GeoLogic – Graphical Interactive Theorem Prover for Euclidean Geometry

Miroslav Olšák(✉)

University of Innsbruck, Innsbruck, Austria
mirek@olsak.net

Abstract. Domain of mathematical logic in computers is dominated by automated theorem provers (ATP) and interactive theorem provers (ITP). Both of these are hard to access by AI from the human-imitation approach: ATPs often use human-unfriendly logical foundations while ITPs are meant for formalizing existing proofs rather than problem solving. We aim to create a simple human-friendly logical system for mathematical problem solving. We picked the case study of Euclidean geometry as it can be easily visualized, has simple logic, and yet potentially offers many high-school problems of various difficulty levels. To make the environment user friendly, we abandoned strict logic required by ITPs, allowing to infer topological facts from pictures. We present our system for Euclidean geometry, together with a graphical application GeoLogic, similar to GeoGebra, which allows users to interactively study and prove properties about the geometrical setup.

Keywords: Euclidean geometry · Logical system

1 Overview

The article discusses GeoLogic 0.2 which can be downloaded from https://github.com/mirefek/geo_logic. It is a logic system for Euclidean geometry together with a graphical application capable of automatic visualization of basic facts (equal angles, equal distances, point being on a line, ...) and allowing user interaction with the logic system. GeoLogic can be used for proving many classical high school geometry problems such as Simson's line, Pascal's theorem, or some problems from International Mathematical Olympiad. Examples of such proofs are available in the package. In this paper, we first explain our motivation, then we describe the underlying logical system, and finally, we present an example of proving the Simson's line to demonstrate GeoLogic's proving and visualization capabilities.

There are many mathematical competitions testing mathematical problem solving capabilities of human beings, presumably most famous of which is the International Mathematical Olympiad (IMO). Writing an automated theorem prover (ATP) that could solve a large portion of IMO problems is a challenge

A. M. Bigatti et al. (Eds.): ICMS 2020, LNCS 12097, pp. 263–271, 2020.
https://doi.org/10.1007/978-3-030-52200-1_26

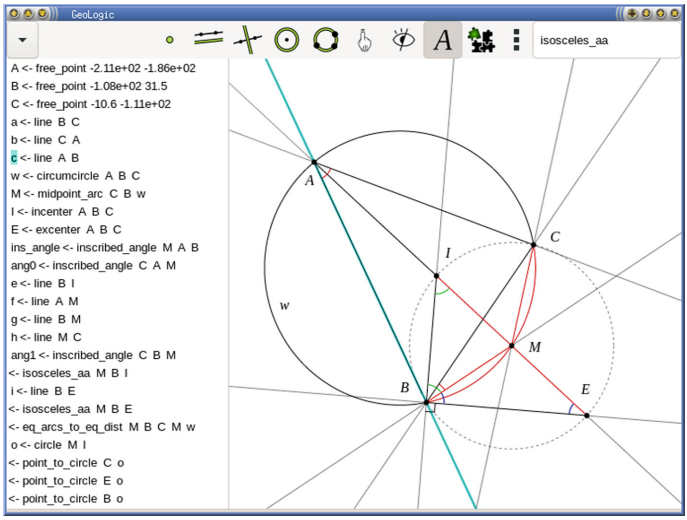


Fig. 1. GeoLogic screenshot

recognized in the field of artificial intelligence [6], and could potentially lead to strong ATPs in general.

IMO, as well as many regional mathematical olympiads divide problems into four categories: algebra, geometry, combinatorics, and number theory. From a human solver's perspective, computers can significantly help with solving geometry problems using an application such as GeoGebra – it allows the user to draw the configuration precisely, and observe how it changes when moving the initial points.

This is one of the reasons why we focused on geometry. Our objective is to capture the steps performed by such human solver in more detail, hoping it could eventually lead to better understanding of human thinking in general.

Therefore, we are building an interactive theorem prover, while preserving usability as an exploration tool. We have implemented a very simple logic, as it is sufficient for Euclidean geometry: most of the geometrical reasoning involves only direct proofs without higher-order logic or case analysis. While some geometrical proofs use case analysis for different topological configurations, we use a different approach. In GeoLogic, we allow inferring topological facts (such as the orientation of a triangle) from the picture (numerical model). This proves only one case of the problem (and its neighborhood), and could potentially lead to inconsistencies caused by numerical errors. However, we believe inconsistency caused by a numerical error is unlikely because we require the fact to be satisfied by a sufficient margin for postulating it.

In the future, we would like to experiment with machine learning agents leading to human-like ATPs for geometry. We would like to also experiment with computer vision components based on the GeoLogic's image output. Another

interesting research direction would be adding tools for case analysis, or proving topological facts, so that a solving process of a problem would consist first from finding a solution in the current GeoLogic's flexible logic, and then transforming it into a rigorous one. We believe that such an approach would be very close to the geometrical problem-solving procedure of human beings.

Finally, even though our main motivation was not to make a pedagogical tool, and we do not market GeoLogic as an application for an arbitrary high school student in its current form, we also believe that GeoLogic can be already interesting for talented students. Our objective of making a user-friendly interactive theorem prover for geometry is well-aligned with educational purposes, and if it will get adopted in the future, it can help us with obtaining data for machine learning experiments.

2 Logical System

The logical system of GeoLogic consists of a *logical core* interacting with *tools*. The logical core contains the following data.

- The set of all geometrical objects constructed so far. Every object can be accessed as a reference (for logical manipulation), or as the numerical object (e.g. coordinates of points, for numerical checking).
- The knowledge database. It consists of a disjoint-set data structure for equality checking, equation systems for ratios and angles, and a lookup table for tools.

The logical core also possesses basic automation techniques for angle and ratio calculations, and deductions around equality.

A *tool* is a general concept for construction steps, predicates, or inference rules. It takes a list of geometrical references on an input (and sometimes additional hyper-parameters), possibly adds some objects and some knowledge to the logical core and returns a list of geometrical references on the output, or fails. A tool always fails if the numerical data do not fit.

Besides that, every tool can be executed in a *check mode* or a *postulate mode*. A tool fails in the check mode (and not in the postulate mode) if it requires a fact which is not known by the knowledge database. Otherwise, the outcomes of the two modes are the same.

Most tools are memoized. When they are called, their input is associated with their output in the lookup table of the logical core. In the next call of the same tool on the same input, the tool does not fail (even in check mode) and returns the stored output (the same logical references). This serves three purposes: computation optimization, functional extensionality, and as a database for predicates. In particular, a primitive predicate `lies_on` is a memoized tool which in postulate mode only checks whether a given point is contained by a given line or circle. If it is not, it fails, otherwise, it returns an empty output. In check mode, however, this tool always fails. It means that the only way how to make this tool executable in the check mode is to have the input already stored

in the lookup table by calling it in the postulate mode before. This differs from topological (coexact) predicates such as `not_on` which in both modes only checks the numerical conditions – whether a given point is not contained by the given line or circle.

By proving a fact (any tool applied to given input) in the logic system, we mean executing certain tools in the check mode (proof), so that in the end the given fact can be also run in the check mode. The graphical interface allows users to run tools in check mode only.

2.1 Composite Tools

A composite tool is a sequence of other tool steps applied to the input objects. More precisely, a composite tool starts with just the input objects, runs several previously defined tools on the objects it has so far, and in the end, it returns some output objects selected from the available created objects. All composite tools are loaded from an external file, so we will explain them together with their format. An example code of the composite tool `angle` follows.

```
angle l0:L l1:L -> alpha:A
  d0 <- direction_of l0
  d1 <- direction_of l1
  alpha <- angle_compute 0 d0 -1 d1 1
```

The first line of a composite tool is a header specifying the tool name, input, and output objects, the other lines define the individual steps. The header line consists of the name, input objects, forward arrow `->`, and output objects separated by space. Every input or output object is given by its label before the colon and its type after the colon. Types are given by letters P (point), L (line), C (circle), A (angle), D (ratio/dimension). Note that the format allows name overloading as long as the input types are different, so there can be an `angle` tool accepting two lines, and also another `angle` tool accepting three points. The lines after header describe the tool steps by output objects, backward arrow `<-`, tool name, and input objects related to the subtool (possibly with numerical hyperparameters) separated by space. Now, we use only labels without types since the parser already knows the input types and it can infer the output types by the used tool. The output labels must be unique unless an anonymous label `_` is used. Among the input parameters, there can be also hyperparameters in the form of integers, floats, or fractions. It is not relevant how we mix the hyperparameters with the standard parameters but the order among hyperparameters, and among parameters matters.

The composite tool we described so far is the simplest composite tool (we call it a *macro*) which runs all its tool steps in the same mode as in what the macro is called. If any of the steps fail, the entire macro fails as well. Next to macros, there can be *axioms* and *lemmata*. The axiomatic tool is such a composite tool that contains a single line `THEN` among the steps. All the steps after `THEN` are then executed in postulate mode, even if the axiomatic tool is

called in a check mode. We call the steps before `THEN` *assumptions* and the steps after `THEN` *implications*. Axiomatic tools are used for wrapping up primitive constructions (see `direction_of`, and `line`), or formulating real axioms (see `isosceles_ss`).

```
direction_of l:L -> a:A
  THEN
  a <- prim__direction_of l

line A:P B:P -> p:L
  <- not_eq A B
  THEN
  p <- prim__line A B
  <- lies_on A p
  <- lies_on B p

isosceles_ss A:P B:P C:P ->
  <- not_eq B C
  <- eq_dist A B A C
  THEN
  <- eq_angle A B C B C A
```

Finally, a *lemma* is similar to the axiomatic tool with the exception that there is a third sequence of steps (called *proof*) following a `PROOF` line. When a lemma is executed in a check-mode, it works the same as an axiomatic tool, but it also calls a *proof check*. The proof check consists of the following steps:

1. opening a new logical core for the following steps,
2. adding the numerical values of input objects as the initial objects,
3. running the assumptions in postulate mode,
4. running the proof in check mode,
5. running the implications in check mode.

If all the tools succeed, the proof check is considered successful. In the following example of `isosceles_aa`, we have a lemma stating that if the angles β, γ in a triangle ABC are equal, so are the sides b, c. This is proven using an axiom `sim_aa_r` which takes two indirectly similar triangles CAB and BAC, checks that they are non-degenerated, and their angles are proven to be equal, and infers that the ratios of the sides of the two triangles are equal.

```
isosceles_aa A:P B:P C:P ->
  <- not_collinear A B C
  <- eq_angle A B C B C A
  THEN
  <- eq_dist A B A C
  PROOF
  <- sim_aa_r C A B B A C
```

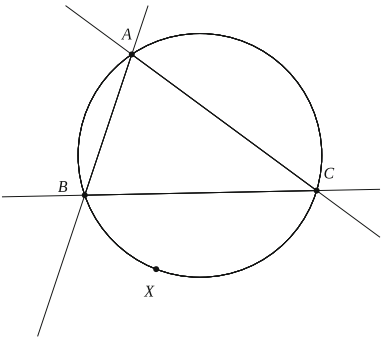
Adding a macro or a lemma to the toolset creates a conservative extension of the logic – anything that is provable with the usage of lemmata and macros can be proven without them.

3 Example – Simson's Line

We provide an example GeoLogic usage on the example of proving Simson's line. We used Geologic's graphical interface to define the following construction steps written as a code. During the construction, we also directly exported pictures from GeoLogic to show how GeoLogic visualizes known facts.

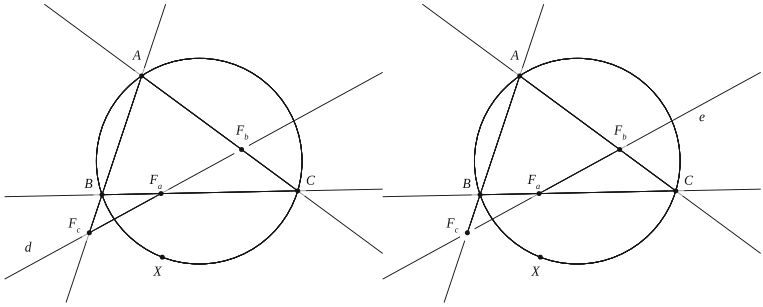
We start by drawing a triangle ABC, and a point X on its circumcircle.

```
A <- free_point -79.20758056640625 -119.095947265625
B <- free_point -126.97052001953125 23.91351318359375
C <- free_point 108.5352783203125 19.20867919921875
a <- line B C
b <- line C A
c <- line A B
o <- circumcircle A B C
X <- m_point_on 0.6169557687823527 o
```



Simson's line is a line passing through feet F_a, F_b, F_c of the point X to the sides of the triangle. However, GeoLogic is not aware (yet) of the fact that these three points are collinear.

```
Fa <- foot X a
Fb <- foot X b
Fc <- foot X c
d <- line Fc Fa
e <- line Fb Fa
```

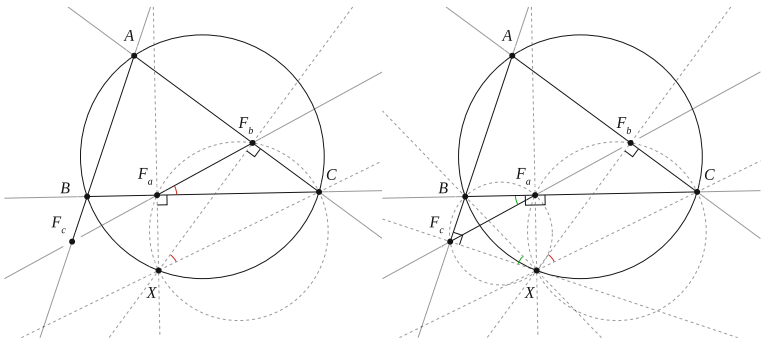


We can use the fact that the angles CF_aX and CF_bX are equal (they are both right angles) to conclude that points C, X, F_a, F_b are concyclic. We consequently use this fact to obtain that the angles F_bF_aC and F_bXC are equal.

```
<- angles_to_concyclic C X Fa Fb
<- concyclic_to_angles Fb C X Fa
```

We can similarly reason that the points B, X, F_a, F_c are concyclic and consequently the angles BF_aF_c and BXF_c are equal.

```
<- angles_to_concyclic B X Fc Fa
<- concyclic_to_angles Fc B Fa X
```



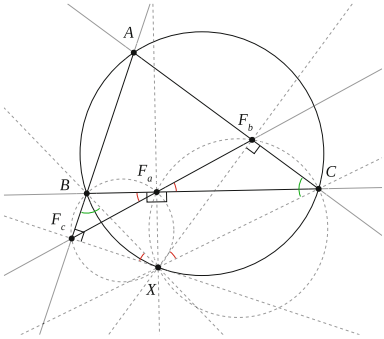
Finally, we use concyclicity of X, A, C, B to conclude that the angle XCA is equal to the complementary angle of ABX.

```
<- concyclic_to_angles X A C B
```

From this point on, GeoLogic's logical core realizes by itself that

$$\angle BF_aF_c = \angle BXF_c = 90° - F_cBX = 90° - F_bCX = CXF_b = CF_aF_b,$$

and since BF_aC are collinear, $F_cF_aF_b$ are collinear as well.



4 Related Work

Jeremy Avigad et al. [1] developed a logical system for formalizing elementary geometrical proofs from Euclid's elements, also distinguishing exact and coexact predicates. Their approach is more formal than ours allowing also proving the coexact statements in the end but it is less extensible by further tools. Michael Beeson et al. [2] connected the interactive theorem prover CoQ with GeoGebra for visualization of the theorem (but not for the proving procedure). Also, note that using a rigid logic system such as in CoQ does not allow numerical checks to be trusted in coexact statements.

The logical core of GeoLogic is partially inspired by General Deduction Database [3] and Full Angle [4] methods for automated synthetic proofs in Euclidean Geometry. These methods are supported by a graphical application Geometry Expert [7] which allows user to state a geometrical problem, run an automated geometrical theorem prover on it, and visualize the proof. Julien Narboux presented a similar graphical interface for construction of geometrical statement translated to CoQ [5]. None of these tools, however, supports constructing and checking proofs in the graphical interface.

5 Conclusion

We designed a semi-formal logic for Euclidean geometry which can be to a great extent controlled with a graphical interface and allows us to prove many standard high school problems. In the future, we would like to perform experiments with machine learning agents.

Acknowledgement. Supported by the ERC starting grant no.714034 SMART.

References

1. Avigad, J., Dean, E., Mumma, J.: A formal system for Euclid's elements. Rev. Symbolic Logic **2**(4), 700–768 (2009). https://doi.org/10.1017/S1755020309990098

2. Beeson, M., Boutry, P., Braun, G., Gries, C., Narboux, J.: GeoCoq (2018). (swh:1:dir:97ce53176b7d5e89d069bc60f49c3fa186831307). (hal-01912024)
3. Chou, S.-C., Gao, X.-S., Zhang, J.-Z.: A deductive database approach to automated geometry theorem proving and discovering. J. Autom. Reasoning **25**, 219–246 (2000). https://doi.org/10.1023/A:1006171315513
4. Chou, S., Gao, X., Zhang, J.: Automated generation of readable proofs with geometric invariants. J Autom. Reasoning **17**, 349–370 (1996). https://doi.org/10.1007/BF00283134
5. Narboux, J.: A graphical user interface for formal proofs in geometry. J. Autom. Reasoning **39**(2), 161–180 (2007). https://doi.org/10.1007/s10817-007-9071-4
6. Selsam, D.: IMO Grand Challenge. https://imo-grand-challenge.github.io/
7. Ye, Z., Chou, S.-C., Gao, X.-S.: An introduction to Java geometry expert. In: Sturm, T., Zengler, C. (eds.) ADG 2008. LNCS (LNAI), vol. 6301, pp. 189–195. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21046-4_10