags can be omitted since they have default values.

Let us describe the attributes of the GEO records in more detail. Several attributes are predefined, i.e., inherited from a "master table". `Id`, `Key` and `Type` identify the record within its table resp. within the data base. `ChangeLog`, `Comment`, `Date`, `PERSON` and `Version` contain information about the history of the given record. In particular, the value of the attribute `PERSON` is a reference to the table `PERSON` that collects information (affiliations, email addresses, etc.) of persons who contributed to *SymbolicData*. This guarantees a fair authorship management of different contributions along the GNU Public License conditions that apply to *SymbolicData* as a whole.

The `CRef` attribute, also inherited from the master table, attaches cross reference information to one of the records in the (primary) data base. In relational data base models such cross references are usually stored in special relation tables that can easily be searched for different keys. We decided to put this cross reference information into one of the main (primary) records and to provide tools to extract it as secondary data in SQL compliant form. This avoids to develop anew elaborated search and select facilities for the primary (XML based) data.

Cross references in the GEO table usually point to the PROBLEMS table that contains descriptions of the geometry theorems to be proved. Note that different GEO records can point to the same PROBLEMS record since there may be different proof schemes for the same theorem. Other cross references point to GEO records as foreign keys, e.g., from the INTPS table of polynomial systems if the system is generated from the GEO record, from the BIB table of bibliographical references, if a paper refers to the proof scheme stored in the record, etc. We will not go into detail about this point but concentrate on the main attributes of the GEO records.

4.2 The Main Attributes of the *SymbolicData* GEO Collection

GEO record 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.

Since the proof schemes should translate into different target systems we need a special language to write them down. The structure of this special

GEO code will be described below. We continue with the description of the other attributes.

The following attributes (with GEO code 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`)

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

`polynomials`	a list of polynomial conditions describing algebraic dependency relations in the given geometric configuration
`constraints`	a list of polynomial non degeneracy conditions
`solution`	a way to solve the algebraic problem (given in extended GEO code syntax)

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

4.3 The Generic GEO Code Syntax

The main reason to invent a generic GEO code language results from 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 can avoid these efforts 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 referring to previously defined geometric objects and rational expressions as arguments the GEO code 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) It should easily be possible to map the generic GEO code to the syntax of the target CAS without name clashes.

(3) It should provide 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 a slightly different meaning compared to the previous paragraph: 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.

To compile lists (e.g., in the `polynomials` part) and to pick up numerators and denominators (e.g., in the `conclusion`) the GEO code syntax provides the additional function names `List`, `Numerator` and `Denominator`.

For (2) note that if the user follows a slightly more restrictive naming convention for symbols they map one to one to each of the target CAS. Due to the common origin almost the same applies to the GeoProver function names and the syntax of rational expressions. Slightly more efforts are required for variable names, since typical names for points (e.g., C, D, E) are protected in some of the target CAS. Below you find the 3-line Perl script (it is part of the *SymbolicData* tools) that translates the generic GEO code to MuPAD.

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

For (3) we systematically assign the variable names `$polys`, `$con` and `$result` to the list of polynomials, to the conclusion (in equational proof schemes) and to the result (in particular to the conclusion in constructive proof schemes) if applicable. This allows easily to monitor the results of the computation. For constructive proof schemes the translated GEO code – with the GeoProver package for the given target CAS previously loaded – should return 0 (or a list of zeroes if several conclusions are to be verified).

A generic solution for the algebraic part of equational proof schemes is given (with the same notational conventions) as value of the attribute `solution`. It uses the following additional "generic" functions:

`geo_gbasis[polys,vars]`
: to compute a lexicographical wrt. *vars* Gröbner basis of *polys*

`geo_normalf[p,polys,vars]`
: to compute the normal form of the polynomial or list of polynomials *p* wrt. the given polynomials *polys* (usually a lexicographical wrt. *vars* Gröbner basis)

`geo_solve[polys,vars]`
: to find the zeroes of the list of polynomials *polys* wrt. *vars*

`geo_solveconstrained[polys,vars,nondegs]`
: that works as `geo_solve` but take the list of polynomials *nondegs* as non-degeneracy conditions

`geo_eliminate[polys,vars,evars]`
: to eliminate the variables *evars* from the polynomials *polys* in the variables *vars*

`geo_eval[con,sol]`
: to substitute the output *sol* of `geo_solve` in the expression *con*

`geo_normal[u]`
: to compute a rational normal form of *u*

`geo_simplify[u]`
: to simplify *u*

There are small supplementary files with collections of function definitions for each of the target CAS that map these generic functions to the respective syntax or give a reliable solution using the algebraic tools provided by the CAS.

5 Using the *SymbolicData* Tools for the GeoProver Source Code Management

Several points around the GeoProver code management suggest to look for a generic solution:

(1) Changes or extensions of the GeoProver have to be incorporated into the package sources for each of the target CAS. This causes problems for the version management and is a permanent source for code inconsistency. Hence one may ask if some of these changes could be done once and in a generic (and consistent) way.

(2) An efficient compilation of geometric configurations usually makes good use of geometric "macros", i.e., shortcuts for construction schemes of standard tasks that are built up from a small number of elementary steps.

For example, 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".

Such macros correspond to nested GEO code function calls

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

The code required to add such a function to the GeoProver packages for each of the target CAS could easily be generated from this generic GEO code statement. Of course, this requires much more efforts than the translation of GEO code proof schemes since the target CAS greatly differ syntactically and even conceptually in the way how functions and packages have to be defined.

(3) The different target CAS have very different, differing from version to version and in most cases not yet thoroughly tuned policies for package documentation. This requires a flexible organization of the GeoProver documentation that keeps the essential parts close to the sources.

Since the *SymbolicData* tools can be combined also with alternative data bases I used them to manage a GeoProver code data base. It consists of

- inline parts for each of the target CAS with the (CAS-specific) "inline" definitions of the most elementary functions,
- a *SymbolicData* `Prover` table that collects information about all GeoProver export functions (one per record),

- additional Perl code (*SymbolicData* action definitions) to compile the package code for the target CAS from these sources.

A typical record (of the function `centroid`) of the `Prover` table is given below. At the moment it provides the function name (attribute `Key`), syntactical information about the function call (attribute `call`), a generic GEO code definition (attribute `code` – if it is absent this function is defined in the inline part), a short (attribute `verbose`) and a more detailed description.

```
###########################################################
# Record 'Prover/centroid'

<Id>        Prover/centroid               </Id>
<Type>      Prover                        </Type>
<Key>       centroid                      </Key>
<call>      centroid[$A::Point,$B::Point,$C::Point]::Point </call>
<verbose>   centroid of the triangle    </verbose>
<code>
intersection_point[median[$A,$B,$C],median[$B,$C,$A]]
</code>
<description>
Centroid of the triangle <math>ABC</math>.
</description>
<Date>      Feb 9 2002                    </Date>

# End of record 'Prover/centroid'
##########################################################
```

The record 'Prover/centroid' in the GeoProver code data base

Now the code base can be extended easily and in a consistent way with new macros: Define a new `Prover` record and rebuild the sources for the target systems. For example, to add a new function `circumcenter` the record should essentially contain the following information:

```
<call> circumcenter[$A::Point,$B::Point,$C::Point]::Point </call>
<verbose>   circumcenter of the triangle    </verbose>
<code>
intersection_point[
midpoint_perpendicular[$A,$B],midpoint_perpendicular[$A,$C]]
</code>
<description>
The circumcenter of the triangle <math>ABC</math>.
</description>
```

6 Benchmark Computations on GEO Records

We conclude this paper with some remarks about the efforts required to really set up benchmark computations on the GEO records with the *SymbolicData* tools. We report on the computations for a beta test of MuPAD 2.5 since they reflect exemplary these efforts.

For a first screening we posed the following general conditions:

(1) We compile all examples into a single input file `/tmp/mupad.in` to avoid multiple startup overhead.

(2) We cancel the computation of a given example with the `traperror` MuPAD function if it spends too much computing time.

(3) We run the computation as batch process

```
mupad-2.5 </tmp/mupad.in >/tmp/mupad.out &
```

We use the *SymbolicData* tools to create the required input file. Note that the tools are driven by the main program `symbolicdata`. It allows to access the different tasks defined in the basic Perl modules through an elaborated actions concept. Its synopsis is

```
symbolicdata [-req file] actions [options] [args]
```

On start-up, `symbolicdata` loads all the basic Perl modules, initializes the data base, parses the command-line arguments up to the mandatory action argument(s), and loads the **global action hash** that specifies, in a well-defined format, all known (or, "registered") actions and their properties. This action hash can easily be extended at run-time using the first (optional) `-req file` argument, where `file` is the name of a Perl module containing the new action definitions. It is loaded *before* the actions are parsed. For more details we refer to [1, 6] and the *SymbolicData* documentation.

We use this extension concept for our goal, compose the required action definition, save it to a file and call it with `symbolicdata`. You find the Perl code for the new *SymbolicData* action `MuPADTrapCode` on page 22.

Some remarks on the code:

- The `req` slot of the action activates the *SymbolicData* Perl module `GEO/GEO.pm` that contains functions to map GEO code to different target CAS. See code part (1).
- The `argvcall` slot is a special action mode to process all arguments at once. It requires to expand the arguments (i.e., sd-file names) and to pick up the corresponding records. This is done in code part (3).

- Code part (2) prints the MuPAD preamble (path setting and initialization code for the MuPAD GeoProver package) to stdout.
- Code part (4) translates the GEO code of the proof scheme of the record `$r` to the MuPAD syntax.
- Code part (5) is (almost) pure MuPAD code. It defines a function `geotest` (anew for each record) with the MuPAD code of the proof scheme for the given record that is called later on with `traperror` and `time` to time or interrupt the computation.

  The remaining code analyzes the output of the `traperror` call and the content of the variable `_result` (the translation of `$result` to MuPAD code) if the computation finished.

  All MuPAD code is sent to stdout and should be redirected to the desired file.

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. Proving elementary geometry theorems using Wu's algorithm. In *Contemp. Math.*, volume 19, pages 243 – 286. AMS, Providence, Rhode Island, 1984.

[3] S.-C. Chou. *Mechanical geometry theorem proving.* Reidel, Dortrecht, 1988.

[4] S.-C. Chou. Automated reasoning in geometries using the characteristic set method and Gröbner basis method. In *Proc. ISSAC-90*, pages 255–260. ACM Press, 1990.

[5] H.-G. Gräbe. GeoProver - a small package for mechanized plane geometry, 1998–2002. With versions for Reduce, Maple, MuPAD and Mathematica.
See `http://www.informatik.uni-leipzig.de/~compalg/software`.

[6] H.-G. Gräbe. The *SymbolicData* benchmark problems collection of polynomial systems. In *Proceedings of ADG-02, Karlsruhe*, 2002. to appear.

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

[8] The *SymbolicData* Project, 2000–2002.
See `http://www.SymbolicData.org`.

[9] 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.

[10] 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.

[11] W.-T. Wu. *Mechanical Theorem Proving in Geometries.* Number 1 in Texts and Monographs in Symbolic Computation. Springer, Wien, 1994.

```
###############################################################
$ACTIONS -> {MuPADTrapCode} =
{
 req => 'GEO/GEO.pm',                              # (1)
 argvcall => sub {
   print Preamble::MuPAD()
     ."Pref::echo(FALSE);Pref::prompt(FALSE);\n";  # (2)
   my $maxtime=200;
   shift; my $arg=ExpandArgv(shift);
   my (@l,$r,@u);
   map push(@$l, Record->new($_)), (@$arg);        # (3)
   for $r (@$l)
   {
     @u=GEO::CreateSolution($r,'MuPAD');           # (4)
     print <<EOT;                                  # (5)
//==> Example $r->{Key}
clear_ndg():
geotest:=proc() begin
$u[1]
end_proc:
print(Unquoted,"##> $r->{Key} starting");
te:=traperror(print(time(geotest())),$maxtime);
if te=0 then
  print(Unquoted,"##> $r->{Key} finished"); print(_result);
elif te=1320 then print(Unquoted,"##> $r->{Key} timed out");
else print(Unquoted,"##> $r->{Key} error");
  print(prog::error(te),lasterror())
end_if:
EOT
   }
   print "quit;\n";
 },
};
###############################################################
```

Perl code for the `MuPADTrapCode` action