GeoLogic – Graphical interactive theorem prover for Euclidean geometry

Miroslav Olšák[1] [0000−0002−9361−1921]

University of 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-immitation 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. In the future, we would like to perform experiments with machine learning agents.

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 vizualisation 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 give a description of the underlying logical system, and finally we present and example of proving the Simson's line to demonstrate GeoLogic's proving and visualisation capabilities.

2 Motivation

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

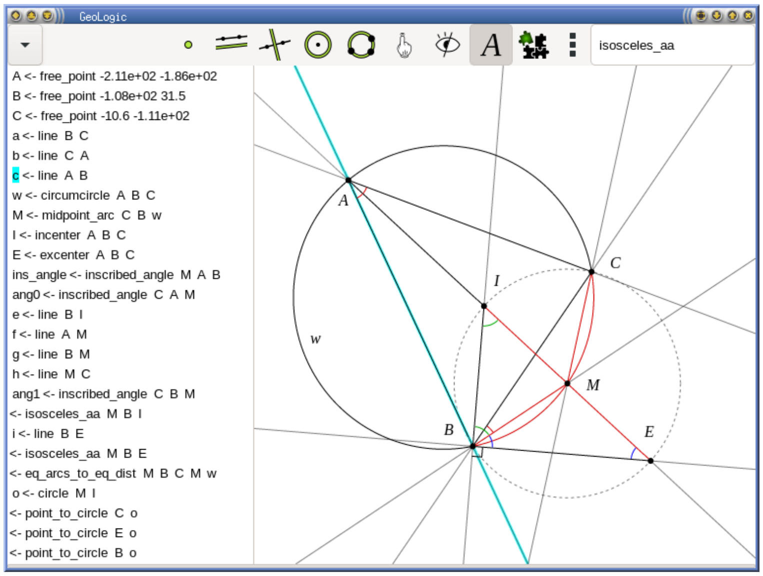


Fig. 1. GeoLogic screenshot

(ATP) that could solve a large portion of IMO problems is a challenge recognized in the field of artificial intelligence [5], and could potentionally 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, computer 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 the usability as an exploration tool. In future, we would like to experiment with machine learning agents leading to human-like ATPs for geometry. Moreover, geometry is concrete and visualisable. This allows to add computer vision components to the future machine learning experiments.

Other advantage of reasoning in Euclidean geometry is that we don't need complex logic. Most of 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. We allow to infer topological facts (such as orientation of a triangle) from the picture (numerical model). This apparently proves only one case of the problem (and its neighborhood), and could potentionally 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, Softening the logic so that it accepts a proof of just one configuration is actually an advantage: It is a common case in Euclidean geometry that a proof of a single configuration can be used for proving the general case either by case analysis and analogies, or by proving that the configuration is the only possible. Therefore, introducing a flexible logic can be seen as providing an intermediate step for finding a formal proof – one first want to proof it in the flexible logic of GeoLogic, and then to transform the GeoLogic's proof into a formal one. Both of the subtasks are typically easier then the original problem, and such proving procedure would reflect how a human solver usually approach the problem.

Finally, even though our main motivation was not to make a pedagogical tool, and we don't 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 an 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.

3 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 possess 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 in 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 fail. 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. Ths differs from topological (coexact) predicates such as `not_on` which in both modes only checks the numerical conditions – whether a given point in 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 user to run tools in check mode only.

3.1 Composite tools

A composite tool is basically a sequence of other tool steps applied to the input objects, or on the outputs of prior tools in the sequence. 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 consisting of name, input objects, forward arrow `->`, and output objects separated by space. Every input or output object is given by its label before colon, and its type after 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. Following lines describe the tool steps by output objects, backward arrow `<-`, tool name and input objects (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 a *macro* which runs all its tool steps in the same mode as in what the macro is called. If any of the steps fails, the entire composite tool fails as well. Next to macros, there can be *axioms* and *lemmata*. 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*. Proof check constists 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.

```
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 tool set creates a conservative extension of the logic – anything that is provable with the usage of lemmata and macros can be proven without them.

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 visualisation 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.

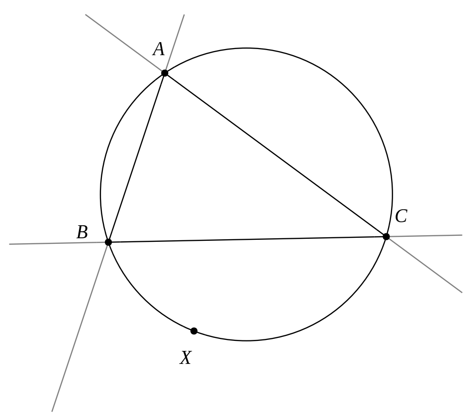
The logical core of GeoLogic is partially inspired by General Deduction Database [3] and Full Angle [4] methods for authomated syntetic proofs in Euclidean Geometry. These methods are supported by a graphical application Geometry Expert [6] which allows user to state a geometrical problem, run an authomated geometrical theorem prover on it, and visualise the proof.

5 Example – Simson's line

We provide an example GeoLogic usage on the example of proving Simson's line. All the Figures in this section are exported from GeoLogic, demonstrating its visualisation of known facts. The code below representing the constructions and reasoning steps was created inside GeoLogic's graphical interface.

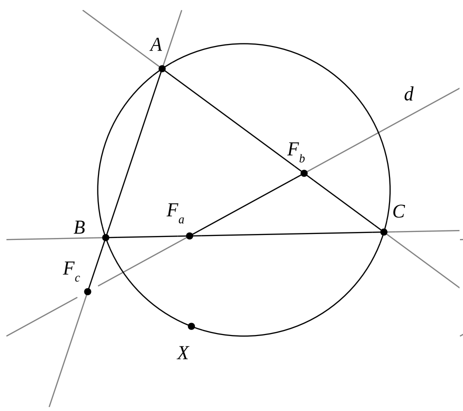
We start by drawing a triangle ABC, and a point X on its circumcircle.

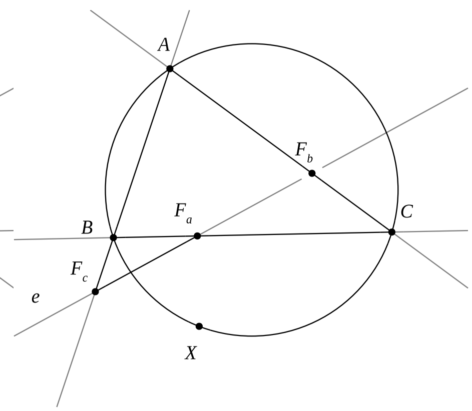
```
A <- free_point -79.20758056640625 -119.095947265625
B <- free_point -126.97052001953125 23.91351318359375
C <- free_point 108.5352783203125 19.20867919921875
a <- line B C
b <- line C A
c <- line A B
o <- circumcircle A B C
X <- m_point_on 0.6169557687823527 o
```



Simson's line is a line passing through foots 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
```



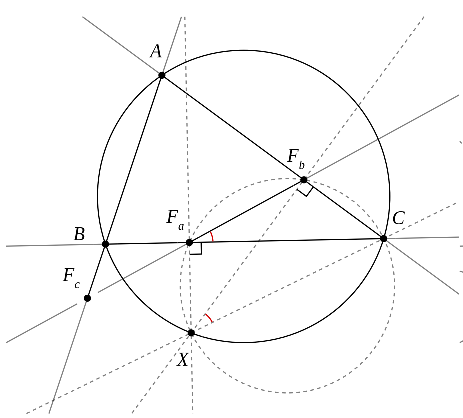


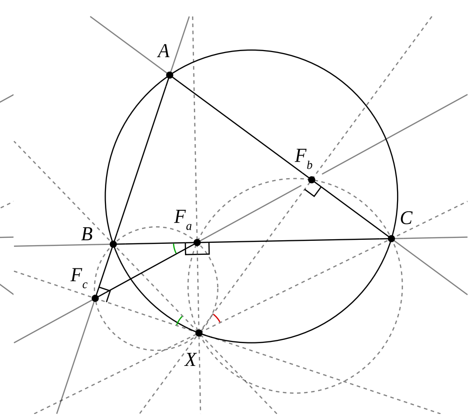
We can use the fact that the angles CF_aX and CF_bX are equal (they are both right angles) to conclude that points C, X, F_a, F_b are concyclic. We consequently use this fact to obtain that the angles F_bF_aC and F_bXC are equal.

```
<- 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
```





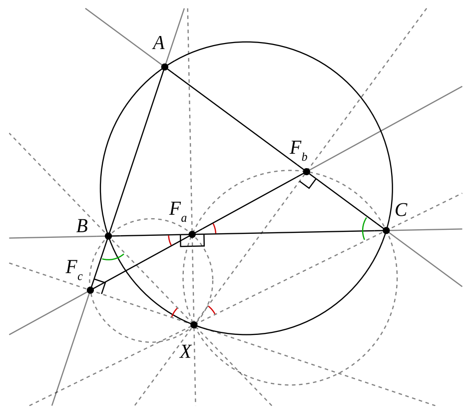
Finally, we use concyclicity of X, A, C, B to conclude that the angle XCA is equal to the complementary angle of ABX.

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

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

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

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



References

1. Avigad, J., Dean, E., & Mumma, J.: A Formal System for Euclid's Elements. The Review of 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, Shang-Ching & Gao, Xiao-Shan & Zhang, Jing-Zhong: 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. Selsam, D.: IMO Grand Challenge `https://imo-grand-challenge.github.io/`
6. Ye Z., Chou SC., Gao XS.: An Introduction to Java Geometry Expert. Sturm T., Zengler C. (eds) Automated Deduction in Geometry. ADG 2008. Lecture Notes in Computer Science, vol 6301. Springer, Berlin, Heidelberg.