GeoLogic – Graphical interactive theorem prover for Euclidean geometry

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

University of Innsbruct, 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 Logical system

GeoLogic uses five geometrical types: point, line, circle, angle and ratio. Note that an "angle" can also represent a direction of a line, and it is considered modulo 180° (the inner unit of an angle is 180°, so it is actually a number between 0 and 1). A "ratio" is a non-zero real number that can represent distances between points, or any products, or powers of such "ratios".

The logical system of GeoLogic consists of a *logical model* interacting with *tools*.

The logical model contains the following data.

- The set of all 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. Particular parts will be explained in Subsection 1.1.

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, they will be discussed later), possibly adds some objects

and some knowledge to the logical core and returns a list of geometrical references on the output, or fails. Besides that, every tool can be executed in a *check mode*, or in a *postulate mode*. A tool always fails if the numerical data do not fit, and it also fails in the check mode, if it requires a fact which is not known by the knowledge database. The sizes and types of the input objects are fixed (predefined) for most tools, the the size and types of the output objects is fixed for every tool.

1.1 Primitive tools

We first discuss the tools directly accessing the knowledge database, in particular:

- Primitive construction
- Equality tools
- Calculating tools
- Strict and non-strict predicates

A tool of a primitive construction is directly linked to a creation of a new geometrical object. An example of such a tool is the tool `prim__line` which takes two distinct points, creates a new line in the logical core passing through these points and returns its reference. Note that this sort of tools always fails in the check-mode, and it does not add any facts to the knowledge database. It is because tools of primitive constructions are always meant to be wrapped in axiomatic composite tools (see Subsection 1.2).

Equality tool is a single tool expecting two objects of the same type, and returning the empty list, or failing. It always fails if the two objects are not numerically equal. In the postulate mode, it forces the two references to be considered as equal. In check mode, it fails if the two references are not considered equal in the knowledge database.

There are two geometrical types that can be used for calculation: angles and ratios. For each of them, there is one tool for constructions of a new object, and one for checking. The terms for angles and ratios are of the following forms respectively

$$\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n, \quad x_1^{\alpha_1} \cdot x_2{}_2^{\alpha} \cdot \cdots \cdot x_n^{\alpha_n},$$

where $x_1, \ldots, x_n$ represent the input objects and $\alpha_1, \ldots, \alpha_n$ integer hyper-parameters. The construction tools `angle_compute` and `ratio_compute` in both modes calculate these terms, add a fractional constant (given as the first hyper-parameter) to them, create corresponding object, add the corresponding equation to the knowledge base and output the reference to the new object. The checking tools verify whether the equation is equal to a given fractional constant (passed as the first hyper-parameter), and returns an empty list. It fails if the equation does not fit numerically, and in the check mode it also fails if the equation is not known by the knowledge database.

The last sort of tools are primitive predicates. They can either *exact* (strict), or *coexact* (topological), the names exact and coexact are taken from [**?**]. An example of an exact predicate is `lies_on` checking whether a point lies on a line (or a circle), and an example of a primitive coexact predicate is `not_on`, checking whether a point is not on a line (or a circle). The convention is that coexact predicates can be checking for only such conditions which are topologically open meaning that a little change in the input does not cause the condition to fail while exact predicate check for other conditions. The output of a predicate is always an empty list (or failure).

The predicate tools belong to *memoized tools*, other memoized tools are composite tools. The first action in any call of a memoized tool is looking to the lookup table, whether the current call is already memoized. If so, it succeeds (in any mode, the lookup table does not distinguish modes). For coexact predicates, this memoization serves only as an optimization. The coexact predicate works in both modes (postulate or check) the same, it checks whether the numerical condition is satisfied and succeeds if so, storing the call into the lookup table. On the other hand, the memoization is a crucial element of the logic system for the exact predicates: In postulate mode, exact predicates work the same way as coexact predicates, but in check mode, exact predicates always fail unless the call is already stored in the lookup table. In other words, the only way how to make an exact predicate to succeed in check mode is to first call the same exact predicate on the same input (the same object references) in postulate mode.

1.2 Composite tools

A composite tool is basically a sequence of other tool steps applied to the input objects, or on the outputs. All composite tools are loaded from an external file, so we will explain them together with their format. And example of a code for the composite tool `angle` can be seen in Figure 1.2.

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

Fig. 1. Example of a basic macro

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 color, and its type after colon. Types given by letters P (point), L (line), C (circle), A (angle), D (ratio, or 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 following describe the tool steps by output

objects, backward arrow `<-`, tool name and input objects separated by space. Now, we use only labels 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 a dummy 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 a *axioms* and *lemmata*. Axiomatic tool is such a 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` in Figure 1.2), or formulating real axioms (see `isosceles_ss` in Figure **??**).

```
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
```

Fig. 2. Examples of axiomatic tools

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. putting 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 implications succeed, the proof check is 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
```

Fig. 3. Examples of a lemma

1.3 Automation

After any modification of the logical core, the core process the newly gained knowledge and makes certain automatic deductions. There are four automation techniques involved:

- gaussian elimination for ratios,
- gaussian elimination for angles,
- basic equality handling,
- additional triggers.

Any equation that GeoLogic can handle uses only multiplication and division. Such equations can be seen as linear equations about the logarithms of the ratios. The logical core keeps the postulated equations in a normalized form, so that it can be easily checked whether any other equation is implied by the given equations. A similar mechanism is involved for angles. However, angles are taken modulo 1, so the logical core only looks whether the checked angle equation can be derived from the postulated equations modulo some rational number, and the rest is left to comparison of the numerical values.

Equality on general objects is handled using a disjoint-set data structure, so its reflexivity, symmetry, or transitivity is guaranteed by design. Whenever equality occurs, the lookup table for tools registers it, and automatically applies fuction extensionality: if the knowledge database knows $x_1 = x'_1, \ldots, x_n = x'_n$, and there is a tool t such that t applied to $x_1, \ldots, x_n$ returned $y_1, \ldots, y_m$ and t applied to $x'_1, \ldots, x'_n$ returned $y'_1, \ldots, y'_m$, then it deduces that $y_1 = y'_1, \ldots, y_m = y'_m$. This extensionality automation is interconnected with the gaussian elimination – whenever extensionality discovers equality between two angles or ratios, it passes the equality as a new equation to the gaussian elimination systems, and vice versa.

Finally, a few extra specific automatic "triggers" are also applied. These usualy correspond to specific equality axioms about geometrical objects which are not direct consequences of extensionality. An example of such an axiom can be seen in Figure 1.3.

```
<- not_eq A B
<- lies_on A l
<- lies_on B l
<- lies_on A l'
<- lies_on B l'
THEN
<- == l l'
```

Example of an axiom which is automatically applied.

1.4 Use in the graphical application

Extra to the mentioned tools, we add *movable* tools which are analogous to already wrapped primitive constructions but with possible extra hyperparameters. Movable tools are used for object that are not fully determined by their parameters such as a free point in the plane, any line passing through a point, or even an ambiguous intersection.

In the user interface, the user applies tools in check mode to previously constructed objects. The user has only access to the top-level objects – the objects that came as a result of a tool he used. Other objects which were used as only temporary objects in the applied tools are still present in the logical core but not available to te user.