Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Geometry Constructions Language

Predrag Janičić

Received: date / Accepted: date

Abstract Geometry Constructions Language (GCL) is a language for explicit descriptions of constructions in Euclidean plane and of their properties. Other mathematical objects can also be described in the language. The language GCL is intuitive and simple, yet it supports arrays, flow control structures, user-defined procedures, etc. The processors for the GCL language — applications GCLC and WINGCLC— enable visualization of described objects and producing of mathematical illustrations, provide different semantical information and support for automated proving of properties of the constructed objects. These features make the tools GCLC and WINGCLC powerful mechanized geometry systems and they have thousands of users worldwide.

Keywords Geometric Constructions · Dynamic Geometry Software · Automated Geometry Theorem Proving

1 Introduction

Euclidean geometry and geometric constructions have important role in mathematics and in mathematical education for thousands of years. In twentieth century, there was a shift from classical, synthetic geometry in favor of algebraic geometry in university education. However, synthetic geometry still holds a very important position in lower levels of mathematical education and also, in recent years, it has been making a comeback to university education, thanks to important applications in computer-aided design, computer graphics, computer vision, robotics, etc.

There is a range of geometry software tools, covering different geometries and geometry problems. Many of them focus on Euclidean geometry and on construction problems. These problems are very suitable for interactive work and animations, typical for *dynamic geometry software* (e.g., *Cinderella*, *Geometer's Sketchpad*, *Cabri*). In dynamic geometry software, the user can create and manipulate geometric constructions. Typically, the user starts a construction with several points, construct new objects depending on the existing ones, and then move the starting points to explore

E-mail: janicic@matf.bg.ac.rs
Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Serbia

how the whole construction changes. Dynamic geometry software can help teachers to illustrate and students to explore and understand abstract concepts in geometry. In addition, dynamic geometry software can be used for producing digital mathematical illustrations. In most of these tools, the user uses a graphical user interface, tools from toolbars, and the point-and-click approach for describing geometric constructions step-by-step. The alternative is describing constructions explicitly, in a suitable geometric language. The language GCL is one such language.

The basic idea behind the GCL language is that geometric constructions are formal procedures made of abstract steps, rather than drawings, and that the abstract (i.e., formal, axiomatic) nature of geometric objects has to be distinguished from their semantics, usual models, and visualizations. Therefore, in GCL one *describes* constructions, rather than *draws* figures.[1] The figure descriptions are declarative and concise descriptions of mathematical contents and from them corresponding illustrations can be generated.

The primary focus of the first versions of the language GCL and its processor GCLC was producing digital illustrations of Euclidean constructions in LaTeX form (hence the name "Geometric Constructions → LaTeX Converter"), but now it is much more than that. For instance, there is support for symbolic expressions, for parametric curves and surfaces, for drawing functions, graphs, and trees, support for flow control, etc. Libraries of GCL procedures provide additional features, such as support for hyperbolic geometry. Complex geometry theorems can be expressed in GCL and proved by the automated geometry theorem provers built into GCLC. So, GCLC now provides mathematical contents directly linked to visual representation and supported by machine–generated proofs and, hence, can serve as a powerful mechanized geometry assistant. WINGCLC is a dynamic geometry tool built on top of GCLC with a graphical user interface and a range of additional functionalities.

The language GCL and its processors GCLC and WINGCLC are under constant development since 1996. The language has been only the subject of extensions, so the full vertical compatibility is kept with the earliest versions. There are command-line versions of GCLC for Windows and for Linux. A version with a graphical user-friendly interface is available only for Windows. The applications GCLC and WINGCLC are accompanied by a detailed user manual and are freely available from `http://www.matf.bg.ac.rs/~janicic/gclc` and from EMIS (The European Mathematical Information Service) servers (`http://www.emis.de/misc/index.html`). They have thousands of users worldwide and their main areas of applications are in:

- publishing, i.e., in producing digital mathematical illustrations;
- storing mathematical contents;
- mathematical education;
- studies of automated geometric reasoning.

Various aspects of GCL, GCLC, and WINGCLC are described in publications referred to in the following text, in appropriate parts.

Overview of the paper. The rest of the paper is organized as follows: Section 2 briefly introduces Euclidean geometric constructions, Section 3 describes the geometry constructions language GCL, while Section 4 describes its processors, Section 5 reviews

[1] In a sense, this approach is in spirit close to the approach used in the TeX/LaTeX system [21, 23]. Within the TeX/LaTeX system, authors (explicitly) describe the layout of their texts.

main areas of applications of GCL and its processors, and Section 6 discusses related languages and tools. In Section 7 we draw final conclusions and we briefly discuss further development of the language GCL and its processors.

2 Geometric Constructions

A geometric construction is a sequence of specific, primitive construction steps. These primitive construction steps are also called *elementary constructions by ruler and compass* and they are:

- construction (by *ruler*) of a line such that two given points belong to it;
- construction of a point such that it is the intersection of two lines (if such a point exists);
- construction (by *compass*) of a circle such that its center is one given point and such that the second given point belongs to it;
- construction of intersections between a given line and a given circle (if such points exist).
- construction of intersections between two given circles (if such points exist).

By using this set of primitive constructions, one can define more involved, compound constructions (e.g., construction of right angle, construction of the midpoint of a segment, construction of the perpendicular bisector of a segment, etc.). In order to describe geometric constructions, it is usual to use higher level constructions as well as the primitive ones.

3 GCL Language

The syntax of the GCL language is very simple and intuitive. It is a high-level language designed for mathematicians and not a machine-oriented script language (which are used internally in some geometry tools). Descriptions of mathematical objects by GCL commands are easily comprehensible to mathematicians and, in the same time, GCL commands enable describing very complex objects in a very few lines. All primitive constructions by ruler and compass and a range of higher-level constructions and isometric transformations are supported in the language. In addition, some objects that are not constructible by ruler and compass (for instance, the image of a point in rotation for the angle of 1°) can also be used.

In order to reduce syntactic overhead and to improve simplicity and readability, the language GCL is format-free (i.e., line-terminations and multiple white spaces are ignored), there are no command separators/terminators, arguments of commands are separated by white spaces, and the use of brackets is very limited. There are several types: *number* (for real numbers), *point*, *line*, *circle*, *conic* (all for objects in Euclidean plane). Again for the sake of simplicity, the language is dynamically typed, i.e., variables are not declared and can change their types during program execution. Elements of one array may have different types. There is support for arrays and there are flow control structures *if-then-else* and *while*-loop, sufficient for the language GCL to be computationally complete (in a sense in which, for instance, the languages C or Pascal are computationally complete). There is support for user-defined procedures and parameters are always passed by reference (so one procedure can return several results),

unless they are numerical constants. Other GCL files (for instance, containing libraries of some procedures) can be included.

An extract of the EBNF description of GCL is given in Figure 1. There are around 150 elementary commands, but they are intuitive, so fundamentals of the language can be acquired in a very short time.

Some GCL commands are aimed at describing a *content* (geometrical or other mathematical objects), while some are aimed at describing a *presentation* (i.e., visualization of the described objects). According to their semantics, GCL commands can be divided into the following groups:

Basic definitions: commands for introducing points, for defining a line on the basis of two selected points, for defining a circle, a numerical constant, etc.

Basic constructions: constructions of intersection points for two lines, for a line and a circle, construction of the midpoint of a given segment, the bisector of an angle, the perpendicular bisector of a segment, the line passing through a given point and perpendicular to a given line; the line passing through a given point and parallel to a given line, etc.

Transformations: commands for translation, rotation, line-symmetry, half-turn, and also for some non-isometric transformations like scaling, circle inversion, etc.

Calculations, expressions, and flow control structures: commands for calculating angles determined by triples of points, distances between points, for generating (pseudo)-random numbers, for calculating symbolic expressions, support for *if-then-else* structures and *while*-loops, etc.

Drawing commands: commands for drawing (in various modes) lines, line segments, circles, arcs, ellipses, etc.

Labelling and printing commands: commands for labelling and marking points, and for printing text;

Cartesian commands: commands for direct access to a user–defined Cartesian system. A user can define a system, its unit, and, within it, he/she can define points, lines, conics, tangents, curves given in parametric form, etc. A similar support is available for 3D Cartesian space.

Low level commands: commands for changing line thickness, color, clipping area, figure dimensions, etc.

Commands for describing animations: commands for specifying animations. Several points can simultaneously move from one position to another and points can be traced (i.e., a loci can be specified).

Commands for automated geometry theorem proving: commands for specifying a geometry conjecture, a level of proof details, a maximal number of proof steps, and a time limit.

A simple example of a GCL program, with one geometry conjecture, is given in Figure 2.

As in all geometry tools, descriptions of constructions in GCL include several (usually very few) starting points and other points and construction steps dependent on these starting points. The starting points are usually referred to as *free points* as they do not depend on other points. While an Euclidean construction is an abstract procedure, there is its counterpart in the standard Cartesian model that can be visualized. In order to visualize a described construction, its free points should be assigned concrete Cartesian plane coordinates. In the given example, one has to select three particular

```
⟨GCL program⟩            ::=  (⟨statement block⟩
                              | ⟨procedure definition⟩
                              | ⟨include directive⟩)*

⟨statement block⟩        ::=  (⟨statement⟩)*

⟨statement⟩              ::=  ⟨elementary command⟩
                              | while "{" ⟨condition⟩ "}"
                                   "{" ⟨statement block⟩ "}"
                              | if-then-else "{" ⟨condition⟩ "}"
                                   "{" ⟨statement block⟩ "}"
                                   "{" ⟨statement block⟩ "}"
                              | call ⟨identifier⟩ "{" ⟨parameter list⟩ "}"

⟨elementary command⟩     ::=  point ⟨identifier⟩ ⟨number⟩ ⟨number⟩
                              | line ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | circle ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | array ⟨identifier⟩ "{" (⟨number⟩)* "}"
                              ...
                              | intersection ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | midpoint ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | bisector ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | perp ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              ...
                              | translate ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | rotate ⟨identifier⟩ ⟨identifier⟩ ⟨number⟩ ⟨identifier⟩
                              | towards ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩ ⟨number⟩
                              | oncircle ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              ...
                              | getx ⟨identifier⟩ ⟨identifier⟩
                              | distance ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | angle ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | random ⟨identifier⟩
                              | expression ⟨identifier⟩ "{" ⟨expression⟩"}"
                              ...
                              | drawsegment ⟨identifier⟩ ⟨identifier⟩
                              | drawline ⟨identifier⟩
                              | drawarc ⟨identifier⟩ ⟨identifier⟩ ⟨number⟩
                              | filltriangle ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | drawbezier3 ⟨identifier⟩ ⟨identifier⟩ ⟨identifier⟩
                              | drawtree ⟨identifier⟩ ⟨number⟩ ⟨number⟩ ⟨number⟩ ⟨number⟩
                                            ⟨tree description⟩
                              | drawgraph_a ⟨identifier⟩ ⟨number⟩ ⟨number⟩
                                            ⟨list of nodes⟩ ⟨list of edges⟩
                              ...
                              | mark_lt ⟨identifier⟩
                              | cmark_lt ⟨identifier⟩
                              | printat_lt ⟨identifier⟩ "{" ⟨text⟩) "}"
                              ...
                              | dim ⟨number⟩ ⟨number⟩
                              | color ⟨number⟩ ⟨number⟩ ⟨number⟩
                              | fontsize ⟨number⟩
                              | linethickness ⟨number⟩
                              ...

⟨procedure definition⟩   ::=  procedure ⟨identifier⟩ "{" ⟨parameter list⟩ "}"
                                   "{" ⟨statement block⟩ "}"

⟨include directive⟩      ::=  ⟨include⟩ ⟨file_name⟩
```

Fig. 1 An extract of the EBNF description of the language GCL

Cartesian points as vertices of the triangle. The similar approach is used for describing other mathematical objects.

Figure 3 illustrates the use of libraries of procedures in GCL, in this case — support for Poincare's disc model of hyperbolic plane (provided by another, 300 lines long GCL file). Figure 4 illustrates the use of flow control structures and recursive procedures in GCL. Figure 5 illustrates the support for Cartesian system and parametric curves. More GCL examples can be found in [16].

```
% free points
point A 10 10
point B 40 10
point C 30 40

% perpendicular bisectors of the sides
med a B C
med b A C
med c B A

% intersections of the bisectors
intersec O_1 a b
intersec O_2 a c

% labelling the points
cmark_lb A
cmark_rb B
cmark_t C
cmark_lt O_1
cmark_rt O_2

% drawing the sides of the triangle ABC
drawsegment A B
drawsegment A C
drawsegment B C

% drawing the circumcircle of the triangle
drawcircle O_1 A

% specifying a conjecture
prove { identical O_1 O_2 }
```

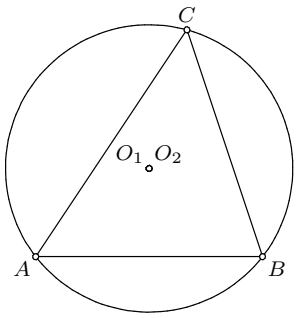

Fig. 2 Example of a GCL description of a geometric construction and a conjecture (left) and the corresponding (LaTeX) output (right)

3.1 Specifying Geometry Conjectures

In GCL, a geometry conjecture is not expressed explicitly in terms of first-order logic, but by the description of a construction itself and by a given goal. Conjectures, given in this form, are universally quantified sentences in the underlying theory of Euclidean geometry with theory of real numbers.

For the construction shown in Figure 2, for any particular three non-collinear points `A`, `B`, and `C`, the points `O_1` and `O_2` (pairwise intersections of the perpendicular bisectors of the sides) are identical. This statement relies only on the specification of the

```
% include support for Poincare's disk model
include hyp.gcl

point O 25 25
point X 45 25
circle k O X
call h_drawabsolute { O k }

point P 10 20
point Q 12 31
point R 30 10

cmark_l P
cmark_lt Q
cmark_b R
call h-drawsegment { O k P Q }
call h-drawsegment { O k Q R }
call h-drawsegment { O k P R }

call h-med { a O k P Q }
call h-med { a1 O k P R }
call h-intersec { X O k a a1 }
call h-drawcircle { O k X P }
```

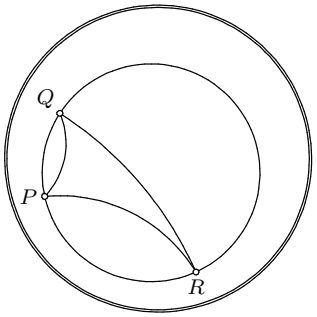

Fig. 3 A GCL description of a geometric construction in hyperbolic plane (left) and the corresponding illustration in Poincare's disc model (right)

```
procedure Koch { A B n }
{
    if_then_else { n>0 }
    {
        expression r { 1/3 }
        towards C A B r
        towards E B A r
        rotate D C -120 A
        expression n' { n-1 }
        call Koch { A C n' }
        call Koch { C D n' }
        call Koch { D E n' }
        call Koch { E B n' }
    }
    {
        drawsegment A B
    }
}

point A 0 10
point B 40 10
call Koch { A B 6 }
```

Fig. 4 A GCL description of Koch's curve (left) and the corresponding (LaTeX) output (right)

construction and not on specific Cartesian coordinates of the points (used only for visualization). The goal of the statement is expressed in GCL by the following command:

```
prove { identical O_1 O_2 }
```

Generally, a conjecture can be either of one of the basic sorts, such as the one above (see also below), or it can be of the form $L = R$, where L and R are expressions

```
ang_picture 3 3 49 49
ang_origin 20 20
ang_unit 6
ang_drawsystem_a

ang_draw_parametric_curve x
        {-2.5; x<4; x+0.05}
        { x; sin(pow(x,2))*cos(x) }

% polar coordinates
number rho 2
ang_draw_parametric_curve phi
        { 0 ; phi<6; phi+0.1}
        { phi*rho*sin(phi)/5 ;
          rho*cos(phi) }
```

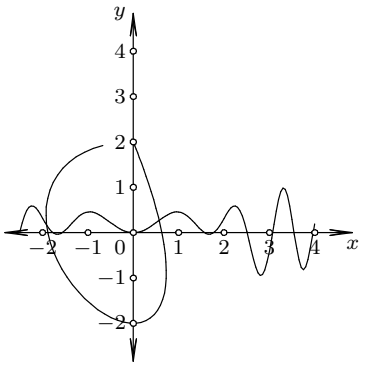

Fig. 5 A GCL description of two parametric curves (left) and the corresponding (LaTeX) output (right)

over geometric quantities (quantities used in the area method for automated theorem proving [7,8]). The following geometric quantities are used:

ratio of directed segments: for four collinear points P, Q, A, and B such that $A \neq B$, it is the ratio $\frac{\overline{PQ}}{\overline{AB}}$;

signed area: it is the signed area S_{ABC} of a triangle ABC or the signed area S_{ABCD} of a quadrilateral $ABCD$;

Pythagoras difference: for three points, P_{ABC} is defined as follows:

$$P_{ABC} = AB^2 + CB^2 - AC^2 \ .$$

For four points, P_{ABCD} is defined as follows:

$$P_{ABCD} = P_{ABD} - P_{CBD} \ .$$

real number: it is a constant real number.

Geometric quantities can be combined together into more complex terms by operators for addition, multiplication and division (written in prefix form as `sum`, `mult`, `ratio`). For instance, the conjecture (corresponding to Ceva's theorem):

$$\left(\left(\frac{\overline{AF}}{\overline{FB}} \cdot \frac{\overline{BD}}{\overline{DC}}\right) \cdot \frac{\overline{CE}}{\overline{EA}}\right) = 1$$

is written in the following way:

```
prove { equal { mult { mult { sratio A F F B }
                            { sratio B D D C } }
                     { sratio C E E A } }
              1 }
```

All supported geometry conjectures of basic sorts can also be stated in terms of geometric quantities, as shown in the following table:

GCL specifications of basic conjectures	semantics	expressed in terms of geometric quantities
`identical A B`	points A and B are identical	$P_{ABA} = 0$
`collinear A B C`	points A, B, C are collinear	$S_{ABC} = 0$
`perpendicular A B C D`	AB is perpendicular to CD	$P_{ACD} = P_{BCD}$
`parallel A B C D`	AB is parallel to CD	$S_{ACD} = S_{BCD}$
`midpoint O A B`	O is the midpoint of AB	$\frac{\overline{AO}}{\overline{OB}} = 1$
`same_length A B C D`	AB has the same length as CD	$P_{ABA} = P_{CDC}$
`harmonic A B C D`	points A, B, C, D are harmonic	$\frac{\overline{AC}}{\overline{CB}} = \frac{\overline{DA}}{\overline{DB}}$

A conjecture may involve geometric quantities only over objects already introduced within the current construction.

More details on specifying geometry conjectures can be found in [32].

3.2 XML Interchange Format

There is a XML counterpart of a fragment of the language GCL — of a core of the language covering geometric constructions [33]. This format is designed with motivation to serve as an interchange format for different geometry tools. The format is supported by a suite of tools including:[2]

- converters from GCL and the language *Eukleides*[3] to XML-based format; these converters were implemented in the programming language C/C++;
- converters from XML-based format to GCL and the language *Eukleides*; these converters were implemented as XSLT files;
- a converter from XML-based format to a HTML form; this converter was implemented as a XSLT file;
- a converter from XML-based format to a natural language form (currently, only for English language); this converter was implemented as a XSLT file;
- a tool for generating figures in SVG format on the basis of GCL code; this tool was implemented in the programming language C++;
- newly defined XML-based format for representing proofs of properties of geometric constructions with a corresponding DTD; the format is adapted for the methods supported by GCLC;
- tools for exporting proofs from automated theorem provers to XML-based form (there is support for all theorem provers built into GCLC); these tools were implemented in the programming language C++;
- a converter for proofs from XML-based form to a simple, readable HTML form; this converter was implemented as a XSLT file.

This suite is suitable for storing geometric constructions, theorems, and proofs and presenting them on Internet, as demonstrated by the repository GeoThms[4] [31]. More details on this suite and the XML-based formats can be found in [33].

[2] Note that the format and the listed tools (except export to SVG) deal only with textual descriptions of constructions and not with figures.

[3] See more about the language *Eukleides* in Sec. 6.

[4] `http://hilbert.mat.uc.pt/~geothms`

4 GCL Processor

The tool GCLC is a processor for the GCL language. It compiles a GCL document and generates a corresponding visual output description in intermediate low-level data structures. This representation consists of simple primitives, such as drawing line segments, circles, and text printing, and can be simply exported to different formats. Thanks to the object-oriented design, processors for other geometric or visual languages can be simply plugged in to use the intermediate format. Also, support for additional export formats can be easily added.

The command-line application GCLC takes a GCL document and a list of parameters as input. If there is an error in the GCL code, GCLC reports it. Otherwise, GCLC produces images in selected formats and generates a log file with Cartesian values of all constructed objects. If there is a geometry conjecture given in the GCL document, then GCLC invokes the selected automated theorem prover and exports its output to LaTeX or XML form.

GCLC is implemented in standard C++, consists of around 1Mb or 40000 lines of code. The executable versions both for Linux and Windows have less than 1Mb.

4.1 Built-in Theorem Provers

The system GCLC has three geometry theorem provers for Euclidean constructive theorems built in (for a survey of automated deduction in geometry, see, for instance [26]):

- a theorem prover based on the area method [7,8].[5] This method belongs to the group of semi-algebraic methods. It produces human-readable proofs (still not traditional, synthetic proofs), with a clear justification for each proof step. The conjecture is expressed by an equality of expressions in geometric quantities. Along the proof, this equality is transformed step-by-step, until it becomes trivial. For more details about this prover, see [17].
- theorem provers based on the Gröbner bases method and on the Wu's method.[6] These methods belong to the group of algebraic methods. For more details about these provers, see [29].

The provers are tightly integrated in GCLC. This means that one can use the prover to reason about a GCL construction without changing and adapting the description of the construction for the deduction process — only the geometry conjecture has to be provided within the `prove` command (see the example in Figure 2). By this, GCLC directly links geometric contents, visual information, and machine–generated proofs.

All conjectures are internally transformed into statements in terms of geometric quantities. The theorem provers consider only abstract specification of the conjecture and do not consider Cartesian values of the points involved (they are used only for visualization). The proofs, given in terms of the supported methods (as yet, there are no object-level proofs, verifiable by theorem proving assistants), can be exported to LaTeX

[5] This theorem prover was developed in collaboration with Pedro Quaresma from University of Coimbra.

[6] These theorem provers were developed in collaboration with Goran Predović from University of Belgrade.

or to XML form, with explanations for each proof step. Proofs are also accompanied by semantical counterparts — as a check whether a conjecture is valid in the specific case, determined by the given Cartesian points.

All three provers can prove hundreds of complex geometry theorems very efficiently, usually in only milliseconds (for a selection of proved theorems, see, for instance, a repository GeoThms [31]).

4.2 Syntactical, Semantical, and Deductive Checks

The GCL processor detects syntactical errors in input GCL files. The processor can also detect semantical errors — situations when, for a given concrete set of geometrical objects, a construction step is not possible. For instance, in the construction shown in Figure 2, it is impossible to construct a line O_1O_2, since the points O_1 and O_2 are identical. Moreover, in this example, construction of the line O_1O_2 is always illegal (for any Cartesian coordinates of the free points) and it can be deductively shown. When GCLC encounters a construction step that is semantically invalid (e.g., two identical points do not determine a line), it reports that the step is illegal with respect to a given set of free points. Then, it automatically invokes the theorem prover to check if a construction step is geometrically sound, i.e., if it is possible in general case. The prover is ran on the critical conjecture (e.g., it tries to prove that two points are identical) and, if successful, it reports that the construction step is always illegal/impossible.

By this verification mechanism, the deductive nature of geometry conjectures and proofs are linked to the semantic nature of models of geometry and, also, to human intuition and geometric visualizations. As we are aware of, GCLC is the only geometry tool with such a verification system. Some related mechanisms are discussed in Sec. 6.

More details on verification of regular constructions within GCLC can be found in [19].

4.3 Graphical User Interface

WINGCLC is a Microsoft Windows application that provides graphical user interface to GCLC (as yet, there is no version with a graphical user interface for Linux). Descriptions in the GCL language are edited in an integrated syntax coloring editor and can be visualized within an internal viewer.

WINGCLC has a range of interactive functionalities and tools typical for dynamic geometry software. Some of them are *watch window* for monitoring values of selected constructed objects in Cartesian plane (so WINGCLC can work as a *geometric calculator*), tools for interactive moving of free points, updating figures, animations, etc. WINGCLC has also functionalities for locating errors in the GCL code and for listing logs for GCL documents.

Figure 6 illustrates some of the mentioned tools and features (loci, animations, *watch windows*, etc.). More details on the graphical user interface of WINGCLC can be found in [20].

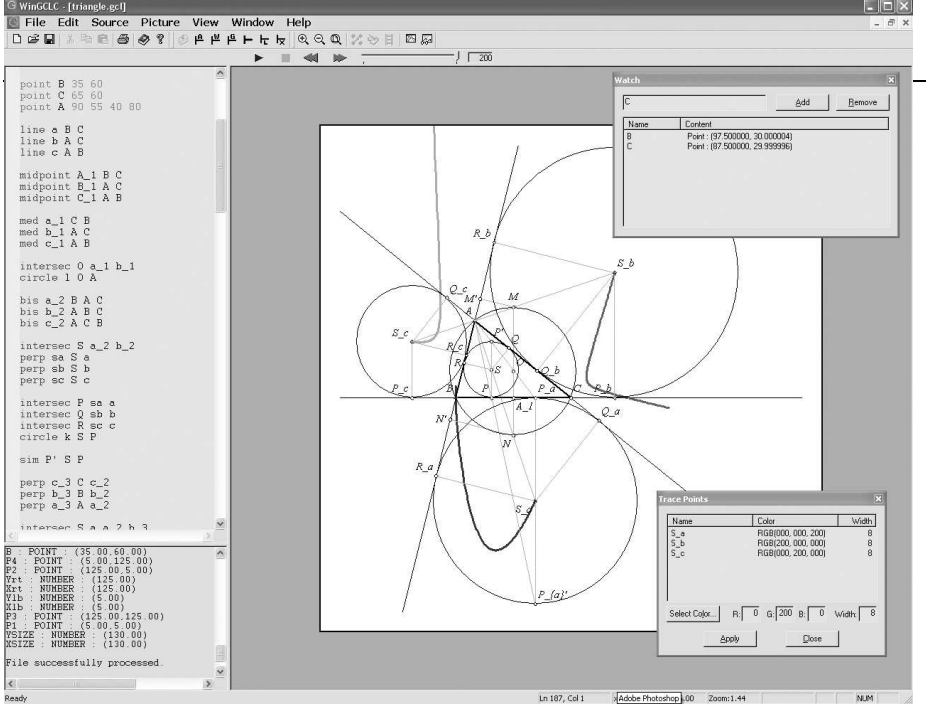

Fig. 6 *Trace* and *watch* windows in WINGCLC

4.4 Export Formats

Figures described in the GCL language can be exported by the tool GCLC to a range of vector formats: a simple LaTeX format, a LaTeX format based on the package `pstricks`, a LaTeX format based on the package `TikZ`, EPS (Encapsulated PostScript), and SVG (Scalable Vector Format). In all exported files there are comments for all commands, so the files are readable and can be a subject of post-editing if necessary. In addition, in WINGCLC there is support for the bitmap format.

5 Areas of Applications

Four main areas of application for the language GCL and its processors are in: publishing (i.e., producing digital mathematical illustrations), storing mathematical contents, mathematical education, and studies of automated reasoning in geometry.

5.1 Producing Digital Illustrations

GCLC can serve as a tool for making digital mathematical illustrations of high quality. Figures in LaTeX format produced by GCLC can be included directly in LaTeX documents and they use LaTeX fonts and formulae which is often essential for high quality illustrations in LaTeX documents (while this is a problem for many other formats and tools). Figures in SVG format are suitable for web-publication. Generated figures in all formats are suitable for different sorts of postprocessing and conversions to other formats. GCLC has been used for producing digital illustrations for a number of mathematical books and journal articles.

5.2 Storing Mathematical Contents

A lot of mathematical contents, both in education and in research, is of visual nature. Mathematical illustrations carry mathematical messages that are represented visually rather than in textual or numerical form. Such messages are, usually, better understandable to a reader when represented visually. On the other hand, this visual information is typically not mathematically rigorous; it is usually approximation and/or interpretation of some mathematical objects, notions, concepts, numerical data, proofs, ideas, etc. It is assumed that the reader (with a support from the given textual explanations, earlier experience and mathematical background, intuition, etc.) can understand the correct mathematical message from the illustration and can interpret the visual information in terms of formal mathematical information. However, that information cannot always be reproduced from the illustration itself. In addition, a mathematician, the author or a reader of a mathematical text, may need to alter an image, to modify some of its characteristics, to make it more general or more specific, and also to store it in a way that enables these sorts of transformations.

A complex geometric construction may be illustrated by an image and can make the understanding of the mathematical text easier. However, as in the wider context of mathematics, without a given context and textual explanations of the constructions, it is unlikely that one can guess the correct specification of the construction. In addition, the Cartesian interpretation of Euclidean geometry is just one of possible interpretations and, hence, potentially misses some of the abstract geometric meaning. Therefore, an image itself does not provide precise geometric message, so it is better to have a formal figure description, rather than mathematical illustration itself. Mathematical contents stored in this way is easy to understand, visualize, maintain, modify and process in different ways.

The language GCL and its processors have been developed along the lines of the given motivation. In GCL, figure descriptions are declarative, precise and concise descriptions of mathematical contents and from them corresponding illustrations can be generated. This way, the GCL language is a mean for storing mathematical contents of visual nature in textual form.[7]

More details on storing mathematical contents and other mathematical knowledge management aspects of the GCL language and its processors are discussed in [30].

5.3 Mathematical Education

In mathematical education, students can interactively use GCLC to make different attempts in making constructions and/or exploring some mathematical (especially geometrical) objects, notions, ideas, problems, proofs, properties, etc. [10]. Rigorously describing geometry objects is similar to programming, so construction problems can help students skilled in programming to understand geometry, while they can also help students acquainted with geometry and construction problems to understand programming, and this applies to various education levels. Visualizations and interactive work

[7] Although all supported picture formats have their advantages, GCL figures are typically stored in their original, source form. This form is not only precise and sufficient for producing pictures, but also very concise: for instance, all figures from a university book with 120 illustrated geometry problems [18] have together (in uncompressed, GCL form) around 100Kb.

make teaching and studying geometry more interesting and more fruitful. The built-in theorem provers can help students link semantical and deductive aspects of geometry.

The language GCL and its processors are taught in a number of high-school and university courses on geometry and on technical writing.[8] Due to the abstraction level required for describing constructions, the language GCL and its processors are not very suitable for use in primary schools.

More details on educational aspects of the GCL language can be found in [10].

5.4 Studies in Automated Reasoning in Geometry

The language GCL is expressible enough to cover a wide range of theorems in Euclidean plane geometry. The GCL processor, the tool GCLC has three powerful automated theorem provers built-in. Despite the fact that all these theorem provers are well-known, widely accepted, and popular (they are probably the three most successful methods for automated theorem proving in Euclidean geometry)[9] there are just a few implementations. Given that the implementation of these methods within GCLC share many mechanisms and portions of code, GCLC can serve as a workbench for testing, comparing, and improving these methods, by combining them together or with some other methods.

6 Related Languages and Tools

The tools GCLC/WINGCLC are related to the family of interactive geometry tools.[10] Only the (commercial) tools *Cabri*[11] and *Geometer's Sketchpad*[12] have history of continuous development longer than GCLC/WINGCLC. GCLC/WINGCLC share a number of features with other geometry tools, but there are also some significant differences and distinctive features. In this section we briefly survey related geometric languages and tools and their features.

Languages for describing geometric constructions. All geometry tools store descriptions of constructions in special purpose formats. Most of them are not intended to be readable and human-editable. There are attempts at developing a common interchange format for different geometry tools [4,33].

Many (almost all) dynamic geometry tools provide, in some form, textual descriptions for the constructions that are described through the graphical interface, in the

[8] Mathematical education can also be closely related to producing mathematical images and to storing mathematical contents. For instance, Zoran Lučić with his students (Faculty of Mathematics, University of Belgrade) produced an electronic version of a number of classical books on geometry, including Euclid's masterpiece — *The Elements* [11]. This is probably the first edition of *The Elements* that includes formal, rigorous description (in the GCL language) of all images, descriptions that directly reflect the accompanying mathematical text.

[9] Wu's method is sometimes considered as the most efficient automated theorem proving method in all categories (not only in geometry). It is also often considered to be one of the "four modern great Chinese inventions", see `http://www.edu.cn/20060215/3173112.shtml`.

[10] An overview of interactive geometry tools can be found here: `http://en.wikipedia.org/wiki/Interactive_geometry_software`.

[11] `http://www.cabri.com/`

[12] `http://www.dynamicgeometry.com/`

point-and-click manner. However, in these tools such descriptions are not editable and cannot be used for two-way communication and for modifying constructions (except, trivially, for deleting constructions steps).

Many dynamic geometry tools, including *Cabri* and *Geometer's Sketchpad*, enable recording construction steps and repeating them later on other given geometric objects. Such recorded sequences of steps — macros (sometimes referred to as "scripts") are stored as files that are not editable and hence do not provide features of programming languages.

For accessing constructions and properties of constructed objects, some geometry tools enable scripting — some of them in general purpose languages, and some of them in custom-developed languages. In that sense, the framework GCL/GCLC/WINGCLC is most closely related to the special-purpose languages *Eukleides* and *CindyScript* with their accompanying tools.

Eukleides[13] (developed from 2001) is an Euclidean geometric drawing language, with a processor `eukleides` (a compiler that converts descriptions of geometric figures to LaTeX or in other vector graphic formats), and a graphical user interface `xeukleides` that enables interactive work. Some dynamic geometry tools (e.g., *GeoProof*) use *Eukleides*, as a high-level output format, suitable for modifying and for generating figures in LaTeX documents. In *Eukleides*, one can describe two points and their midpoint in a very similar manner as in GCL:

```
A = point(0,0)
B = point(2,0)
C = barycenter(A,B)
draw(segment(A,B))
```

Only with *Eukleides* and GCL, descriptions of the constructions are stored in files exactly the same as they were written by the user. In contrast to GCL, in *Eukleides* there are no arrays, flow control structures, support for user-defined procedures, and support for automated theorem proving.

Cinderella.2[14] has its custom built scripting language — *CindyScript*. It is designed to allow high-level interaction with geometric constructions created in the geometric part of the program. *CindyScript* is functional language, yet functions may have side effects, affecting constructions (e.g., by drawing operations). *CindyScript* does not have explicit typing of values and there are no declarations of variables and functions. Any value of any type can be assigned to any variable. A variable is created when it is assigned for the first time. For defining a function, the name of the function, a parameter list, and the body have to be given, but types of arguments and return value are not required. For entering *CindyScript* programs the user uses a script editor and specifies the occasion on which the program will be executed (e.g., after every move of the construction). Scripts can read and change most of the parameters of the elements of a geometric construction. For instance, `A.xy=(B+C)/2` sets the point `A` to be the midpoint of `B` and `C`. Starting with *Cinderella* version 2.1, points can also be created and destroyed on the fly from *CindyScript*.

Xcas[15] is a computer algebra system with support for dynamic geometry. All geometric instructions can be described in the custom-built programming language with several syntax styles.

[13] `http://www.eukleides.org`

[14] `http://doc.cinderella.de`

[15] `http://www-fourier.ujf-grenoble.fr/~parisse/giac.html`

GeoGebra[16] uses JavaScript for scripting. Scripts can be used not only for changing states of the constructed objects, but also for creating buttons for user-defined actions.

DrGeo II[17] uses the Smalltalk programming language as a script language. The language can also be used to define interactive figures. A script depends on a set of items in the geometric figure and the returned value of the script is printed in the drawing area.

Kig[18] uses the Python scripting language. Each user-defined function creates one new object from several existing objects and shows it. All kinds of objects can be used in the Python code and all kinds of objects can be returned. The user selects the arguments for the script through the graphical interface and enter a code for the script object. For instance, the following code returns and shows the midpoint of the segment with endpoints `a` and `b`:

```
def calc(a, b):
m = (a.coordinate()+b.coordinate())/2; return Point(m)
```

In all of the above tools (except *Eukleides*), support for scripting has been added only recently and in later stages of development, following the need for easier describing complex constructions and operations, for situations in which not even macros are sufficient. As an example, Kig's manuals reads: "if you have some fancy way of calculating an interesting point on a conic, then instead of messing with complex constructions and macros, you could just write down in Python code how the point is to be calculated and then Kig will show it for you". Difficulties with describing complex constructions in the point-and-click manner only bring the above tools to the need of procedural descriptions of constructions and operations, and hence — closer to the motivation with which GCLC was built. Still, there is a substantial difference: in the above tools, the primary mean for describing constructions is the point-and-click approach with available buttons and menu options for various constructions steps. Instructions given by scripts make a substantially different way of communication. Moreover, expressiveness of scripts typically does not match the expressiveness available by the graphical interface (with missing possibilities on both sides). On the other hand, in GCLC/WINGCLC, there is only one, consistent and expressive way for describing constructions. Although developing a point-and-click mode for using GCL is always an option, it is not likely that further development will go in that direction. Namely, the main comparative strength of GCLC/WINGCLC is the powerful, expressive, custom-built language. A move to the point-and-click approach would make just yet another geometry tool in that family, typically not-suitable for describing complex geometric configurations. Also, the largest part of the GCL language would remain unaccessible through the graphical interface (since there are around 150 commands) and there would still be a need for using scripting. In addition, there seem to be many users that prefer explicit textual descriptions to the point-and-click approach.

Related to the GCL language is also the language *Gool* (an interpreter for it is still not available) [24, 25]. *Gool* is an object-oriented language for specifying and visualizing geometric objects, for reasoning about their properties, and for creating interactive documents automatically. In describing geometric objects in *Gool*, one has to deal with a number of technical issues, which could be difficult for mathematicians not familiar

[16] `http://www.geogebra.org`
[17] `http://wiki.laptop.org/go/DrGeo`
[18] `http://edu.kde.org/kig/`

with programming. Most computations in *Gool* will be based on algebraic methods, which could often make geometric intuition and information lost.

Geolog[19] is a logic programming language suitable for expressing geometry conjectures and proving them in traditional, readable manner using coherent logic [2,13]. *Geoprolog* is an interpreter for *Geolog*. The language *Geolog* and its interpreter do not address presentation issues and there are no visualizations of the geometric contents described.

There are several widely used low-level languages and vector based formats for graphics in which geometric objects can be described. Encapsulated PostScript[20] is a graphics file format derived from PostScript with support for a number of geometric objects. MetaPost[21] is a language with interpreter, that shares Donald Knuth's Metafont's declarative syntax for manipulating lines, curves, points, and geometric transformations. It produces figures in Encapsulated PostScript from geometric/algebraic descriptions. PSTricks[22] is a set of macros for describing figures directly within TEX or LATEX code. PGF/TikZ[23] is a pair of languages for producing vector graphics from geometric/algebraic descriptions. The top-level PGF and TikZ commands are invoked as TEX macros. The listed (and related) languages are all much lower level compared to GCL and actually GCL can transform complex descriptions of figures into some of the above languages, using them as export formats.

Automated proving features. While most dynamic geometry tools focus on visualizing geometric objects and relationships, some of them also aim at checking properties of constructed objects. Many dynamic geometry systems use numerical and some use probabilistic methods to test geometry conjectures. Several tools go a step further and also address properties of constructed objects in deductive terms, by employing methods for automated theorem proving. These features are very important as they directly link visual and semantical geometric information with deductive properties and machine–generated proofs. We briefly review tools with automated proving and related features.

Cinderella uses randomized theorem checking for analyzing constructions, for example — for checking whether a new element is identical to an already existing element [22]. It is not a symbolic, deductive theorem proving method, but a probability method for checking whether a conjecture is likely a theorem.

GEOTHER[24] is an environment that combines drawing routines and interface written in Java with five algebraic theorem provers implemented in Maple [34]. On the bases of the textual description of a conjecture, GEOTHER automatically produces dynamic diagrams, i.e., assigns coordinates to the involved points in an appropriate manner.

MMP/Geometer[25] automates geometric diagram generation, geometry theorem proving, and geometry theorem discovering [14]. MMP/Geometer implements Wu's method, the area method, and the geometry deductive database method. Conjectures are given in a restricted pseudo-natural language or in a point-and-click manner.

[19] http://www.csupomona.edu/~jrfisher/www/geolog/
[20] http://www.adobe.com/devnet/postscript/
[21] http://www.tug.org/metapost.html
[22] http://tug.org/PSTricks/
[23] http://sourceforge.net/projects/pgf/
[24] http://www-calfor.lip6.fr/~wang/GEOTHER/
[25] http://www.mmrc.iss.ac.cn/~xgao/software.html

GeoView[26] is a tool that combines a dynamic geometry drawing tool *GeoplanJ* with *PCoq*, a user interface for the general purpose theorem prover *Coq* [1]. The statements of plane geometry theorems and their proofs are (manually) constructed and then verified within *Coq* theorem prover. Dynamic geometry figures can be automatically generated from *PCoq* theorem statements.

Discover and related tools *webDiscovery* and *GDI*, combine a dynamic geometry environment with computer algebra systems, such as *CoCoA* and *Mathematica*, and are capable of loci generation, automated theorem proving, and automatic discovery in Euclidean geometry [3,5,6]. For a user-defined geometric construction, conditions for some property to hold can be automatically derived, using algebraic theorem proving methods (e.g., Gröbner basis method). Also, the system offers an interactive way for investigating and conjecturing in geometry.

Geometry Explorer is a dynamic geometry tool that produces human-readable proofs of properties of constructed objects, using the full-angle method [35]. It can produce diagrammatic proof visualizations that aim to be more intuitive than textual proofs.

GeoProof [27] is an interactive geometry tool that can communicate with the *Coq* proof assistant to perform interactive proofs of geometry theorems [27,28]. It can also use automatic theorem provers based on the area method, Wu's method, and Gröbner basis method for generating formal, verifiable proofs. User-defined constructions are automatically translated into *Coq*'s syntax.

Geometry Expert[28] (*GEX*) is a dynamic geometry tool focused on automated theorem proving and it implements Wu's, Gröbner basis, vector, full angle, and the area methods [9]. *Java Geometry Expert*[29] (*JGEX*), under development from 2004, is a new, Java version of *GEX* [36]. *JGEX* combines dynamic geometry, automated geometry theorem proving, and, as its most distinctive part, visual dynamic presentation of proofs. It provides a series of visual effects for presentation of proofs and proofs can be visualized either manually or automatically. Within the program distribution, there are more than six hundred examples of proofs.

Theorema is a general mathematical tool with support for several methods for automated proving of theorems in geometry [12].

All of the above tools with deductive mechanisms (all the listed tools except *Cinderella*) were built with focus on automated theorem proving and available support for constructions and visualization typically directly reflect their proving capabilities. GCLC/WINGCLC was built to have a wider scope (automated theorem proving capabilities were added in later stages of development) and has a richer support for visualizing mathematical, not only geometrical, objects.

In the above tools, geometry conjectures are given either following constructions described in the point-and-click manner, or as formal statements that do not address presentation issues. None of these tools uses a language such as GCL in which conjectures are given within a single specification that describes both geometric contents and the presentation issues.

Unlike some of the listed tools, in GCL/WINGCLC there is no support for interactive proofs and for visualization of proofs (generated either manually or automatically). In

[26] `http://www-sop.inria.fr/lemme/geoview/geoview.html`
[27] `http://home.gna.org/geoproof/`
[28] `http://www.mmrc.iss.ac.cn/gex/`
[29] `http://woody.cs.wichita.edu/`

GCL/WINGCLC there is no support for discovering properties of constructed objects. Also, proofs that produce GCLC are not verifiable by general theorem proving systems.

Additional features. Most of the dynamic geometry tools focus on Euclidean geometry. However, some of them have also support for other geometries or for other fields of mathematics and physics. For example, C.a.R.[30] has support for hyperbolic and elliptic geometry. Cinderella.2 has native support for projective and hyperbolic geometry, but also provides an environment for doing interactive physics experiments. GCLC provides support for hyperbolic geometry (for Poincaré's disc model) via the library of GCL functions and, similarly, can provide support for other theories. In GCL, there are also features that go beyond Euclidean and analytical geometry. For instance, in GCLC there is support for visualizing functions, graphs, trees, etc., so GCLC can substitute a wide range of tools for producing mathematical illustrations and can serve as a general mathematical illustration tool.

Some tools, like *GeoGebra*, can deal with algebraic expressions and perform symbolic computations, including finding derivatives and integrals of functions [15]. Some tools, like C.a.R., can perform a range of numeric computations, including numeric integration. GCLC provides support for simple calculations (including, for example, trigonometric functions). Symbolic algebraic computations (like computing Gröbner bases) are used by the theorem provers, but are not accessible to the user.

Some geometry tools have multilingual support (e.g., *Cabri*, *C.a.R*, *Cinderella*, *Euklides*, *GeoGebra*, *Kig*, *KSEG*[31]). There is only the English version of GCLC/WINGCLC.

Most (if not all) dynamic geometry tools provide support for various export formats, and many of them have support for LaTeX. In GCLC, there is currently support for three LaTeX-based formats and the user can use the full power or LaTeX in labelling objects and printing text and, hence, have high-quality digital illustrations.

7 Conclusions and Future Work

We presented the Geometry Constructions Language (GCL), a language for explicit describing constructions in Euclidean plane and their properties. The basic idea behind the GCL language is that geometric constructions are formal procedures, rather than drawings. Therefore, in GCL, one explicitly *describes* constructions and figures, rather than *draws* figures.

After years of development, the GCL processor GCLC is much more than a geometry and visualization tool. There is support for symbolic expressions, for parametric curves and surfaces, for flow control, etc. The built-in theorem provers can automatically prove a number of complex geometry theorems. WINGCLC is an interactive version of GCLC, a dynamic mathematical tool with a range of functionalities. These features make the tools GCLC and WINGCLC powerful mechanized geometry systems, with the main applications in producing digital mathematical illustrations, storing mathematical contents, mathematical education, and in studies of automated reasoning in geometry. The systems are publicly available and have thousands of users worldwide

[30] `http://mathsrv.ku-eichstaett.de/MGF/homes/grothman/java/zirkel/doc_en/`

[31] `http://www.mit.edu/~ibaran/kseg.html`

Modern mathematical tools have to combine powerful support for visualization, symbolic and numerical calculations, and for automated theorem proving. The tools GCLC and WINGCLC aim at this direction. Possible directions for further development of the language GCL and its processors are in: improving support for symbolic and numeric calculations (e.g., adding support for symbolic derivations, numeric integration, etc.), in developing new theorem provers, able to generate human-readable proofs (e.g., based on coherent logic [2,13]), in developing tools for automated solving of construction problems, and in developing support for automated discovery of geometry theorems.

Acknowledgements. I am grateful to the anonymous reviewers, to Pedro Quaresma, and to Julien Narboux for comments made on earlier versions of this paper.

References

1. Yves Bertot, Frédérique Guilhot, and Loic Pottier. Visualizing geometrical statements with geoview. *Electr. Notes Theor. Comput. Sci.*, 103:49–65, 2004.
2. Marc Bezem and Thierry Coquand. Automating coherent logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2005*, volume 3835 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
3. Francisco Botana. A web-based intelligent system for geometric discovery. In *International Conference on Computational Science*, volume 2657 of *Lecture Notes in Computer Science*, pages 801–810. Springer, 2003.
4. Francisco Botana. Format exchange in dynamic geometry. In *Workshop Intergeo-España*, 2007.
5. Francisco Botana and Tomás Recio. Towards solving the dynamic geometry bottleneck via a symbolic approach. In *Automated Deduction in Geometry*, volume 3763 of *Lecture Notes in Computer Science*, pages 92–110. Springer, 2006.
6. Francisco Botana and José Valcarce. A dynamicsymbolic interface for geometric theorem discovery. *Computers and Education*, 38:2135, 2002.
7. C. C. Chou, OU X. S. Gao, and J. Z. Zhang. Automated production of traditional proofs for constructive geometry theorems. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, 1993.
8. S.C. Chou, X.S. Gao, and J.Z. Zhang. *Machine Proofs in Geometry*. World Scientific, Singapore, 1994.
9. S.C. Chou, X.S. Gao, and J.Z. Zhang. An Introduction to Geometry Expert. In M. A. McRobbie and Slaney J. K., editors, *CADE 13*, volume 1104 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.
10. Mirjana Djorić and Predrag Janičić. Constructions, instructions, interactions . *Teaching Mathematics and its Applications*, 23(2):69–88, 2004.
11. Zoran Lučić et. al. Euclid's Elements. on-line at: `http://www.matf.bg.ac.rs/nastavno/zlucic/`.
12. Bruno Buchberger et.al. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 2006.
13. John Fisher and Marc Bezem. Skolem machines and geometric logic. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *4th International Colloquium on Theoretical Aspects of Computing — ICTAC 2007*, volume 4711 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
14. Xiao-Shan Gao and Qiang Lin. Mmp/geometer a software package for automated geometric reasoning. In Franz Winkler, editor, *Automated Deduction in Geometry: 4th International Workshop, (ADG 2002)*, volume 2930 of *Lecture Notes in Computer Science*, pages 44–66. Springer-Verlag, 2004.
15. Markus Hohenwarter and Karl Fuchs. Combination of dynamic geometry, algebra and calculus in the software system GeoGebra. In *Proceedings of Computer Algebra Systems and Dynamic Geometry Systems in Mathematics Teaching Conference*, 2004.

16. Predrag Janičić. GCLC – A Tool for Constructive Euclidean Geometry and More than That. In Nobuki Takayama, Andres Iglesias, and Jaime Gutierrez, editors, *Proceedings of International Congress of Mathematical Software (ICMS 2006)*, volume 4151 of *Lecture Notes in Computer Science*, pages 58–73. Springer-Verlag, 2006.
17. Predrag Janičić and Pedro Quaresma. System description: Gclcprover + GeoThms. In Ulrich Furbach and Natarajan Shankar, editors, *International Joint Conference on Automated Reasoning (IJCAR-2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 145–150. Springer-Verlag, 2006.
18. Predrag Janičić. *Zbirka zadataka iz geometrije*. Skripta Internacional, Beograd, 1st edition 1997, 6th edition 2005. Collection of problems in geometry (in Serbian).
19. Predrag Janičić and Pedro Quaresma. Automatic verification of regular constructions in dynamic geometry systems. In F. Botana and T. Recio (Eds.), editors, *Automated Deduction in Geometry*, volume 4869 of *Lecture Notes in Artificial Intelligence*, pages 39–51. Springer-Verlag, 2007.
20. Predrag Janičić and Ivan Trajković. WinGCLC — a Workbench for Formally Describing Figures. In *Proceedings of the 18th Spring Conference on Computer Graphics (SCCG 2003)*, pages 251–256, Budmerice, Slovakia, April, 24-26 2003. ACM Press, New York, USA.
21. Donald Knuth. *TeXBook*. Addison Wesley Professional, 1986.
22. Ulrich Kortenkamp and Jürgen Richter-Gebert. Using automatic theorem proving to improve the usability of geometry software. In *Workshop on Mathematical User Interfaces*, 2004.
23. Leslie Lamport. *LaTeX: A Document Preparation System*. Addison Wesley Professional, 1994.
24. Tielin Liang and Dongming Wang. Towards a geometric-object-oriented language. In Hoon Hong and Dongming Wang, editors, *Automated Deduction in Geometry*, volume 3763 of *Lecture Notes in Computer Science*, pages 130–155. Springer, 2006.
25. Tielin Liang and Dongming Wang. On the design and implementation of a geometric-object-oriented language. *Frontiers of Computer Science in China*, 1(2):180–190, 2007.
26. Noboru Matsuda and Kurt Vanlehn. Gramy: A geometry theorem prover capable of construction. *Journal of Automated Reasoning*, 32:3–33, 2004.
27. Julien Narboux. A Graphical User Interface for Formal Proofs in Geometry. *Journal of Automated Reasoning*, 39(2), 2007.
28. Julien Narboux. Geoproof, a user interface for formal proofs in geometry. In *Mathematical User-Interfaces Workshop*, 2007.
29. Goran Predović. Automated geometry theorem proving based on Wu's and Buchberger's methods. Master's thesis, Faculty of Mathematics, University of Belgrade, 2008. supervisor: Predrag Janičić.
30. Pedro Quaresma and Predrag Janičić. Integrating dynamic geometry software, deduction systems, and theorem repositories. In J.M. Borwein and W.M. Farmer, editors, *Mathematical Knowledge Management (MKM-2006)*, volume 4108 of *Lecture Notes in Artificial Intelligence*, pages 280–294. Springer-Verlag, 2006.
31. Pedro Quaresma and Predrag Janičić. Geothms — a web system for euclidean constructive geometry. *Electronic Notes in Theoretical Computer Science*, 174(2):35–48, 2007.
32. Pedro Quaresma and Predrag Janičić. Framework for the Constructive Geometry. Technical Report TR2006/001, Center for Informatics and Systems of the University of Coimbra, 2006.
33. Pedro Quaresma, Predrag Janičić, Jelena Tomašević, Milena Vujošević-Janičić, and Dušan Tošić. XML-based Format for Geometry — XML-based Format for Descriptions of Geometrical Constructions and Geometrical Proofs. In J. M. Borwein, E. M. Rocha, and J. F. Rodrigues, editors, *Communicating Mathematics in Digital Era*, pages 183–197. A K Peters, Ltd. Wellesley, MA, USA, 2008.
34. Dongming Wang. Geother 1.1: Handling and proving geometric theorems automatically. In *Automated Deduction in Geometry*, volume 2930 of *Lecture Notes in Artificial Intelligence*, pages 194–215. Springer-Verlag, 2004.
35. Sean Wilson and Jacques Fleuriot. Combining dynamic geometry, automated geometry theorem proving and diagrammatic proofs. In *Workshop on User Interfaces for Theorem Provers (UITP)*, 2005.
36. Zheng Ye, Shang-Ching Chou, and Xiao-Shan Gao. An Introduction to Java Geometry Expert. In *Automated Deduction in Geometry*, 2008.