Building a Theorem Prover for Euclidean Geometry

David Toh Hui Kai, Christ's College

Project Supervisor: Prof. Lawrence Paulson

Introduction

Automated theorem-proving is a method of developing formal proofs of mathematical statements using mechanised proof assistants. Given a problem, a theorem prover may be deployed to automatically search for a proof, or solution, to the problem, via some inference procedure.

Euclidean geometry lends itself well to inferential reasoning, given the rigid and largely rule-based nature of the subject. **Deductive database (DD) provers** are a class of provers which repeatedly invoke fundamental geometric rules and theorems, building up a database of intermediate results which can be reasoned to be true, and stopping when the problem statement is eventually proven. Existing work has modelled the inferential reasoning as a planning problem (Gelernter, 1995), as well as the process of establishing the fixed-point closure of a set of geometric predicates (Chou, et al., 2000).

Considerable headway has also been reported by **algebraic reasoning (AR)** solvers, which convert the geometric problem to an algebraic problem in a Cartesian, barycentric, or some other coordinate system, before performing algebraic reduction. A hybrid class of solvers using **semi-geometric (SG)** methods have also seen success, and focus on algebraic manipulation of explicitly geometric quantities, such as (Chou, et al., 1994)'s area method and (Wilson & Fleuriot, 2005)'s implementation of the full-angle method.

DD provers produce human-readable and logically comprehensible proofs, and are thus most useful from a problem-solving perspective. We discuss the issues faced by DD provers and address them using various approaches existent in literature – via the use of custom data structures, search algorithms, heuristics, and selective invocation of AR/SG methods.

Representation and Redundancy

Geometric statements often assume the form of equivalence classes capable of holding an arbitrary number of elements. For example, a typical geometry problem might have an arbitrarily number of points on a circle ω centered at O, expressible using the equivalence class

$$OnCircle(\omega, O) = \{P_1, P_2, P_3, \dots\}$$

Thus, design choices need to be taken when implementing both our logical grammar and the deduction ruleset, to avoid introducing redundancy. Many different types of redundancy exist: for

instance, in the above example, it would be highly superfluous to store all pairs of segments (OP_i, OP_j) of equal length, which would use $O(n^2)$ space. Another major source of redundancy comes from introducing multiple ways to construct the same object.

(Chou, et al., 2000) employed lexicographical ordering of variables, and equivalence classes with representative elements for some predicates. They additionally described hard-coded measures to reduce redundant deductions, for example by skipping the storage of facts which do not provide any new information (such as $OA = OB \Rightarrow OAB \cong OBA$).

(Matsuda & Lehn, 2004) used semantic networks to relate geometric elements to one another. They also grouped postulates sharing the same sub-configuration of points and objects into "diagrammatic schemas", which consisted of a semantic network, as well as a list of postulates on the sub-configuration of elements.

(Trinh, et al., 2024) used a graph database structure to represent equivalence classes, with edges (or multiedges) between nodes to facilitate later deduction traceback.

Algebraic Manipulation

Proofs often require some degree of length or angle chasing, involving the four basic operations, which are best handled extra-logically:

$$\frac{AX}{XC} = \left(\frac{AB}{BC}\right)^2 \cdot \left(\frac{AM}{MB}\right) \quad \text{or} \quad \angle(PQ, QR) = \angle(PQ, QS) + 2 \cdot \angle(ST, TV) - \frac{\pi}{2}$$

(Olšák, 2020) described an interactive proof verifier, GeoLogic, that has no inference automation, instead only automating algebraic manipulation. GeoLogic first describes the use of Gaussian elimination to manipulate both angles and log-distances. It also simply performs numeric checks to validate edge cases.

(Trinh, et al., 2024) combined their DD engine with an AR system also utilising Gaussian elimination.

Construction Search Space

The most significant challenge to geometric proof generation is that of auxiliary constructions - adding additional points, lines, circles, triangles (e.g. via rotation and translation) etc. to the diagram. The vast majority of problems with appreciable difficulty can only be solved in this manner.

A set of construction rules is thus needed to limit the space of possible constructions. Additionally, we require a set of construction heuristics to determine which to keep and which to discard.

(Chou, et al., 2000) first described the construction of new points as the Skolemisation of existential quantifiers, and presented a set of hard-coded construction rules. Additionally, they described two construction heuristics: checking if new properties about the original diagram are deduced, and limiting the depth of constructions to one layer.

(Matsuda & Lehn, 2004) described a non-hard-coded rule for introducing new geometric constructions: so that rules which were previously not unifiable may now be unifiable. (For example, if we want to use Pascal's theorem and we only have 2 out of 3 pairs of intersections, we might introduce the third one.)

(Stojanović, et al., 2010) attempted to implement geometric proof-search using coherent logic, with a few heuristics to control the search space, such as early pruning of goals when selecting rules to apply. Their implementation was named ArgoCLP.

(Bak, 2020) automated the generation of geometry problems via the selective addition of constructions, and has a complicated set of heuristics for choosing constructions and avoiding redundancy.

Project Proposal

This project aims to design and implement a theorem prover specific to Euclidean geometry, integrating the different standalone techniques and optimisations variously used by past literature as described in the previous section. The theorem prover will integrate elements of both DD and AR as described in (Trinh, et al., 2024). Here is an outline of how the theorem prover will work:

1. AR: Represent all relevant angle and length quantities as linear combinations of other angles and lengths in a matrix.
2. DD: From its existing database, as well as the AR's equations, iterate through the available deduction rules to generate new deductions.
3. Return success if the statement we are proving is found.
4. Repeat steps 1 to 3 until we have reached the closure of both the DD and the AR matrix (i.e. both stop changing).
5. Either add a construction to the diagram and repeat from step 1 (extension), or fail.

The theorem prover will be developed in C++ to provide a comparable level of performance assessment against existing implementations.

Success Criteria

The base requirements for the project's success are as follows:

- A logical grammar capable of representing most geometry problems unambiguously, together with a parser and lexer;
- A set of propositional rules, heavily similar to those specified in (Chou, et al., 2000), capable of describing the following elementary geometric axioms: parallel and perpendicular relations, basic angle relations (e.g. corresponding angles, alternate angles etc.), congruent and similar triangles, midpoints, cyclicity and circumcenter;
- A data structure specification that addresses some or all of the representability and redundancy issues outlined above. This is likely to take the form of a graph data structure, heavily similar to (Trinh, et al., 2024), using the ordering and equivalence techniques outlined in (Chou, et al., 2000) and (Bak, 2020);
- An algebraic manipulation engine for determining the closure of algebraic relations involving angles and log-distances, using a Gaussian elimination approach as implemented in (Olšák, 2020) and (Trinh, et al., 2024);
- A deductive engine for determining the closure of geometric relations. The deduction process must address some or all of the redundancy issues outlined above, and will likely do so employing the techniques in (Chou, et al., 2000) and (Stojanović, et al., 2010).

The majority of these criteria involve implementing and integrating, with minor tweaks, work done in existing literature.

Extensions

The highest-priority extension of the project will involve adding the capability of performing constructions to the solver.

Construction rules may consist of one or more of the following:

- The hard-coded construction rules described in (Chou, et al., 2000).
- The unification-based construction rule described in (Matsuda & Lehn, 2004).

We will experiment with different construction heuristics, including the ones outlined in (Bak, 2020). In particular, I believe it is worth investigating different construction depth limits.

Other extensions include:

- Encoding additional rules powerful enough to describe more complex theorems, such as Pascal's theorem, the "Chicken Feet Lemma", or the concept of isogonal conjugates;
- Providing an explicit traceback algorithm to determine the minimal set of immediate ancestor statements required for a proof, in line with the implementation described in (Trinh, et al., 2024);
- Storing previously solved problems as rules that can then be themselves invoked.
- Extending the available memory of the DD and AR databases by using a DBMS capable of storing intermediate computations on disk, and modifying the data structure specification accordingly, similar to that of (Baeta & Quaresma, 2023).

Evaluation

The success of our implementation may be evaluated by variously testing the theorem prover on some of the problems from the following problem sets, both throughout the development process as well as at the end of the project:

- The 500 elementary problems contained in (Chou, et al., 1994);
- The 110 intermediate problems contained in (Chou, et al., n.d.);
- The 100 Olympiad-level problems contained in this problem set, as well as other open-source competition geometry problem sets;
- The geometry problems of the IMO Shortlist which are encodable by our logical grammar.

As there is no standardised logical grammar in literature, the work required to manually translate these problems into our solver is unavoidable.

The performance of the theorem prover will be evaluated on the following four heuristics:

- (Core heuristic) Correctness of problem encoding
- (Core heuristic) Solve rate
- (Core heuristic) Memory usage
- (Extension heuristic) Runtime, with various other open-source implementations serving as benchmarks

Timetable and Resources

This project will primarily be done on my personal laptop (Acer Swift SFA16-41, 32GB RAM, AMD Ryzen 7 Pro 6850U, integrated graphics card, 1TB storage). Development will be done in Ubuntu 22.04 and minimally require the use of clang, CMake, and git for version management.

This project may require some usage of HPC resources, or alternatively cloud computing resources, in the later stages of runtime evaluation on more complex problems (because my laptop has a very weak cooling system).

The proposed timetable is as described below:

1	Michaelmas	Oct 13 – Oct 26	Locally install and investigate the implementations described in (Chou, et al., 2000), (Matsuda & Lehn, 2004), (Olšák, 2020), (Bak, 2020), and (Trinh, et al., 2024)
2		Oct 27 – Nov 9	Implement a simple logical grammar and rule list Design the data structure specification Translate around 20 problems for runtime evaluation Scaffold the deductive engine
3		Nov 10 – Nov 23	Finish designing the data structure specification Finish building the simple deductive engine Write unit tests for the deductive engine
4		Nov 24 – Dec 7	Scaffold the algebraic manipulation engine (and find a way to implement Gaussian elimination in C++ stably) Modify the data specification as necessary to accommodate the algebraic engine
5	Christmas Break	Dec 8 – Dec 21	Build the algebraic manipulation engine Write unit tests for the algebraic manipulation engine Begin integrating both engines to allow for the inference of their joint closure
6		Dec 22 – Jan 11	Continue integrating both engines, making modifications where necessary to optimise for redundancy issues and runtime Write unit tests for the overall proof assistant Translate around 50 elementary problems and 20 intermediate problems for runtime evaluation
7		Jan 12 – Jan 25	Buffer time for engine integration, optimisation, and unit testing Progress Report
8	Lent	Jan 26 – Feb 8	Write implementations of both the unification-based and hard-coded construction rules, modifying the data structure specification as necessary Write implementations of the simple construction heuristics outlined in (Chou, et al., 2000)
9		Feb 9 – Feb 22	Finish implementing and optimising construction engine Experiment with optimising construction rules and heuristics Write unit tests for construction engine

			Translate around 20 intermediate problems for runtime evaluation
10		Feb 23 – Mar 8	Continue optimising construction rules and heuristics Translate more problems for runtime evaluation Time for other extensions
11	Easter Break	Mar 9 – Mar 22	Translate more problems for runtime evaluation Time for other extensions Write dissertation
12		Mar 23 – Apr 5	Time for other extensions Write dissertation
13		Apr 6 – Apr 19	Write dissertation
14	Easter	Apr 20 – May 3	Write dissertation

Bibliography

Baeta, N. & Quaresma, P., 2023. Towards a geometry deductive database prover. *Annals of Mathematics and Artificial Intelligence 91,* pp. 851-863.

Bak, P., 2020. *Automated Generation of Planar Geometry Olympiad Problems,* s.l.: s.n.

Botana, F., Kovács, Z. & Recio, T., 2021. A Mechanical Geometer. *Mathematics in Computer Science,* Volume 15, pp. 631-641.

Chou, S.-C., Gao, X.-S. & Zhang, J.-Z., 1994. *Machine Proofs in Geometry: Automated Production of Readable Proofs for Geometry Theorems.* s.l.:World Scientific.

Chou, S.-C., Gao, X.-S. & Zhang, J.-Z., 1996. Automated Generation of Readable Proofs with Geometric Invariants II. Theorem-Proving with Full Angles. *Journal of Automated Reasoning,* Volume 17, pp. 349-370.

Chou, S.-C., Gao, X.-S. & Zhang, J.-Z., 2000. A Deductive Database Approach to Automated Geometry Theorem Proving and Discovering. *Journal of Automated Reasoning 25(3),* pp. 219-246.

Chou, S.-C., Gao, X.-S. & Zhang, J.-Z., n.d. *A Collection of 110 Geometry Theorems and Their Machine Produced Proofs Using Full-Angles.* s.l., s.n.

Gelernter, H., 1995. Realization of a Geometry Theorem-proving Machine. *Computers and Thought,* pp. 134-152.

Janicic, P., Narboux, J. & Quaresma, P., 2012. The Area Method: A Recapitulation. *Journal of Automated Reasoning.*

Kovács, Z., Recio, T. & Weitzhofer, S., 2012. *Implementing Theorem Proving in GeoGebra by Exact Check of a Statement in a Bounded Number of Test Cases.* s.l., s.n.

Matsuda, N. & Lehn, K. V., 2004. GRAMY: A Geometry Theorem Prover Capable of Construction. *Journal of Automated Reasoning 32,* pp. 3-33.

Olšák, M., 2020. *GeoLogic -- Graphical interactive theorem prover for Euclidean geometry.* s.l., s.n., pp. 263-271.

Stojanović, S., Pavlović, V. & Janičić, P., 2010. *A Coherent Logic Based Geometry Theorem Prover Capable of Producing Formal and Readable Proofs.* s.l., s.n., pp. 201-220.

Trinh, T. H. et al., 2024. Solving olympiad geometry without human demonstrations. *Nature,* 625(7995), pp. 476-482.

Wilson, S. & Fleuriot, J. D., 2005. *Combining Dynamic Geometry, Automated Geometry Theorem Proving and Diagrammatic Proofs.* s.l., s.n.

Winkler, F., 1990. Gröbner bases in geometry theorem proving and simplest degeneracy conditions. *Mathematica Pannonica 1.1,* pp. 15-32.

Wu, W.-T., 1978. On the Decision Problem and the Mechanisation of Theorem-Proving in Elementary Geometry. *Scientia Sinica 21(2).*