GeoGebra Automated Reasoning Tools
A Tutorial

Zoltán Kovács, Tomás Recio and M. Pilar Vélez

January 2017

1 Introduction

The software tool GeoGebra (`http://www.geogebra.org`) is capable of supporting the teaching of Euclidean plane geometry theorems by using symbolic computations. For these purposes some tools for the automatic proving and discovery of geometric theorems have been developed.

The novel technologies are still in an experimental phase in the classrooms. This document summarizes the technical possibilities by showing some examples also.

2 Starting GeoGebra

GeoGebra is available on many platforms, including

- desktop or laptop computers with various operating systems installed,
- tablets, and
- phones.

Also embedded GeoGebra applets are available on web pages, including GeoGebra Materials (`https://www.geogebra.org/materials/`) with millions of freely available teaching materials.

The available tools on the different platforms may however differ. Also the user experience on the various platforms may be different: the symbolic computations may need a high amount of calculations and the underlying hardware or software components may or may not support some steps completely.

The recommended platform for classroom use may vary. The fastest results can be obtained by fast desktop (or laptop) computers, but in this case the software must be downloaded and installed by the user. Some examples in this tutorial may work only in the "desktop" version (which is created for Microsoft Windows, Apple Macintosh and Linux desktop and laptop computers). On the other hand, the "web" version does not require installation by the user: it will run in a modern web browser, and the teacher is able to prepare a list of examples as GeoGebra applets in advance before the classroom use by using GeoGebra Materials, for example. The "web" version is however slower: the symbolic computations may be slower by a magnitude.

GeoGebra recently runs on tablets and phones also. In some cases these platforms provide faster user experience than the web version does, but the smaller screen size may prevent the users from investigating geometry theorems in details. It is encouraged that teachers do experiments by using these kind of modern devices, but their use for automated reasoning is still experimental.

There is continuous work on GeoGebra's automated reasoning tools. The best practice is to always use the latest version. A weekly update can always be expected for all versions, excluding the Mac App Store version which is updated about monthly. The list of newest changes can be found at `http://dev.geogebra.org/trac/timeline`—this is intended only for advanced users and developers.

3 Automated Reasoning Tools

Automated reasoning tools are a collection of GeoGebra tools and commands ready to conjecture, discover, adjust and prove geometric statements in a dynamic geometric construction.

First the user needs to draw a geometric figure by using certain tools listed by default on top of the main window in GeoGebra. After constructing the figure, GeoGebra has many ways to promote investigating geometrical properties of a figure by various tools and settings:

1. By *dragging the free objects* their dependent objects can be visually investigated.

2. The *Relation* tool helps comparing objects and obtaining relations.

3. By setting the *trace* of a constructed object on/off the movement of an object will be visualized when its parent objects are changing.

4. The *Locus* tool shows the trace of an object for all possible positions of a parent object (while it moves on a path).

5. By typing the `Relation` or `Locus` command in GeoGebra's Input Bar more refined information can be obtained.

These methods are usually well known by the GeoGebra community, and therefore they are well documented and many examples can be found on them at GeoGebra Materials (`https://www.geogebra.org/materials/`). On the other hand, currently GeoGebra also offers symbolic automated reasoning tools for generalizing the observed/conjectured geometric properties:

6. The Relation tool and command can be used to recompute the results symbolically,

7. The `LocusEquation` command refines the result of the Locus command by displaying the algebraic equation of the graphical output.
8. The `LocusEquation` command can investigate implicit loci.
9. The `Envelope` command computes the equation of a curve which is tangent to a family of objects while a certain parent of the object moves on a path.

3.1 High and low level tools

GeoGebra provides the above high level methods to promote investigating geometry theorems. These icons are considered as "high level" tools because of their ease of use and thus they can be directly shown in classrooms. There are also further ways to learn more on the mathematical background or just to help in troubleshooting. Those "low level" methods are listed in the Appendix, and therefore not suggested for direct use among students.

Obviously, some of the listed methods are easier, and others are more difficult. Using the command line in the Input Bar in GeoGebra can be considered as a more difficult way for most users. That is, it can be suggested that a teacher first shows the easier methods, and later demonstrates the other ways when the pupils have enough experiments done.

3.2 Tools with symbolic support

As mentioned above some automated reasoning tools are provided of symbolic support. This feature allow to verify in a mathematically rigorous way general statements of elementary Geometry that have been conjectured by the user.

A general hint that the user should start GeoGebra on startup in "graphing calculator" mode. This turns on showing the labels on each newly added object—this can be crucial for the Relation tool and command when reporting on various relations.

In most cases, however, the axes are not really necessary to be shown: they can be switched off when the *Move* tool is active (it is the leftmost icon showing an arrow cursor) by right-click in the *Graphics View* and de-selecting the *Axes* setting.

In some cases the *Algebra View* is not needed to be displayed—unless the equations of the implicit curves are to be investigated in details, this can however be done also by changing the object's label to contain its value, too. (To do so, by right-clicking on the object, choosing *Object Properties*, the user needs to set *Show Label* to *Value* on the *Basic* tab.)

3.2.1 The Relation tool and command

GeoGebra's Relation tool and command shows a message box that gives the user information about the relation between two or more objects. This command allows the user to find out numerically (that is, for the drawing construction with assigned coordinates) whether

- two lines are perpendicular,
- two lines are parallel,
- two (or more) objects (points, segment lengths, polygon areas) are equal,
- a point lies on a line or conic,
- a line is tangent or a passing line to a conic,
- three points are collinear,
- three lines are concurrent (or parallel),
- four points are concyclic (or collinear).

Some of these checks can also be performed symbolically, that is, the statement can be verified rigorously for the general case (with arbitrary coordinates) and not only for the pictured concrete geometric construction.

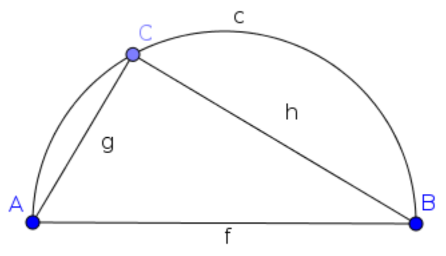
When using the Relation *tool*, the user points on two objects to get the message box shown. Alternatively, two, three or four objects can be selected by the selection rectangle to invoke the message box.

When using the Relation *command*, the user types one of the following formulas in the Input Bar:

- `Relation[ <Object>, <Object> ]`
- `Relation[ { <Object>, <Object> } ]`
- `Relation[ { <Object>, <Object>, <Object> } ]`
- `Relation[ { <Object>, <Object>, <Object>, <Object> } ]`

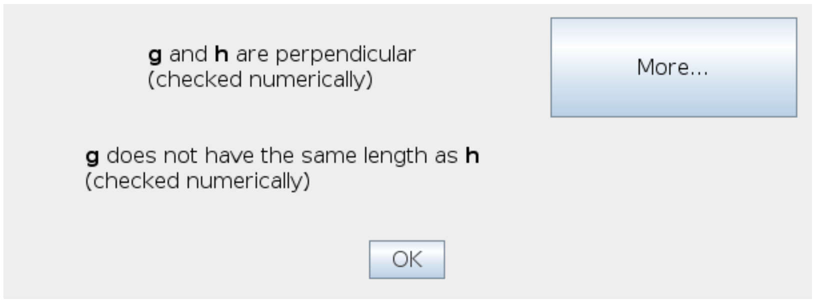
When the message box is shown with one or more true numerical statements on the objects, there may be a button "More..." shown if there is symbolic support for the given statement. When clicking "More...", shortly the numerical statement will be updated with a more general symbolic one.

Example (Thales' circle theorem)



1. By using the *Segment* tool, construct AB.

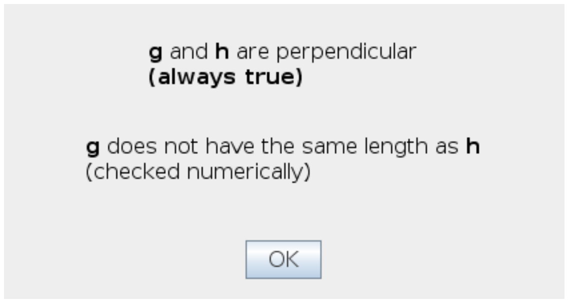
2. By choosing the *Semicircle through 2 Points* tool, create arc c.
3. Put point C on c.
4. Create segments AC and BC and denote them by g and h, respectively.
5. Compare g and h by using the Relation tool and pointing on g and h by the mouse, or type `Relation[g,h]` in the Input Bar. The following message will be shown:



g and **h** are perpendicular
(checked numerically)

g does not have the same length as **h**
(checked numerically)

6. Click "More..."—the message will be changed as follows:



g and **h** are perpendicular
(always true)

g does not have the same length as **h**
(checked numerically)

Remark that the Relation command (step 5) looks for relations between g and h from the coordinates and equations assigned to the drawn construction. However by clicking "More..." (step 6) we verify that g and h are perpendicular for any points A and B we can choose at step 1.

3.2.2 The `LocusEquation` command

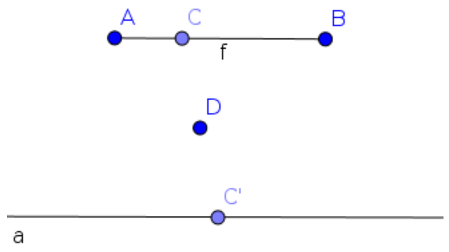
This command calculates the equation of a locus and plots this as an implicit curve. There are two kinds of uses:

- **Explicit locus.** Given an input point $\mathcal{I}$ on a path $\mathcal{P}$, some construction steps, and an output point $\mathcal{O}$. The task is to determine the equation $\mathcal{E}$ of $\mathcal{O}$ while $\mathcal{I}$ is moving on $\mathcal{P}$, and then plot $\mathcal{E}$. $\mathcal{I}$ is usually called *mover*, $\mathcal{O}$ is the *tracer*. $\mathcal{E}$ is called *locus equation*, and its graphical visualization is the *locus*.

  The syntax of the command is

```
LocusEquation[ <Point Tracer>, <Point Mover> ].
```

Example



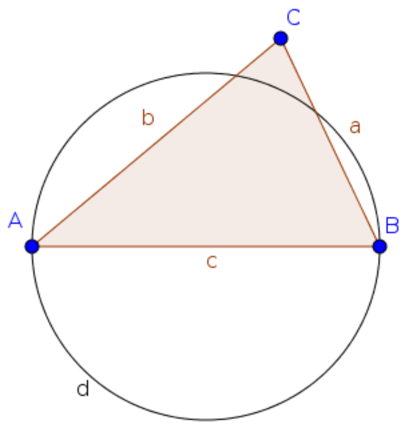
1. By using the *Segment* tool, construct AB. This automatically creates segment f.
2. Put point C on f.
3. Create point D by using the *Point* tool.
4. By using the *Reflect about Point* tool, reflect C about D. This defines C'.
5. Type `LocusEquation[C’,C]` in the Input Bar. Now the implicit curve a will be computed and plotted. This should be a segment (the mirror image of f about D), but GeoGebra needs to handle f as a line instead of a segment (for algebraic geometrical reasons), thus its mirror image is also a line.
6. Try dragging each draggable objects. It can be visually concluded that the mirror image of a segment about a point is always parallel to the preimage.

- **Implicit locus.** Given an input point $\mathcal{I}$, either as a free point, or on a path $\mathcal{P}$. There are some construction steps given. The user claims a Boolean condition $\mathcal{C}$ on some objects of the construction. The task is to determine an equation $\mathcal{E}$ such that for all points $\mathcal{I}'$ of it, if $\mathcal{I} = \mathcal{I}'$, then $\mathcal{C}$ holds. Again, $\mathcal{E}$ is called locus equation, and its graphical representation is the locus.

  The syntax of the command is

```
LocusEquation[ <Boolean Expression>, <Point> ].
```

Example



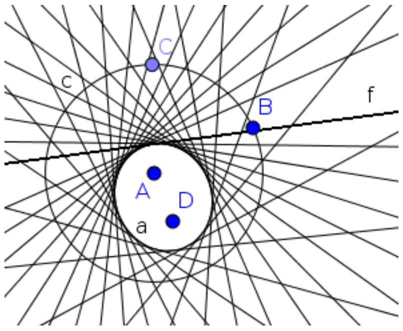
1. By using the *Polygon* tool, construct triangle ABC. Now segments a, b and c will be automatically introduced by GeoGebra.
2. Type `LocusEquation[a^2+b^2==c^2,C]` in the Input Bar. Now the implicit curve d will be computed and plotted, which seems to be a circle. Note that *two* equal signs must be entered; another possibility is to use $\overset{?}{=}$ (by clicking the $\boxed{\alpha}$ icon, or inserting this symbol from an external application by using Copy and Paste).
3. Try dragging each draggable objects. It can be visually concluded that if C lies on a circle whose diameter is AB, then—because of the right property of the triangle—$a^2 + b^2 = c^2$ indeed follows.

3.2.3 The `Envelope` command

This command computes the equation of a curve which is tangent to a family of objects while a certain parent of the object moves on a path.

More precisely, given an input point $\mathcal{I}$ on a path $\mathcal{P}$, some construction steps, and an output path $\mathcal{O}$, either a line or a circle. The task is to determine the equation $\mathcal{E}$ of a curve $\mathcal{C}$ which is tangent to $\mathcal{O}$, while $\mathcal{I}$ is moving on $\mathcal{P}$. Then finally plot $\mathcal{E}$. $\mathcal{I}$ is called the mover. $\mathcal{E}$ is called the *envelope equation*, and its graphical visualization is the *envelope*.

Example



1. By using the *Circle with Center through Point* tool, construct circle c with center A and circumpoint B.
2. Put point C on c.
3. Create an arbitrary point D inside c.
4. Construct the *Perpendicular Bisector* f of segment CD by using its endpoints.
5. Type `Envelope[f,C]` in the Input Bar. Now the implicit curve a will be computed and plotted, and it seems to be an ellipse.

3.3 Technical notes

The following notes are important restrictions for each automated reasoning tool in GeoGebra which uses symbolic computations:

- Not all GeoGebra tools and construction steps are supported.
- The supported tools work only for a restricted set of geometric objects, i.e. using points, lines, circles, conics.
- Rays and line segments will be treated as infinite lines. Circle arcs will be treated as circles.
- Too complicated locus or envelope computations may return ‘undefined’ in the Algebra View.
- If there is no locus or envelope, then the implicit curve is the empty set $0 = 1$. Example: for an arbitrary point P

  ```
  LocusEquation[false,P]
  ```

  returns the empty set.

- If the locus or the envelope is the whole plane, then the implicit curve is the equation $0 = 0$. Example: for an arbitrary point P

  ```
  LocusEquation[true,P]
  ```

  returns the whole plane.
- Sometimes extra branches of the curve will appear that were not in the original locus or envelope.
- The graph of the implicit curve may be inaccurate in some cases.

4 Classroom uses: conjecture, proof and generalization

Technically the easiest symbolic tool is the Relation tool in the list above. On the other hand, some teaching scenarios may require different tools to consider, or more than one tool, but in a different order than listed above.

4.1 Thales' circle theorem

In many traditional math classes Thales' circle theorem is stated in an explicit form: if C is on a semicircle, the segments g and h are perpendicular. Actually this theorem can be formulated by using an open ended question: *Let ABC be an arbitrary triangle. What is the geometric locus of C if the angle at C must be right?*

In this approach it may make more sense to use the technically more difficult `LocusEquation[g⊥h,C]` command first, than finishing the construction and use the Relation tool or command directly. What is more, the output of the `LocusEquation` command can suggest a conjecture for the pupils, namely that the curve is indeed a circle. The Algebra View shows the equation of the locus, this can be however difficult for younger learners to identify.

Finally, Thales' circle theorem can be generalized towards the theorem of the inscribed angle in a triangle. In this case the condition is no longer $g \perp h$, but that the angle between them equals to a fix one. GeoGebra currently supports entering this kind of investigation with the syntax

```
LocusEquation[AreCongruent[α,β],C]
```

if α is fixed and $\beta = \angle ACB$.

To sum up, in this approach

1. an implicit locus is computed with GeoGebra,
2. a conjecture for the output curve is made by the pupil,
3. the conjecture is checked by the Relation tool or command in GeoGebra,

4. the proof can be optionally worked out by paper and pencil by the pupil,

5. the theorem can be generalized by plotting further implicit loci with GeoGebra—as further experiments for the pupil.

4.2 Further examples

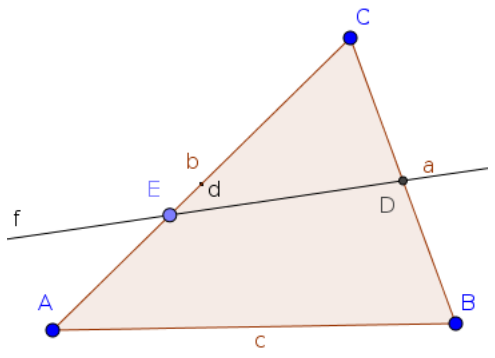
The triangle inequality can be translated to an equality which can be turned into an investigation of degenerated triangles. As a generalization, the synthetic definition of conic sections can be mentioned.

Another application is to derive the locus equation in triangle ABC with the condition $a \overset{?}{=} b$, here C is to be found (step 1). Clearly, C must lie on the bisector of segment AB (step 2). As by explicitly putting C on the bisector, GeoGebra confirms that $AC = BC$ when starting the Relation tool's symbolic machinery (step 3). After proving the statement by traditional means (step 4), a generalization can be obtained by typing e.g. `LocusEquation[a==2b,C]`: this can be an interesting experiment for advanced learners, too (step 5).

4.3 A worked out example: The midline theorem

Here step-by-step instructions are provided on a possible way on investigating the midline theorem by using GeoGebra's automated reasoning tools.

Step 1



1. By using the *Polygon* tool, construct triangle ABC. This will automatically create segments a, b and c.

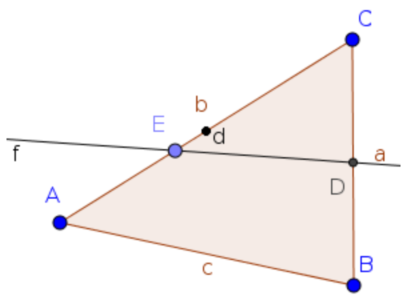
2. By using the *Midpoint or Center* tool, create the midpoint D of a.

3. Put point E on b.

4. Create line f which joins D and E.

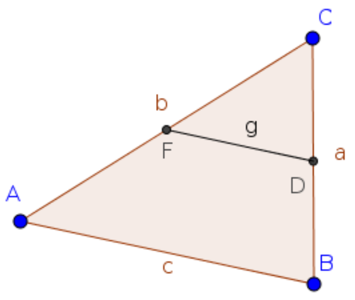
5. Ask GeoGebra on the requirement for E in order to have f parallel to c: type `LocusEquation[c||f,E]` in the Input Bar. Now the implicit curve d will be computed and plotted, and it seems to be a single point. Note: it may to be useful to change the line thickness of the implicit curve d, and also to increase ist layer number to ensure that other objects do not hide it. Both settings can be changed in the Object properties window.

Step 2



6. Drag the free objects and conjecture that E must be the midpoint of b.

Step 3



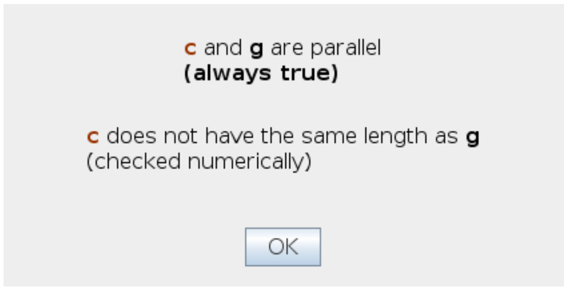
7. Make the objects E, f and d invisible by hiding them.

8. Create the midpoint F of segment b.

9. Join D and F by segment g.

10. Use the *Relation* tool to compare c and g. They seem to be parallel.

11. Click “More…” in the popup window and check symbolically that they are indeed parallel.



c and g are parallel
(always true)

c does not have the same length as g
(checked numerically)

The pupils may continue with step 4 if they need an elegant way to prove this statement, or just stop here if there is no time for further work in the classroom.

Also, in step 5 further questions can be raised. c and g do not have the same length—but can g be computed by using the length of c? Maybe $c = 1.5 \cdot g$, or maybe more? The GeoGebra command `Relation[c,1.5g]` will result in the answer that c and $1.5g$ are not equal, but maybe there is another constant than 1.5 which results in a positive answer... Even if there is no time for further work in the classroom, some pupils may find these questions interesting and they can continue thinking on them alone or in groups—but in some sense *independently*, using the computer as an expert system.

5 Appendix

5.1 Low level GeoGebra tools

Automated reasoning tools in GeoGebra are completed by some low level tools prepared for learning more and in a more accurate way about geometric properties.

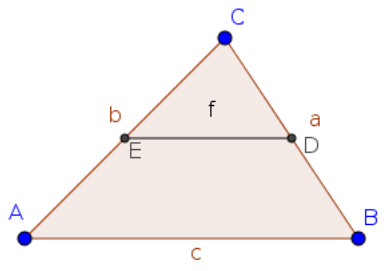
5.1.1 The `Prove` command

The `Prove` command decides if a geometric statement is true in general. It has three possible outputs:

- *true* means that the statement is always true, or true under some non-degeneracy conditions.
- *false* means that the statement is false in general. GeoGebra uses algebraic geometry in many cases to decide such questions. In algebraic geometry "generally true" (true in "most" cases) and "generally false" (false in "most" cases) are not opposite properties, that is, a statement can be not "generally true" and not "generally false" at the same time. GeoGebra interprets this special case as *false* (since it is not generally true).
- *undefined* means that GeoGebra cannot decide because of some reason:

  - The statement cannot be translated into a model which can be further investigated. This usually means that algebraization of the statement failed because of
    - ∗ theoretical impossibility (e.g. using a transcendent function as a construction step, for example, sine of x),
    - ∗ missing implementation in GeoGebra.
  - The translated statement in algebraic geometry is too difficult to solve. This means that either there are too many variables, or the equations are hard to handle by the solver algorithm. This results in either a timeout or an out of memory error.
  - The solver algorithm was able to investigate the situation, but the result is ambiguous: either the statement is false, or it is true under certain conditions—but the algorithm was not able decide which case is present.
  - There was an internal error in GeoGebra during the computations.

Example



1. Construct the triangle ABC by using the *Polygon* tool.
2. Construct the midpoints D and E of sides a and b, respectively, by using the *Midpoint or Center* tool.
3. By using the *Segment* tool, create f by joining D and E.
4. Type `Prove[f‖c]` to obtain *true* in the Algebra View as Boolean Value d. Note that the parallel sign must be entered by using either
   - the list of the mathematical symbols by clicking the $\boxed{\alpha}$ icon in the Input Bar, or
   - inserting this symbol externally by using Copy and Paste.
   - Alternatively, `f‖c` can be substituted by `AreParallel[f,c]` also.
5. Type `Prove[c==3f]`. Now the answer is *undefined*, because GeoGebra cannot decide if the statement is false or it is true under certain conditions. In such cases the `ProveDetails` command can help (see below). Note that *two* equal signs must be entered; another possibilities are to use

- $\overset{?}{=}$ (by clicking the $\boxed{\alpha}$ icon, or inserting this symbol from an external application by using Copy and Paste), or
- alternatively, `AreEqual[c,3f]`. In this case the full command is `Prove[AreEqual[c,3f]]`

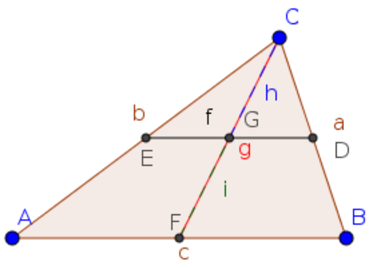
5.1.2 The `ProveDetails` command

The `ProveDetails` command has similar behavior like the `Prove` command has, but it may use different algorithms in the decision process, and may provide more information on the results. It has three possible outputs:

- {*true*} means that the statement is always true.
- {*true*, {...}} if the statement is true under some non-degeneracy or essential conditions: these conditions are listed in the internal braces. (If the list remains "...", it means that no synthetic translation could be found.) If the conjunction of the negated conditions is true, then the statement is true.
- {*false*} means that the statement is false in general. See the comments at the `Prove` command for more details on this.

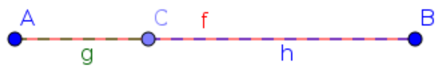
Example (continued)

6. Type `ProveDetails[c==3f]`. Now the answer is {*false*}.
7. Type `ProveDetails[c==2f]`. Now the answer is {*true*}.



8. Now let F be the midpoint of c, and let us denote segment CF by g. Let G be the intersection point of f and g. Finally, let us denote segments CG and FG by h and i, respectively. In this case `ProveDetails[h==i]` returns {*true*,{ *"AreCollinear[A,B,C]"*}} which means that if A, B and C are not collinear, then $h = i$.

Another example



1. Let AB a segment, denoted by f.
2. Let C be a point on f.
3. Let us denote segments AC and BC by g and h, respectively.
4. Type `ProveDetails[f==g+h]`. Now the answer is

   $$\{true,\{\text{“}g{+}f{=}h\text{”},\ \text{“}h{+}f{=}g\text{”}\}\}$$

   which means that if $g + f \neq h$ and $h + f \neq g$, then $f = g + h$.

5.1.3 Debugging

Starting GeoGebra via command line there are more possibilities to investigate the results. Here the method on a typical Linux installation is demonstrated.

The user needs to start GeoGebra by the following command:

```
geogebra --logfile=/dev/stdout --logshowcaller=false \
        --logshowtime=false --logshowlevel=false
```

A typical output looks like as follows:

```
Using AUTO
Using BOTANAS_PROVER
A = (3.42, 1.86) /* free point */
// Free point A(v1,v2)
B = (10.48, 3.1) /* free point */
// Free point B(v3,v4)
f = Segment[A, B] /* Segment [A, B] */
C = Point[f] /* Point on f */
// Constrained point C(v5,v6)
Hypotheses:
1. -v5*v4+v6*v3+v5*v2-v3*v2-v6*v1+v4*v1
g = Segment[A, C] /* Segment [A, C] */
h = Segment[C, B] /* Segment [C, B] */
Processing numerical object
Hypotheses have been processed.
giac evalRaw input: evalfa(expand(ggbtmpvarf))
giac evalRaw output: ggbtmpvarf
input = expand(ggbtmpvarf)
result = ggbtmpvarf
eliminate([ggbtmpvarf-((ggbtmpvarg)+(ggbtmpvarh))=0,ggbtmpvarh^2=v11^2,ggbtmpvarg^2=v12^2,
     ggbtmpvarf^2=v13^2],[ggbtmpvarh,ggbtmpvarg,ggbtmpvarf])
giac evalRaw input: evalfa(eliminate([ggbtmpvarf-((ggbtmpvarg)+(ggbtmpvarh))=0,ggbtmpvarh^2=
     v11^2,ggbtmpvarg^2=v12^2,ggbtmpvarf^2=v13^2],[ggbtmpvarh,ggbtmpvarg,ggbtmpvarf]))
Running a probabilistic check for the reconstructed Groebner basis. If successfull, error
     probability is less than 1e-07 and is estimated to be less than 10^-18. Use
     proba_epsilon:=0 to certify (this takes more time).
// Groebner basis computation time 0.000448 Memory -1e-06M
giac evalRaw output: {v11^4-2*v11^2*v12^2+v12^4-2*v11^2*v13^2-2*v12^2*v13^2+v13^4}
input = eliminate([ggbtmpvarf-((ggbtmpvarg)+(ggbtmpvarh))=0,ggbtmpvarh^2=v11^2,ggbtmpvarg^2=
     v12^2,ggbtmpvarf^2=v13^2],[ggbtmpvarh,ggbtmpvarg,ggbtmpvarf])
```

```
result = {v11^4-2*v11^2*v12^2+v12^4-2*v11^2*v13^2-2*v12^2*v13^2+v13^4}
giac evalRaw input: evalfa(eliminate([ggbtmpvarf-((ggbtmpvarg)+(ggbtmpvarh))=0,ggbtmpvarh=
     v11,ggbtmpvarg=v12,ggbtmpvarf=v13],[ggbtmpvarh,ggbtmpvarg,ggbtmpvarf]))
Running a probabilistic check for the reconstructed Groebner basis. If successfull, error
     probability is less than 1e-07 and is estimated to be less than 10^-18. Use
     proba_epsilon:=0 to certify (this takes more time).
// Groebner basis computation time 0.000592 Memory -1e-06M
giac evalRaw output: {v11+v12-v13}
input = eliminate([ggbtmpvarf-((ggbtmpvarg)+(ggbtmpvarh))=0,ggbtmpvarh=v11,ggbtmpvarg=v12,
     ggbtmpvarf=v13],[ggbtmpvarh,ggbtmpvarg,ggbtmpvarf])
result = {v11+v12-v13}
giac evalRaw input: evalfa(simplify({v11^4-2*v11^2*v12^2+v12^4-2*v11^2*v13^2-2*v12^2*v13^2+
     v13^4}/{v11+v12-v13}))
giac evalRaw output: {v11^3-v11^2*v12+v11^2*v13-v11*v12^2-2*v11*v12*v13-v11*v13^2+v12^3+v12
     ^2*v13-v12*v13^2-v13^3}
input = simplify({v11^4-2*v11^2*v12^2+v12^4-2*v11^2*v13^2-2*v12^2*v13^2+v13^4}/{v11+v12-v13
     })
result = {v11^3-v11^2*v12+v11^2*v13-v11*v12^2-2*v11*v12*v13-v11*v13^2+v12^3+v12^2*v13-v12*
     v13^2-v13^3}
giac evalRaw input: evalfa(factor(v11^3-v11^2*v12+v11^2*v13-v11*v12^2-2*v11*v12*v13-v11*v13
     ^2+v12^3+v12^2*v13-v12*v13^2-v13^3))
giac evalRaw output: (v11-v12-v13)*(v11-v12+v13)*(v11+v12+v13)
input = factor(v11^3-v11^2*v12+v11^2*v13-v11*v12^2-2*v11*v12*v13-v11*v13^2+v12^3+v12^2*v13-
     v12*v13^2-v13^3)
result = (v11-v12-v13)*(v11-v12+v13)*(v11+v12+v13)
Trying to detect polynomial -v13-v12+v11
-v13-v12+v11 means h = f + g
Trying to detect polynomial v13-v12+v11
v13-v12+v11 means f + h = g
Trying to detect polynomial v13+v12+v11
v13+v12+v11 means f + g + h = 0, uninteresting
Thesis equations (non-denied ones):
2. v11^2-v6^2-v5^2+2*v6*v4-v4^2+2*v5*v3-v3^2
3. v12^2-v6^2-v5^2+2*v6*v2-v2^2+2*v5*v1-v1^2
4. v13^2-v4^2-v3^2+2*v4*v2-v2^2+2*v3*v1-v1^2
Thesis reductio ad absurdum (denied statement), product of factors:
(v13^4-2*v13^2*v12^2+v12^4-2*v13^2*v11^2-2*v12^2*v11^2+v11^4)*v14-1
that is,
5. -1+v14*v13^4-2*v14*v13^2*v12^2+v14*v12^4-2*v14*v13^2*v11^2-2*v14*v12^2*v11^2+v14*v11^4
substitutions: {v1=0, v2=0}
Eliminating system in 8 variables (5 dependent)
giac evalRaw input: evalfa([[ff:=\"\"],[aa:=eliminate2([v12^2-v6^2-v5^2,v11^2-v6^2-v5^2+2*v6
     *v4-v4^2+2*v5*v3-v3^2,-1+v14*v13^4-2*v14*v13^2*v12^2+v14*v12^4-2*v14*v13^2*v11^2-2*v14*
     v12^2*v11^2+v14*v11^4,v13^2-v4^2-v3^2,-v5*v4+v6*v3],revlist([v6,v11,v12,v13,v14]))],[bb
     :=size(aa)],[for ii from 0 to bb-1 do ff+=(\"[\"+(ii+1)+\"]: [1]:
     unicode95uunicode91u1]=1\");cc:=factors(aa[ii]);dd:=size(cc);for jj from 0 to dd-1 by 2
      do ff+=(\"  unicode95uunicode91u\"+(jj/2+2)+\"]=\"+cc[jj]); od; ff+=(\" [2]: \"+cc[1])
     ;for kk from 1 to dd-1 by 2 do ff+=(\",\"+cc[kk]);od;od],[if(ff==\"\") begin ff:=[0]
     end],ff][5])
Running a probabilistic check for the reconstructed Groebner basis. If successfull, error
     probability is less than 1e-07 and is estimated to be less than 10^-7. Use
     proba_epsilon:=0 to certify (this takes more time).
// Groebner basis computation time 0.000249 Memory -1e-06M
giac evalRaw output: "[1]: [1]:  unicode95uunicode91u1]=1  unicode95uunicode91u2]=1 [2]:
     1,1"
Considering NDG 1...
Found a better NDG score (0.0) than Infinity
Statement is GENERALLY TRUE
Benchmarking: 38 ms
STATEMENT IS TRUE (yes/no: TRUE)
OUTPUT for ProveDetails: null = {true, {"f + h = g", "h = f + g"}}
```

There is intentionally no easier way to show the users this type of output. However, the last few lines of the debug information is available in GeoGebra in the *Help* menu, by choosing *About/License*, and clicking *System Information*—this copies the latest debug messages into the clipboard.