FormalGeo: The First Step Toward Human-like IMO-level Geometric Automated Reasoning

Xiaokai Zhang[1], Na Zhu[1,2], Yiming He[1,2], Jia Zou[1,2], Qike Huang[1,2], Xiaoxiao Jin[1,2], Yanjun Guo[1,2], Chenyang Mao[1,2], Zhe Zhu[1,2], Dengfeng Yue[1,2], Fangzhen Zhu[1], Yang Li[1], Yifan Wang[1,2], Yiwen Huang[1], Runan Wang[1,2], Cheng Qin[1,2], Zhenbing Zeng[3], Shaorong Xie[1], Xiangfeng Luo[1], and Tuo Leng[1,2,*]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, China
[2] Institute of Artificial Intelligence, Shanghai University, Shanghai, China
[3] College of Sciences, Shanghai University, Shanghai, China

Abstract. This is the first paper in a series of work we have accomplished over the past three years. In this paper, we have constructed a complete and compatible formal plane geometry system. This will serve as a crucial bridge between IMO-level plane geometry challenges and readable AI automated reasoning. Within this formal framework, we have been able to seamlessly integrate modern AI models with our formal system. AI is now capable of providing deductive reasoning solutions to IMO-level plane geometry problems, just like handling other natural languages, and these proofs are readable, traceable, and verifiable. We propose the *geometry formalization theory* (GFT) to guide the development of the geometry formal system. Based on the GFT, we have established the *FormalGeo*, which consists of 88 geometric predicates and 196 theorems. It can represent, validate, and solve IMO-level geometry problems. we also have crafted the *FGPS* (formal geometry problem solver) in Python. It serves as both an interactive assistant for verifying problem-solving processes and an automated problem solver. We've annotated the *formalgeo7k* and *formalgeo-imo* datasets. The former contains 6,891 (expand to 133,818 through data augmentation) geometry problems, while the latter includes 18 (expand to 2,627 and continuously increasing) IMO-level challenging geometry problems. All annotated problems include detailed formal language descriptions and solutions. *AI* can be integrated into formal systems in various roles. It can act as a parser, enabling the autoformalization of natural language and geometric diagrams. It can also serve as a solver, ruuning search tree pruning. Implementation of the formal system and experiments validate the correctness and utility of the GFT. The backward depth-first search method only yields a 2.42% problem-solving failure rate on formalgeo7k. We can incorporate deep learning techniques to achieve lower one. The source code of FGPS and datasets are available here.

Keywords: Formal mathematics · Human-like automated reasoning · IMO-level Geometry problem solving

*Corresponding author: Email - tleng@shu.edu.cn

1 Introduction

Since the inception of mathematics, it has inherently encompassed both structure and computation. These two facets not only interact with each other but also mutually reinforce each other. With the advent of modern computing, it has started to exert its influence on mathematics in two distinct ways. On one hand, it serves as a mathematical tool, empowering mathematical computations more than ever before, thereby directly impacting the balance of values and methodologies within mathematics. On the other hand, as a mathematical medium, it indirectly reshapes the content and structure of mathematics through innovative applications, ushering in an unprecedented era of prosperity and development for the field.

Within this transformative landscape, the field of mathematical mechanization emerged, situated at the intersection of mathematics and computer science. The translation and conversion of mathematical knowledge into a language comprehensible to computers are evidently the first and indispensable steps in the fusion of mathematics and computer science. This is the essence of formal mathematics.

Formal mathematics serves as the foundation for computer-aided mathematical problem-solving. It employs a symbol system that adheres to a particular artificial grammar, enabling the representation of any concept, proposition, and inference. It is only when mathematical knowledge is rigorously formalized that the problem-solving process in mathematics can be described as a deterministic algorithm and implemented within a computer. Over the course of several decades, numerous formal mathematics systems and tools have emerged, such as Lean, Isabelle and Coq.

The development of artificial intelligence (AI) has introduced a new paradigm for computer-aided mathematical problem-solving. AI-assisted mathematical problem solving is a rapidly developing area which aims to apply deep learning technology to math problem. AI systems can assume various roles within formal mathematical systems. They can serve as mathematical problem solvers [57,34,22,26,23], automatically generating problem solutions. They can act as mathematical problem parsers [47,12,55], assisting humans in converting mathematical knowledge into formal descriptions. Moreover, they can even take on the role of mathematical problem proposers [14,35,31], suggesting mathematical conjectures.

A complete mathematical formal system is an indispensable component of AI-assisted mathematical problem-solving. On the one hand, a unified format for mathematical problem descriptions and datasets helps in avoiding interference from other factors and serves as a standard for evaluating the capabilities of AI models. On the other hand, the answers generated by AI models must be verified for credibility, and manual verification is both inefficient and error-prone. This necessitates the capability for computers to automatically validate the answers to mathematical problems. The Stanford 2021 AI100 report [28] designates the IMO grand challenge [13] as a landmark event in the development of artificial intelligence: To create an AI system capable of receiving a formal problem rep-

resentation, generating a formal (i.e., machine-checkable) proof for the problem, and attaining a gold medal in the International Mathematical Olympiad. Serving as a bridge that connects the two crucial research domains of AI and mathematics, formal mathematics has garnered increasing attention from researchers.

The intersection and fusion of AI and formal mathematics have yielded a series of achievements. However, as an essential branch of formalized mathematics, the formalization and mechanized solving of geometric problems have still made slow progress. As depicted in Fig. 1, this domain has long been constrained by three major challenges: inconsistent knowledge form, unreadable solving process and non-mechanized solving method.

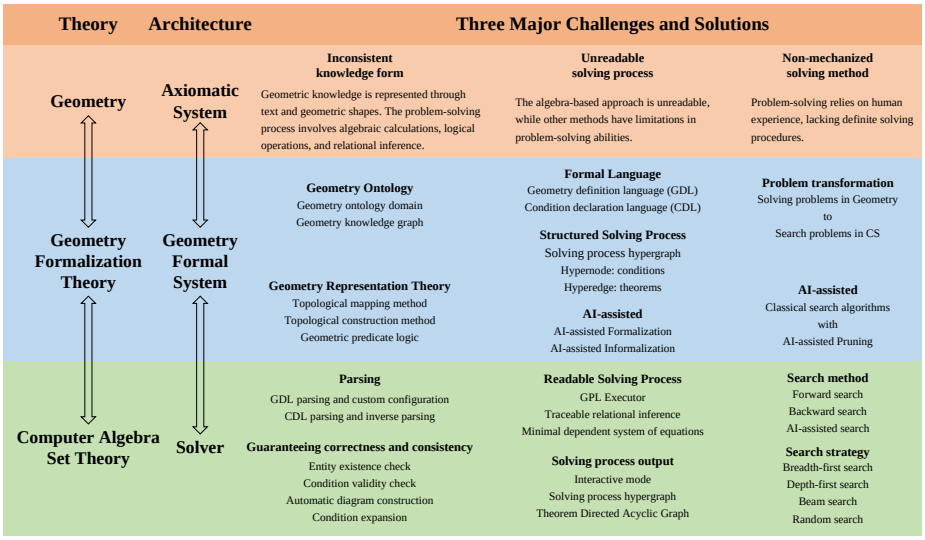


Fig. 1. Three major challenges and solutions in formal plane geometry.

Addressing the first challenge, we introduced the geometry formalization theory (GFT) to unify the representation of geometric knowledge, including geometry ontology and geometry representation theory. geometry ontology provides a comprehensive overview of plane geometry from a highly abstract philosophical perspective, creating the geometry ontology domain. The geometry ontology domain consists of four quadrants based on the dimensions of number-shape and dynamic-static. Each quadrant maps the relationships between modern geometric axiom systems, geometry formal systems, and solvers across three hierarchical levels. Leveraging the geometry ontology domain, we can ensure the comprehensiveness of formal system design. Geometry representation theory investigates how to represent static geometric knowledge and dynamic problem-solving processes. Based on topological mapping method and topological combination

method, we can transform geometric diagrams into textual or symbolic descriptions. Using geometry predicate logic, we can unify relation reasoning, logical operations, and algebraic calculations into rigorous formal representations.

Addressing the second challenge, we designed a set of geometry formal languages to serve as a bridge for communication between humans and computers. geometry formal languages consist of geometry definition language and condition declaration language, where the former is used to personalize the configuration of solvers and import into the formal system, and the latter is used for inputting problem descriptions. Geometry formal languages possess rigorous syntactic descriptions that can be mechanically processed by computers, while their syntax is similar to predicate logic, providing good readability. We abstract the process of geometry problem-solving as a hyper tree, where tree nodes represent known conditions, tree edges represent geometric theorems, and the problem-solving process is a path from the root node to the target leaf node. We can also utilize modern AI techniques to automatically translate between natural language and formal language, further enhancing readability.

Addressing the third challenge, we transformed the mathematical problems of geometric problem-solving into computational search problems in the field of computer science, achieving both forward search and backward search. Forward search starts with known conditions and continuously applies theorems to obtain new conditions until the target condition is achieved. Backward search starts from the problem goal, expands the goal into multiple subgoals according to theorems, and repeats this process until all new subgoals are known conditions. The vast search space gives rise to the problem of combinatorial explosion, with the search time for difficult problems exhibiting exponential growth. Pruning of the search tree is essential. In addition to classical pruning techniques such as Monte Carlo Tree Search and Alpha-Beta pruning, we leverage deep learning as a powerful pruning method to accelerate the solving process.

Our contributions can be summarized as follows:

1.We propose the GFT, which comprises geometry ontology and geometry representation theory. This theory provides a comprehensive framework for the field of plane geometry, unifying the representation of symbolic geometric knowledge and graphic geometric knowledge. It encompasses various operations, including relation reasoning, logical operations, and algebraic calculations, within a unified representation framework.

2.We have established the *FormalGeo* system based on the GFT, which consists of 88 geometric predicates and 196 theorems. It can represent, validate, and solve geometry problems from STA-level to IMO-level.

3.We have crafted the formal geometry problem solver (FGPS) in Python. It serves as both an interactive assistant for verifying problem-solving processes and an automated problem solver, utilizing various methods such as forward search, backward search and AI-assisted search and various strategy such as depth-first search, breadth-first search, random search and beam search. FGPS incorporates features such as formal statement parsing, condition validity checks, automatic

diagram construction, and condition expansion. It is capable of executing backtrackable, interpretable algebraic equation solving and relational reasoning.

4.We've annotated the formalgeo7k and formalgeo-imo datasets. The former contains 6,891 (expand to 133,818) geometry problems, while the latter includes 18 (expand to 2,627 and continuously increasing) IMO-level challenging geometry problems. Each problem comprises a complete natural language description, geometric shapes, formal language annotations, and theorem sequences annotations.

5.We conducted experiments on the formalgeo7k, comparing two search-based problem-solving methods (forward and backward) and 4 search strategies (breadth-first, depth-first, random, beam) in terms of their success rate, time-consuming, and search step. The forward random search method yields a 39.7% problem-solving accuracy rate, and we can incorporate deep learning techniques to achieve higher one.

2 Related Work

Gelernter et al. developed the pioneering automated geometry problem-solving system known as the Geometry Theorem Prover [18], which employed a backward search approach to solve pre-formalized problems. Nevins pointed out that the forward chaining method [nevins1975plane] 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 [15,36],geometric knowledge extraction [38,37,51,20,59], geometric diagram parsing [40,55,45], geometric theorem proving [53,16,25], and geometry problem solving [41,58,1,2,39,52].

Wen-Tsun proposed the Wu's Method [46], which transforms geometry problem into a system of algebraic equations consisting of polynomials and inequalities and leverages various algebraic techniques to solve these equations. The study of algebraic approaches to geometry problems has given rise to a range of research achievements, such as Buchberger's Gröbner bases method [3], numerical parallel methods [48], polynomial system triangulation elimination algorithm [17], cylindrical algebraic decomposition for solving inequalities [11], dimensionality reduction methods [30], and software tools like GEOTHER [43].

Zhang proposed the point elimination method based on geometric invariants [54]. 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 [7,8,27], leading to the development of practical software tools such as Geometry Explorer [44], Geometry Expert [10] and Java Geometry Expert [50]. The method based on geometric invariant can also be extended to solid geometry [9] and non-Euclidean geometry [49].

The machine proof of geometric theorems can generally be categorized into the three aforementioned approaches [24]: search-based synthesis methods, algebra methods, and points elimination methods based on geometric invariants. Synthesis method can provide proofs of traditional style but can only prove a small subset of carefully chosen plane geometry theorems due to the limited computational power of computers and the combinatorial explosion inherent in these methods. Algebra methods can handle a wide range of problem types, but the solving process typically involves algebraic expressions that are difficult to manually verify and obtain readable proof procedures. Methods based on geometric invariants can provide readable proof procedures, but the types of problems that can be solved are limited by the types of geometric invariants available.

Geometry problem solving has been gaining more attention in the NLP community recently. Several geometry formal systems and datasets have been constructed, such as Geometry3K [29], GeoQA [6], GeometryQA [42]. Geometry3K 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 [51], which focuses on geometric relation extraction, and PGDP5k [19], 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 employs the formal method of Program, transforming the geometric problem-solving process 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+ [4], UniGeo [5], and PGPS9K [56]. 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 geometric problem-solving processes. Additionally, adding new rules requires modifying the solver's code, making them less extensible. GeometryQA 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 programmatic approach but is more structured.

Shared benchmarks and datasets have significantly advanced research in AI-assisted geometric problem solving. Several AI systems, such as CL-based model [21], SCA [32], GeoDRL [33], have been constructed to achieve higher success rates in problem solving. As problem-solving success rates continue to improve, there is a growing demand for datasets with higher quality and difficulty. Previous work focused on AI system research but overlooked research into geometry formalization theory. Expanding datasets in existing systems requires substantial modifications to solver code, making it challenging to extend both the formal systems and datasets.

3 Geometry Formalization Theory

In the realm of geometry, a typical problem consists of known conditions described in natural language, the problem objective, and a geometric diagram. The problem-solving process can be construed as the application of multiple theorems, involving relational reasoning and algebraic computation. The study of GFT focuses on how to transform geometric problems and their solution processes, described in natural language and images, into a unified and precise formal language for mechanical processing by computers. GFT comprises 3 major components: Geometry Ontology, Geometry Representation Theory, and Geometry Formal Language.

Geometry Ontology studies the fundamental ontology within the field of geometry and the relationships between these ontological elements. It employs the Geometry Ontology Domain, which provides a comprehensive and systematic summary of the knowledge of Euclidean plane geometry. The Geometry Ontology Domain further refines into the Geometry Knowledge Graph, guiding the design of geometry formal systems.

Geometry Representation Theory investigates how to express geometric knowledge using formal language. Formal systems are abstract descriptions and simulations of the real world, with a one-to-one correspondence to the real world. Consistency theory explores how to establish a correct formal system. Under the guidance of consistency theory, we propose Geometry Representation Theory. When transforming various geometric knowledge into a unified formal language, we must ensure the consistency of static representations (such as the formal representation of diagrams) and the consistency of dynamic processes (such as theorems described in formal language).

Geometry Formal Language takes the form of structured geometric knowledge, divided into two categories: Geometry Definition Language (GDL) and Condition Declaration Language (CDL). GDL includes Predicate Definition Language and Theorem Definition Language. The former defines various types of geometric relations and properties, while the latter defines various theorems that may be used in the problem-solving process. During the initialization phase of a geometric problem solver, GDL is used to configure the solver, achieving its shareability and extensibility. CDL is employed to describe the known conditions of individual geometric problems and is divided into three categories: construction statements, condition statements, and objective statements. CDL allows us to input geometric problems into the solver.

3.1 Geometry Ontology

Geometry Ontology Domain consists of two dimensions: Number and Shape, and Static and Dynamic. Number refers to precise and quantitative descriptions of geometric knowledge, while Shape pertains to generalized and qualitative descriptions of geometric knowledge. Static refers to various geometric knowledge elements, such as the properties of geometric figures and their interrelationships.

Dynamic encompasses rules for transforming different types of geometric knowledge, including common knowledge and theorems.

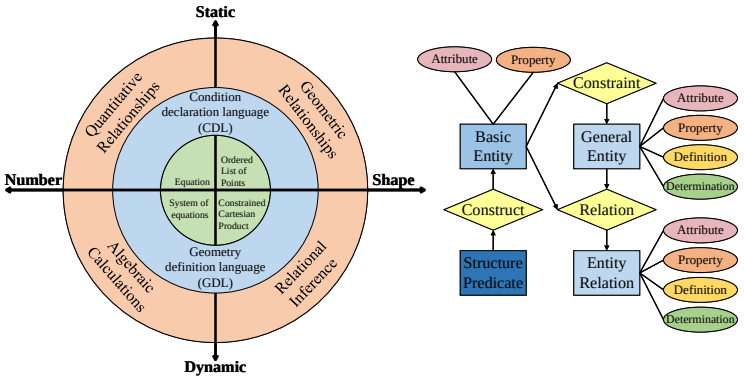


Fig. 2. Geometry Ontology Domain (left). It consists of two dimensions: Number and Shape, and Static and Dynamic and further subdivided into three levels: Axiomatic Systems, Formal Systems, and Solvers. Simplified Geometry knowledge graph (right). Rectangles represent static geometric knowledge, while circles represent dynamic processes of transforming geometric knowledge. An example can be found in App. A.2.

These two dimensions divide the Geometry Ontology Domain into four quadrants, each further subdivided into three levels of mapping relationships: Axiomatic Systems, Formal Systems, and Solvers, as shown in Fig. 2. The outermost layer, axiomatic systems, refers to geometric knowledge described in natural language or diagrams, such as problem conditions and theorem definitions. The intermediate layer, Formal Systems, transforms vague and uncertain natural language descriptions and knowledge in different forms from text and diagrams into precise, human-readable, and computable formal languages, serving as a bridge between humans and computers. The innermost layer, Solvers, represents the specific internal form of geometric knowledge within a computer, including data structures for problem conditions and applying methods for theorems. App. A.1 provides examples for various components of the Geometry Ontology Domain to facilitate understanding.

To further extend domain of geometry ontology domain, incorporating object-oriented thinking, and utilizing entity-relationship diagrams, we construct geometry knowledge graph as shown in Fig. 2. Construction statements contain all the structural information of geometric figures. Starting from three types of construction statements, we derive all basic entities. By imposing further constraints on basic entities, we obtain general entities. Basic entities and general entities interact internally and with each other, forming entity relationships. Attributes represent quantifiable descriptions of geometric objects. Construction statements, basic entities, general entities, entity relationships, and attributes describe the static aspects of geometric knowledge. Properties, definitions, and

judgments describe the dynamic processes of transforming geometric knowledge, i.e., theorems. The knowledge graph of geometry provides a detailed representation of the relationships and hierarchical structure among various geometric knowledge components. Using the knowledge graph of geometry ensures that the constructed formal system is more comprehensive and avoids omissions.

3.2 Geometry Representation Theory

App. B introduces the consistency theory of formal systems. Guided by this theory, when constructing a geometry formal system, we must ensure the consistency of static representation and the consistency of dynamic processes.

Consistency in Static Representation In the field of geometry, various geometric knowledge is conveyed through textual descriptions or geometric diagrams. Geometric knowledge in textual form is presented using natural language and mathematical symbols, making it relatively straightforward to transform into structured formal language. However, the challenge lies in formalizing the geometric knowledge implicit in geometric diagrams, which necessitates establishing a reversible mapping between geometric diagrams and their formal representations.

We classify the information inherent in geometric diagrams into two categories: topological structure information (TSI) and metric information (MI). TSI defines the fundamental structure of a diagram and serves as a crucial basis for classifying different geometric shapes. MI further characterizes various properties of the diagram, such as the length of lines and the measure of angles. MI is closely related to TSI and is relatively amenable to summarization and formalization. The primary challenge lies in formalizing the TSI.

The most fundamental elements that compose a geometric diagram are points, which can be used to describe the TSI of the diagram. We can define a set of rules that utilize the points constituting the diagram to depict its TSI.

Closed geometric figures can be transformed into topologically equivalent circles, as shown in Fig. 3. Unfolding the circle from any position of the topologically equivalent circle into a straight line, the relative positions of points on the line record the topological structural information of the original diagram. We can use an ordered list of points as the formal representation of the geometric topological structural information. For non-closed geometric figures, we can first transform them into closed geometric diagrams before proceeding with formalization, as shown in Example.4. of Fig. 3. By unfolding the topologically equivalent circle from different positions, we may obtain different ordered lists of points. All these ordered lists together form a set, serving as the formal representation of the topological structural information of geometric diagram. Any element in the set contains all the topological structural information of the original diagram.

The basic transformations of a geometric diagram can be defined as several operations on its formal representation, as shown in App. C.

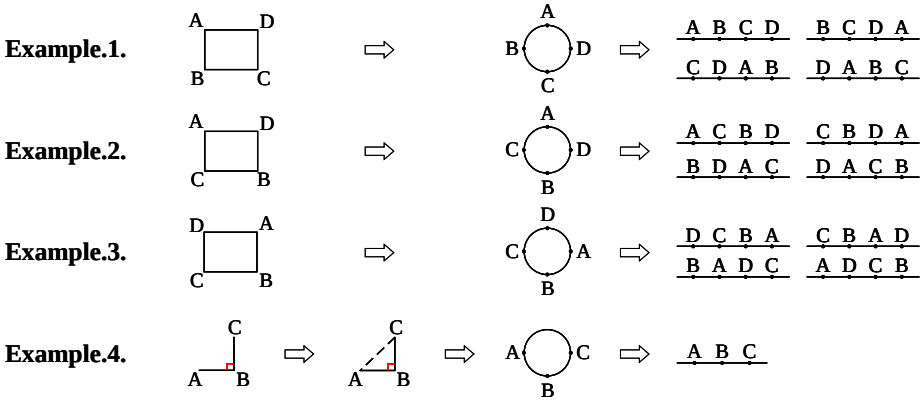


Fig. 3. Examples of topological mapping method

The above method is called the topological mapping method, which can be used for the formalization of a simple geometric diagram. In practice, a geometric problem's diagram is often composed of a combination of simple geometric diagrams. We propose the topological construction method, which utilizes the formal representations of multiple simple geometric diagrams to obtain the formal representation of a composite diagram.

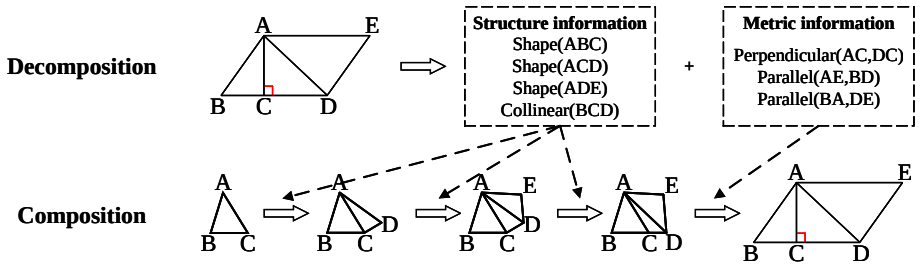


Fig. 4. An example of topological construction method

The topological construction method decomposes the composite diagram into simple geometric diagrams while preserving the TSI between them, TSI of each simple geometric diagram and MI of each simple geometric diagram. When constructing a diagram, it first reconstructs the basic structure of the original diagram based on the TSI and then adjusts the diagram further according to the MI to obtain the original diagram. The topological construction method is a constructive drawing method that is not affected by the order of construction statements, as shown in Fig. 4. We have implemented topological construction

method in FGPS, and the algorithm description and time complexity analysis can be found in App. D.

The process of constructing a geometric diagram can be denoted as $\oplus$. If diagram C is composed of diagrams A and B, their formal representations of TSI are represented as sets R_a, R_b, and R_c, respectively. Diagram A contains m points, while diagram B contains n points. There are k ($k \geq 2$)common points shared between A and B. Two elements $P_a = (p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i, \ldots, p_{i+k}, p_{s_2}^{(a)}, \ldots, p_m^{(a)})$ and $P_a = (p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+k}, \ldots, p_i, p_{s_2}^{(b)}, \ldots, p_n^{(b)})$ are selected from sets R_A and R_B. $\oplus$ is defined in two steps, as shown in Eq. 1 and Eq. 2. In the first step, $\otimes$ operation is applied to P_a and P_b to obtain the element P_c. In the second step, the P_c undergoes the *rotate* (Eq.22) i times to construct the set R_c.

$$\begin{aligned} & P_a \otimes P_b \\ =&(p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i, p_{s_2}^{(b)}, \ldots, p_n^{(b)}, p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+k}) \end{aligned} \tag{1}$$

$$\begin{aligned} & R_a \oplus R_b \\ =&\{rotate^{(i)}(P_a \otimes P_b) | i = 1, 2, \ldots, m + n - 2k + 2\} \\ =&R_c \end{aligned} \tag{2}$$

The formal representation of composite diagrams, along with $\oplus$, forms a semigroup, satisfying closure, commutative law and associative law. The proof can be found in App. D.

Consistency in Dynamic Process In essence, geometric theorems are rules governing the transformation of various geometric knowledge, involving relation reasoning, logical operations, and algebraic calculations. Geometric predicate logic (GPL) translates the diverse operations inherent in geometric theorems into a unified formal language, enabling precise theorem descriptions that can be mechanically executed by computers.

Geometric knowledge comprises geometric relations and quantitative relations, with geometric relations as the core, and quantitative relations are attributes of geometric relations. Multiple geometric relations can lead to new relations through relation reasoning, where elements in these new relations can be derived through operations on the elements of existing relations.

GPL categorizes operations in geometry into external relation reasoning and internal relation reasoning, as illustrated in Fig. 5. If the operation results in a new relation with a different structure, it is termed external relation reasoning, also known as relation composition. Confining a relation with constraints to obtain a stricter relation while preserving the original relation's structure constitutes internal relation reasoning. Depending on the type of constraint applied, this can be further categorized into geometric constraints and algebraic constraints on relations. Relation reasoning is subdivided into basic operations &, |, and $\sim$, which correspond to logical operations AND, OR, and NOT, respectively.

Geometric relations are denoted as $R(v_1, v_2, \ldots, v_n), v \in V$, where R is referred to as the relation name, specifying the type of geometric relation, and

Relational inference	**Relation Composition** (**External** relational inference)		**Geometric constraints** (**Internal** relational inference)			**Algebraic constraints** (**Internal** relational inference)		
GPL Conjunction Words	**&**	|	**&**	|	~	**&**	|	~
Definition	Constrained Cartesian Product	Union	Constrained Cartesian Product	Union	Difference	Algebraic operations with Constrained Cartesian Product	Union	Difference

Fig. 5. Definition of geometric predicate logic

V represents point variables, describing the TSI of the geometric relation. A geometric relation with N elements is represented as a set $R(v_1, v_2, \ldots, v_n) = \{(p_1^{(i)}, p_2^{(i)}, \ldots, p_n^{(i)}) | i = 1, 2, \ldots, N\}$, where p are the points constituting the geometric relation. For any element $r_i = (p_1^{(i)}, p_2^{(i)}, \ldots, p_n^{(i)})$ of a geometric relation, we define $r_i(v_j)$ as the value of the position v_j of element r_i, i.e., $r_i(v_j) = p_j$. Quantity relations can be represented by algebraic constraints denoted as R_A, where $R_A(r) = 1$ indicates that an element satisfies the constraint. Given relations $R_1(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)})$ and $R_2(v_1^{(2)}, v_2^{(2)}, \ldots, v_m^{(2)})$, they have k common variables and their point variables are denoted as V_1 and V_2. We can obtain a new relation R_3 through GPL.

& is referred to as the constrained Cartesian product and is analogous to the logical operation AND. The operation is denoted as $R_1 \& R_2 \rightarrow R_3$. Initially, the Cartesian product operation is applied to R_1 and R_2, yielding $R_3^{'}$. Constraints are imposed on $R_3^{'}$ to select elements that adhere to the constraint conditions. These constraints necessitate that the elements within $R_3^{'}$ exhibit identical values at the common variable positions of R_1 and R_2, as shown in Eq. 3, where $\times$ represents the Cartesian product. Then, duplicate common variables are eliminated, leading to the derivation of a new geometric relation R_3, as shown in Eq. 4.

$$\begin{aligned} & R_3^{'}(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}, v_1^{(2)}, v_2^{(2)}, \ldots, v_m^{(2)}) \\ =& \{(r_i^1, r_j^2) | (r_i^1, r_j^2) \in R_1 \times R_2, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2\} \end{aligned} \tag{3}$$

$$\begin{aligned} & R_1(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}) \& R_2(v_1^{(2)}, v_2^{(2)}, \ldots, v_m^{(2)}) \\ =& \{(r(v_1^{(3)}), r(v_2^{(3)}), \ldots, r(v_{m+n-k}^{(3)})) | r \in R_3^{'}, v^{(3)} \in V_1 \cup V_2\} \\ =& R_3(v_1^{(3)}, v_2^{(3)}, \ldots, v_{m+n-k}^{(3)}) \end{aligned} \tag{4}$$

The definition of & is more straightforward when the second relation is a quantitative relation. The operation is denoted as $R_1 \& R_A \rightarrow R_3$. The point variables of R_A are a subset of those in R_1. We take the elements from R_1 and incorporate them into R_A corresponding to the point variables to construct algebraic constraints. The set of all elements that satisfy these algebraic constraints forms a new geometric relation R_3, as shown in Eq. 5.

$$R_3(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}) = \{r | r \in R_1, R_A(r) = 1\} \tag{5}$$

| corresponds to the logical operation OR, represented as $R_1|R_2 \to R_3$, as shown in Eq. 6. | is commonly nested together with & in practical applications.

$$R_3(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}) = \{r | r \in R_1 \cup R_2\} \tag{6}$$

$\sim$ corresponds to the logical operation NOT, represented as $\sim R_1 \to R_3$, as shown in Eq. 7. E is defined as all possible elements within a certain relation, specifically, all permutations of known points that conform to the structure of V_1.

$$R_3(v_1^{(1)}, v_2^{(1)}, \ldots, v_n^{(1)}) = \{r | r \in E - R_1\} \tag{7}$$

GPL satisfies commutative, associative, and distributive laws, as demonstrated in App. E. The nested use of GPL connectives with geometric and numeric relations provides a powerful expressive capability and can be employed for the formalization of theorems, as illustrated in App. A.4.

3.3 Geometry Formal Language

Formal languages are categorized into geometry definition language (GDL) and conditional declaration language (CDL). The former is used to define entities, attributes, theorems, and other elements of a geometry formal system, while the latter is employed for declaring known conditions and problem-solving objectives in geometric problems.

GDL comprises predicate definition language and theorem definition language. Predicate definition language is used to define different types of geometric relations and geometric properties. A typical predicate definition statement includes the name of the geometric relation, point variables, existential constraints on entities, format validity constraints, and automatic extension. The solver can read and interpret predicate definition language to ensure the legitimacy of input problem conditions.

CDL consists of three main parts: construction statements, condition statements, and goal statements. Construction statements describe the TSI of the geometric problem's diagram, such as basic shapes, collinear, and cocircular. Condition statements are used to input the known conditions in geometric problems, including both geometric and algebraic relations. goal statements declare the problem-solving objectives.

Both types of formal languages share the same syntax format, which is similar to predicate logic syntax. The fundamental concepts are predicates and terms. Predicates are used to define categories of geometric knowledge, encompassing geometric relations and algebraic relations. Terms specify the specific content of this knowledge. If it's a geometric relation, the term is an ordered sequence of points; if it's an algebraic relation, the term is an expression composed of geometric attributes, operators, free variables, and numbers. Functions are mappings

from individual geometric relation terms to individual algebraic relation terms. Through such mappings, ordered sequences of points can be used to represent numeric relations, unifying the representation formats of geometric and numeric relations. Examples of formal language can be found in App. A.3, App. A.4, and App. A.5.

4 Geometry Formal System: FormalGeo

Guided by GFT, we have constructed the geometry formal system, FormalGeo. This formal system comprises 75 predicates and 196 theorems (see App. A.3 and App. A.4) and can represent, verify, and solve geometric problems ranging from SAT-level to IMO-level. Based on FormalGeo, we annotated the formalgeo7k and formalgeo-imo datasets.

The geometric definition language serves as the specific form of the geometry formal system. The design of the geometry formal system includes the design of the predicate definition language and the theorem definition language. This section introduces the design methodology of FormalGeo and provides an overview of formalgeo7k and formalgeo-imo datasets.

Utilizing the simplified geometric knowledge graph presented in Fig. 2 as a template, we formulate the predicate definition language and theorem definition language for FormalGeo.

4.1 Predicate Definition Language

Structure predicate are used to describe the TSI of geometric figures. FormalGeo comprises three fundamental structure predicates: Shape, Collinear, and Cocircular. In the problem-solving initialization phase, the solver executes topological construction method based on recognized structure predicate statements, automatically expanding to 6 basic entities: Point, Line, Angle, Polygon, Arc and Circle. Basic entities are a more intuitive representation of the TSI of geometric figures and serve as the core for various geometric and algebraic relations. Structure predicate and basic entity together are referred to as construction predicate, which describe the TSI of the figures.

By applying additional constraints to basic entities, general entities are obtained. Various geometric entities are interconnected, forming entity relations. Properties are used to describe the MI of entities and entity relations, in conjunction with free variables, operators, and real numbers, to establish quantitative relations. General entity, entity relation, and property are collectively referred to as custom predicate. Fig. 6 illustrates the hierarchical structure and extension relationships among various geometric predicates. The solver can automatically extend conditions based on the extension rules defined in the predicate definition language. It should be noted that the principle of extension rules is followed that construction predicate are only extended by other construction predicate, and custom predicate are only extended by other custom predicate.

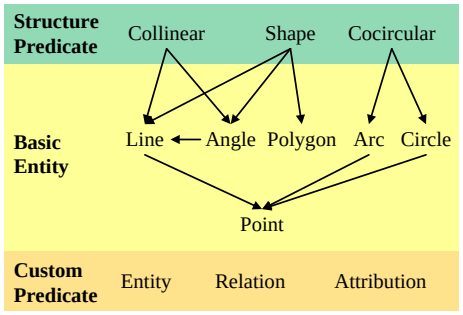


Fig. 6. Hierarchy of predicates

4.2 Theorem Definition Language

Geometric theorems in a formal system are described using geometric predicate logic, consisting of premises and conclusions. Based on the characteristics of their premises and conclusions, geometric theorems can be broadly categorized into three types: properties, definitions, and determinations. If the premises of a theorem contain only one geometric relation, the theorem falls into the category of properties or definitions. Definitions are considered common knowledge and are automatically invoked and extended by the solver, while properties require explicit invocation. If the conclusion of a theorem contains only one geometric relation, the theorem serves as a determination for that particular geometric relation.

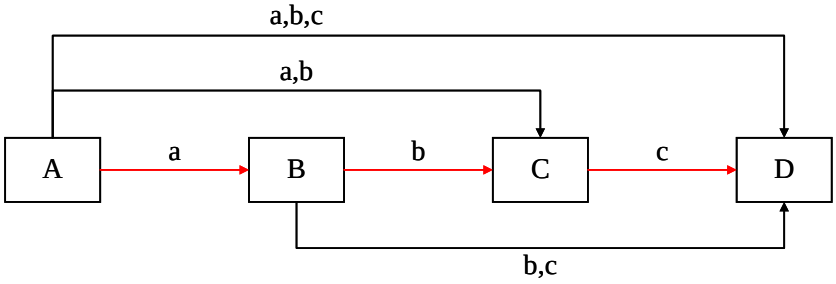


Fig. 7. Hierarchy of theorems

As previously mentioned, geometric theorems are rules for converting various geometric knowledge, and these rules exhibit a hierarchical structure, as depicted in Fig. 7, where each arrow represents a method of theorem definition. When defining theorems, we can directly combine multiple predicates to create a determination theorem, as shown in Eq. 8, or we can break it down into several theorems, as shown in Eq. 9. Different ways of defining theorems can impact

the speed of theorem application in the problem-solving process. Therefore, it is essential to explore the most suitable method for defining theorems.

$$A\&a\&b\&c \to D \tag{8}$$

$$A\&a \to B, B\&b \to C, C\&c \to D \tag{9}$$

We introduce the concept of the abstract hierarchy of theorems to describe the level of structure in theorem definitions. The abstract hierarchy of a theorem, denoted as K, represents the minimum number of theorems required to derive a higher-level predicate from lower-level predicates. Along the red paths in Fig. 7, we have $K_{A\to B} = 1$, $K_{A\to C} = 2$, and $K_{A\to D} = 3$.

Theorems are intended to facilitate the solving of problems. We prioritize theorem definition based on solving time, while temporarily disregarding other factors such as readability. For search-based problem-solving algorithms, the solving time T is influenced by the length of the theorem sequence d, the number of theorems N in theorem library, and the average application time $\bar{t}$ of a single theorem. This can be defined more specifically in Eq. 10. Here, K is directly proportional to d and inversely proportional to N and $\bar{t}$.

$$T \propto f(d, N, \bar{t}) \propto (N + N^2 + \cdots + N^d)\bar{t} \tag{10}$$

In our practical observations, we have found that there exists an approximate inverse relationship between T and K. In the process of defining theorems, FormalGeo tends to favor higher abstract hierarchy. We leave more detailed comparative experiments and mechanistic analyses for future work.

4.3 Datasets

Most of the existing datasets for geometry problem-solving 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 formalazation theory support, resulting in incoherent and incomplete 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 have annotated formalgeo7k and formalgeo-imo.

Our data is collected from various sources, including Geometry3k [29], GeoQA [6], GeoQA+ [4], 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 4 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. A annotated problem is illustrated in

in Fig. 10. 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.

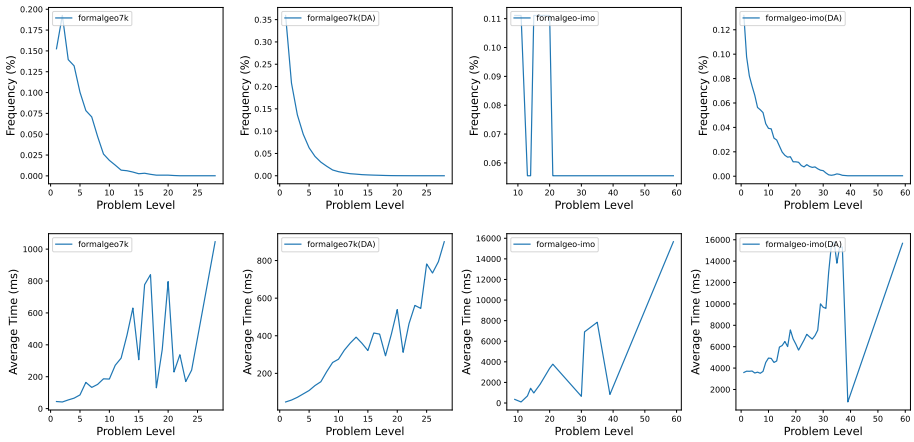


Fig. 8. Distribution of problem (the top 4). Average Time of interactive verification (the bottom 4). DA represents data augmentation.

We use the length of theorem sequences required for problem solving as a rough metric for assessing problem difficulty. The difficulty distribution in 2 datasets is illustrated in the Fig. 8. All annotated and expanded problems have been verified by the solver, and their average solution times varying with problem difficulty also show in the Fig. 8.

The number of questions with a difficulty level of 15 or higher in formalgeo7k is quite small, leading to significant fluctuations. formalgeo-imo follows the same reasoning. After data augmentation, datasets exhibit a larger scale of data and a smoother difficulty curve. In general, more challenging problems require longer solving time. Therefore, the length of problem solving theorem sequences can serve as a rough measure of problem difficulty.

5 Geometry Problem Solver: FGPS

Guided by GFT, we have constructed a geometric problem solver, FGPS. Any geometric formal system designed based on GFT and any geometric problem that

adheres to the syntax of formal language can be input into FGPS for verification and solution.

In this section, we introduce the implementation of the core solving engine. Detailed descriptions about solver's structure and other functionalities can be found in the App. F.

5.1 GPL Executor

The process of geometric problem solving can be represented as a sequence of theorem applications. Theorems are defined using GPL, and, as a result, the process of geometric problem solving within the solver 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 4 steps:

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.

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.

In the GPL execution phase, the solver reads geometric and quantitative relations sequentially and performs relational inference (Eq. 3 $\sim$ 7) 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 Eq. 11, which includes 5 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 1 quantitative relation $R_A(v_1, v_2)$.

$$R_1\&(R_2|(\sim R_3|R_A)\&R_4\&R_5) \tag{11}$$

During the GDL parsing phase, it is expanded into a DNF according to the distributive law, as shown in Eq. 12. This DNF consists of 3 simple conjunctions, with each simple conjunction serving as a theorem branch.

$$R_1\&R_2|R_1\&\sim R_3\&R_4\&R_5|R_1\&R_A\&R_4\&R_5 \tag{12}$$

In the GDL reordering phase, let's 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 Eq. 13.

$$R_1\&R_5\&R_4\&R_A \tag{13}$$

In the GPL execution phase, the GDL statements are read and executed in order, and the process is as shown in Eq. 14.

$$R_1\&R_5\&R_4\&R_A \rightarrow R_{1,5}\&R_4\&R_A \rightarrow R_{1,5,4}\&R_A \rightarrow R_{1,5,4,A} \tag{14}$$

5.2 Minimum Dependency Equations

The known conditions of geometric problems can be categorized into geometric and quantitative relationships. Quantitative relationships eventually represented as a set of algebraic equations or inequalities. When performing algebraic constraint in the execution of GPL, the satisfaction of 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 $(t = 1, 2, \ldots)$, 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.

$$|B_t \cap M_t| > 0 \tag{15}$$

$$min(|B_t - M_t|) \tag{16}$$

$$max(|B_t \cap M_t|) \tag{17}$$

The selection criteria for x_t are as follows:

1.B_t must intersect with M_t, as shown in Eq. 15. If they do not intersect, it implies that x_t is unrelated to G_t.

2.Under the condition of satisfying Eq. 15, adding x_t should introduce as few new unknown variables as possible, as depicted in Eq. 16. 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 equation 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 Eq. 15 and Eq. 16, the equation to be added should encompass more unknown variables, as demonstrated in Eq. 17. 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.

6 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 apply 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. A detailed description of the search algorithms can be found in App. F.

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 2 Intel i9-10900X, 1 AMD Ryzen 9 5900X, and 1 AMD Ryzen 9 7950X, 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 seconds, the best success rate for problem-solving was approximately 30%. When the timeout for each problem was increased to 600 seconds, the specific results are as follows.

An overview of search-based automated problem-solving results is presented in Tab. 1. 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

Table 1. An overview of search results.

method strategy		result (%)		
		solved	unsolved	timeout
FW	BFS	38.86	**7.42**	53.72
FW	DFS	36.16	9.80	54.05
FW	RS	**39.71**	9.07	51.22
FW	BS	25.28	38.72	**36.00**
BW	BFS	**35.44**	2.68	61.88
BW	DFS	33.73	**2.42**	63.84
BW	RS	34.05	2.65	63.30
BW	BS	34.39	12.86	**52.74**

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 trigonometric.

In accordance with the length of the theorems required for problem-solving, we roughly categorize the difficulty of the questions into 6 levels, denoted as $l_1(length <= 2)$, $l_2(3 <= length <= 4)$, $l_3(5 <= length <= 6)$, $l_4(7 <= length <= 8)$, $l_5(9 <= length <= 10)$, $l_6(length >= 11)$, 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 Tab. 2. As 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 2. 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	**17.35**	**8.63**	3.77
FW	DFS	36.16	55.75	**40.04**	22.94	12.38	7.03	4.11
FW	RS	**39.71**	59.24	**40.04**	**33.68**	16.38	5.43	**4.79**
FW	BS	25.28	46.12	22.60	13.47	5.83	2.88	0.34
BW	BFS	35.44	**67.22**	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 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 Fig. 9.

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 of backward search, discarding potential solusation branches does not lead to solution failure but takes longer search time.

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

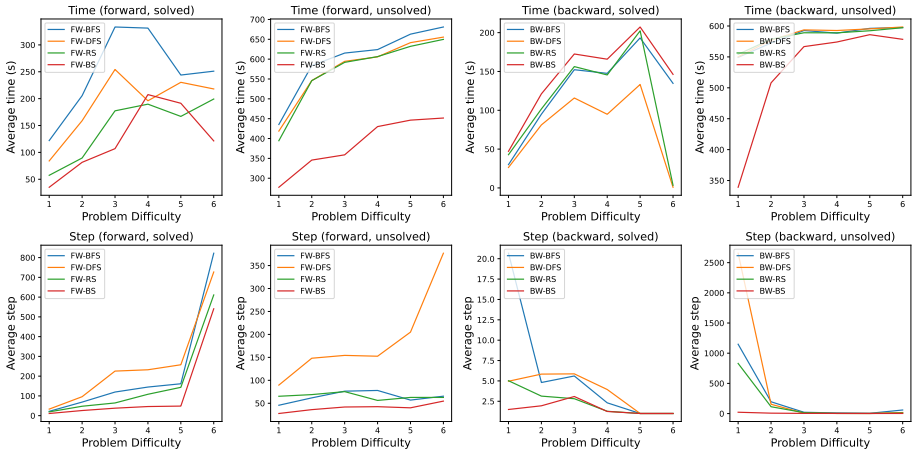


Fig. 9. Average search time (the top 4). Average search step (the bottom 4). The raw data for the charts can be found in App. F

slightly higher than BS. Backward search does not exhibit a significantly superior strategy.

7 Conclusion

We have introduced GFT, which includes geometry ontology and geometry representation theory, to guide the formalization of geometric problems. Building upon GFT, we have developed the geometric formal system FormalGeo and constructed the solver FGPS. Furthermore, we have annotated the geometric problem-solving dataset, formalgeo7k and formalgeo-imo. Experiments on 2 datasets have demonstrated the correctness and utility of GFT. We have also analyzed the success rate and efficiency of the solver's automatic problem-solving algorithm.

In the future, we plan to enhance GFT to make it more comprehensive and endowed with stronger representational capabilities. We also intend to further improve FormalGeo by expanding the types of predicates and theorems, as well as annotating formalgeo-imo datasets. Additionally, we aim to apply deep learning techniques to search tree pruning for the automatic solving of IMO-level geometric problems.

Acknowledgement

Thanks to the reviewers for their extensive constructive feedback. Thanks to all researchers involved in academic discussions and datasets annotation. The

research was supported by Geometric Cognitive Reasoning Group of Shanghai University (GCRG, SHU).

References

1. Alvin, C., Gulwani, S., Majumdar, R., Mukhopadhyay, S.: Synthesis of geometry proof problems. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 28 (2014)
2. Alvin, C., Gulwani, S., Majumdar, R., Mukhopadhyay, S.: Synthesis of solutions for shaded area geometry problems. In: The Thirtieth International Flairs Conference (2017)
3. Buchberger, B.: Applications of gröbner bases in non-linear computational geometry. Mathematical aspects of scientific software pp. 59–87 (1988)
4. 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. pp. 1511–1520 (2022)
5. Chen, J., Li, T., Qin, J., Lu, P., Lin, L., Chen, C., Liang, X.: Unigeo: Unifying geometry logical reasoning via reformulating mathematical expression. arXiv preprint arXiv:2212.02746 (2022)
6. 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. pp. 513–523 (2021)
7. 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. pp. 284–291 (1993)
8. 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 (1994)
9. Chou, S.C., Gao, X.S., Zhang, J.Z.: Automated production of traditional proofs in solid geometry. Journal of Automated Reasoning **14**(2), 257–291 (1995)
10. Chou, S.C., Gao, X.S., Zhang, J.Z.: An introduction to geometry expert. In: CADE. vol. 1104, pp. 235–239 (1996)
11. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition–preliminary report. ACM SIGSAM Bulletin **8**(3), 80–90 (1974)
12. Cunningham, G., Bunescu, R., Juedes, D.: Towards autoformalization of mathematics and code correctness: Experiments with elementary proofs. In: Proceedings of the 1st Workshop on Mathematical Natural Language Processing (MathNLP). pp. 25–32 (2022)
13. Daniel, S., Leonardo, d.M., Kevin, B., Reid, B., Percy, L., Sarah, L., Freek, W.: Imo grand challenge (2019), https://imo-grand-challenge.github.io/
14. Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., et al.: Advancing mathematics by guiding human intuition with ai. Nature **600**(7887), 70–74 (2021)
15. Gan, W., Yu, X.: Automatic understanding and formalization of natural language geometry problems using syntax-semantics models. International Journal of Innovative Computing, Information and Control **14**(1), 83–98 (2018)
16. Gan, W., Yu, X., Zhang, T., Wang, M.: Automatically proving plane geometry theorems stated by text and diagram. International Journal of Pattern Recognition and Artificial Intelligence **33**(07), 1940003 (2019)

17. Gao, X.S., Chou, S.C.: On the dimension of an arbitrary ascending chain. CHINESE SCIENCE BULLETIN-ENGLISH EDITION- **38**, 799–799 (1993)
18. Gelernter, H.L.: Realization of a geometry theorem proving machine. In: IFIP congress. pp. 273–281 (1959)
19. Hao, Y., Zhang, M., Yin, F., Huang, L.L.: Pgdp5k: A diagram parsing dataset for plane geometry problems. In: 2022 26th International Conference on Pattern Recognition (ICPR). pp. 1763–1769. IEEE (2022)
20. Huang, L., Yu, X., He, B.: A novel geometry problem understanding method based on uniform vectorized syntax-semantics model. In: 2022 International Conference on Intelligent Education and Intelligent Research (IEIR). pp. 78–85. IEEE (2022)
21. Jian, P., Guo, F., Wang, Y., Li, Y.: Solving geometry problems via feature learning and contrastive learning of multimodal data. CMES-COMPUTER MODELING IN ENGINEERING & SCIENCES **136**(2), 1707–1728 (2023)
22. 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. Advances in Neural Information Processing Systems **35**, 8360–8373 (2022)
23. Jiang, A.Q., Welleck, S., Zhou, J.P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., Wu, Y.: Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In: The Eleventh International Conference on Learning Representations (2022)
24. Jingzhong, Z., Yongbin, L.: Automatic theorem proving for three decades. Journal of Systems Science and Mathematical Sciences **29**(9), 1155 (2009)
25. Kovács, Z., Yu, J.H.: Automated discovery of geometrical theorems in geogebra. arXiv preprint arXiv:2202.04627 (2022)
26. Lample, G., Lacroix, T., Lachaux, M.A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., Martinet, X.: Hypertree proof search for neural theorem proving. Advances in Neural Information Processing Systems **35**, 26337–26349 (2022)
27. Li, H.: Symbolic computation in the homogeneous geometric model with clifford algebra. In: Proceedings of the 2004 international symposium on Symbolic and algebraic computation. pp. 221–228 (2004)
28. 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 preprint arXiv:2210.15767 (2022)
29. 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). pp. 6774–6786 (2021)
30. Lu, Y.: Practical automated reasoning on inequalities: Generic programs for inequality proving and discovering. Proceedings of the Third Asian Technology Confer ence in Mathematics. Tsukuba, Japan pp. 24–35 (1998)
31. Mishra, C., Moulik, S.R., Sarkar, R.: Mathematical conjecture generation using machine intelligence. arXiv preprint arXiv:2306.07277 (2023)
32. Ning, M., Wang, Q.F., Huang, K., Huang, X.: A symbolic character-aware model for solving geometry problems (2023)
33. 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. pp. 13468–13480 (2023)

34. Polu, S., Han, J.M., Zheng, K., Baksys, M., Babuschkin, I., Sutskever, I.: Formal mathematics statement curriculum learning. In: The Eleventh International Conference on Learning Representations (2022)
35. Raayoni, G., Gottlieb, S., Manor, Y., Pisha, G., Harris, Y., Mendlovic, U., Haviv, D., Hadad, Y., Kaminer, I.: Generating conjectures on fundamental constants with the ramanujan machine. Nature **590**(7844), 67–73 (2021)
36. Rao, Y., Xie, L., Guan, H., Li, J., Zhou, Q.: A method for expanding predicates and rules in automated geometry reasoning system. Mathematics **10**(7), 1177 (2022)
37. 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. Computational Linguistics **45**(4), 627–665 (2020)
38. 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. pp. 773–784 (2017)
39. 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). pp. 251–261 (2017)
40. Seo, M.J., Hajishirzi, H., Farhadi, A., Etzioni, O.: Diagram understanding in geometry questions. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 28 (2014)
41. 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. pp. 1466–1476 (2015)
42. 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). pp. 964–972 (2021)
43. Wang, D.: Geother: A geometry theorem prover. In: Automated Deduction—Cade-13: 13th International Conference on Automated Deduction New Brunswick, NJ, USA, July 30–August 3, 1996 Proceedings. pp. 166–170. Springer (2005)
44. Wilson, S., Fleuriot, J.D.: Geometry explorer: A tool for generating diagrammatic full-angle method proofs. In: Automated deduction in geometry: extended abstracts. pp. 144–150 (2006)
45. Wong, M.F., Qi, X., Tan, C.W.: Euclidnet: Deep visual reasoning for constructible problems in geometry. In: 2nd MATH-AI Workshop at NeurIPS'22: Toward Human-Level Mathematical Reasoning (2022)
46. Wu, W.T.: On the decision problem and the mechanization of theorem proving in elementary geometry. Scientia Sinica **21**, 157–179 (1978)
47. Wu, Y., Jiang, A.Q., Li, W., Rabe, M., Staats, C., Jamnik, M., Szegedy, C.: Autoformalization with large language models. Advances in Neural Information Processing Systems **35**, 32353–32368 (2022)
48. Yang, L., Zhang, J., Li, C.: A prover for parallel numerical verification of a class of constructive geometry theorems. In: Proc. IWMM. vol. 92, pp. 244–250 (1992)
49. 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: International Workshop on Automated Deduction in Geometry Toulouse, France, September 27–29, 1996 Selected Papers 1. pp. 171–188. Springer (1997)

50. Ye, Z., Chou, S.C., Gao, X.S.: An introduction to java geometry expert. In: Automated Deduction in Geometry: 7th International Workshop, ADG 2008, Shanghai, China, September 22-24, 2008. Revised Papers 7. pp. 189–195. Springer (2011)
51. 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: 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Workshop on Math AI for Education (MATHAI4ED) (2021)
52. Yu, X., Gan, W., Wang, M.: Understanding explicit arithmetic word problems and explicit plane geometry problems using syntax-semantics models. In: 2017 International Conference on Asian Language Processing (IALP). pp. 247–251. IEEE (2017)
53. Yu, X., Wang, M., Gan, W., He, B., Ye, N.: A framework for solving explicit arithmetic word problems and proving plane geometry theorems. International Journal of Pattern Recognition and Artificial Intelligence **33**(07), 1940005 (2019)
54. 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. Annals of Mathematics and Artificial Intelligence **13**(1-2), 109–137 (1995)
55. Zhang, M.L., Yin, F., Hao, Y.H., Liu, C.L.: Plane geometry diagram parsing. In: Raedt, L.D. (ed.) Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22. pp. 1636–1643. International Joint Conferences on Artificial Intelligence Organization (7 2022). https://doi.org/10.24963/ijcai.2022/228, https://doi.org/10.24963/ijcai.2022/228, main Track
56. Zhang, M.L., Yin, F., Liu, C.L.: A multi-modal neural geometric solver with textual clauses parsed from diagram. arXiv preprint arXiv:2302.11097 (2023)
57. Zheng, K., Han, J.M., Polu, S.: minif2f: a cross-system benchmark for formal olympiad-level mathematics. In: International Conference on Learning Representations (2022)
58. Zhong, X., Fu, H., Yu, Y., Liu, Y.: Interactive learning environment based on knowledge network of geometry problems. In: 2015 10th International Conference on Computer Science & Education (ICCSE). pp. 53–58. IEEE (2015)
59. Zhou, W., Xu, R., Guan, H., Zhao, J., Rao, Y.: Research on geometry problem text understanding based on bidirectional lstm-crf. In: 2022 9th International Conference on Digital Home (ICDH). pp. 121–127. IEEE (2022)

A Examples

Guided by the GFT, we have developed the geometry formal system, FormalGeo. In this section, we provide several examples related to the design of FormalGeo predicates and theorems to help readers better understand the GFT. App. A.1 and App. A.2 correspond to the application of geometry ontology, while App. A.3, App. A.4, and App. A.5 correspond to the application of geometry representation theory.

A.1 Geometry ontology domain

The geometry ontology domain comprehensively summarizes and categorizes geometric knowledge. It is divided into 4 quadrants along 2 dimensions: dynamic versus static, and number versus shape. Each quadrant encompasses mappings for axiomatic system, formal system, and solver, as depicted in Tab. 3.

Table 3. Examples of geometry ontology domain.

Quadrants	Tiers	Examples
1 Static Shape	axiomatic system	Triangle ABC is an equilateral triangle.
	formal system	EquilateralTriangle(ABC)
	solver	$R_{ET} = \{ABC, BCA, CAB\}$
2 Static Number	axiomatic system	The length of line AB is equal to the length of line CD.
	formal system	Equal(LengthOfLine(AB),LengthOfLine(CD))
	solver	$ll_{AB} - ll_{CD} = 0$
3 Dynamic Number	axiomatic system	A triangle with two equal legs is an isosceles triangle.
	formal system	Triangle(ABC)& Equal(LengthOfLine(AB),LengthOfLine(AC))→ IsoscelesTriangle(ABC)
	solver	GPL defined in Eq. 5
4 Dynamic Shape	axiomatic system	If AB is parallel to CD, and CD is parallel to EF, then AB is parallel to EF.
	formal system	Parallel(AB,CD)&Parallel(CD,EF)→Parallel(AB,EF)
	solver	GPL defined in Eq. 3 and Eq. 4

A.2 Geometry knowledge graph

The geometry knowledge graph is an extension of the geometry ontology domain, which structurally presents the design process of predicate definition language

and theorem definition language, preventing omissions or redundancies. FormalGeo comprises n predicates and m theorems, and its geometry knowledge graph is illustrated in Fig. 11.

A.3 Predicate definition language

FormalGeo comprises 88 predicates, including 25 fundamental predicates (Tab. 6) built into the solver and 12 entities (Tab. 7), 30 entity relationships (Tab. 8), and 21 attributions (Tab. 9) defined using the predicate definition language. The structured relationships between predicates are depicted in Fig. 11.

The detailed statements for defining a predicate is as shown in the Tab. 4, including the predicate name and point variable declaration, validity check declaration, multiple representations, and automatic expansion. Additionally, when defining attributes, it also includes symbolic form declaration.

Table 4. Detailed statement examples for defining a predicate.

name	item	content
IsMidpointOfLine(M,AB)		Point(M)
	ee_check	Line(AB)
		Collinear(AMB)
	fv_check	M,AB
	multi	M,BA
	extend	Equal(LengthOfLine(AM),LengthOfLine(MB))
LengthOfLine(AB)	ee_check	Line(AB)
	sym	ll
	multi	BA

A.4 Theorem definition language

Theorems are defined using the GPL, comprising two parts: premises and conclusions, as shown in the Tab. 5. FormalGeo encompasses 196 theorems, and their structured relationships are illustrated in Fig. 11.

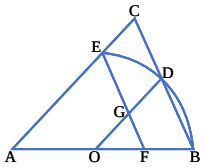
Table 5. Detailed statement examples for defining a theorem.

name	item	content
midpoint_of_line_judgment(M,AB)	premise	Collinear(AMB)& Equal(LengthOfLine(AM),LengthOfLine(MB))
	conclusion	IsMidpointOfLine(M,AB)
vertical_angle (AOC,BOD)	premise	Collinear(AOB)&Collinear(COD)& Angle(AOC)&Angle(BOD)
	conclusion	Equal(MeasureOfAngle(AOC),MeasureOfAngle(BOD))

A.5 Condition declaration language

We can use CDL to transform the description of geometric problems into a formal language. CDL consists of three main parts: construction statements, condition statements, and goal statements. An example of a transformed problem is illustrated in Fig. 10, with theorem_seqs denoting the annotated problem-solving theorem sequence.

△ABC is an isosceles △, F is the midpoint of OB, O is the center of the circle O, OG = GD.
To prove: quadrilateral CEGD is a parallelogram.



construction_cdl

Shape(CE,ODE,DC)	Collinear(CEA)
Shape(ODE,EG,GD)	Collinear(CDB)
Shape(EA,AO,OG,GE)	Collinear(AOFB)
Shape(GO,OF,FG)	Collinear(EGF)
Shape(DG,GF,FB,BD)	Collinear(OGD)
Shape(OBD,DB)	Cocircular(O,BDE)

theorem_seqs

radius_of_circle_property_length_equal(1,OB,O)
radius_of_circle_property_length_equal(1,OD,O)
isosceles_triangle_judgment_line_equal(1,OBD)
isosceles_triangle_property_angle_equal(1,ABC)
isosceles_triangle_property_angle_equal(1,OBD)
parallel_judgment_corresponding_angle(2,DG,CE,B)
midsegment_of_triangle_judgment_midpoint(1,FG,OBD)
midsegment_of_triangle_property_parallel(1,FG,OBD)
parallel_property_collinear_extend(2,DB,GF,C)
parallel_property_collinear_extend(2,FG,DC,E)
parallelogram_judgment_parallel_and_parallel(1,CEGD)

condition_cdl

IsMidpointOfLine(F,OB)
IsCentreOfCircle(O,O)
IsoscelesTriangle(ABC)
Equal(LengthOfLine(OG),LengthOfLine(GD))

goal_cdl

Relation(Parallelogram(CEGD))

problem_answer

Parallelogram(CEGD)

Fig. 10. An example of annotated problems in FormalGeo7k

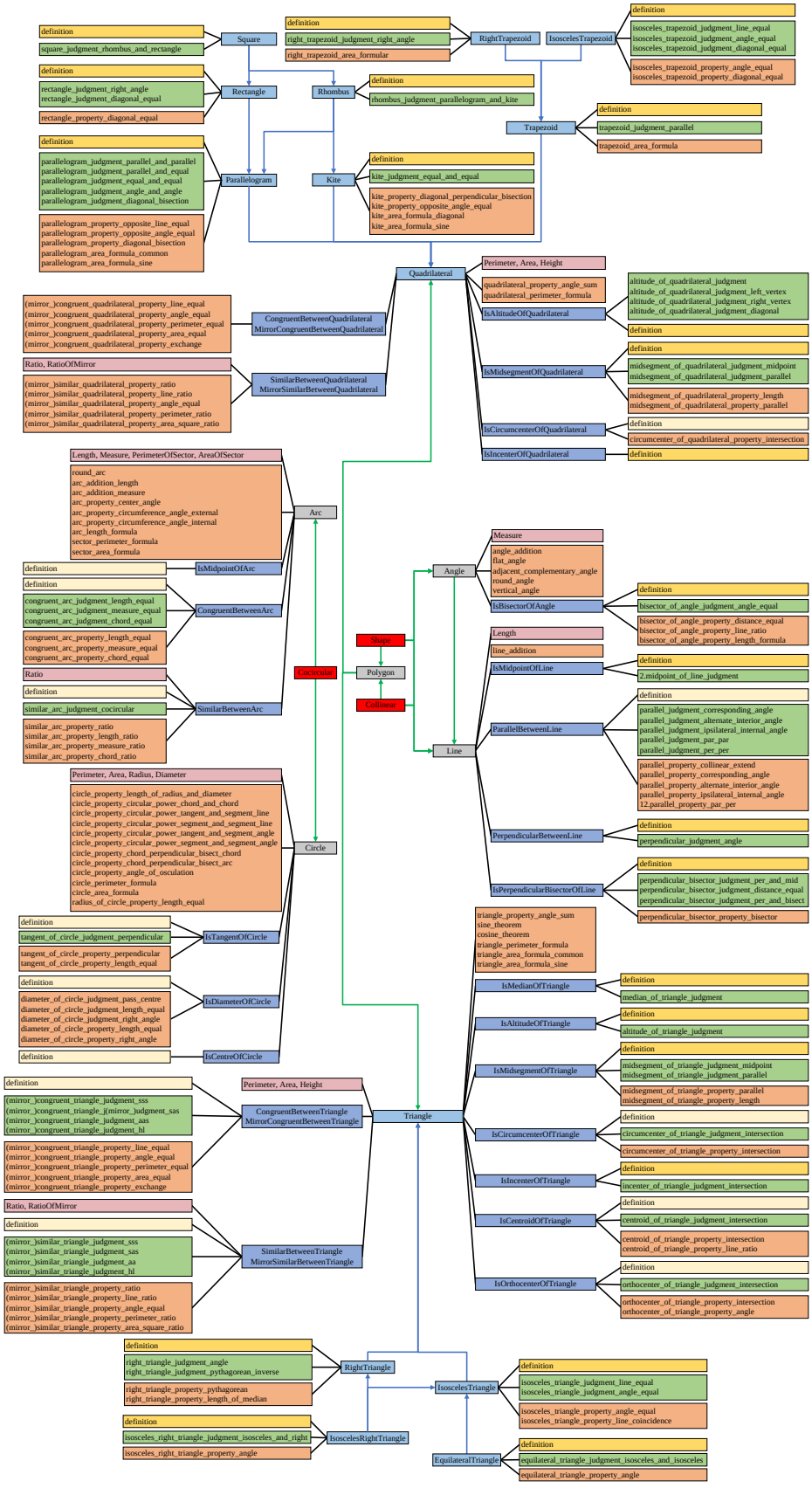


Fig. 11. Geometry knowledge graph of FormalGeo

Table 6. Predicates built into the solver.

id	type	name	examples
1	Construction	Shape	Shape(AB,BC,CA)
2	Construction	Collinear	Collinear(ABCD)
3	Construction	Cocircular	Cocircular(O,ABC)
4	BasicEntity	Point	Point(A)
5	BasicEntity	Line	Line(AB)
6	BasicEntity	Arc	Arc(OAB)
7	BasicEntity	Angle	Angle(ABC)
8	BasicEntity	Polygon	Polygon(ABCD)
9	BasicEntity	Circle	Circle(O)
10	Algebra	Equal	Equal(a,b)
11	Algebra	Equation	Equation(a-b)
12	Attribution	Free	Free(x)
13	Operation	Add	Equal(Add(a,b,c),1)
14	Operation	Sub	Equal(Sub(a,b),1)
15	Operation	Mul	Equal(Mul(a,b,c),1)
16	Operation	Div	Equal(Div(a,b),1)
17	Operation	Pow	Equal(Pow(a,b),1)
18	Operation	Mod	Equal(Mod(a,b),1)
19	Operation	Sqrt	Equal(Sqrt(a),1)
20	Operation	Sin	Equal(Sin(a),1/2)
21	Operation	Cos	Equal(Cos(a),1/2)
22	Operation	Tan	Equal(Tan(a),1)
23	Target	Value	Value(a)
24	Target	Equal	Equal(a,b)
25	Target	Relation	Relation(RightTriangle(ABC))

Table 7. Entities defined using the predicate definition language.

id	type	examples
26	Entity	RightTriangle(ABC)
27	Entity	IsoscelesTriangle(ABC)
28	Entity	IsoscelesRightTriangle(ABC)
29	Entity	EquilateralTriangle(ABC)
30	Entity	Kite(ABCD)
31	Entity	Parallelogram(ABCD)
32	Entity	Rhombus(ABCD)
33	Entity	Rectangle(ABCD)
34	Entity	Square(ABCD)
35	Entity	Trapezoid(ABCD)
36	Entity	IsoscelesTrapezoid(ABCD)
37	Entity	RightTrapezoid(ABCD)

Table 8. Relations defined using the predicate definition language.

id	type	examples
38	Relation	IsMidpointOfLine(M,AB)
39	Relation	IsMidpointOfArc(M,OAB)
40	Relation	ParallelBetweenLine(AB,CD)
41	Relation	PerpendicularBetweenLine(AC,BC)
42	Relation	IsPerpendicularBisectorOfLine(AB,CD)
43	Relation	IsBisectorOfAngle(BD,ABC)
44	Relation	IsMedianOfTriangle(AD,ABC)
45	Relation	IsAltitudeOfTriangle(AD,ABC)
46	Relation	IsMidsegmentOfTriangle(DE,ABC)
47	Relation	IsCircumcenterOfTriangle(O,ABC)
48	Relation	IsIncenterOfTriangle(O,ABC)
49	Relation	IsCentroidOfTriangle(O,ABC)
50	Relation	IsOrthocenterOfTriangle(O,ABC)
51	Relation	CongruentBetweenTriangle(ABC,DEF)
52	Relation	MirrorCongruentBetweenTriangle(ABC,DEF)
53	Relation	SimilarBetweenTriangle(ABC,DEF)
54	Relation	MirrorSimilarBetweenTriangle(ABC,DEF)
55	Relation	IsAltitudeOfQuadrilateral(EF,ABCD)
56	Relation	IsMidsegmentOfQuadrilateral(EF,ABCD)
57	Relation	IsCircumcenterOfQuadrilateral(O,ABCD)
58	Relation	IsIncenterOfQuadrilateral(O,ABCD)
59	Relation	CongruentBetweenQuadrilateral(ABCD,EFGH)
60	Relation	MirrorCongruentBetweenQuadrilateral(ABCD,EFGH)
61	Relation	SimilarBetweenQuadrilateral(ABCD,EFGH)
62	Relation	MirrorSimilarBetweenQuadrilateral(ABCD,EFGH)
63	Relation	CongruentBetweenArc(OAB,OCD)
64	Relation	SimilarBetweenArc(OAB,OCD)
65	Relation	IsDiameterOfCircle(AB,O)
66	Relation	IsTangentOfCircle(PA,O)
67	Relation	IsCentreOfCircle(P,O)

Table 9. Attributions defined using the predicate definition language.

id	type	examples
68	Attribution	LengthOfLine(AB)
69	Attribution	MeasureOfAngle(ABC)
70	Attribution	PerimeterOfTriangle(ABC)
71	Attribution	AreaOfTriangle(ABC)
72	Attribution	HeightOfTriangle(ABC)
73	Attribution	RatioOfSimilarTriangle(ABC)
74	Attribution	RatioOfMirrorSimilarTriangle(ABC)
75	Attribution	PerimeterOfQuadrilateral(ABCD)
76	Attribution	AreaOfQuadrilateral(ABCD)
77	Attribution	HeightOfQuadrilateral(ABCD)
78	Attribution	RatioOfSimilarQuadrilateral(ABCD)
79	Attribution	RatioOfMirrorSimilarQuadrilateral(ABCD)
80	Attribution	LengthOfArc(OAB)
81	Attribution	MeasureOfArc(OAB)
82	Attribution	RatioOfSimilarArc(OAB)
83	Attribution	RadiusOfCircle(O)
84	Attribution	DiameterOfCircle(O)
85	Attribution	PerimeterOfCircle(O)
86	Attribution	AreaOfCircle(O)
87	Attribution	PerimeterOfSector(OAB)
88	Attribution	AreaOfSector(OAB)

B Consistency theory of formal system

A formal system is an abstraction of the real world that simplifies and clarifies problems or concepts by removing certain details and complexities. Its primary objective is to provide a precise, consistent, and reliable method for studying and solving problems while eliminating ambiguity and uncertainty. By formalizing problems, we can apply mathematical, logical, and computational methods to address a wide range of real-world issues.

In the real world, a concept c, which could represent anything, the properties of things, relationships between things, etc., can be represented by a symbol $s^{(c)}$ within the formal system. A single concept may have multiple symbol representations within the formal system, denoted as the set R_c. The rules governing the conversion of concepts in the real world are represented as various rules in formal system, denoted as $\xrightarrow{r}$. These rules define relationships and operations between symbols, determining the methods of inference and computation.

When designing a formal system, it is essential to ensure the consistency of static representations. For any symbol representation $s_i^{(c)}$ of a concept c, there should exist an operation *multi* to obtain the symbol set R_c, as shown in Eq. 18. The mapping between c and R_c should be a reversible mapping, as indicated in Eq. 19.

$$multi(s_i^{(c)}) = R_c, s_i^{(c)} \in R_c \tag{18}$$

$$f(c) = R_c, f^{-1}(R_c) = c \tag{19}$$

Additionally, the design of a formal system should ensure the consistency of dynamic processes. For any rule $\xrightarrow{r}$ designed to govern the conversion of concepts c_a and c_b in the real world, it should satisfy the condition that, for any symbol representation $s_i^{(c_a)}$ of c_a, the application of $\xrightarrow{r}$ results in and can only yield the symbol representation $s_j^{(c_b)}$ of c_b.

$$c_a \xrightarrow{r} c_b \tag{20}$$

$$s_i^{(c_a)} \xrightarrow{r} s_j^{(c_b)}, s_i^{(c_a)} \in R_{c_a}, s_j^{(c_b)} \in R_{c_b} \tag{21}$$

C Topological mapping method

When we utilize topological mapping method to transform the TSI of a diagram into a formal representation, we can define the fundamental transformations of the diagram as operations on its TSI formal representation.

The formal representation of the TSI of a geometric diagram is denoted as $(p_1, p_2, \ldots, p_n)$. Clockwise rotation of the diagram can be defined as moving the first point in the sequence to the end of the sequence, as shown in Eq. 22. Reflection can be defined as reversing the order of the sequence, as shown in Eq. 23. Translation, shearing, and scaling do not alter the TSI of the diagram, as shown in Eq. 24.

$$rotate((p_1, p_2, \ldots, p_n)) = (p_2, \ldots, p_n, p_1) \tag{22}$$

$$reflect((p_1, p_2, \ldots, p_n)) = (p_n, p_{n-1}, \ldots, p_1) \tag{23}$$

$$linear((p_1, p_2, \ldots, p_n)) = (p_1, p_2, \ldots, p_n) \tag{24}$$

D Topological construction method

The operation $\oplus$ defined by Eq. 1 and Eq. 2 is meaningful only when two shapes are non-overlapping simple closed shapes and when the two shapes share adjacent edges (the number of common points > 2).

The formal representation of composite diagrams, along with $\oplus$, forms a semigroup, satisfying closure, commutative law and associative law. The formal representation of composite diagrams is defined as the collection of the formal representations of all its constituent shapes, so closure is evidently satisfied. Next, we will prove that the operation $\oplus$ satisfies commutative law and associative law.

If diagram C is composed of diagrams A and B, their formal representations of TSI are represented as sets R_a, R_b, and R_c, respectively. Diagram A, B, C

contains m, n, o points, respectively. There are x ($x \geq 2$)common points shared between A and B and y ($y \geq 2$)common points shared between B and C. The formal representations of their TSI can be represented as Eq. 25, Eq. 26 and Eq. 27.

$$P_a = (p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, \ldots, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}) \tag{25}$$

$$\begin{aligned} P_b =& (p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, \ldots, p_i^{(a,b)}, \\ & p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, \ldots, p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}) \end{aligned} \tag{26}$$

$$P_c = (p_1^{(c)}, \ldots, p_{s_1}^{(c)}, p_{j+y}^{(b,c)}, \ldots, p_j^{(b,c)}, p_{s_2}^{(c)}, \ldots, p_n^{(c)}) \tag{27}$$

We shall begin by proving the commutative law. As demonstrated in Eq. 28 and Eq. 29, $P_a \otimes P_b = rotate^{(i)}(P_b \otimes P_a)$, where $i = |\{p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}\}|$. Then the commutative law can be demonstrated by Eq. 30.

$$\begin{aligned} & P_a \otimes P_b \\ =& (p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, \ldots, \\ & p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}, p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}) \end{aligned} \tag{28}$$

$$\begin{aligned} & P_b \otimes P_a \\ =& (p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}, p_1^{(a)}, \ldots, \\ & p_{s_1}^{(a)}, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, \ldots, p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}) \end{aligned} \tag{29}$$

$$\begin{aligned} & R_a \oplus R_b \\ =& \{rotate^{(i)}(P_a \otimes P_b) | i = 1, 2, \ldots, l + m - 2x + 2\} \\ =& \{rotate^{(i)}(P_b \otimes P_a) | i = 1, 2, \ldots, l + m - 2x + 2\} \\ =& R_b \oplus R_a \end{aligned} \tag{30}$$

Next, we proceed to prove the associative law. As demonstrated in Eq. 31 and Eq. 32, $(P_a \otimes P_b) \otimes P_c = P_a \otimes (P_b \otimes P_c)$, where Then the commutative law can be demonstrated by Eq. 33.

$$\begin{aligned} & (P_a \otimes P_b) \otimes P_c \\ =& (p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, \ldots, \\ & p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}, p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}) \\ & \otimes (p_1^{(c)}, \ldots, p_{s_1}^{(c)}, p_{j+y}^{(b,c)}, \ldots, p_j^{(b,c)}, p_{s_2}^{(c)}, \ldots, p_n^{(c)}) \\ =& (p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, p_{s_2}^{(c)}, \ldots, p_n^{(c)}, p_1^{(c)}, \ldots, \\ & p_{s_1}^{(c)}, p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}, p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}) \end{aligned} \tag{31}$$

$$
\begin{aligned}
&P_a \otimes (P_b \otimes P_c) \\
=&(p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, \ldots, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)}) \\
&\otimes (p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, \ldots, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, p_{s_2}^{(c)}, \ldots, \\
&p_n^{(c)}, p_1^{(c)}, \ldots, p_{s_1}^{(c)}, p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}) \\
=&(p_1^{(a)}, \ldots, p_{s_1}^{(a)}, p_i^{(a,b)}, p_{s_2}^{(b)}, \ldots, p_{s_3}^{(b)}, p_j^{(b,c)}, p_{s_2}^{(c)}, \ldots, p_n^{(c)}, p_1^{(c)}, \ldots, \\
&p_{s_1}^{(c)}, p_{j+y}^{(b,c)}, p_{s_4}^{(b)}, \ldots, p_m^{(b)}, p_1^{(b)}, \ldots, p_{s_1}^{(b)}, p_{i+x}^{(a,b)}, p_{s_2}^{(a)}, \ldots, p_l^{(a)})
\end{aligned}
\tag{32}
$$

$$
\begin{aligned}
&(R_a \oplus R_b) \oplus R_c \\
=&\{rotate^{(i)}((P_a \otimes P_b) \otimes P_c) | i = 1, 2, \ldots, l + m + n - 2x - 2y + 4\} \\
=&\{rotate^{(i)}(P_a \otimes (P_b \otimes P_c)) | i = 1, 2, \ldots, l + m + n - 2x - 2y + 4\} \\
=&R_a \oplus (R_b \oplus R_c)
\end{aligned}
\tag{33}
$$

The aforementioned properties enable us to construct diagrams from any two simple components of a composite diagram without considering the order of construction. This laws makes the TCM an order-independent diagram construction method. When formalizing problems and implementing TCM, we no longer need to be concerned about the order of construction statements.

We have implemented TCM in FGPS, and the algorithm description can be found in Alg. 1, where the *multi* function extends a single TSI formal representation of a diagram to the set of all formal representations, as defined in Eq. 2, and the *combine* function is used to achieve the combination representation of two simple diagrams, as defined in Eq. 1.

The computational cost of the algorithm per iteration is shown in Tab. 10, where n represents the number of simple closed diagrams that make up the complex diagram. For the complex diagram, it takes at most $n-1$ combinations of simple figures to obtain the complex diagram. The number of combined diagrams generated after each iteration depends on the number of edges in the simple diagrams. For example, when n triangles are combined, the number of quadrilaterals obtained is approximately $3n/2$, denoted as $k_i \cdot n$.

Table 10. Time Complexity Analysis of the implemented TCM.

iteration	len(Units)	len(Combs)	combination times	len(NewCombs)
1	n	n	$n \cdot n$	$k_1 \cdot n$
2	n	$k_1 \cdot n$	$n \cdot k_1 \cdot n$	$k_2 \cdot n$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$n-1$	n	$k_{n-2} \cdot n$	$n \cdot k_{n-2} \cdot n$	0

The time complexity of the implemented TCM is given by Eq. 34.

$$n^2 + k_1 n^2 + \cdots + k_{n-2} n^2 = (1 + k_1 + \cdots + k_{n-2}) n^2 \approx O(n^3) \tag{34}$$

Algorithm 1 Topological construction method

Input: *Units*: list, TSI of simple diagrams that make up combined diagram.
Output: *Results*: list, constructed TSI of combined diagram.

```
Units ← multi(Units)
Combs ← Units
Results ← Combs
while len(Combs) > 0 do
  Initialize an list NewCombs
  for i = 1 to len(Units) do
    unit ← Units[i]
    for j = 1 to len(Combs) do
      comb ← Combs[j]
      new_comb ← combine(unit, comb)
      if new_comb is not None and new_comb is not in Results then
        new_combs ← multi(new_comb)
        for k = 1 to len(new_combs) do
          add new_combs[k] to NewCombs
          add new_combs[k] to Results
        end for
      end if
    end for
  end for
  Combs ← NewCombs
end while
```

E Geometry predicate logic

The GPL operation & is, in fact, a more general form of operation $\oplus$. $\oplus$ requires that the two geometric relations involved in the operation are formalized TSI, and the result has a fixed form. & is not limited to the same type of geometric relations and can even operate across different types of relations, such as geometric and quantitative relations. The result of the operation is a set of point variables, which can be combined into specific structures as needed. In addition, GPL has introduced the | and $\sim$ to enhance its expressive power.

For the & operation, when the second relation is a quantitative relation, it is denoted as $R_1 \& R_A$. The operation is only meaningful when the point variables of R_A are a subset of those in R_1. As for the | operation, it is only meaningful when the two relations involved have the same point variable structure. The & operation satisfies commutative law and associative law, while the combined operation of & and | satisfies the distributive law.

$$R_1(v_1^{(1)}, v_2^{(1)}, \ldots, v_l^{(1)}) \tag{35}$$
$$R_2(v_1^{(2)}, v_2^{(2)}, \ldots, v_m^{(2)}) \tag{36}$$
$$R_3(v_1^{(3)}, v_2^{(3)}, \ldots, v_n^{(3)}) \tag{37}$$

There are 3 relations as shown in Eq. 35, Eq. 36 and Eq. 37. In the subsequent proof process, for the sake of clarity, we temporarily omit the operation of removing duplicate variables, as defined in Eq. 4.

$$\begin{aligned}
&R_1\&R_2\\
=&\{(r_i^1, r_j^2)|(r_i^1, r_j^2) \in R_1 \times R_2, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2\}\\
=&\{(r_j^2, r_i^1)|(r_j^2, r_i^1) \in R_2 \times R_1, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2\}\\
=&R_2\&R_1
\end{aligned} \tag{38}$$

$$\begin{aligned}
&(R_1\&R_2)\&R_3\\
=&\{(r_i^1, r_j^2, r_k^3)|(r_i^1, r_j^2, r_k^3) \in (R_1 \times R_2) \times R_3, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2,\\
&r_i^1(u) = r_k^3(u), u \in V_1 \cap V_3, r_j^2(w) = r_k^3(w), w \in V_2 \cap V_3\}\\
=&\{(r_i^1, r_j^2, r_k^3)|(r_i^1, r_j^2, r_k^3) \in R_1 \times (R_2 \times R_3), r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2,\\
&r_i^1(u) = r_k^3(u), u \in V_1 \cap V_3, r_j^2(w) = r_k^3(w), w \in V_2 \cap V_3\}\\
=&R_1\&(R_2\&R_3)
\end{aligned} \tag{39}$$

$$\begin{aligned}
&R_1\&(R_2|R_3)\\
=&\{(r_i^1, r_j^2)|(r_i^1, r_j^2) \in R_1 \times (R_2 \cup R_3), r_i^1(v) = r_j^2(v), v \in V_1 \cap (V_2 \cup V_3)\}\\
=&\{(r_i^1, r_j^2)|(r_i^1, r_j^2) \in R_1 \times R_2, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_2\}\\
&\cup \{(r_i^1, r_j^2)|(r_i^1, r_j^2) \in R_1 \times R_3, r_i^1(v) = r_j^2(v), v \in V_1 \cap V_3\}\\
=&(R_1\&R_2)|(R_1\&R_3)
\end{aligned} \tag{40}$$

The proof of ommutative law, associative law and distributive law can be found in Eq. 38, Eq. 39 and Eq. 40. We utilize the laws of the Cartesian product $\times$ to prove the laws of the &. The elements of the relation R are treated as unordered sets. Therefore, (r_i^1, r_j^2) and (r_j^2, r_i^1) are considered equivalent in Eq. 38. As mentioned earlier, | is meaningful only when the two relations involved have the same point variable structure. Hence, in Eq. 40, we have $V_2 = V_3 = V_2 \cup V_3$. The aforementioned laws allow us to simplify GPL statements before their execution, speeding up the process.

F FGPS

The structure of FGPS can be divided into five main components: Main Logic Control, Core Solving Engine, Formal Language Parser, Data Loader, and AI Interface. The relationships between these modules are illustrated in Fig. 12.

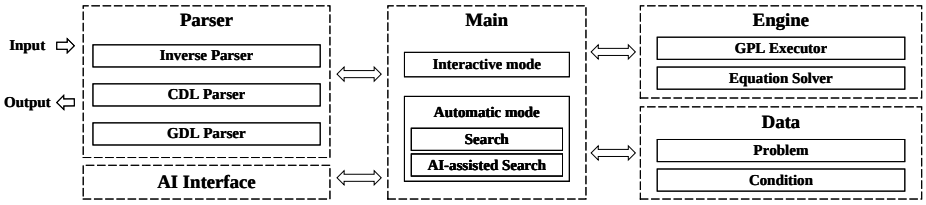


Fig. 12. The architecture of FGPS

Main is 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 (breadth-first, depth-first, random, beam) 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 inference and Equation Solver for algebraic computation.

Parser facilitates bidirectional conversion between formal language and machine language. It consists of 3 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 inference. 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 2 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.

Guided by GFT and modular design, FGPS boasts exceptional extensibility beyond its fundamental features like formal language parsing, GPL execution, human-readable problem-solving processes, and structured output. By invoking FGPS's core modules, we've 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 starts from the known conditions of the problem and continuously apply 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 Fig. 13. The description of the forward search algorithm is provided in Alg. 2. The function *get_expandable*() traverses the search tree based on pre-defined strategies (BFS, DFS, RS and BS) and returns nodes with the EXPANDABLE state. The

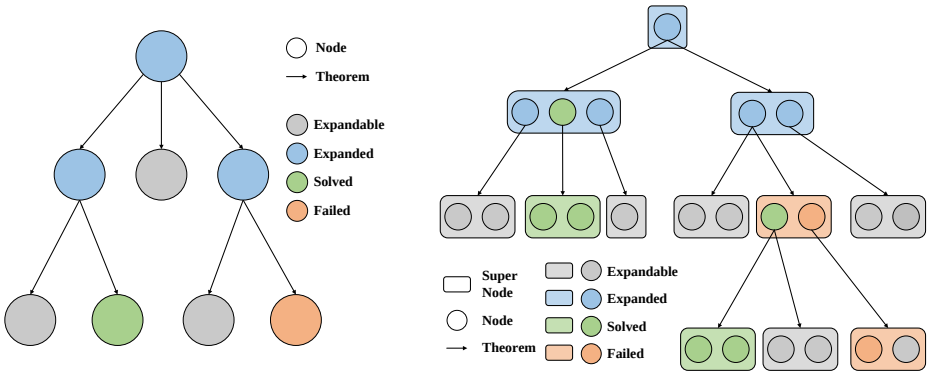


Fig. 13. Forward search Tree (left). Backward search Tree (right).

function *apply_theorem*() applies the theorem associated with the current node and returns whether the problem 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.

Algorithm 2 Forward search

Input: *tree*: a tree with the known problem conditions as the root node.
Output: *theorem_seqs*: list of theorem sequence for problem solving.
Initialize an list *theorem_seqs*
node ← *tree.get_expandable*()
while *node* is not *None* **do**
  solved ← *node.apply_theorem*()
  if *solved* **then**
    node.state ← SOLVED
    theorem_seqs ← *node.get_theorem_seqs*()
    break
  end if
  node.state ← EXPANDED
  l ← *node.expand*()
  if *l* = 0 **then**
    node.state ← FAILED
  end if
  node ← *tree.get_expandable*()
end while

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 subgoals, hypernodes representing sets of subgoals, and edges representing theorems, as illustrated in Fig. 13. The description of the backward search algorithm is provided in Alg. 3. The function *get_expandable*() traverses the search tree based on pre-defined strategies, returning hypernodes 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 subgoals 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 3 Backward search

Input: *super_tree*: a super tree with the problem goal as the root super node.
Output: *theorem_seqs*: list, theorem sequence for problem solving.
Initialize an list *theorem_seqs*
super_node ← *super_tree.get_expandable*()
while *super_node* is not *None* **do**
  for $i = 1$ to *len*(*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
      l ← *node.expand*()
      if $l = 0$ **then**
        node.state ← FAILED
        break
      else
        node.state ← EXPANDED
      end if
    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

We conducted experiments on the FormalGeo7k. Search time and search step of different search methods and strategies can be found in Tab. 11.

Table 11. Experimental Results

metric	group	method	strategy	data						
				total	l_1	l_2	l_3	l_4	l_5	l_6
time	solved	FW	BFS	185.05	121.97	205.51	333.33	331.26	243.92	251.11
		FW	DFS	132.35	84.11	158.91	254.29	196.02	230.39	218.05
		FW	RS	92.21	57.38	89.55	177.34	189.74	166.96	199.31
		FW	BS	58.76	35.11	81.52	106.97	207.55	191.40	121.36
		BW	BFS	57.67	30.00	94.97	152.26	147.55	193.21	134.65
		BW	DFS	46.45	26.34	81.17	115.83	94.85	133.23	0.85
		BW	RS	65.82	42.79	101.71	156.27	145.50	202.37	3.37
		BW	BS	75.55	47.16	121.17	172.50	165.80	207.28	146.26
	unsolved	FW	BFS	574.94	435.34	583.68	615.29	624.07	662.96	680.99
		FW	DFS	548.81	418.77	545.90	594.45	605.73	641.54	655.50
		FW	RS	542.46	394.49	545.83	591.90	606.38	632.24	649.77
		FW	BS	355.74	277.24	345.70	358.92	430.08	446.34	451.65
		BW	BFS	579.24	549.25	572.43	592.65	588.48	596.18	597.49
		BW	DFS	581.50	550.13	577.12	593.69	592.71	595.34	598.39
		BW	RS	580.74	554.16	578.65	589.05	588.93	592.29	597.24
		BW	BS	513.86	339.10	508.03	566.64	574.09	585.96	578.42
step	solved	FW	BFS	58.44	21.85	68.67	119.56	144.80	161.44	822.18
		FW	DFS	87.16	33.15	95.95	225.57	232.25	257.59	727.17
		FW	RS	41.89	19.14	47.63	64.32	108.26	143.41	611.00
		FW	BS	18.64	10.99	25.97	37.86	46.19	48.44	541.00
		BW	BFS	15.14	20.68	4.80	5.60	2.31	1.00	1.00
		BW	DFS	5.17	4.96	5.83	5.86	3.94	1.00	1.00
		BW	RS	4.34	5.02	3.13	2.81	1.27	1.00	1.00
		BW	BS	1.67	1.49	1.94	3.08	1.25	1.00	1.00
	unsolved	FW	BFS	63.65	45.66	61.73	76.33	77.90	56.90	65.42
		FW	DFS	154.25	89.57	148.09	154.30	152.47	205.00	376.90
		FW	RS	66.38	65.22	68.86	75.37	56.18	62.72	62.72
		FW	BS	37.44	27.72	36.11	41.85	42.56	40.07	54.67
		BW	BFS	266.77	1148.76	195.45	21.53	10.16	4.49	59.20
		BW	DFS	517.90	2656.32	154.26	6.99	2.18	1.62	17.20
		BW	RS	181.74	830.49	113.67	15.27	2.14	1.34	10.16
		BW	BS	7.17	22.11	6.84	3.03	2.48	1.73	2.01