# Using GCLC and its Theorem Provers for Teaching Geometry

Milena Marić[1], Predrag Janičić[2]

[1]Architectural Technical High-school, Belgrade,

[2]Faculty of Mathematics, Belgrade

Computer Algebra and Dynamic Geometry in Mathematics Education, Novi Sad, 2012.

## Overview

# Motivation

- It is all fine to use dynamic geometry tools in education, but there are some basic issues to be addressed first.
- Do the pupils (and at what extend) understand formulations of geometry problems they are required to solve?
- Do the pupils understand the notion of mathematical proof?
- Are the teachers aware of these issues and how do they deal with these?

# Example geometry problem from a textbook

- We focus on geometry of triangles for high-school pupils.

**Problem**

Prove that in a right-angle triangle, the right angle bisector also bisects the angle formed by the altitude and the median over the hypotenuse.

- To understand the problem, the pupil must know what is:
  1. a right triangle
  2. an angle bisector
  3. an altitude over the given side
  4. a median over the given side
  5. a hypotenuse

# Possible problems

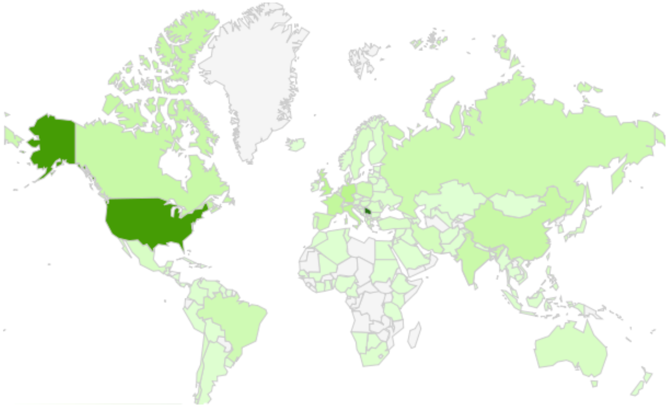
- For the above problem, it may be problematic to the pupil:
  1. to understand the problem;
  2. to make a corresponding illustration;
  3. to prove the conjecture.

# GCLC

- In our high-school geometry course we use the GCLC tool.
- GCLC: a tool for mathematical education, for producing mathematical illustrations, and a research tool.
- Basic principle: *a construction is a formal procedure, not an image*.
- Uses procedural (constructive) specifications of geometry figures.
- Freely available from `http://www.matf.bg.ac.rs/~janicic/gclc`, versions for Windows and Linux

# GCLC (2)

- Used in high-schools and university courses, and for publishing worldwide



# Declarative geometry language

- A geometry configurations relevant for some conjecture is typically specified in a declarative way.
- Declarative specification lists constraints that hold.
- It is important to identify a set of relations that can be encountered in formulations of problems.
- Make clear which relations are primitive (given by axioms) and which are derived/defined.

# Descriptive language — relations

- Following the above idea and the textbook used in the course, we have formulated relations that build our declarative geometry language.
- Some examples:

| | |
|---|---|
| $incident(A, l)$ | Point $A$ is on line $l$. |
| $between(A, B, C)$ | Point $B$ is between points $A$ and $C$. |
| $congruent(A, B, C, D)$ | Segment $AB$ is congruent to $CD$. |
| $sameside(A, B, S)$ | Points $A$, $B$ and $S$ are collinear and $A$ and $B$ are on the same side of $S$. |
| $midpoint(S, A, B)$ | Point $S$ is a midpoint the segment $AB$. |
| $perpendicular(A, B, C, D)$ | Lines $AB$ and $CD$ are perpendicular. |
| $righttriangle(A, B, C)$ | $\triangle ABC$ has right angle at point $A$ |

# Example problem

## Example Problem — a declarative formulation

$$\begin{aligned} \mathit{righttriangle}(A, B, C) \wedge \mathit{altitudefoot}(H, C, A, B) \wedge \mathit{midpoint}(T, A, B) \quad \Rightarrow \\ \mathit{bisector}(A, C, B) = \mathit{bisector}(H, C, T) \end{aligned}$$

# Primitive and defined relations

- *incident*, *between*, and *congruent* are the only primitive basic relations.
- Other relations can be defined in terms of simpler relations.
- For example, $righttriangle(A, B, C)$ can be reduced to $perpendicular(A, B, A, C)$.

# Degenerate cases

- Degenerate cases are important!
- Textbook usually does not make clear what happens in degenerate cases.
- For example, does $triangle(A, B, C)$ imply that $A$, $B$ and $C$ are disjoint?

## How to visualize a problem given in the declarative way?

- Declarative specifications can be precise, but they don't show how to construct (and visualize) the given configuration!
- Also, dynamic geometry software does not accept declarative specifications (no matter if it accepts a text input or is GUI based)!
- For declarative specifications of geometry configurations there may be simple procedural counterparts (sequences of construction steps).

# From declarative to procedural/constructive

- Procedural problem formulations use functions instead of relations (e.g., $p = intersection(l_1, l_2)$ instead of $intersection(p, l_1, l_2)$).
- Procedural specifications help students to understand the problem (to take into account all constraints)
- Procedural specifications help students to make visualizations (an illustrations of the problem)
- Procedural specifications can be simply visualized using dynamic geometry software.

# Procedural language — a set of construction steps

- It is necessary to specify all construction steps that can be used when formulating a problem.
- Construction steps might be either elementary (e.g., constructing a line trough two points) or derived (e.g., constructing a segment bisector).
- The teacher should insist that pupils can reduce each derived step to a sequence of primitive steps (if possible).
- The choice of possible construction steps determines the constructibility for problems.

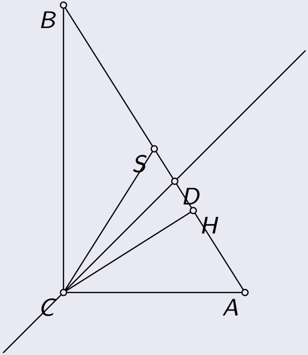
# Procedural language — construction steps

Some construction steps:

| | |
|---|---|
| *freepoint*() | construct a free point |
| *line*($A$, $B$) | construct a line trough the two given points |
| *circle*($O$, $A$, $B$) | construct a circle centered at $O$ with a radius $AB$ |
| *intersect*($l_1$, $l_2$) | construct an intersection of lines $l_1$ and $l_2$ |
| *midpoint*($A$, $B$) | construct a midpoint of the segment $AB$ |
| *angle*($A$, $B$, $\alpha$) | angle of $\alpha$ degrees. |

# Example problem

## Problem — a procedural description




$$
\begin{aligned}
A &= freepoint() \\
C &= freepoint() \\
ac &= line(A, C) \\
bc &= perpenicular(C, ac) \\
B &= point\_on\_line(bc) \\
h &= perpenicular(C, ab) \\
S &= midpoint(A, B) \\
cd &= bisector(A, C, B) \\
D &= intersection(ab, cd)
\end{aligned}
$$

# Translating declarative formulations to procedural specifications

- There are two approaches and two goals in solving this translation problem:
  1. Teach pupils to make a construction based on a declarative problem formulation (and a corresponding procedural formulation).
  2. Create software that can automatically convert a descriptive to a procedural formulation.

# Automatic conversion — pro et contra?

- The problem can be very hard since it can require solving arbitrary complex geometric constructive problem.
- However, experience shows that many textbook problems can be easily constructed.
- At all levels of education, automatic conversion would be welcome, but only as support tool.

# A harder example

**Example**

In an isoceles triangle $\triangle ABC (AC = BC)$, the line $p$ contains the point $C$ and intersects $AB$ in $M$ such tat $AC = AM$ and $CM = MB$. ...

Although it is not explicit in the specification, the above configuration is possible only in the special case when $\angle CAB = \angle CBA = 72^\circ$, so the construction must start from this special triangle.

## How to construct a conversion algorithm?

- Often an atomic statement of the declarative specification implies an object that can be simply described by a sequence of construction steps.
- Example:

$$
\mathit{righttriangle}(A,B,C) \quad \left| \quad \begin{array}{l} A = \mathit{freepoint}() \\ C = \mathit{freepoint}() \\ ac = \mathit{line}(A,C) \\ bc = \mathit{perpendicular}(C,ac) \\ B = \mathit{point\_on\_line}(bc) \end{array} \right.
$$

# How to construct a conversion algorithm?

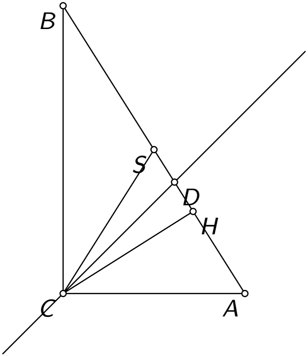
- Sometimes, an object is constrained by multiple relations.
- Example:

| | |
|---|---|
| $righttriangle(A, B, C)$ | $A = freepoint()$ |
| $angle(C, B, A) = 30^\circ$ | $C = freepoint()$ |
| | $ac = line(A, C)$ |
| | $bc = perpendicular(C, ac)$ |
| | $k_1 = circle(C, A, C)$ |
| | $k_2 = circle(A, A, C)$ |
| | $(M, M_1) = intersect\_circles(k_1, k_2)$ |
| | $ab = line(A, M)$ |
| | $B = intersect\_lines(ac, ab)$ |

# Proving geometry conjectures

- After visualizing the problem, it would be good if the pupil can prove if the main (or some intermediate) statement holds.
- The same languages used to describe the problem should be also used to specify and check the statements.
- All lemmas used must be made explicit.

# Example problem: A proof




### Proof

$\angle ACD = \angle DCB$ since $DC$ is a bisector of the angle $ACB$. $BS = SC$ since $S$ is a center of a described circle of the $\triangle ABC$. So, $\angle SCB = \angle SBC$. $\angle ACH = \angle SBC$ since they have perpendicular sides. Therefore, $\angle ACH = \angle SCB$. Finally, $\angle HCD = \angle DCS$.

# Example problem: A proof (2)

- The previous proof uses several statements and assumptions (without proving them).
- For example:
  1. Against equal sides of a triangle there are equal angles.
  2. The midpoint of the hypotenuse is the center of the circumcentre of a right triangle.
  3. Several implicit assumptions on the arrangement and order of points.

# Automated proving

- It would be good it the pupil can use help or an automated theorem prover — for the main or some intermediate statements.
- Along with a true/false answer, it would be good if the prover could present a classic, synthetic proof.
- There are several automated theorem provers for geometry and some are also freely available.
- Most of them are based on analytic geometry and do not produce classic proofs.

# Experiments

- Experiments performed with one class of first grade pupils (14 years old) in Architectural Technical High School.
- Each pupil had to use GCLC to make illustrations of several textbook problems.
- The next phase would be to apply built-in theorem provers.

# Experiments: Observations

- The precise GCLC syntax posed problems for pupils at early stages.
- Good pupils managed to overcome this and managed to produce illustrations.
- Some pupils had problems when conversion from declarative to procedural form required solving a non-trivial construction problems.
- Pupils who failed to produce illustrations, were required only to make precise declarative formulations.
- Some managed to do this (with some errors), but some didn't.

# Conclusions

- We have analyzed standard high-school geometry curriculum from a DGS perspective.
- Our experience show that some pupils experience problems even before they approach a DGS, because:
    - They don't understand the problem;
    - They can't reformulate the problem;
    - They don't understand the conjecture or what can make its proof.

# Conclusions (2)

- Our experience suggest that understanding formulations and reformulations of geometry problems is very important and should be addressed both within course materials and by software tools
- Our experience also suggest that the notion of proof should be made clear to pupils and support of automated theorem provers would be beneficial.