








Article

# FGeo-SSS: A Search-Based Symbolic Solver for Human-like Automated Geometric Reasoning

Xiaokai Zhang <sup>1</sup>, Na Zhu <sup>1,2</sup>, Yiming He <sup>1,2</sup>, Jia Zou <sup>1,2</sup>, Cheng Qin <sup>1,2</sup>, Yang Li <sup>1</sup> and Tuo Leng <sup>1,2,\*</sup>

<sup>1</sup> School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; xiaokaizhang@shu.edu.cn (X.Z.)

<sup>2</sup> Institute of Artificial Intelligence, Shanghai University, Shanghai 200444, China

\* Correspondence: tleng@shu.edu.cn

**Abstract:** Geometric problem solving (GPS) has always been a long-standing challenge in the fields of automated reasoning. Its problem representation and solution process embody rich symmetry. This paper is the second in a series of our works. Based on the Geometry Formalization Theory and the FormalGeo geometric formal system, we have developed the Formal Geometric Problem Solver (FGPS) in Python 3.10, which can serve as an interactive assistant or as an automated problem solver. FGPS is capable of executing geometric predicate logic and performing relational reasoning and algebraic computation, ultimately achieving readable, traceable, and verifiable automated solutions for geometric problems. We observed that symmetry phenomena exist at various levels within FGPS and utilized these symmetries to further refine the system's design. FGPS employs symbols to represent geometric shapes and transforms various geometric patterns into a set of symbolic operation rules. This maintains symmetry in basic transformations, shape constructions, and the application of theorems. Moreover, we also have annotated the formalgeo7k dataset, which contains 6981 geometry problems with detailed formal language descriptions and solutions. Experiments on formalgeo7k validate the correctness and utility of the FGPS. The forward search method with random strategy achieved a 39.71% problem-solving success rate.

**Keywords:** symmetry in geometric problem; formal mathematics; geometric problem solving; geometric symbolic solver



check for updates

**Citation:** Zhang, X.; Zhu, N.; He, Y.; Zou, J.; Qin, C.; Li, Y.; Leng, T.

FGeo-SSS: A Search-Based Symbolic Solver for Human-like Automated Geometric Reasoning. *Symmetry* **2024**, *16*, 404. <https://doi.org/10.3390/sym16040404>

Academic Editor: Alexei Kanel-Belov

Received: 10 March 2024

Revised: 24 March 2024

Accepted: 25 March 2024

Published: 30 March 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Symmetry refers to the property of an object remaining invariant under a certain transformation, which is widely present in various aspects of the real world, such as the periodic repetition of musical notes [1], the time/space translational symmetry of physical properties [2], and the bilateral symmetry of organisms [3]. This article designs a formal symbolic solver for the formal representation and automated solution of planar geometric problems and explores the symmetry involved in various aspects of geometric problem solving (GPS). GPS has always been a long-standing challenge [4–6] in the fields of mathematical reasoning and Artificial Intelligence (AI), owing to the cross-modal forms of knowledge and the absence of automated solving methods. As depicted in Figure 1, a geometry problem typically consists of geometric texts and images.

Based on the Geometry Formalization Theory and the FormalGeo geometric formal system [7], we have developed the Formal Geometric Problem Solver (FGPS) in Python. We elucidate the symmetry phenomena present in FGPS and how these symmetries can be utilized to improve solver design:

- Geometric shapes consist of several points, and when performing basic transformations such as rotation and scaling on the shapes, the relative positional information among the points does not change; that is, the shape's topological structure information is symmetrical with respect to basic transformations. A complex geometric shape

is often composed of several basic shapes, and the order of combination of the basic shapes does not change the final complex shape constructed; that is, the construction result is symmetrical relative to the construction order. We have implemented the above formal representation methods and construction methods, transforming the symmetry and asymmetry into a series of rules for symbolic operations. The implementation of these symmetries helps to ensure the comprehensiveness and correctness of the FGPS design.

- The process of GPS can be seen as the application of theorems, where each application derives new conditions, ultimately leading to the derivation of the solution objective. The application order of geometric theorems can form a directed acyclic graph (DAG), and any theorem sequence obtained through its topological sorting can solve the current geometric problem. The solvability of geometric problems with respect to the topological sorting of the theorem’s DAG is symmetrical. By leveraging this symmetry, we expand the number of problems according to the process of GPS, effectively addressing issues such as high labeling costs and scarcity of datasets.
- FGPS incorporates two symmetric problem-solving algorithms: forward search, which starts from known conditions and continuously applies theorems to reach the problem’s goal, and backward search, which starts from the problem’s goal and expands it into sub-goals until all sub-goals are known conditions. We organize the process of GPS into a hypertree with conditions as hypernodes and theorems as hyperedges. The hypertrees obtained from the two algorithms are mirror-symmetrical. We designed the backward search algorithm by analyzing the symmetrical processes of forward search, and the solution generation process of backward search is also based on forward search.

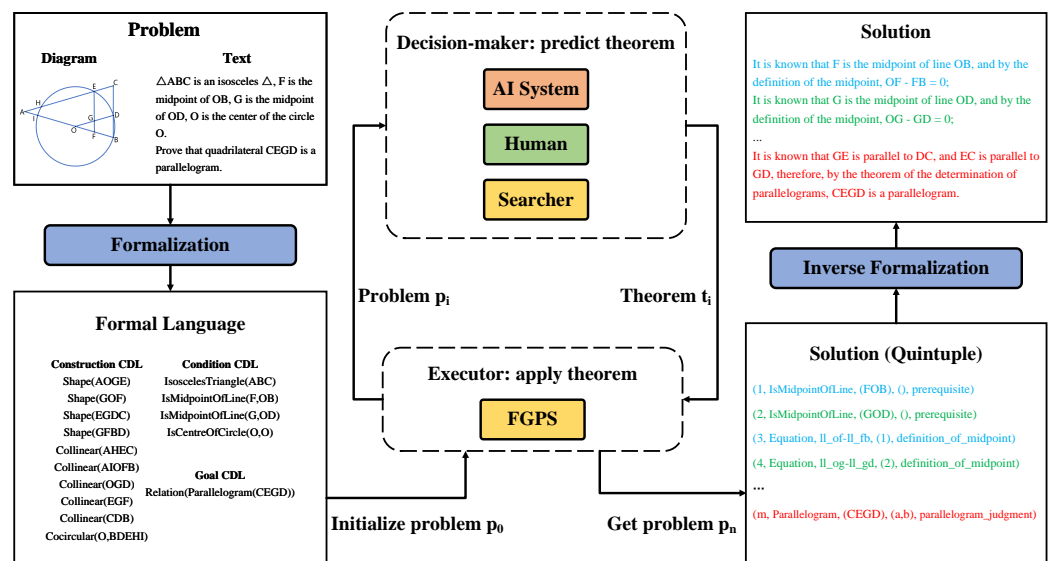


Figure 1. Interactive geometric problem solving based on FGPS.

Traditional methods for GPS can generally be divided into three categories [8]: synthesis methods, algebraic methods, and invariant-based point elimination methods. Synthesis methods, such as the backward search method [9], forward chaining method [10] and deductive database method [11], is essentially a search-based method. Algebraic methods are based on coordinates, such as Wu’s method [12], transforming GPS into a computational problem. Invariant-based point elimination methods [13] find that the solution methods for geometric problems are embedded in their geometric shape construction. Due to limitations in computing power and the readability of the solution process, traditional methods can solve only a limited type and difficulty of geometric problems.

The rapid development of AI has provided new ideas for GPS. New neural network architectures have given models the ability to model geometric knowledge. New learning

and training methods have made it easier for people to teach computers geometric a priori knowledge. Hence, AI-assisted synthesis methods, i.e., heuristic search methods, have attracted much attention from researchers. Such works can generally be divided into two categories [14]: symbolic approaches [15] and probabilistic approaches [16]. Symbolic methods require the prior construction of a formal system, and then geometric problem texts and images are parsed into a unified formal language. The AI predictor predicts theorems that may be needed in problem solving according to the formal language. The formal reasoner applies theorems and updates the known conditions of the problem. Direct theorem prediction can be modeled as a sequence generation task, and step-by-step theorem prediction can be modeled as a multi-classification task. Probabilistic methods model GPS as a sequence generation task with multimodal inputs, directly encoding geometric problem texts and images into a unified internal representation, then learning from human examples to generate problem-solving programs. These programs are subsequently executed by an executor to obtain the solution.

Nevertheless, both symbolic and probabilistic methods focus on the study of learning methods and model structures, neglecting the study of formal systems. Existing formal systems are roughly implemented using programming languages, and defining new predicates and theorems requires changing the solver's code, making it difficult to expand. The reasoning process cannot record process information, making the problem-solving process unreadable and unverifiable by humans. There is little research on the formal theory of geometry, and no systematic sorting of the structure and knowledge in the geometric field. These shortcomings severely limit the representational ability of formal systems, making it almost impossible to solve higher difficulty problems, such as International Mathematical Olympiad (IMO)-level geometric problems [17]. There is an urgent need for research on geometric formal systems and solvers.

Based on FormalGeo [7], we constructed a search-based symbolic solver to address these issues. FormalGeo is a geometric formal system built on Geometry Formalization Theory, featuring good readability. The syntax of the geometric formal language is similar to predicate logic and close to natural language, providing excellent readability and establishing a bridge for communication between humans and computers. The geometric formal language includes Geometric Definition Language (GDL) and Condition Declaration Language (CDL). FGPS parses GDL and CDL to configure the formal system and input geometric problem. FormalGeo transforms the process of GPS into a series of applications of theorems, defined using Geometric Predicate Logic (GPL). FGPS can parse and execute GPL, achieving traceable geometric relational reasoning and algebraic equation solving. The known conditions of a geometric problem are stored internally in FGPS as quintuples (condition ID, condition type, condition body, premises, and theorem). Based on the premises and theorem of conditions, we reorganize and structure them, transforming the process of GPS into a hypertree with conditions as hypernodes and theorems as hyperedges. Thus, we achieve formal language input of problems and structured output of solutions, ultimately realizing readable, traceable, and verifiable solutions for geometric problems.

FGPS can run in two modes: interactive and automated. As shown in Figure 1, in the interactive mode, an external decision-maker is required to provide the next theorem information. FGPS applies the theorem and updates the problem status. When the external decision-maker is a human, FGPS can serve as an interactive proof assistant to help humans verify the proof process; when the decision-maker is an AI, it can conveniently implement heuristic search methods. When FGPS operates in automatic mode, it can use various methods (forward search, backward search, and heuristic search) combined with various search strategies (breadth-first search, depth-first search, random search, and random beam search) to solve problems, automatically removing redundant theorems after achieving the solution goal.

Our contributions can be summarized as follows:

- We crafted FGPS in Python. It serves as both an interactive assistant for verifying problem-solving processes and an automated problem solver that utilizes a variable

search-based method and strategy. FGPS incorporates functions such as formal statement parsing, condition validity checks, automatic diagram construction, and condition expansion. It is capable of performing readable, traceable, and verifiable algebraic equation solving and relational reasoning.

- We explored the symmetry phenomena inherent in basic geometric transformations, geometric constructions, geometric problem solving, and the design of the solving system, and utilized these symmetries to further refine the system's design. The two symmetrical parts can verify and complement each other, making the design of FGPS more comprehensive and avoiding omissions.
- We annotated the formalgeo7k dataset, which contains 6981 (expanded to 133,818 using data augmentation method). Each problem comprises a complete natural language description, geometric shapes, formal language annotations, and theorem sequences annotations. Additionally, we have attempted to annotate 18 geometry IMO-level problems, forming the dataset formalgeo-imo.
- We conducted experiments on formalgeo7k, comparing different search methods and strategies in terms of their problem-solving success rate, search time, and search step. The forward search method combined with random search strategy achieved a 39.7% problem-solving accuracy rate.

## 2. Related Works

Gelernter et al. developed the pioneering automated GPS system known as the Geometry Theorem Prover [9], which employed a backward search approach to solve pre-formalized problems. Nevins pointed out that the forward chaining method [10] can also be effective by efficiently representing the known conditions of the problem and limiting the typical application of those conditions. The development of geometry problem solving has led to the emergence of various downstream tasks, including geometry problem formalization [18,19], geometric knowledge extraction [20–24], geometric diagram parsing [25–27], geometric theorem proving [28–30], and geometry problem solving [31–36]. Such methods are essentially a search-based method, which requires humans to predefine the search space or provide the system with a priori knowledge, namely theorems. Meanwhile, geometric problems need to be highly formalized to suit search formats. Theoretically, given a complete formal system and a priori knowledge, the search-based solver could provide solutions to any problem. However, in practice, we cannot determine if the given a priori knowledge is complete. Additionally, due to the exponential explosion characteristic of search-based methods, it is impossible to provide a correct solution within reasonable memory and time constraints.

Wen-Tsun proposed Wu's method [12], which transforms geometric problem into a system of algebraic equations consisting of polynomials and inequalities. Then, by utilizing the properties of algebraic computation, these algebraic expressions are simplified and solved, transforming complex algebraic expressions into forms understandable by humans, and thus interpreting their geometric meanings. The study of algebraic approaches to geometry problems has given rise to a range of research achievements, such as Buchberger's Gröbner bases method [37], numerical parallel methods [38], polynomial system triangulation elimination algorithm [39], cylindrical algebraic decomposition for solving inequalities [40], dimensionality reduction methods [41], and software tools like GEOTHER 1.0 [42]. These methods fully leverage the computational power of computers, but their problem-solving processes are not easily comprehensible to humans.

Zhang proposed the point elimination method based on geometric invariants [13]. This approach employs constructive methods to describe problems and is capable of generating concise and meaningful readable proofs for a large number of non-trivial geometric problems. Subsequently, research on machine proofs of geometric theorems based on geometric invariants rapidly advanced [43–45], leading to the development of practical software tools such as Geometry Explorer 1.1 [46], Geometry Expert 1.0 [47], and Java Geometry Expert 0.8 [48]. The method based on geometric invariant can also

be extended to solid geometry [49] and non-Euclidean geometry [50]. Point elimination method is a mechanized method but requires a lot of effort in discovering new invariants and shape construction, and the types of geometric problems that can be solved are limited.

GPS has been gaining more attention in the natural language processing community recently. Several geometry formal systems and datasets have been constructed, such as Geometry3K [15], GeoQA [16], and GeometryQA [51]. Geometry3K [15] translates the known conditions of geometric problems into formal statements, defining theorems as a set of rules for converting between formal statements. This approach, referred to as "formal language", is also used in GeoRE [22], which focuses on geometric relation extraction, and PGDP5k [52], which is designed for geometric image parsing; while these methods are intuitive, they lack theoretical guidance, are not comprehensive, and are not easily extensible with additional predicates and theorems. GeoQA [16] employs the formal method of the "program" approach, transforming the process of GPS into a sequence of programs consisting of variables and operators. Executing this program sequence yields the solution. Subsequent work extended the number and types of questions and rules, resulting in GeoQA+ [53], UniGeo [54], and PGPS9K [55]. These formal methods can represent algebraic and symbolic problem-solving processes, but compared to formal language methods, they are less intuitive and cannot represent traditional solutions. Additionally, adding new rules requires modifying the solver's code, making them less extensible. GeometryQA [51] employs a formal method known as the expression tree, which transforms the problem-solving process into a solving tree composed of operators and variables. This method is similar to the program approach but is more structured. Shared benchmarks and datasets have significantly advanced research in AI-assisted GPS. Several AI systems, such as the CL-based model [56], SCA [57], GeoDRL [14], and LANS [58], have been constructed to achieve higher problem-solving success rates.

The integration of logic and neural networks has increasingly captured researchers' interest in recent years. Early research endeavors, such as Neural Turing Machines [59], Memory Networks [60], and Neural Programmer [61], replaced discrete algorithms with end-to-end differentiable counterparts, which enable models to not only optimize via gradient descent but also to incorporate discrete computational and reasoning capabilities. Subsequent studies infused this concept into logical reasoning, proposing many neurosymbolic approaches. Notable examples include Neural Theorem Provers [62], DeepProbLog [63], Logical Tensor Networks [64], Neural Logic Machines [65], and Conditional Theorem Provers [66], which have been extensively applied in the area of knowledge graph reasoning [67,68]. In the field of automated theorem proving, simple reasoning rules are inducted as tactics, while large language models (LLMs) learn higher-level reasoning rules from experience, predicting the tactics necessary for problem solving. These tactics are then fed into a symbolic reasoner for verification. Approaches such as SCL [69], Thor [70], and HTTPS [71] have been successful in solving IMO-level problems [72]. Furthermore, the reasoning capabilities of LLMs [73,74] have become a focal point of current research [75,76].

### 3. FGPS

In this section, we introduce the specific implementation of FGPS. FGPS is built upon Geometry Formalization Theory and the FormalGeo formal system [7].

#### 3.1. Architecture

As illustrated in Figure 2, FGPS can be divided into five components:

- Main is the control module of FGPS, invoking other modules to enable interactive problem solving and automated problem solving. The automated solving component implements both forward search and backward search, allowing for the configuration of various search strategies and defining interfaces for AI-assisted searches.
- Engine is the core component of FGPS, responsible for parsing and executing GPL and consists of two sub-modules, GPL executor for relational reasoning and equation solver for algebraic computation.

- Parser facilitates bidirectional conversion between formal language and machine language. It consists of three sub-modules. GDL parser parses GPDL and GTDL into machine language, enabling custom configuration of the solver. CDL parser parses the formal describing of problems into machine language for subsequent reasoning. Inverse parser translates machine language back into formal language, facilitating the verification and checking of the solution process.
- Data preserves all details of the problem-solving process and comprises two sub-modules. The problem module ensures the correctness and consistency of the problem input conditions, implementing automatic diagram construction, condition auto-expansion, and validity checks. The condition module is responsible for data storage.
- AI interface defines the interface for interaction between the AI system and FGPS. Both the AI automatic formalization and the AI problem solver can be seamlessly integrated with FGPS.

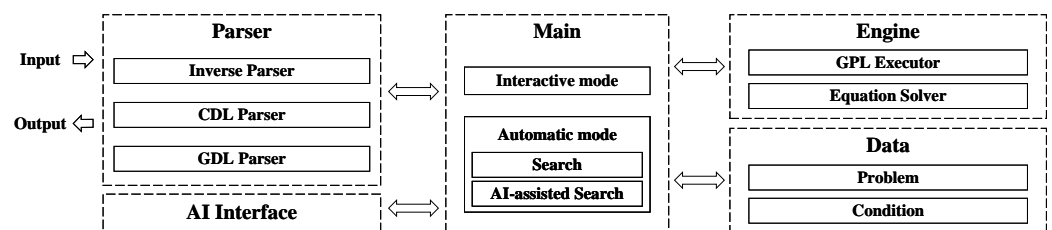


Figure 2. The components of FGPS.

Guided by Geometry Formalization Theory and modular design, FGPS boasts excellent extensibility, allowing for easy customization of the formal system and the definition of external interfaces.

### 3.2. GPL Executor

The process of GPS can be represented as a sequence of theorem applications and theorems are defined using GPL. As a result, the process of GPS within FGPS is essentially the execution of GPL. GPL statements can consist of multiple logical conjunction words, geometric relations, and quantitative relations nested together. The application process can be divided into three steps:

1. In the GPL parsing phase, the solver expands complex GPL statements into Disjunctive Normal Form (DNF) using the distributive law. Each simple conjunction represents a branch of the theorem. This not only meets the requirements for backward reasoning and facilitates the generation of sub-goals but also speeds up theorem execution by skipping irrelevant branches.
2. In the GPL ordering phase, for each branch of the theorem, the solver adjusts the positions of geometric relations and quantitative relations within simple conjunctions according to the commutative law. The guiding principles for this adjustment are as follows: 1. Transforming relation composition into geometric constraints. 2. Moving geometric constraints forward. 3. Moving algebraic constraints backward. This approach not only helps filter out geometric relation elements that do not comply with the constraints, preventing the explosion of combinatorics caused by Cartesian product operations, but also reduces the number for algebraic equation solving, thereby improving theorem application speed.
3. In the GPL execution phase, the solver reads geometric and quantitative relations sequentially and performs relational reasoning in the order of their appearance.

The GPL execution process can be illustrated with an example. Suppose that we have a theorem defined as shown in Equation (1), which includes five geometric relations  $R_1(v_1, v_2)$ ,  $R_2(v_2, v_3)$ ,  $R_3(v_2)$ ,  $R_4(v_2, v_3)$ , and  $R_5(v_2)$  and one quantitative relation

$R_A(v_1, v_2)$ . It should be noted that the detailed definition of  $\&$ ,  $|$  and  $\sim$  can be found in Geometry Formalization Theory [7].

$$R_1 \& (R_2 | (\sim R_3 | R_A) \& R_4 \& R_5) \quad (1)$$

During the GDL parsing phase, it is expanded into a DNF according to the distributive law, as shown in Equation (2). This DNF consists of three simple conjunctions, with each simple conjunction serving as a theorem branch.

$$R_1 \& R_2 | R_1 \& \sim R_3 \& R_4 \& R_5 | R_1 \& R_A \& R_4 \& R_5 \quad (2)$$

In the GDL reordering phase, let us take branch  $R_1 \& R_A \& R_4 \& R_5$  as an example. It adjusts the order of its statements according to the commutative law, resulting in the form shown in Equation (3).

$$R_1 \& R_5 \& R_4 \& R_A \quad (3)$$

In the GPL execution phase, the GDL statements are read and executed in order, and the process is as shown in Equation (4).

$$R_1 \& R_5 \& R_4 \& R_A \rightarrow R_{1,5} \& R_4 \& R_A \rightarrow R_{1,5,4} \& R_A \rightarrow R_{1,5,4,A} \quad (4)$$

### 3.3. Core Engine

This subsection introduces the key methods that enable FGPS to perform geometric relational reasoning and algebraic equation solving. These methods are organized within a unified framework and constitute the core solving engine of FGPS.

#### 3.3.1. Traceable Geometric Relational Reasoning

We transformed the execution process of GPL into its two most fundamental operations:  $R_1 \& R_2$  and  $R_1 \& R_A$ .  $R_1$  and  $R_2$  are geometric relations, and  $R_A$  is an quantitative relation. Let us assume the existing geometric relations  $R_1(a, b) = \{(X, Y), (M, N)\}$ ,  $R_2(b, c) = \{(Y, Z)\}$ , quantitative constraint  $R_A(a, b): Equal(Attr(a), Attr(b))$ , and the known algebraic equations being  $\{attr\_m - attr\_n = 0\}$ .

For the  $R_1 \& R_2$  type of operation, first, a Cartesian product operation is executed to obtain preliminary reasoning results  $R'$ . Subsequently, intersection is performed on their point variables to obtain common variables, and each value of the common variables in  $R'$  is checked for correctness, filtering out the elements that meet the requirements. Then, a union of point variables is taken to reorganize the reasoning results.

Let us take  $R_1(a, b) \& R_2(b, c)$  as example. First, we perform the Cartesian product operation:

$$R_1(a, b) \& R_2(b, c) = \{(X, Y, Y, Z), (M, N, Y, Z)\} \quad (5)$$

We filter out the elements based on common variables:

$$R'(a, b, b, c) = \{(X, Y, Y, Z)\} \quad (6)$$

Remove duplicates variables:

$$R_{1,2}(a, b, c) = \{(X, Y, Z)\} \quad (7)$$

For the  $R_1 \& R_A$  type of operation, the intersection of point variables is taken to obtain common variables, and then the values of these common variables are substituted into the quantitative relation  $R_A$ , filtering out the elements that conform to the algebraic constraints.

Let us take  $R_1(a, b) \& R_A(a, b)$  as example. First, we construct equations:

$$R_1(a, b) \& R_A(a, b) = \{attr\_a - attr\_b = 0, attr\_m - attr\_n = 0\} \quad (8)$$

We construct a system of equations based on the known algebraic equations and check whether the equations hold true.

$$R'_A(a, b) = \{attr\_m - attr\_n = 0\} \quad (9)$$

Filter the elements in  $R_1$  based on whether the equations hold true.

$$R_{1,A}(a, b) = \{(M, N)\} \quad (10)$$

In the above reasoning process, the premises of the new relations are recorded synchronously, achieving traceable geometric relational reasoning.

### 3.3.2. Minimum Dependency Equation Constructing

The known conditions of geometric problems can be categorized into geometric and quantitative relationships. Quantitative relationships are eventually represented as a set of algebraic equations or inequalities. When performing algebraic constraint in the execution of GPL, the satisfaction of the algebraic constraint under the known algebraic equations or inequalities of the problem is checked.

Algebraic constraints can be transformed into algebraic expressions represented by  $a$ , creating the target equation  $g - a$ . Among the several known equations  $X$  in the problem conditions, those relevant to  $g - a$  are selected to construct the target equation group  $G$ , which is subsequently solved. If  $g = 0$  is obtained as a solution, the algebraic constraints are satisfied. Typically, only a few equations in  $X$  are related to  $g - a$ , and this subset of equations is referred to as the minimum dependency equations.

The solving of equations accounts for the majority of the time spent in the entire process of solving geometric problems. Accelerating the equation solving process is crucial for enhancing the speed of geometric problem solving. To address this, we propose a method for constructing the minimum dependency equations. Without loss of generality, we examine the intermediate process of constructing  $G$ . At time  $t$  ( $t = 1, 2, \dots$ ),  $G_t$  contains  $t$  equations and  $m$  unknowns, with the set of unknowns denoted as  $M_t$ . We need to select a candidate equation  $x_t$  from  $X$  to add to  $G_t$  in a way that increases the likelihood of obtaining a solution for the unknown  $g$ . The set of unknowns in  $x_t$  is represented as  $B_t$ . This process is repeated until  $|M_t| = t$  or no new equations can be added.

$$|B_t \cap M_t| > 0 \quad (11)$$

$$\min(|B_t - M_t|) \quad (12)$$

$$\max(|B_t \cap M_t|) \quad (13)$$

The selection criteria for  $x_t$  are as follows:

1.  $B_t$  must intersect with  $M_t$ , as shown in Equation (11). If they do not intersect, it implies that  $x_t$  is unrelated to  $G_t$ .
2. Under the condition of satisfying Equation (11), adding  $x_t$  should introduce as few new unknown variables as possible, as depicted in Equation (12). The closer the number of  $t$  and  $|M_t|$  are, the higher the likelihood of solving  $G_t$ . In the initial stages of constructing  $G$ , which only contains  $g - a$ ,  $M_1 - 1 > 1$ . The number of added equations each time is a fixed value 1. If we aim to minimize the gap between  $t$  and  $|M_t|$ , we should try to introduce as few new equations when selecting  $x_t$ .
3. Under the condition of satisfying Equations (11) and (12), the equation to be added should encompass more unknown variables, as demonstrated in Equation (13). These additional unknown variables are often associated with other equations within  $G_t$ , providing more choices for simplifying  $G_t$ . If there are multiple equations that satisfy these conditions, we can choose any of them at random.



### 3.4. GPS Methods

By invoking FGPS's core modules, we have developed both an interactive solver and a search-based problem solver. Next, we will introduce the forward search algorithm and the backward search algorithm.

Forward search (FW) starts from the known conditions of the problem and continuously applies theorems to derive new conditions until the goal is achieved. The search process involves the construction of a search tree, with nodes representing sets of known conditions and edges denoting theorems, as depicted in Figure 3. The description of the forward search algorithm is provided in Algorithm 1. The function *get\_expandable()* traverses the search tree based on predefined strategies (BFS, DFS, RS, and RBS) and returns nodes with the EXPANDABLE state. The function *apply\_theorem()* applies the theorem associated with the current node and returns whether the problem is solved. The function *get\_theorem\_seqs()* returns a list of theorems applied from the root node to the current node. The function *expand()*, guided by the known conditions of the current node, checks the list of applicable theorems and extends new nodes.

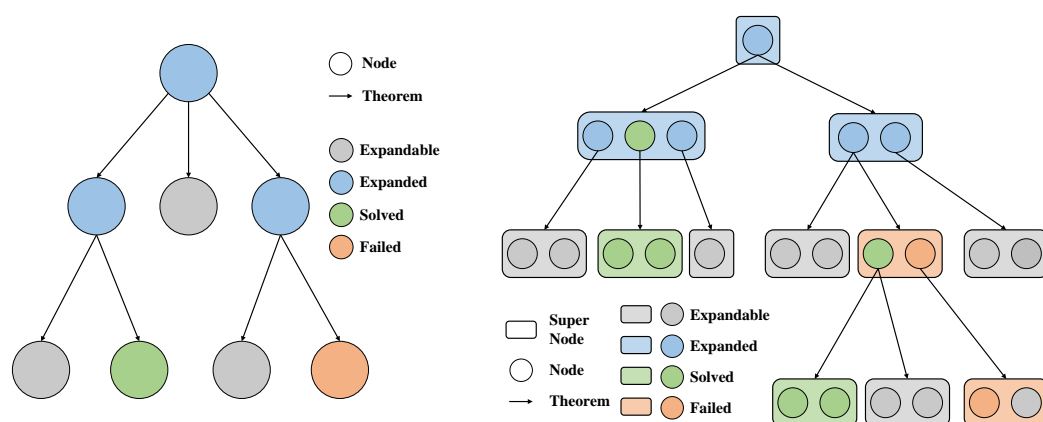


Figure 3. Forward search tree (left). Backward search tree (right).

#### Algorithm 1 Forward search.

**Input:** *tree*: a tree with the known problem conditions as the root node.

**Output:** *theorem\_seqs*: list of theorem sequences for problem solving.

Initialize a list *theorem\_seqs*

*node*  $\leftarrow$  *tree.get\_expandable()*

**while** *node* is not *None* **do**

*solved*  $\leftarrow$  *node.apply\_theorem()*

**if** *solved* **then**

*node.state*  $\leftarrow$  SOLVED

*theorem\_seqs*  $\leftarrow$  *node.get\_theorem\_seqs()*

**break**

**end if**

*node.state*  $\leftarrow$  EXPANDED

**if** *node.expand()* = 0 **then**

*node.state*  $\leftarrow$  FAILED

**end if**

*node*  $\leftarrow$  *tree.get\_expandable()*

**end while**

The complexity of forward search is related to the size of the search space (the number of defined theorems  $M$ ) and the difficulty of the problem (the length of the problem-solving theorem sequence  $L$ ). We need to apply theorems to expand each node of the forward search tree. The number of theorems that can be applied varies for different types of nodes, and we take the average number,  $k_f M$ , as the number of applied theorems for each node

expansion. From the beginning to the end of problem solving, the search tree expands to the  $M$  level, and we need to store all nodes of the search tree. The time complexity and space complexity are shown in Equation (14) and Equation (15), respectively.

$$T(M, L) = \sum_{l=1}^L (k_f M)^l = \frac{k_f M (k_f M^L - 1)}{k_f M - 1} = O(M^L) \quad (14)$$

$$S(M, L) = \sum_{l=0}^L (k_f M)^l = \frac{k_f M^{L+1} - 1}{k_f M - 1} = O(M^L) \quad (15)$$

Backward search (BW), on the other hand, begins with the problem-solving goal, expands it into multiple sub-goals, and repeats this process until all sub-goals are resolved. The search process involves the construction of a search tree, with nodes representing sub-goals, supernodes representing sets of sub-goals, and edges representing theorems, as illustrated in Figure 3. The description of the backward search algorithm is provided in Algorithm 2. The function *get\_expandable()* traverses the search tree based on predefined strategies, returning supernodes with the EXPANDABLE state. The function *node.check()* updates the state of the superNode based on the known problem conditions, while the function *super\_node.check()* updates its own state based on the states of its nodes. The function *expand()* extends the current goal into several sub-goals based on the list of theorems. The function *update()* propagates the state update from child nodes to parent nodes, starting from the leaves and progressing up to the root. The function *get\_theorem\_seqs()* provides a list of theorems applied from the current node to the root node.

---

#### Algorithm 2 Backward search.

---

**Input:** *super\_tree*: a super tree with the problem goal as the root super node.

**Output:** *theorem\_seqs*: list of theorem sequences for problem solving.

Initialize a list *theorem\_seqs*

*super\_node* ← *super\_tree.get\_expandable()*

**while** *super\_node* is not None **do**

**for**  $i = 1$  to  $\text{len}(\text{super\_node.nodes})$  **do**

*node* ← *super\_node.nodes*[ $i$ ]

*node.check()*

**if** *node.state* is SOLVED **then**

      continue

**else if** *node.state* is FAILED **then**

      break

**else if** *node.expand()* = 0 **then**

*node.state* ← FAILED

      break

**else**

*node.state* ← EXPANDED

**end if**

**end for**

*super\_node.check()*

*super\_tree.update()*

**if** *super\_tree.solved* **then**

*theorem\_seqs* ← *super\_tree.get\_theorem\_seqs()*

    break

**end if**

*super\_node* ← *super\_tree.get\_expandable()*

**end while**

---

Similar to the forward search algorithm, the backward search algorithm is related to  $M$  and  $L$ . Let us assume that each node can be expanded into an average of  $k_b M$  supernodes,

and each supernode contains an average of  $k_s$  nodes. In the node expansion phase, the algorithm needs to expand one node to  $k_b M \times k_s$  nodes. In the backpropagation phase, algorithm needs to update the state of  $(k_s k_b M)^l$  nodes from  $l$  levels up to root node. Therefore, its overall time complexity is as shown in Equation (16). The algorithm needs to save all nodes during the running process, and its space complexity is as shown in Equation (17).

$$T(M, L) = \sum_{l=1}^L (k_s k_b M)^l + \sum_{l=1}^L l (k_s k_b M)^l = O(LM^L) \quad (16)$$

$$S(M, L) = \sum_{l=1}^L (k_s k_b M)^l = O(M^L) \quad (17)$$

#### 4. Datasets

Most of the existing datasets for GPS suffer from the following issues: 1. Limited data volume or non-open source availability. 2. Lack of annotations or incomplete and low-quality annotations. 3. Absence of formalization theory support, resulting in incoherent and inconsistent formal systems. 4. Low scalability, defining new predicates and theorems require solver's code modifications. 5. Lower difficulty level of the problems. To address the aforementioned issues, we annotated datasets formalgeo7k and formalgeo-imo. We conducted a comparative analysis with existing works, as shown in Table 1.

**Table 1.** Comparative analysis with existing GPS datasets.

Datasets	FM	Size	Comparative Metrics									
			FW	BW	IS	AS	NT	PT	ET	VC	SS	
GEOS [31]	FL	186	✓		✓			✓				
GEOS++ [20]	FL	1406	✓		✓			✓				
Geometry3K [15]	FL	3002	✓		✓			✓				
GeoQA [16]	P	5010	✓		✓			✓				
GeometryQA [51]	P	1398	✓		✓			✓				
GeoQA+ [53]	P	7528 (5010 from [16])	✓		✓			✓				
UniGeo [54]	P	14,541 (4998 from [16])	✓		✓			✓	✓			
PGPS9K [55]	P	9022 (2891 from [15])	✓		✓			✓				
formalgeo7k	FL	6981 (DA to 133,818)	✓	✓	✓		✓	✓	✓	✓	✓	✓
formalgeo-imo	FL	18 (DA to 2627)	✓	✓	✓		✓	✓	✓	✓	✓	✓

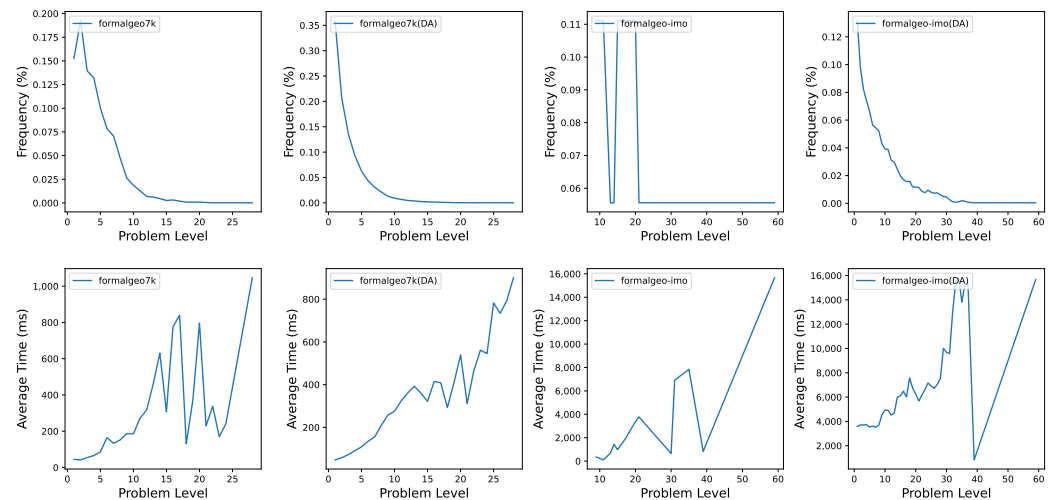
✓ indicates that the dataset and its formal system have such features. FM denotes formalized methods, FL denotes formal language, P denotes program, and DA denotes data augmentation. The 9 comparative metrics are forward solving method, backward solving method, interactive solving, automatic solving, numerical target, proving target, extensible, validity check, and structured solution.

Our data are collected from various sources, including Geometry3k [15], GeoQA [16], GeoQA+ [53], and online resources. We carefully curated, classified, deduplicated, and standardized the problem statements. The creation of the formalgeo7k involved 16 trained master's students over a period of around 13 weeks. The creation of the formalgeo-imo involved four trained master's students over a period of around 1 week. Excluding the time spent on collaboration and dataset allocation, annotating datasets took approximately 1000 person-hours.

formalgeo7k comprises 6981 geometric problems that are accompanied by natural language descriptions, geometric diagrams, formal language annotations, and solution theorem sequence annotations. An annotated problem is illustrated in Figure 1 (omitting the theorem annotations). The problem-solving process can be represented as a hypertree with conditions as hypernodes and theorems as hyperedges. The solution theorem sequence is a path from the root node (known conditions) to a leaf node (the problem-solving objective). By selecting any intermediate node along this path as the problem-solving objective, we can generate new problems, allowing us to expand the problem number to

133,818. formalgeo-imo is constructed with the same standards but with more challenging problem difficulty.

We used the length of theorem sequences required for problem solving as a rough metric for assessing problem difficulty. All annotated and expanded problems have been verified by the solver, and their average solution times varying with problem difficulty are also show in Figure 4. The number of questions with a difficulty level of 15 or higher in formalgeo7k is quite small, leading to significant fluctuations. After data augmentation, datasets exhibited a larger scale of data and a smoother difficulty curve. In general, more challenging problems require longer solving time.



**Figure 4.** Distribution of problem (the top 4). Average time of interactive verification (the bottom 4). DA represents data augmentation.

## 5. Experiments

We conducted experiments on the formalgeo7k, comparing different search methods and strategies in terms of problem-solving success rate, solution time, and the number of steps required for problem solving.

Forward search (FW) starts from the known conditions of the problem and continuously applies theorems to derive new conditions until the goal is achieved. Backward search (BW), on the other hand, begins with the problem-solving goal, expands it into multiple sub-goals, and repeats this process until all sub-goals are resolved.

The search-based methods construct a search tree during the problem-solving process. We have the flexibility to choose various strategies to traverse the search tree and reach the goal. Breadth-first search (BFS) begins by expanding the top-level nodes of the search tree and then proceeds layer by layer into the depth. Depth-first search (DFS) recursively selects nodes from the search tree from shallow to deep and continues this process. Random search (RS) randomly selects an expandable node at each stage of expansion. Beam search (BS) selects  $k$  nodes in each stage of expansion and can be viewed as a trade-off between BFS and RS.

We conducted experiments on two Intel i9-10900X, one AMD Ryzen 9 5900X, and one AMD Ryzen 9 7950X (The Intel processors were sourced from Intel Corporation, Santa Clara, California, USA. The AMD processors were sourced from Advanced Micro Devices, Inc., Sunnyvale, California, USA.), running the search algorithms using multiple processes while maintaining a CPU utilization rate of 80%. The maximum search depth was set to 15, and the beam size was set to 20. The total duration of the experiments was approximately 3 days. When the timeout for each problem was 300 s, the best success rate for problem solving was approximately 30%. When the timeout for each problem was increased to 600 s, the specific results are as follows.

An overview of search-based automated problem-solving results is presented in Table 2. The highest problem-solving success rate was achieved by forward random search, reaching 39.708%. Most of the remaining problems were due to timeouts. As timeout settings are extended and computational resources increase, the proportion of timeout problems is expected to decrease. The number of unsolved problems using beam search was significantly higher compared to other strategies. This is because when selecting  $k$  branches, beam search occasionally discards the correct branch. Other contributing factors may include code bugs, equation solving timeouts, and the omission of theorems related to trigonometry.

**Table 2.** An overview of search results.

Method	Strategy	Result (%)		
		Solved	Unsolved	Timeout
FW	BFS	38.86	<b>7.42</b>	53.72
FW	DFS	36.16	9.80	54.05
FW	RS	<b>39.71</b>	9.07	51.22
FW	BS	25.28	38.72	<b>36.00</b>
BW	BFS	<b>35.44</b>	2.68	61.88
BW	DFS	33.73	<b>2.42</b>	63.84
BW	RS	34.05	2.65	63.30
BW	BS	34.39	12.86	<b>52.74</b>

The bold data indicates the best results in the FW or BW methods.

In accordance with the length of the theorems required for problem solving, we roughly categorized the difficulty of the questions into six levels, denoted as  $l_1$  ( $length < 3$ ),  $l_2$  ( $3 \leq length \leq 4$ ),  $l_3$  ( $5 \leq length \leq 6$ ),  $l_4$  ( $7 \leq length \leq 8$ ),  $l_5$  ( $9 \leq length \leq 10$ ), and  $l_6$  ( $length > 10$ ), with corresponding problem numbers of 2407, 1898, 1247, 824, 313, and 292. The success rates for solving geometric problems of varying difficulty are presented in Table 3. As the problem difficulty increases, the success rate of problem solving rapidly declines. This phenomenon can be attributed to the fact that search-based problem-solving methods exhibit exponential growth in solving time as the length of the theorem sequence increases, often resulting in timeouts before achieving the goal. For problems of lower difficulty, backward search demonstrate a relatively higher success rate, while forward search outperforms in the case of more challenging problems.

**Table 3.** Results of success rates.

Method	Strategy	Success Rates (%)						
		Total	$l_1$	$l_2$	$l_3$	$l_4$	$l_5$	$l_6$
FW	BFS	38.86	59.95	38.62	28.55	<b>17.35</b>	<b>8.63</b>	3.77
FW	DFS	36.16	55.75	<b>40.04</b>	22.94	12.38	7.03	4.11
FW	RS	<b>39.71</b>	59.24	<b>40.04</b>	<b>33.68</b>	16.38	5.43	<b>4.79</b>
FW	BS	25.28	46.12	22.60	13.47	5.83	2.88	0.34
BW	BFS	35.44	<b>67.22</b>	33.72	11.15	6.67	6.07	1.03
BW	DFS	33.73	65.93	30.82	8.90	6.55	5.11	0.68
BW	RS	34.05	66.64	31.66	8.66	5.83	4.47	0.68
BW	BS	34.39	67.10	31.35	9.46	6.31	5.75	1.03

The bold data indicates the best results in the FW and BW methods.

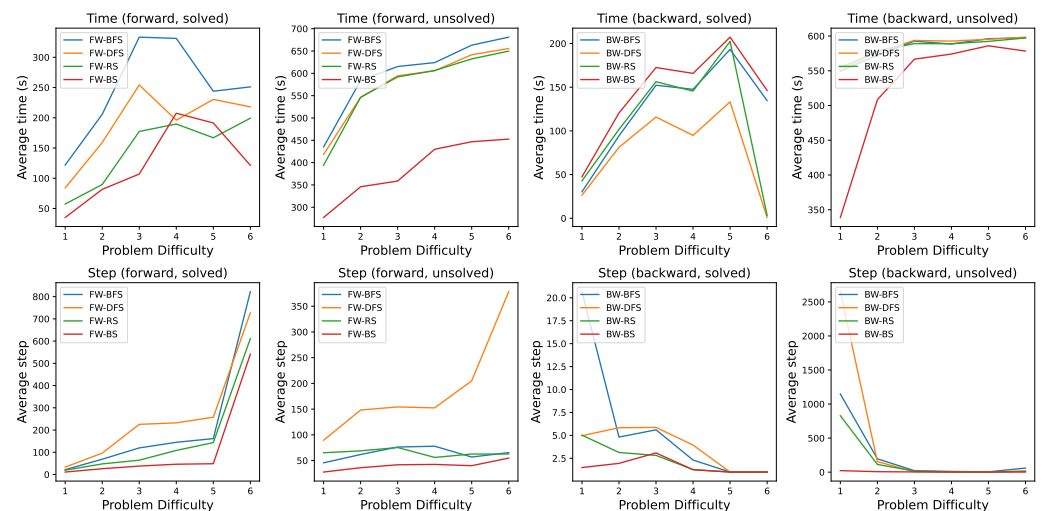
The efficiency of the problem-solving algorithm can be measured by the search time and step. The experimental results of search-based automated problem-solving algorithms on the formalgeo7k are presented in Figure 5.

In terms of average search time, backward search is slightly better than forward search overall. For solved problems, the search time is roughly proportional to the difficulty of the problems when problems are of low difficulty. However, as the difficulty increases,

the search time for both forward search and backward search decreases. On the one hand, this is because there are very few successfully solved high-difficulty problems, leading to significant statistical errors. On the other hand, when dividing the difficulty of problems, we only consider the length of the solution theorem sequence but do not consider the time required for each theorem execution. The solved high-difficulty problems are precisely those that require less solution time. For unsolved problems, the search time is roughly proportional to the difficulty of the problems.

Comparing different search strategies, it can be observed that in the forward search, BFS has a slightly lower success rate compared to the RS, but it takes the most time. BS has the lowest success rate but the least time consumption. For forward search, RS is the optimal strategy as it has the highest success rate and only slightly higher time consumption than BS that has the lowest success rate. In backward search, BFS is the optimal strategy, with the highest success rate and only slightly higher time consumption than DFS.

We observe a significant difference in the solution time of the BS strategy in backward search for solved and unsolved problems. This difference may be due to the characteristics of the backward search, where even if possible solution branches were discarded in previous steps, they may be reconstructed in later search steps. Therefore, as for BS strategy in backward search, discarding potential solution branches does not lead to solution failure but takes longer search time.



**Figure 5.** Average search time (the top 4). Average search step (the bottom 4).

Regarding the search step, forward search statistics are based on the number of nodes, while backward search statistics are based on the number of super nodes. Hence, they cannot be directly compared. The search step length in forward search is positively correlated with the difficulty of problems, while in backward search, it is negatively correlated with problem difficulty. The results of backward search are counterintuitive, and this could be because, for higher difficulty problems, the super nodes in backward search may contain more nodes, leading to increased time spent traversing a single super node and a reduction in the total number of traversed super nodes. Additionally, it can be observed that the search step length for unsolved problems in backward search is significantly higher than the average step length for solved problems. This is because, compared to forward search, backward search is less likely to halt, and it continues searching even if it misses a potential solution branch.

Comparing different strategies, DFS has the highest search step length, BS has the lowest search step length, and RS and BFS strategies have approximately the same average step length. For forward search, RS strategy is still the optimal strategy because it has the highest success rate and its search step length is only slightly higher than BS. Backward search does not exhibit a significantly superior strategy.

## 6. Conclusions

Based on the Geometry Formalization Theory and the FormalGeo geometric formal system, we constructed FGPS, which can serve as an interactive assistant for verifying problem-solving processes and as an automated problem solver that utilizes variable search-based methods and strategies. Moreover, we explored the symmetry phenomena inherent in basic geometric transformations, geometric constructions, geometric problem solving, and the design of the solving system, and utilize these symmetries to further refine the system's design. We annotated GPS datasets formalgeo7k and formalgeo-imo, the former contains 6981 geometry problems with detailed formal language descriptions and solutions. Experiments have demonstrated the correctness and efficiency of FGPS. In the future, we plan to further refine the formal system to annotate geometry problems at the IMO-level. We also plan to apply deep learning techniques to search tree pruning for the automatic solving of IMO-level geometric problems.

**Author Contributions:** Conceptualization, X.Z. and T.L.; methodology, X.Z., N.Z., Y.H., J.Z. and T.L.; software, X.Z.; validation, X.Z., C.Q. and Y.L.; writing—original draft preparation, X.Z.; writing—review and editing, X.Z. and T.L.; supervision, T.L.; funding acquisition, T.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the National Natural Science Foundation of China grant 12071282.

**Data Availability Statement:** The project is available at <https://github.com/BitSecret/FGPS> (accessed on 24 March 2024).

**Acknowledgments:** We thank the reviewers for their helpful comments.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

- Balasubramanian, K. Symmetry, combinatorics, artificial intelligence, music and spectroscopy. *Symmetry* **2021**, *13*, 1850.
- Elliott, J.P.; Dawber, P.G. *Symmetry in Physics*; Macmillan: London, UK, 1979; Volume 1.
- Toxvaerd, S. The emergence of the bilateral symmetry in animals: A review and a new hypothesis. *Symmetry* **2021**, *13*, 261.
- Daniel, S.; Leonardo, D.M.; Kevin, B.; Reid, B.; Percy, L.; Sarah, L.; Freek, W. IMO Grand Challenge. 2019. Available online: <https://imo-grand-challenge.github.io/> (accessed on 24 March 2024).
- XTXMarkets. Artificial Intelligence Mathematical Olympiad Prize (AIMO Prize), 2023. Available online: <https://aimoprize.com/> (accessed on 24 March 2024).
- Littman, M.L.; Ajunwa, I.; Berger, G.; Boutilier, C.; Currie, M.; Doshi-Velez, F.; Hadfield, G.; Horowitz, M.C.; Isbell, C.; Kitano, H.; et al. Gathering strength, gathering storms: The one hundred year study on artificial intelligence (AI100) 2021 study panel report. *arXiv* **2022**, arXiv:2210.15767.
- Zhang, X.; Zhu, N.; He, Y.; Zou, J.; Huang, Q.; Jin, X.; Guo, Y.; Mao, C.; Zhu, Z.; Yue, D.; et al. FormalGeo: The first step toward human-like IMO-level geometric automated reasoning. *arXiv* **2023**, arXiv:2310.18021.
- Jingzhong, Z.; Yongbin, L. Automatic theorem proving for three decades. *J. Syst. Sci. Math. Sci.* **2009**, *29*, 1155.
- Gelernter, H.L. Realization of a geometry theorem proving machine. In Proceedings of the IFIP Congress, Paris, France, 15–20 June 1959; pp. 273–281.
- Nevins, A.J. Plane geometry theorem proving using forward chaining. *Artif. Intell.* **1975**, *6*, 1–23.
- Chou, S.C.; Gao, X.S.; Zhang, J.Z. A deductive database approach to automated geometry theorem proving and discovering. *J. Autom. Reason.* **2000**, *25*, 219–246.
- Wu, W.T. On the decision problem and the mechanization of theorem proving in elementary geometry. *Sci. Sin.* **1978**, *21*, 157–179.
- Zhang, J.Z.; Chou, S.C.; Gao, X.S. Automated production of traditional proofs for theorems in Euclidean geometry I. The Hilbert intersection point theorems. *Ann. Math. Artif. Intell.* **1995**, *13*, 109–137.
- Peng, S.; Fu, D.; Liang, Y.; Gao, L.; Tang, Z. GeoDRL: A self-learning framework for geometry problem solving using reinforcement learning in deductive reasoning. In *Findings of the Association for Computational Linguistics: ACL 2023*; Association for Computational Linguistics: Toronto, ON, Canada, 2023; pp. 13468–13480.
- Lu, P.; Gong, R.; Jiang, S.; Qiu, L.; Huang, S.; Liang, X.; Zhu, S.c. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Virtual, 1–6 August 2021; pp. 6774–6786.

16. Chen, J.; Tang, J.; Qin, J.; Liang, X.; Liu, L.; Xing, E.; Lin, L. GeoQA: A geometric question answering benchmark towards multimodal numerical reasoning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*; Association for Computational Linguistics: Toronto, ON, Canada, 2021; pp. 513–523.
17. Trinh, T.H.; Wu, Y.; Le, Q.V.; He, H.; Luong, T. Solving olympiad geometry without human demonstrations. *Nature* **2024**, *625*, 476–482.
18. Gan, W.; Yu, X. Automatic understanding and formalization of natural language geometry problems using syntax-semantics models. *Int. J. Innov. Comput. Inf. Control* **2018**, *14*, 83–98.
19. Rao, Y.; Xie, L.; Guan, H.; Li, J.; Zhou, Q. A Method for expanding predicates and rules in automated geometry reasoning system. *Mathematics* **2022**, *10*, 1177.
20. Sachan, M.; Dubey, K.; Xing, E. From textbooks to knowledge: A case study in harvesting axiomatic knowledge from textbooks to solve geometry problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, 9–11 September 2017; pp. 773–784.
21. Sachan, M.; Dubey, A.; Hovy, E.H.; Mitchell, T.M.; Roth, D.; Xing, E.P. Discourse in multimedia: A case study in extracting geometry knowledge from textbooks. *Comput. Linguist.* **2020**, *45*, 627–665.
22. Yu, W.; Wang, M.; Wang, X.; Zhou, X.; Zha, Y.; Zhang, Y.; Miao, S.; Liu, J. Geore: A relation extraction dataset for chinese geometry problems. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Workshop on Math AI for Education (MATHAI4ED)*, Online, 6–14 December 2021.
23. Huang, L.; Yu, X.; He, B. A novel geometry problem understanding method based on uniform vectorized syntax-semantics model. In *Proceedings of the 2022 International Conference on Intelligent Education and Intelligent Research (IEIR)*, Wuhan, China, 18–20 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 78–85.
24. Zhou, W.; Xu, R.; Guan, H.; Zhao, J.; Rao, Y. Research on geometry problem text understanding based on bidirectional LSTM-CRF. In *Proceedings of the 2022 9th International Conference on Digital Home (ICDH)*, Guangzhou, China, 28–30 October 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 121–127.
25. Seo, M.J.; Hajishirzi, H.; Farhadi, A.; Etzioni, O. Diagram understanding in geometry questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Quebec, QC, Canada, 27–31 July 2014; Volume 28.
26. Zhang, M.L.; Yin, F.; Hao, Y.H.; Liu, C.L. Plane geometry diagram parsing. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Vienna, Austria, 23–29 July 2022; Raedt, L.D., Ed.; International Joint Conferences on Artificial Intelligence Organization: San Francisco, CA, USA, 2022; pp. 1636–1643. <https://doi.org/10.24963/ijcai.2022/228>.
27. Wong, M.F.; Qi, X.; Tan, C.W. EuclidNet: Deep visual reasoning for constructible problems in geometry. In *Proceedings of the 2nd MATH-AI Workshop at NeurIPS'22: Toward Human-Level Mathematical Reasoning*, New Orleans, LA, USA, 3 December 2022.
28. Yu, X.; Wang, M.; Gan, W.; He, B.; Ye, N. A framework for solving explicit arithmetic word problems and proving plane geometry theorems. *Int. J. Pattern Recognit. Artif. Intell.* **2019**, *33*, 1940005.
29. Gan, W.; Yu, X.; Zhang, T.; Wang, M. Automatically proving plane geometry theorems stated by text and diagram. *Int. J. Pattern Recognit. Artif. Intell.* **2019**, *33*, 1940003.
30. Kovács, Z.; Yu, J.H. Automated discovery of geometrical theorems in geoGebra. *arXiv* **2022**, arXiv:2202.04627.
31. Seo, M.; Hajishirzi, H.; Farhadi, A.; Etzioni, O.; Malcolm, C. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 17–21 September 2015; pp. 1466–1476.
32. Zhong, X.; Fu, H.; Yu, Y.; Liu, Y. Interactive learning environment based on knowledge network of geometry problems. In *Proceedings of the 2015 10th International Conference on Computer Science & Education (ICCSE)*, Cambridge, UK, 22–24 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 53–58.
33. Alvin, C.; Gulwani, S.; Majumdar, R.; Mukhopadhyay, S. Synthesis of geometry proof problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Quebec, QC, Canada, 27–31 July 2014; Volume 28.
34. Alvin, C.; Gulwani, S.; Majumdar, R.; Mukhopadhyay, S. Synthesis of solutions for shaded area geometry problems. In *Proceedings of the Thirtieth International Flairs Conference*, Marco Island, FL, USA, 22–24 May 2017.
35. Sachan, M.; Xing, E. Learning to solve geometry problems from natural language demonstrations in textbooks. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (SEM 2017)*, Vancouver, BC, Canada, 3–4 August 2017; pp. 251–261.
36. Yu, X.; Gan, W.; Wang, M. Understanding explicit arithmetic word problems and explicit plane geometry problems using syntax-semantics models. In *Proceedings of the 2017 International Conference on Asian Language Processing (IALP)*, Singapore, 5–7 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 247–251.
37. Buchberger, B. Applications of Gröbner bases in non-linear computational geometry. In *Mathematical Aspects of Scientific Software*; Springer: New York, NY, USA, 1988; pp. 59–87.
38. Yang, L.; Zhang, J.; Li, C. A prover for parallel numerical verification of a class of constructive geometry theorems. In *Proceedings of the IWMM*, St. Malo, France, 17–19 September 1992; Volume 92, pp. 244–250.
39. Gao, X.S.; Chou, S.C. On the dimension of an arbitrary ascending chain. *Chin. Sci.-Bull.-Engl. Ed.* **1993**, *38*, 799–799.
40. Collins, G.E. Quantifier elimination for real closed fields by cylindrical algebraic decomposition—preliminary report. *ACM SIGSAM Bull.* **1974**, *8*, 80–90.



41. Lu, Y. Practical automated reasoning on inequalities: Generic programs for inequality proving and discovering. In Proceedings of the Third Asian Technology Conference in Mathematics, Tsukuba, Japan, 24–28 August 1998; pp. 24–35.
42. Wang, D. GEOTHER: A geometry theorem prover. In *Automated Deduction—CADE-13: Proceedings of the 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, 30 July–3 August 1996*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 166–170.
43. Chou, S.C.; Gao, X.S.; Zhang, J.Z. Automated geometry theorem proving by vector calculation. In Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, Kiev, Ukraine, 6–8 July 1993; pp. 284–291.
44. Chou, S.C.; Gao, X.S.; Zhang, J.Z. *A Collection of 110 Geometry Theorems and Their Machine Produced Proofs Using Full-Angles*; Washington State University: Washington, WA, USA, 1994.
45. Li, H. Symbolic computation in the homogeneous geometric model with Clifford algebra. In Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain, 4–7 July 2004; pp. 221–228.
46. Wilson, S.; Fleuriot, J.D. Geometry Explorer: A tool for generating diagrammatic full-angle method proofs. In *Automated Deduction in Geometry: Extended Abstracts*; Dialnet: Universidad de La Rioja, Spain, 2006; pp. 144–150.
47. Chou, S.C.; Gao, X.S.; Zhang, J.Z. An introduction to geometry expert. In Proceedings of the CADE, New Brunswick, NJ, USA, 30 July–3 August 1996; Volume 1104, pp. 235–239.
48. Ye, Z.; Chou, S.C.; Gao, X.S. An introduction to java geometry expert. In *Automated Deduction in Geometry: Proceedings of the 7th International Workshop, ADG 2008, Shanghai, China, 22–24 September 2008*; Revised Papers 7; Springer: Berlin/Heidelberg, Germany, 2011; pp. 189–195.
49. Chou, S.C.; Gao, X.S.; Zhang, J.Z. Automated production of traditional proofs in solid geometry. *J. Autom. Reason.* **1995**, *14*, 257–291.
50. Yang, L.; Gao, X.S.; Chou, S.C.; Zhang, J.Z. Automated production of readable proofs for theorems in non-Euclidean geometries. In *Automated Deduction in Geometry: Proceedings of the International Workshop on Automated Deduction in Geometry, Toulouse, France, 27–29 September 1996*; Selected Papers 1; Springer: Berlin/Heidelberg, Germany, 1997; pp. 171–188.
51. Tsai, S.h.; Liang, C.C.; Wang, H.M.; Su, K.Y. Sequence to general tree: Knowledge-guided geometry word problem solving. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), Virtual, 1–6 August 2021; pp. 964–972.
52. Hao, Y.; Zhang, M.; Yin, F.; Huang, L.L. PGDP5K: A diagram parsing dataset for plane geometry problems. In Proceedings of the 2022 26th International Conference on Pattern Recognition (ICPR), Montreal, QC, Canada, 21–25 August 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1763–1769.
53. Cao, J.; Xiao, J. An augmented benchmark dataset for geometric question answering through dual parallel text encoding. In Proceedings of the 29th International Conference on Computational Linguistics, Gyeongju, Republic of Korea, 12–17 October 2022; pp. 1511–1520.
54. Chen, J.; Li, T.; Qin, J.; Lu, P.; Lin, L.; Chen, C.; Liang, X. UniGeo: Unifying geometry logical reasoning via reformulating mathematical expression. *arXiv* **2022**, arXiv:2212.02746.
55. Zhang, M.L.; Yin, F.; Liu, C.L. A multi-modal neural geometric solver with textual clauses parsed from diagram. *arXiv* **2023**, arXiv:2302.11097.
56. Jian, P.; Guo, F.; Wang, Y.; Li, Y. Solving geometry problems via feature learning and contrastive learning of multimodal data. *CMES-Comput. Model. Eng. Sci.* **2023**, *136*, 1707–1728.
57. Ning, M.; Wang, Q.F.; Huang, K.; Huang, X. A symbolic character-aware model for solving geometry problems. *arXiv* **2023**, arXiv:2308.02823.
58. Zhang, M.L.; Li, Z.Z.; Yin, F.; Liu, C.L. LANS: A layout-aware neural solver for plane geometry problem. *arXiv* **2023**, arXiv:2311.16476.
59. Graves, A.; Wayne, G.; Danihelka, I. Neural turing machines. *arXiv* **2014**, arXiv:1410.5401.
60. Weston, J.; Chopra, S.; Bordes, A. Memory networks. *arXiv* **2014**, arXiv:1410.3916.
61. Neelakantan, A.; Le, Q.V.; Sutskever, I. Neural programmer: Inducing latent programs with gradient descent. In Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016.
62. Rocktäschel, T.; Riedel, S. End-to-end differentiable proving. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 3791–3803.
63. Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; De Raedt, L. Deepproblog: Neural Probabilistic Logic Programming. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–14.
64. Badreddine, S.; Garcez, A.d.; Serafini, L.; Spranger, M. Logic tensor networks. *Artif. Intell.* **2022**, *303*, 103649.
65. Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; Zhou, D. Neural logic machines. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
66. Minervini, P.; Riedel, S.; Stenetorp, P.; Grefenstette, E.; Rocktäschel, T. Learning reasoning strategies in end-to-end differentiable proving. In Proceedings of the 37th International Conference on Machine Learning, Virtual, 13–18 July 2020; Volume 119, pp. 6938–6949.
67. Sadeghian, A.; Armandpour, M.; Ding, P.; Wang, D.Z. Drum: End-to-end differentiable rule mining on knowledge graphs. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 15347–15357.
68. Qu, M.; Chen, J.; Xhonneux, L.P.; Bengio, Y.; Tang, J. RNNLogic: Learning logic rules for reasoning on knowledge graphs. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.

69. Polu, S.; Han, J.M.; Zheng, K.; Baksys, M.; Babuschkin, I.; Sutskever, I. Formal mathematics statement curriculum learning. In Proceedings of the Eleventh International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2022.
70. Jiang, A.Q.; Li, W.; Tworkowski, S.; Czechowski, K.; Odrzygóźdź, T.; Miłoś, P.; Wu, Y.; Jamnik, M. Thor: Wielding hammers to integrate language models and automated theorem provers. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 8360–8373.
71. Lample, G.; Lacroix, T.; Lachaux, M.A.; Rodriguez, A.; Hayat, A.; Lavril, T.; Ebner, G.; Martinet, X. Hypertree proof search for neural theorem proving. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 26337–26349.
72. Zheng, K.; Han, J.M.; Polu, S. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In Proceedings of the International Conference on Learning Representations, Virtual, 3–7 May 2021.
73. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. GPT-4 technical report. *arXiv* **2023**, arXiv:2303.08774.
74. Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. 2024. Available online: <https://www.anthropic.com> (accessed on 24 March 2024).
75. Huang, J.; Chang, K.C.C. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023*; Association for Computational Linguistics: Toronto, ON, Canada, 2023; pp. 1049–1065.
76. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D.; et al. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.