PAVOL JOZEF ŠAFÁRIK UNIVERSITY IN KOŠICE
FACULTY OF SCIENCE

Automated Generation of Planar Geometry Olympiad Problems

MASTER'S THESIS

Field of Study: Informatics
Institute: Institute of Computer Science
Supervisor: doc. RNDr. Stanislav Krajči, PhD.
Consultant: Mgr. Michal Rolínek, PhD.

Košice 2020 **Bc. Patrik Bak**

Acknowledgments

I would like to express my gratitude to: my supervisor Stanislav Krajči, for his support in granting me full academic freedom to pursue my own ideas; my consultant Michal Rolínek, whose expertise and advice took this text to an entirely new dimension; Pavel Šalom, for his immensely careful reading of the entire formal model and problem generation sections; and Dávid Uhrik, for countless remarks on every part of the text in terms of both content and style.

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Patrik Bak
Študijný program:	Informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	Informatika
Typ záverečnej práce:	Diplomová práca
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Automated generation of planar geometry olympiad problems

Názov SK: Automatické generovanie planimetrických úloh Matematickej olympiády

Cieľ: Design and implement software that is able to generate planar geometry olympiad problems by extending an initial configuration with new geometrical objects and subsequently finding non-trivial theorems in the generated configurations.

Literatúra:
[1] Rajiv Bagai, Vasant Shanbhogue, Jan M. Żytkow, and Shang-Ching Chou. Automatic theorem generation in plane geometry. In: International Symposium on Methodologies for Intelligent Systems. 1993. 415–424.
[2] Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. A deductive database approach to automated geometry theorem proving and discovering. Journal of Automated Reasoning. 2000, 25 (3), 219–246
[3] Andreas Poulos. A research on the creation of problems for mathematical competitions. Teaching of Mathematics. 2017, 20 (1)

Vedúci:	doc. RNDr. Stanislav Krajči, PhD.
Konzultant:	Mgr. Michal Rolínek, PhD.
Oponent:	RNDr. Ondrej Krídlo, PhD.
Ústav :	ÚINF - Ústav informatiky
Riaditeľ ústavu:	RNDr. Ondrej Krídlo, PhD.

Dátum schválenia: 02.04.2020

Abstract

We present a system that generates planar geometry problems suitable for mathematical competitions such as the International Mathematical Olympiad. Our solution consists of the following assets: (1) a novel, provably correct, problem generation algorithm, (2) bulk filtering algorithms of easy problems inspired by geometry theorem proving methods, primarily the deductive database method, (3) a novel problem quality ranking algorithm. A C# implementation of the system has been developed and made available on GitHub https://github.com/PatrikBak/GeoGen. The developed algorithms have been tested in many small and large-scale experiments that generated, rated, and sorted thousands of problems, nine of which are enclosed in the appendix. Five generated problems have been proposed to the International Mathematical Olympiad, whilst two generated problems have already been accepted to the Czech-Slovak mathematical contests.

Keywords: geometry problem generation, geometry theorem proving, mathematical olympiad problems, planar geometry problems

Abstrakt

V práci predstavujeme systém, ktorý generuje úlohy z rovinnej geometrie vhodné do matematických súťaží ako Medzinárodná Matematická Olympiáda. Naše riešenie obsahuje tieto komponenty: (1) nový, dokázateľne korektný algoritmus generovania úloh, (2) algoritmy hromadného filtrovania ľahkých úloh inšpirované metódami dokazovania geometrických viet, primárne metódou deduktívnej databázy, (3) nový algoritmus hodnotenia kvality úloh. K dispozícií je C# implementácia systému na GitHube https://github.com/PatrikBak/GeoGen. Vyvinuté algoritmy boli otestované v malých aj šikoroškálových experimentoch, kde sa vygenerovali, ohodnotili a usporiadali tisíce úloh, z nich deväť je v prílohe. Päť vygenerovaných úloh bolo navrhnutých na Medzinárodnú Matematickú Olympiádu a dve vygenerované úlohy boli už akceptované do Česko-Slovenských matematických súťaží.

Kľúčové slová: generovanie geometrických úloh, dokazovanie geometrických viet, úlohy matematickej olympiády, rovinné geometrické úlohy

Contents

Introduction . . . 8

1 Related work . . . 10

1.1 Geometry problem generation . . . 10

1.2 Geometry theorem proving . . . 10

2 Methods . . . 12

2.1 Failures of the naive approach . . . 12

2.1.1 Configuration generation . . . 12

2.1.2 Test case experiment . . . 13

2.1.3 Theorem finding . . . 14

2.1.4 Recognizing olympiad problems . . . 15

2.2 Formal model . . . 17

2.2.1 Construction . . . 17

2.2.2 Configuration . . . 20

2.2.3 Isomorphism . . . 26

2.3 Problem generation . . . 32

2.3.1 Configuration generation . . . 32

2.3.2 Theorem finding . . . 40

2.4 Filtering problems . . . 40

2.4.1 Theorem proving . . . 41

2.4.2 Complementary methods . . . 51

2.5 Problem ranking . . . 52

2.5.1 Symmetry rating . . . 53

2.5.2 Level rating . . . 54

2.5.3 Complementary rating . . . 55

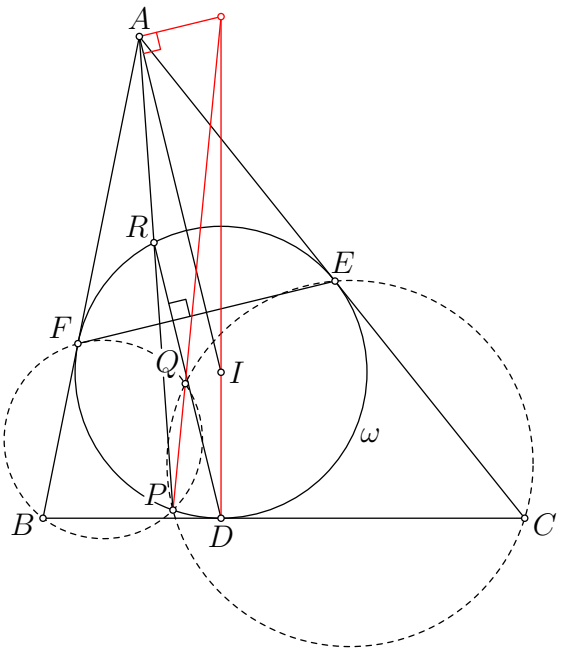
2.5.4 Configuring weights . . . 56

3 Experiments and results . . . 57

3.1 Implementation 57
3.1.1 Visualization 57
3.2 Large-scale experiments 57
3.2.1 Run-time 58
3.3 Results 58
Conclusion 60
Resumé 61
Bibliography 63
Appendix 65

Introduction

Geometry problems have always been an integral part of **mathematical competitions**. The most famous contest, the International Mathematical Olympiad (IMO)[1], takes place every year and consists of six difficult problems, two of which frequently belong to planar geometry. For instance, the hardest problem of the IMO 2019 can be seen in Figure 1.



Let I be the incenter of acute triangle ABC with $AB \neq AC$. The incircle ω of ABC is tangent to sides BC, CA, and AB at D, E, and F, respectively. The line through D perpendicular to EF meets ω again at R. Line AR meets ω again at P. The circumcircles of triangles PCE and PBF meet again at Q.

Prove that lines DI and PQ meet on the line through A perpendicular to AI.

Figure 1 The hardest problem of the International Mathematical Olympiad 2019.

Such problems are created by **experienced geometers**. This requires years of training in problem-solving and a great amount of creativity. A conventional way to design these problems is to examine various geometric situations by introducing new objects and proving relations among them.

Our ultimate goal is to **automate this process**, i.e. we aim to extend a given initial geometric situation to arrive at geometry problems. Ideally, these problems should already be difficult and interesting enough to be considered for mathematical competitions. The work of the geometer is then to solve and judge the results, and produce the final output.

[1] http://imo-official.org/

Solution. Our solution is introduced in Section 2.1, where we analyzed the problems and failures of the naive approach. The outcome of the analysis is a system consisting of three main components:

1. *Generation* of geometry problems (Section 2.3).
2. *Filtering* unsuitable problems (Section 2.4).
3. *Ranking* of the remaining problems (Section 2.5).

We also designed a nontrivial formal mathematical model (Section 2.2) used to describe the generation algorithm in depth and formally prove its correctness.

Results. The developed solution has been thoroughly tested in experiments described in Chapter 3. The conducted experiments resulted into thousands of intriguing problems, nine of these problems are to be seen in Appendix. Five generated problems have been proposed to the International Mathematical Olympiad. Two generated problems have already been accepted to the Czech-Slovak mathematical contests, specifically Figure 3.3.1 and Figure 3.3.2.

Contribution. Existing geometry research and software in this area focuses mainly on analyzing a given geometric situation, specifically discovering and proving theorems using various deductive approaches (see Chapter 1). Our contribution is the focus on problem generation, which brought many unprecedented challenges resulting in further technical contributions such as:

- *Memory efficient generation* of geometric situations.
- *Fast bulk theorem proving* of non-olympiad problems.
- *Problem quality estimation* of potential olympiad problems.

Chapter 1

Related work

1.1 Geometry problem generation

There have not been many attempts at the automation of geometry problem generation. An early attempt can be noted in [1]. The developed program works by generating relations among points and lines until there is an inconsistency that can be translated into a theorem by negating the last added relation. These inconsistencies are detected by an algebraic theorem prover. The system can rediscover a few well-known theorems such as Pappus's hexagon theorem [10, p. 67].

High school geometry problem generation is addressed in [19]. These are computational problems targeted to practice common techniques such as Pythagoras' theorem, triangle similarity, etc. The system cleverly combines various methods including generation of figures and automated deduction to generate diverse questions.

1.2 Geometry theorem proving

Unlike problem generation, theorem proving in geometry is a very developed area. The most successful methods turn out to be algebraic ones, such as Wu's method [3, 4, 20] or Gröbner bases method [2, 15]. These methods work with algebraic equations expressed via Cartesian coordinates. The main disadvantage is that they merely provide a yes or no answer, i.e. they do not produce traditional readable proofs.

Other approaches are called *coordinate-free* methods. The most notable ones are:

- The **area method** [8, 13] is built based on expressing geometric properties by means of lengths and areas. It is very powerful and can provide readable proofs of various easy theorems, such as Ceva's theorem [10, p. 4]. Though proofs of more complicated olympiad-like theorems would be tedious and unnatural.
- The **full angle method** [6, 5, 23], commonly known as *angle chasing*, which uses powerful properties of angles and cyclic quadrilaterals. Unlike the area method, this method practically always produces short natural proofs. It is not very powerful on its own as many (especially olympiad) problems require a combination of different methods (though rarely is angle chasing not among them).
- The **deductive database method** [7] uses a database of high-level inference rules that are used to infer new conclusions until a *fixpoint* is reached, where no new conclusion can be made. This method can entail aspects of the previous two methods (as can be seen in [7]). Produced proofs are very close to human-like proofs

since using high-level lemmas is the most natural way for geometers to think while solving.

Most harder geometry problems, such as olympiad ones, require an introduction of auxiliary points (or other objects) in their coordinate-free proofs. Doing this unwisely would lead to a combinatorial explosion of options, subsequently resulting in slow provers. Better approaches can be seen in [7, 17, 22].

Despite all the effort, non-algebraic proving methods still have a relatively smaller scope of solvable problems than algebraic provers. Therefore, most geometry software implement more than one method [11, 21, 24]

Some systems use a randomized theorem check approach, i.e. the conjectured theorems are verified numerically by means of analytic geometry. [14, 16]. This is the most practical way to find all theorems in a given geometric situation, due to the chance of producing incorrect results being next to zero. Also, the maximal running time can be guaranteed.

Chapter 2

Methods

Most olympiad geometry problems possess the following structure: (i) Description of a geometric situation; (ii) Theorem to be proven (for example Figure 1). Our focus will be entirely on problems of this type. The described structure is the prime motivation to separate the problem generation into:

1. Generation of configurations of geometry objects;
2. Finding and processing theorems in the generated configuration.

The upcoming sections will build on this idea and explain the developed solutions in more detail.

2.1 Failures of the naive approach

Before delving into actual solutions, we shall address the issues of the naive approach. They will inherently serve as the foundation for the final astute algorithms.

2.1.1 Configuration generation

The core idea is to extend a configuration with new objects to obtain new configurations. These new objects are added via geometry constructions such as Midpoint. This process is then repeated with newly generated configurations. The number of iterations is fixed prior to each generation. See Figure 2.1.1 for an illustration.

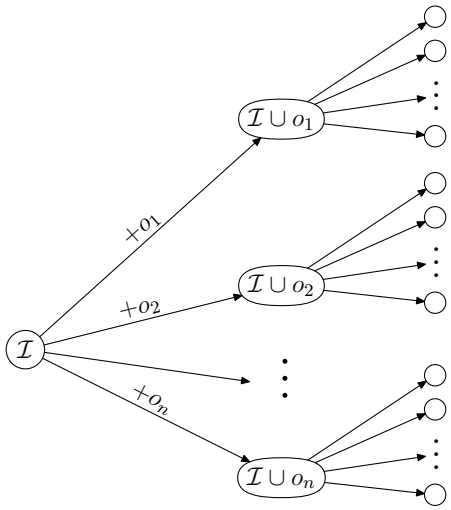


Figure 2.1.1 A graph of a generation process that generates geometric configurations. It starts with an initial configuration $\mathcal{I}$. This is then extended by objects $o_1, o_2, \ldots, o_n$. The obtained configurations are iteratively extended again in this manner, to a certain depth fixed prior to the generation.

Isomorphism. The described straightforward approach leads to generating *isomorphic* configurations. These are, informally speaking, configurations where one can be formed from the other by relabeling (see Figure 2.1.2).

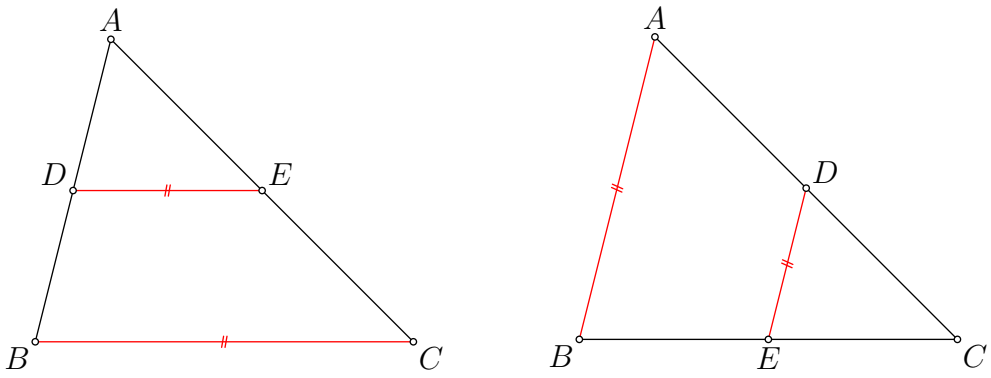


Figure 2.1.2 An example of isomorphic problems. On the left figure, points D, E are the midpoints of AB, AC, respectively, and we are to prove $BC \parallel DE$. Relabeling $(A, B, C) \mapsto (C, A, B)$ leads to the problem on the right figure.

2.1.2 Test case experiment

In the upcoming parts, we will illustrate the gravity of generated configurations, and later on the number of theorems. To that end, we designed a test case experiment that will perform the generation algorithm from Section 2.1.1. The initial configuration is a triangle with no additional objects. The used constructions are displayed in Table 2.1.1. The initial triangle will be extended by at most 4 objects.

CircleWithDiameter	Orthocenter
Excenter	ParallelLine
Excircle	ParallelLineToLineFromPoints
ExternalAngleBisector	ParallelogramPoint
Incenter	PerpendicularBisector
Incircle	PerpendicularLine
InternalAngleBisector	PerpendicularLineAtPointOfLine
IntersectionOfLineAndLineFromPoints	PerpendicularLineToLineFromPoints
IntersectionOfLines	PerpendicularProjection
IntersectionOfLinesFromPoints	PerpendicularProjectionOnLineFromPoints
Median	PointReflection
Midpoint	ReflectionInLine
MidpointOfArc	ReflectionInLineFromPoints
MidpointOfOppositeArc	TangentLine
OppositePointOnCircumcircle	

Table 2.1.1 Typical olympiad constructions used in the test case experiment Section 2.1.2, and also in the large-scale experiments from Chapter 3.

A typical olympiad problem consists of a triangle and 4-7 additional objects. Table 2.1.2 compares the combinatorial explosion, with and without isomorphism handling, when generating triangle problems with at most 4 additional objects (iterations). The results suggest that isomorphism ought to be taken into account.

Iterations	Non-isomorphic configurations	Run time	All configurations	Run time
1	22	0.4 sec	142	0.1 sec
2	1,377	0.7 sec	6,878	1.7 sec
3	241,073	2.2 min	1,392,396	8.2 min
4	$\approx$ 20,000,000	$\approx$ 20 hours	$\geq$ 100,000,000	$\geq$ 4 days

Table 2.1.2 Quantities of generated configurations in the test case experiment described in Section 2.1.2. The table shows that it is vital to handle configuration isomorphism (Figure 2.1.2).

Memory usage. Table 2.1.2 points towards the need to handle generation in a memory-efficient manner. Empirical measurements show that by storing intermediate results we would reach 10GB memory usage within approximately 45 minutes. Such an approach would not allow for long-running generations, like those described in Section 3. Our solution handles the generation of non-isomorphic configurations with storing linearly many configurations in the number of iterations, which in practice is at most 5. The details of this nontrivial algorithm will be presented in Section 2.3.1.

2.1.3 Theorem finding

There is a very limited range of types of geometry theorems that appear in olympiad problems. Our focus will be on those listed in Table 2.1.3.

Collinear points	Parallel lines	Tangent circles
Concyclic points	Concurrent lines	Line tangent to circle
Equal line segments	Perpendicular lines	Point lies on line or circle

Table 2.1.3 The types of investigated theorems throughout the thesis.

As we can see in Table 2.1.4, the number of theorems in configurations can very easily get too high to be handled manually.

Iterations	Configurations	Configurations with a theorem	All theorems
1	22	20	64
2	1,377	1,317	4,224
3	241,073	230,601	872,442

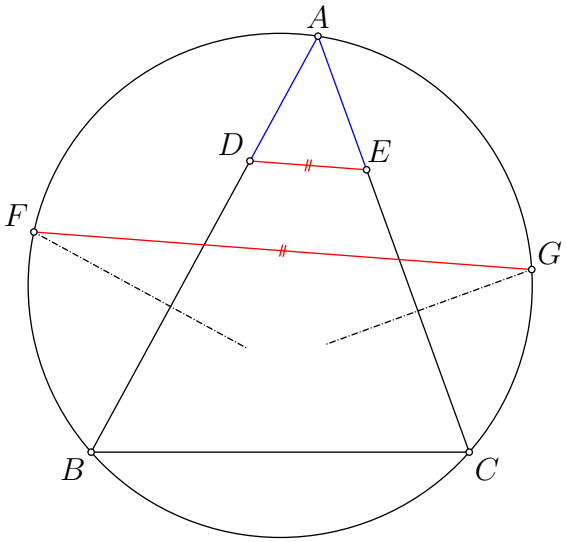
Table 2.1.4 Quantities of configurations and theorems in the performed test case experiment from Section 2.1.2. Only the theorems that use the last object of a configuration are counted in, thus excluding apparent duplicates.

2.1.4 Recognizing olympiad problems

The ultimate goal is to produce olympiad problems. The notion of a problem being olympiad is very subjective, but problem selection committees generally anticipate these three main features:

1. *Difficulty.* Olympiad problems are normally unapproachable for most high-school students lacking former training.
2. *Originality.* Problems proposed at international competitions must be dissimilar to previously proposed problems.
3. *Attractiveness.* A general appeal is difficult to describe. However, one significant beauty aspect present in most olympiad problems is *symmetry*.

   A problem starting with a triangle is recognized as symmetrical if there is a relabeling of two of the triangle's vertices that represents the same problem (see Figure 2.1.3).



Let Γ be the circumcircle of acute-angled triangle ABC. Points D and E lie on segments AB and AC, respectively, such that $AD = AE$. The perpendicular bisectors of BD and CE intersect the minor arcs AB and AC of Γ at points F and G, respectively.

Prove that the lines DE and FG are parallel (or are the same line).

Figure 2.1.3 First problem of the International Mathematical Olympiad 2018. This problem is symmetric, because relabeling $(A, B, C) \mapsto (A, C, B)$, yielding $(D, E, F, G) \mapsto (E, D, G, F)$, is the same problem.

Uninteresting problems. Many theorems from Table 2.1.4 are not suitable for mathematical competitions. Two types of such theorems are easily recognizable:

1. *Construction-related theorems* follow from the properties related to a construction, e.g. if A, B, C are points and D is the point opposite to A on the circumcircle of ABC, then trivially A, B, C, D are concyclic, and a little less trivially $AB \perp BD$ and $AC \perp CD$ (see Figure 2.1.4).
2. *Simplifiable theorems* are the ones containing an object that is not necessary to state the theorem, e.g. see Figure 2.1.5.

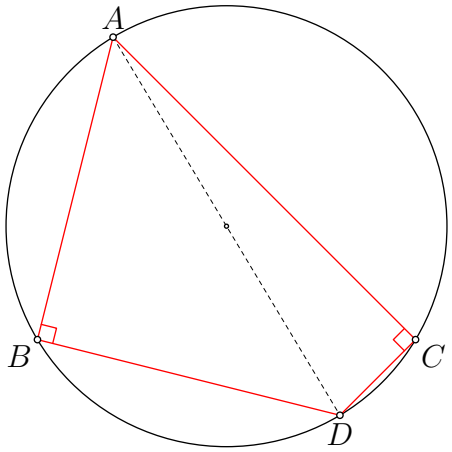


Figure 2.1.4 Thales's theorem. It is a *construction-related* theorem, since D can be viewed as the point opposite to A on the circumcircle of ABC.

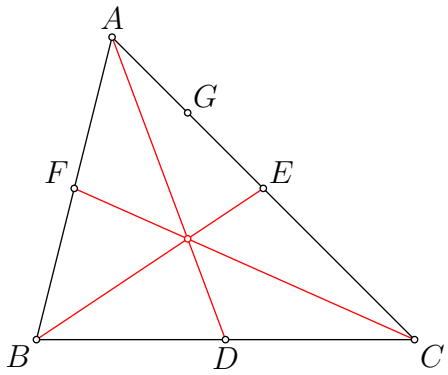


Figure 2.1.5 The median concurrency theorem with an unnecessary point G, the midpoint of AE. Such a situation is called a *simplifiable theorem*.

In Table 2.1.5 we can see the distribution of construction-related and simplifiable theorems among all the theorems from the examined test case experiment. The data shows that there are still far too many theorems processable by hand.

Iterations	Construction-related theorems	Simplifiable theorems	Remaining theorems
1	44	0	0
2	2,839	69	1,118
3	502,908	126,031	240,503

Table 2.1.5 The distribution of easily recognizable construction-related (Figure 2.1.4) and simplifiable (Figure 2.1.5) theorems among the theorems from Table 2.1.4.

Asymmetry exclusion. Restricting ourselves to generating only symmetric problems partially addresses the demand for *attractiveness* in olympiad problems. Table 2.1.6 shows this requirement significantly decreases the number of theorems to process. On the other hand, the need for further methods remains apparent.

Iterations	Symmetric theorems	Asymmetric theorems
1	0	0
2	203	915
3	11,718	228,785

Table 2.1.6 The distribution of symmetric theorems among the theorems from Table 2.1.5 that are neither construction-related, nor simplifiable.

Exclusion of easy problems. The main challenge lies in recognizing easy problems among those that are neither construction-related, nor simplifiable. A significant

amount of them do not comply with the difficulty demands, which is noticeable due to the following observation:

A triangle with 3 points extended by 3 objects yields around 11,000 symmetric theorems (see Table 2.1.6). These contain too few points to be considered *difficult*, which means they are not olympiad. Now, by extending any configuration containing a sub-triangle with 3 points we will encounter these 11,000 theorems again. In many cases, they are all symmetric and cannot be simplified.

This analysis outlines one of many reasons for the inevitability of proposing sophisticated methods for finding difficult theorems.

Solution. Our system outlined in Section 2.4.1 is able to filter out approximately 95.6% of easy symmetry theorems in a very low computational time, as can be seen in Table 2.1.7.

The remaining 512 theorems are subjected to the method that *merges* similar results (the second method in Section 2.4.2), which leaves 250 theorems, i.e. a reduction by another approximately 51.1%.

For the sake of completeness, let us mention that the theorems unfiltered by either of the methods are sorted via the ranking system described in Section 2.5.

Iterations	Symmetric theorems	Unfiltered theorems	Run time
1	0	0	0.3 sec
2	203	0	2.4 sec
3	11,718	512	2.8 min

Table 2.1.7 The results of our theorem-proving-based filtering system from Section 2.4 on the symmetric theorems from the performed test case experiment described in Section 2.1.2.

2.2 Formal model

In this section we will provide a formal description of informally introduced notions in Section 2.1. This will allow for a precise description and a correctness proof of the configuration generation algorithm in Section 2.3.

2.2.1 Construction

As a first major step, we will provide a formal definition of a *construction*. To begin, let us introduce object types. In the entire thesis, we will work only with three types of them: points, lines, and circles.

Definition (Object Type). An *object type* is either of the three constants: Point, Line, Circle.

The focus of the thesis is on planar problems, hence we shall define our objects in $\mathbb{R}^2$.

Definition (Objects). By Objects(Point), Objects(Line), Objects(Circle), we denote the sets of all $\mathbb{R}^2$ points, lines, circles, respectively.

Constructions expect specific inputs, for example Incenter takes three distinct points making a Triangle. Generally, an input for a construction is a tuple of objects with certain types. Let us define a set of such tuples:

Definition (Geometry Object Base). Let $T = (t_1, \ldots, t_n)$ be a non-empty sequence of object types. A *geometry object base* $\mathcal{B}_T$ is a subset of Objects$(t_1) \times \cdots \times$ Objects(t_n). The sequence T will be called the *object types* of $\mathcal{B}_T$.

Example 2.2.1. Consider the following geometry object bases:

- Triangle is the set of all triples of distinct non-collinear points, therefore a geometry object base with object types (Point, Point, Point).
- TwoConcentricCircles is the set of pairs of circles sharing a center, and so the object types here are (Circle, Circle).
- LineAndTwoPoints is the set of triples (l, p_1, p_2), where l is a line and p_1, p_2 are points, hence the object types are (Line, Point, Point).

Geometry object bases have their symmetries, for example if points (A, B, C) make a triangle, then (C, A, B) also make a triangle. These reorderings must also match the designated types, for example if (l, P, Q) is a pair of a line and two points, then we cannot exchange the line with either point.

Definition (Invariant Permutation). Let $\mathcal{B}_T$ be a geometry object base with object types $T = (t_1, \ldots, t_n)$. A permutation σ of the set $\{1, \ldots, n\}$ is called an *invariant permutation of* $\mathcal{B}_T$ if $(t_{\sigma(1)}, \ldots, t_{\sigma(n)}) = (t_1, \ldots, t_n)$ and also $(x_1, \ldots, x_n) \in \mathcal{B}_T$ if and only if $(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) \in \mathcal{B}_T$.

Example 2.2.2. Consider the following geometry object bases:

- Triangle stays a triangle after any reordering of its points, i.e. all 6 possible permutations are invariant ones,
- RightTriangle having the right angle at its first vertex can exchange the last two points, i.e. it has two invariant permutations $(1, 2, 3)$ and $(1, 3, 2)$.
- LineAndTwoPoints, which is Objects(Line) $\times$ Objects(Point) $\times$ Objects(Point), also has the two invariant permutations $(1, 2, 3)$ and $(1, 3, 2)$.

Theorem 2.2.1. Let $\mathcal{B}_T$ be a geometry object base and let S be the set of all of its invariant permutation. Then S with composition is a group.

Proof. Let $T = (t_1, \ldots, t_n)$. We need to verify the group axioms:

1. (Closure) Let σ_1 and σ_2 be two invariant permutations of $\mathcal{B}_T$. Then $\sigma_1 \circ \sigma_2$ is a permutation of $\{1, \ldots, n\}$. We will show that it is an invariant one.

By the definition of an invariant permutation:

$$(t_{(\sigma_1 \circ \sigma_2)(1)}, \ldots, t_{(\sigma_1 \circ \sigma_2)(n)}) = (t_{\sigma_2(1)}, \ldots, t_{\sigma_2(n)}) = (t_1, \ldots, t_n).$$

Similarly $(x_1, \ldots, x_n) \in \mathcal{B}_T$ if and only if $(x_{\sigma_2(1)}, \ldots, x_{\sigma_2(n)}) \in \mathcal{B}_T$, which holds if and only if $(x_{(\sigma_1 \circ \sigma_2)(1)}, \ldots, x_{(\sigma_1 \circ \sigma_2)(n)}) \in \mathcal{B}_T$.

2. (Associativity) Let σ_1, σ_2, σ_3 be invariant permutations. Based on the proven closure, permutations $\sigma_1 \circ (\sigma_2 \circ \sigma_3)$ and $(\sigma_1 \circ \sigma_2) \circ \sigma_3$ are also invariant ones. Clearly, they are equal, because permutation composition is associative.
3. (Identity) Obviously, the identity permutation i on $\{1, \ldots, n\}$ is an invariant permutation of $\mathcal{B}_T$, and for any permutation σ it holds that $\sigma \circ i = i \circ \sigma = \sigma$.
4. (Inverse) Let σ be an invariant permutation. We will prove σ^{-1} is also an invariant permutation. We have

$$(t_{\sigma^{-1}(n)}, \ldots, t_{\sigma^{-1}(n)}) = (t_{(\sigma \circ \sigma^{-1})(n)}, \ldots, t_{(\sigma \circ \sigma^{-1})(n)}) = (t_1, \ldots, t_n),$$

and

$$(x_1, \ldots, x_n) = (x_{(\sigma \circ \sigma^{-1})(1)}, \ldots, x_{(\sigma \circ \sigma^{-1})(n)}) \in \mathcal{B}_T,$$

which holds if and only if $(x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(n)}) \in \mathcal{B}_T$. □

After having defined a geometry object base, we may use it as a domain for constructions. Naturally, a construction is a function that produces new objects from given ones, for example, ReflectionInLine takes a point and a line and produces the reflection of the point in the line.

Definition (Construction). Let $T = (t_1, \ldots, t_n)$ be a non-empty sequence of object types. A *construction* $\mathcal{F}_T$ is a function, which maps a geometry object base to a set Objects(o), where o is an object type. The sequence T will be called the *input types of* $\mathcal{F}_T$ and o will be called the *output type of* $\mathcal{F}_T$.

Example 2.2.3. Consider the following constructions:

- Midpoint has input types (Point, Point), an output type Point, and it is defined for a geometry base, which consist of all pairs of points (P, Q). Moreover, it holds that $\mathsf{Midpoint}(P, Q) = \frac{1}{2}(P + Q)$.
- IntersectionOfLines has input types (Line, Line), an output type Line, it is defined for a geometry base of line pairs (k, l) such that $k \nparallel l$, and IntersectionOfLines(k, l) is the only intersection point of the lines k and l.
- ReflectionInLine has input types (Point, Line), an output type Point, and it is defined for pairs (p, l), where p is a point and l is a line.

Rather than in the analytic interpretation of constructions, we will be interested in their symmetries. Specifically, under which reorderings of the input arguments a construction yields the same object. For example, InternalAngleBisector(B, A, C) is the same

as InternalAngleBisector(C, A, B), but not as InternalAngleBisector(A, B, C). While reordering arguments, we need to assure that the types correspond. For example, we cannot reorder arguments of ReflectionInLine(P, l). All of that is incorporated in the following definition:

Definition (Invariant Permutation). Let $\mathcal{F}_T$ be a construction with $T = (t_1, \ldots, t_n)$. A permutation σ of the set $\{1, \ldots, n\}$ is called an *invariant permutation of* $\mathcal{F}_T$ if σ is an invariant permutation of $\mathrm{dom}(\mathcal{F}_T)$ and $\mathcal{F}_T(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) = \mathcal{F}_T(x_1, \ldots, x_n)$.

Example 2.2.4. Consider the following constructions:

- Midpoint takes two points in any order, i.e it has two invariant permutations $(1, 2)$ and $(2, 1)$,
- PointReflection expects two points and their order matters, i.e. it has only one invariant permutation $(1, 2)$,
- IntersectionOfLineAndLineFromPoints assumes a line l and two points A and B, producing the intersection point of the lines l and AB, i.e. the two points can be interchanged, but not with l, which gives exactly two invariant permutations $(1, 2, 3)$ and $(1, 3, 2)$,
- IntersectionOfLinesFromPoints needs four points A, B, C, D and defines the intersection point of lines AB and CD. Clearly, we can exchange points A and B or C and D, but also exchange the pairs (A, B) and (C, D). One can see this leads to 8 invariant permutations:

$$(1, 2, 3, 4),\ (1, 2, 4, 3),\ (2, 1, 3, 4),\ (2, 1, 4, 3),$$
$$(3, 4, 1, 2),\ (3, 4, 2, 1),\ (4, 3, 1, 2),\ (4, 3, 2, 1).$$

Theorem 2.2.2. Let $\mathcal{F}_T$ be a construction and let S be the set of all of its invariant permutations. Then S with composition is a group.

Proof. The proof is entirely analogous to that of Theorem 2.2.1 (besides a few additional steps with $\mathcal{F}_T$). Details are omitted. □

2.2.2 Configuration

The main idea how to represent configurations is by means of graph theory, specially directed acyclic graphs with a fixed order of their vertices and edges:

Definition (Ordered Graph). Suppose that G is a directed graph with ordered vertices $v_1, \ldots, v_m$, where for each vertex v_i there exists an ordering π_i of its direct successors. We call G an *ordered graph* if it satisfies:

1. There exists a positive integer $n \leq m$ such that the first n vertices have no outgoing edges.
2. For every edge from v_i to v_j it holds that $i > j$.

The first n vertices will be called *leaves*.

Remark. Further on, we will not access π_i directly, instead we will write the direct successors in the order induced by π_i, referring to this sequence as *ordered successors* (leaving out "direct" for brevity).

As mentioned before, we will use an ordered graph to represent configurations. To obtain a good understanding of the motivation, consider the following example: ABC is a triangle, l is the perpendicular bisector of BC, D is the reflection of A in l. To model this formally as an ordered graph, we will do this:

- The ordered vertices of the graph will be A, B, C, l, D.
- The vertices A, B, C will be the leaves. The ordered direct successors of l will be B and C, whereas the ordered direct successors of D will be A and l.
- The graph will need to be *vertex-labeled* to preserve what construction was used to define each non-leaf object. In our example, l was defined as PerpendicularBisector and D was defined via ReflectionInLine.
- A configuration will be defined with respect to a geometry object base. In our example it is a Triangle.
- Vertices will be labeled with object types. The types of leaves will be inferred from the object base. In our example, all three leaves A, B, C will have a type Point. The types of non-leaves will be the output type of the construction with which they are labeled. In our example, l will be a Line and D will be a Point.
- The types of ordered successors of a vertex must correspond to the input types of the particular construction, so that we cannot define ReflectionInLine(l, A).

Definition (Configuration). Let $\mathcal{B}_T$ be a geometry object base with $T = (t_1, \ldots, t_n)$. A *configuration* $\mathcal{C}_{\mathcal{B}_T}$ is a vertex-labeled ordered graph G satisfying:

1. Each vertex is labeled with an object type.
2. It has exactly n leaves $v_1, \ldots, v_n$ labeled with respective types $t_1, \ldots, t_n$.
3. Each non-leaf vertex labeled with a type t and ordered successors $v'_1, \ldots, v'_m$ with respective types $T' = (t'_1, \ldots, t'_m)$ is also labeled with a construction $\mathcal{F}_{T'}$ with an output type t.

The vertices of G will be called *objects of* $\mathcal{C}_{\mathcal{B}_T}$. The leaves of G will be called *base objects of* $\mathcal{C}_{\mathcal{B}_T}$, whereas the other vertices will be called *constructed objects of* $\mathcal{C}_{\mathcal{B}_T}$.

Example 2.2.5. Consider the following configurations:

1. Let l be a line and A, B be points. Let P be the intersection point of AB and l, M the midpoint of AB, and N the reflection of P in M.

   To represent this situation, we have $\mathcal{B}_T$ = LineAndTwoPoints and a configuration $\mathcal{C}_{\mathcal{B}_T}$ as the left graph on Figure 2.2.1 with ordered vertices l, A, B, P, M, N.

The base objects l, A, B are labeled with respective types Line, Point, Point. The remaining objects P, M, N are constructed, and have respective constructions IntersectionOfLineAndLineFromPoints, Midpoint, PointReflection, therefore they all have a type Point.

2. Let $ABCD$ be a quadrilateral and M, N the midpoints of AB and CD, respectively. This can be represented as configuration $\mathcal{C}_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Quadrilateral}$ and ordered objects A, B, C, D, M, N. Note that the graph of the configuration has two components (see the right graph on Figure 2.2.1).

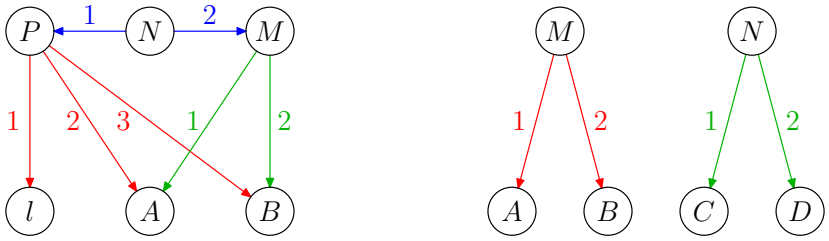


Figure 2.2.1 Graphs of the configurations from Example 2.2.5.

To precisely describe obvious equalities such as $\mathsf{Midpoint}(A, B) = \mathsf{Midpoint}(B, A)$, let us define object equivalence. Note that the definition will be recursive, to contain even more complex relations, including $\mathsf{Midpoint}(\mathsf{Midpoint}(A, B), \mathsf{Midpoint}(C, D)) = \mathsf{Midpoint}(\mathsf{Midpoint}(C, D), \mathsf{Midpoint}(B, A))$. Finally, we will make use of invariant permutations of a construction, which are essentially the underlying reason for objects being equivalent.

Definition (Object Equivalence). Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be configurations with the same sets of base objects. We say that two objects o_1 of $\mathcal{C}^1_{\mathcal{B}_T}$ and o_2 of $\mathcal{C}^2_{\mathcal{B}_T}$ are *equivalent*, denoted by $o_1 \sim o_2$, if either condition is met:

a) Both objects o_1 and o_2 are base objects and $o_1 = o_2$,

b) Both objects o_1 and o_2 are constructed objects labeled with the same construction c with respective ordered successors $s^1_1, \ldots, s^1_n$ and $s^2_1, \ldots, s^2_n$ such that there exists an invariant permutation σ of c satisfying $s^1_i \sim s^2_{\sigma(i)}$ for every $1 \leq i \leq n$.

Remark. Note that this recursive definition is correct, which easily follows from the following observation: Let $o^1_1, \ldots, o^1_{m_1}$ and $o^2_1, \ldots, o^2_{m_2}$ be the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively, and let $o_1 = o^1_a$ and $o_2 = o^2_b$. Since a configuration is an ordered graph, it must hold that $\{s^1_1, \ldots, s^1_n\} \subset \{o^1_1, \ldots, o^1_{a-1}\}$ and $\{s^2_1, \ldots, s^2_n\} \subset \{o^2_1, \ldots, o^2_{b-1}\}$.

Example 2.2.6. Consider configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ such that: Configuration $\mathcal{C}^1_{\mathcal{B}_T}$ has ordered objects A, B, C, O, M, where O is the circumcenter of ABC and M is the midpoint of AO (see the left graph on Figure 2.2.2). Configuration $\mathcal{C}^2_{\mathcal{B}_T}$ has ordered objects C, B, A, O', M', where O' is the circumcenter of BCA

and M' is the midpoint of $O'A$ (see the right graph Figure 2.2.2). All the pairs of equivalent objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are as follows:

1. Trivially $A \sim A$, $B \sim B$, $C \sim C$.
2. $O \sim O'$ for $\sigma = (3, 1, 2)$, and also $O' \sim O$ for $\sigma = (2, 3, 1)$.
3. $M \sim M'$ and $M' \sim M$ for $\sigma = (2, 1)$,

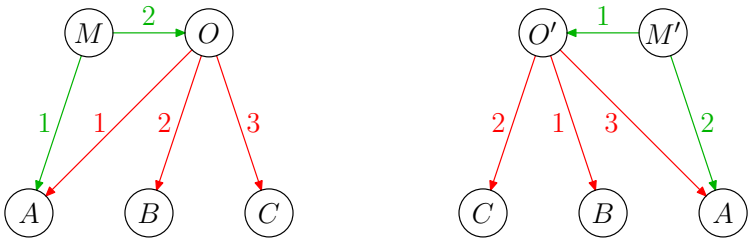


Figure 2.2.2 Graphs of the configurations from Example 2.2.6.

Lemma 2.2.1. For any object o of a configuration $\mathcal{C}_{\mathcal{B}_T}$, it holds that $o \sim o$.

Proof. Let $o_1, \ldots, o_m$ be the objects of $\mathcal{C}_{\mathcal{B}_T}$, where the first n of them are base objects. We will prove that $o_i \sim o_i$ for $1 \leq i \leq m$, by induction on i. For $1 \leq i \leq n$, the claim is obvious, because object equivalence becomes object equality. Assume that $o_j \sim o_j$ for all $1 \leq j \leq i$, where $i \geq n$. We need to show $o_{i+1} \sim o_{i+1}$.

Clearly, o_{i+1} is a constructed object with a construction c, let $s_1, \ldots, s_p$ be its ordered successors. Since $\{s_1, \ldots, s_p\} \subset \{o_1, \ldots, o_i\}$, we have $s_j \sim s_j$ for $1 \leq j \leq p$ by the induction hypothesis. Hence, by taking σ as the identity invariant permutation of c, we have $o_{i+1} \sim o_{i+1}$, which concludes the proof by induction. □

Lemma 2.2.2. Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be configurations with the same sets of base objects, o_1 and o_2 any objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively. If $o_1 \sim o_2$, then $o_2 \sim o_1$.

Proof. Let $o^1_1, \ldots, o^1_{m_1}$ and $o^2_1, \ldots, o^2_{m_2}$ be the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively, where the first n of them are base objects. We will prove for $1 \leq i \leq m_1$ and $1 \leq j \leq m_2$ that if $o^1_i \sim o^2_j$, then $o^2_j \sim o^1_i$. The proof will be done by induction on $\max\{i, j\}$.

If $\max\{i, j\} \leq n$, then o^1_i and o^2_j are base objects, i.e. their equivalence becomes their equality, and therefore the claim is obvious.

Assume that we have proven the statement for every o^1_i and o^2_j such that $\max\{i, j\} \leq t$, where $t \geq n$. We will now prove it for such i and j that $\max\{i, j\} = t+1$. Suppose that $o^1_i \sim o^2_j$. Since $\max\{i, j\} = t + 1$, either $i = t + 1 > n$, or $j = t + 1 > n$, which means that both objects o^1_i and o^2_j must be constructed objects with the same construction c. Let $s^1_1, \ldots, s^1_p$ and $s^2_1, \ldots, s^2_p$ be their respective ordered successors. Then there exists an invariant permutation σ of c such that $s^1_k \sim s^2_{\sigma(k)}$ for $1 \leq k \leq p$.

By Theorem 2.2.2, we know that σ^{-1} is also an invariant permutation of c. Clearly, it holds that $\{s_1^1, \ldots, s_p^1\} \subset \{o_1^1, \ldots, o_{i-1}^1\}$ and $\{s_1^2, \ldots, s_p^2\} \subset \{o_1^2, \ldots, o_{j-1}^2\}$, and since $\max\{i-1, j-1\} = t$, the induction hypothesis gives $s_{\sigma(k)}^2 \sim s_k^1$, for $1 \leq k \leq p$, hence $s_k^2 \sim s_{\sigma^{-1}(k)}^1$, for $1 \leq k \leq p$, which finally means $o_i^2 \sim o_j^1$. This concludes the proof by induction. □

Lemma 2.2.3. Let $\mathcal{C}_{\mathcal{B}_T}^1, \mathcal{C}_{\mathcal{B}_T}^2, \mathcal{C}_{\mathcal{B}_T}^3$ be configurations with the same sets of base objects, and o_1, o_2, o_3 any objects of $\mathcal{C}_{\mathcal{B}_T}^1$, $\mathcal{C}_{\mathcal{B}_T}^2$, $\mathcal{C}_{\mathcal{B}_T}^3$, respectively. If $o_1 \sim o_2$ and $o_2 \sim o_3$, then $o_1 \sim o_3$.

Proof. Let $o_1^1, \ldots, o_{m_1}^1$ and $o_1^2, \ldots, o_{m_2}^2$ and $o_1^3, \ldots, o_{m_3}^3$ be the objects of $\mathcal{C}_{\mathcal{B}_T}^1$, $\mathcal{C}_{\mathcal{B}_T}^2$, $\mathcal{C}_{\mathcal{B}_T}^3$, respectively, where the first n of them are base objects. We will prove for $1 \leq i \leq m_1$ and $1 \leq j \leq m_2$ and $1 \leq k \leq m_3$ that if $o_i^1 \sim o_j^2$ and $o_j^2 \sim o_k^3$ then $o_i^1 \sim o_k^3$. The proof will be done by induction on $\max\{i, j, k\}$.

If $\max\{i, j, k\} \leq n$, then o_i^1, o_j^2, o_k^3 are all base objects, i.e. their equivalence becomes their equality, and therefore the claim is obvious.

Assume that we have proven the statement for all o_i^1, o_j^2, o_k^3 such that $\max\{i, j, k\} \leq t$, where $t \geq n$. We will now prove it for such i, j, k that $\max\{i, j, k\} = t+1$. Suppose that $o_i^1 \sim o_j^2$ and $o_j^2 \sim o_k^3$. Since $\max\{i, j, k\} = t+1$, either $i = t+1 > n$, or $j = t+1 > n$, or $k = t+1 > n$ which means that all objects o_i^1, o_j^2, o_j^3 must be constructed objects with the same construction c. Let $s_1^1, \ldots, s_p^1$ and $s_1^2, \ldots, s_p^2$ and $s_1^3, \ldots, s_p^3$ be their respective ordered successors. Then there exist invariant permutations σ_1 and σ_2 of c such that $s_l^1 \sim s_{\sigma_1(l)}^2$ and $s_l^2 \sim s_{\sigma_2(l)}^3$ for $1 \leq l \leq p$.

By Theorem 2.2.2, $\sigma_2 \circ \sigma_1$ is also an invariant permutation of c. Clearly, it holds that $\{s_1^1, \ldots, s_p^1\} \subset \{o_1^1, \ldots, o_{i-1}^1\}$ and $\{s_1^2, \ldots, s_p^2\} \subset \{o_1^2, \ldots, o_{j-1}^2\}$ and $\{s_1^3, \ldots, s_p^3\} \subset \{o_1^3, \ldots, o_{k-1}^3\}$, and since $\max\{i-1, j-1, k-1\} = t$, the induction hypothesis gives that the relations $s_l^1 \sim s_{\sigma_1(l)}^2$ and $s_{\sigma_1(l)}^2 \sim s_{\sigma_2(\sigma_1(l))}^3$ imply $s_l^1 \sim s_{(\sigma_2 \circ \sigma_1)(l)}^3$ for every $1 \leq l \leq p$, which finalizes the proof of $o_i^1 \sim o_k^3$, therefore the proof by induction is concluded. □

One may have noticed that in our definition of a configuration, nothing prevented it from having equivalent objects simultaneously, for example having $\mathsf{Midpoint}(A, B)$ and $\mathsf{Midpoint}(B, A)$. In reality, this is not wanted, hence the following definition:

Definition (Correct Configuration). We say that a configuration is *correct* if it does not contain two distinct equivalent objects.

Example 2.2.7. Consider a configuration $\mathcal{C}_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ and ordered objects A, B, C, P, Q, where P is the circumcenter of ABC and Q is the circumcenter of CBA (see Figure 2.2.3) One can check that $\mathcal{C}_{\mathcal{B}_T}$ is not correct, because $P \sim Q$ for $\sigma = (3, 2, 1)$.

Notice that the order of some objects of a configuration does not have to matter, for example in a triangle ABC with its midpoints of AB and AC, the midpoints can be interchanged. It will become essential to recognize such cases. Moreover, we will take

into account object equivalence, i.e. a triangle ABC with midpoints of AB and AC is the same as a triangle ABC with its midpoints of CA and BA.

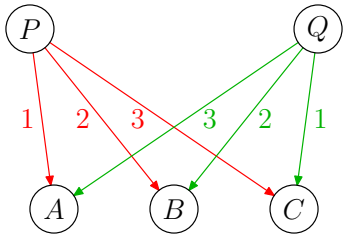


Figure 2.2.3 The graph of the configuration from Example 2.2.7.

Definition (Equivalent Configurations). We say that two configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ with the same sets of base objects are *equivalent*, denote by $\mathcal{C}^1_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$, if there exists a bijection between the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ such that if $o \mapsto o'$, then $o \sim o'$.

Example 2.2.8. Consider configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ such that: Configuration $\mathcal{C}^1_{\mathcal{B}_T}$ has ordered objects A, B, C, O, M, N, where O is the circumcenter of ABC, M is the midpoint of BC, N is the midpoint of AM (see the left graph on Figure 2.2.4). Configuration $\mathcal{C}^2_{\mathcal{B}_T}$ has ordered objects A, C, B, M', O', N', where M' is the midpoint of BC, O' is the circumcenter of CBA, N' is the midpoint of $M'A$ (see the right graph on Figure 2.2.4). These configurations are equivalent, as proven by the bijection $f\colon (A, B, C, O, M, N) \mapsto (A, B, C, O', M', N')$ mapping pairs of equivalent objects.

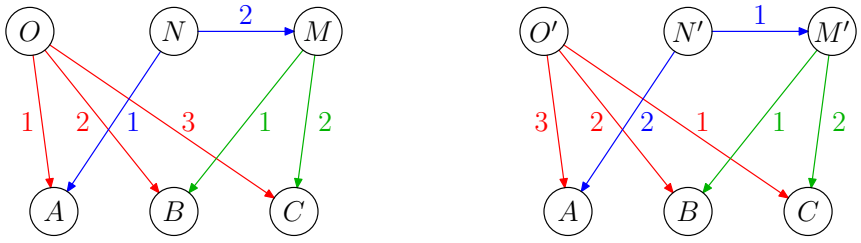


Figure 2.2.4 Graphs of the configurations from Example 2.2.8.

Theorem 2.2.3. Configuration equivalence is an equivalence relation.

Proof. Let us prove the required axioms:

- (Reflexivity) Let $\mathcal{C}_{\mathcal{B}_T}$ be a configuration. The identity bijection f on its objects satisfies $o \sim f(o)$, as per Lemma 2.2.1.
- (Symmetry) Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be configurations such $\mathcal{C}^1_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$. This means that there exists a bijection f between the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ such that for every object o of $\mathcal{C}^1_{\mathcal{B}_T}$ we have $o \sim f(o)$. By Lemma 2.2.2, we also have $f(o) \sim o$, therefore for every object o of $\mathcal{C}^2_{\mathcal{B}_T}$ we have $o \sim f^{-1}(o)$, which proves $\mathcal{C}^2_{\mathcal{B}_T} \equiv \mathcal{C}^1_{\mathcal{B}_T}$.

- (Transitivity) Let $\mathcal{C}^1_{\mathcal{B}_T}, \mathcal{C}^2_{\mathcal{B}_T}, \mathcal{C}^3_{\mathcal{B}_T}$ be configurations such $\mathcal{C}^1_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T} \equiv \mathcal{C}^3_{\mathcal{B}_T}$. This means that there exists bijection f_1 between the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ such that for every object o of $\mathcal{C}^1_{\mathcal{B}_T}$ we have $o \sim f_1(o)$, and also there exists bijection f_2 between the objects of $\mathcal{C}^2_{\mathcal{B}_T}$ and $\mathcal{C}^3_{\mathcal{B}_T}$ such that for every object o of $\mathcal{C}^2_{\mathcal{B}_T}$ we have $o \sim f_2(o)$. By Lemma 2.2.3, we have for every object o of $\mathcal{C}^1_{\mathcal{B}_T}$ that $o \sim f_2(f_1(o))$, which proves $\mathcal{C}^1_{\mathcal{B}_T} \equiv \mathcal{C}^3_{\mathcal{B}_T}$. □

To capture that some configuration is just an extension of another (equivalent) one, we define a subconfiguration.

Definition (Subconfiguration). Let $\mathcal{C}_{\mathcal{B}_T}$ be a configuration with objects $o_1, \ldots, o_m$. We say that a configuration $\mathcal{C}'_{\mathcal{B}_T}$ with objects $o'_1, \ldots, o'_{m'}$ and the same set of base objects as $\mathcal{C}_{\mathcal{B}_T}$ is a *subconfiguration of* $\mathcal{C}_{\mathcal{B}_T}$, if $m' \leq m$ and for every $1 \leq i \leq m'$ it holds that $o_i \sim o'_i$.

Example 2.2.9. Consider configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ such that: Configuration $\mathcal{C}^1_{\mathcal{B}_T}$ has ordered objects A, B, C, M, O, where M is the midpoint of BA, O is the circumcenter of CBA (see the left graph on Figure 2.2.5). Configuration $\mathcal{C}^2_{\mathcal{B}_T}$ has ordered objects A, B, C, M', O', N', where M' is the midpoint of AB, O' is the circumcenter of BAC, N' is the point reflection of B in C (see the right graph on Figure 2.2.5). Clearly $A \sim A$, $B \sim B$, $C \sim C$, $M \sim M'$, $O \sim O'$, therefore $\mathcal{C}^1_{\mathcal{B}_T}$ is a subconfiguration of $\mathcal{C}^2_{\mathcal{B}_T}$.

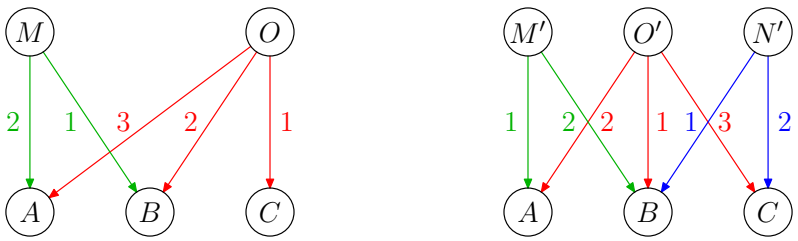


Figure 2.2.5 Graphs of the configurations from Example 2.2.9.

Lemma 2.2.4. Let $\mathcal{C}^1_{\mathcal{B}_T}, \mathcal{C}^2_{\mathcal{B}_T}, \mathcal{C}^3_{\mathcal{B}_T}$ be configurations with the same base objects. If $\mathcal{C}^1_{\mathcal{B}_T}$ is a subconfiguration of $\mathcal{C}^2_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ is a subconfiguration of $\mathcal{C}^3_{\mathcal{B}_T}$, then $\mathcal{C}^1_{\mathcal{B}_T}$ is a subconfiguration of $\mathcal{C}^3_{\mathcal{B}_T}$.

Proof. Let $o^1_1, \ldots, o^1_{m_1}$ and $o^2_1, \ldots, o^2_{m_2}$ and $o^3_1, \ldots, o^3_{m_3}$ be the objects of $\mathcal{C}^1_{\mathcal{B}_T}, \mathcal{C}^2_{\mathcal{B}_T}, \mathcal{C}^3_{\mathcal{B}_T}$, respectively. Clearly, $m_1 \leq m_2 \leq m_3$, and for every $1 \leq i \leq m_1$ we have $o^1_i \sim o^2_i$ and $o^2_i \sim o^3_i$, therefore by Lemma 2.2.3 also $o^1_i \sim o^3_i$. This proves that $\mathcal{C}^1_{\mathcal{B}_T}$ is a subconfiguration of $\mathcal{C}^3_{\mathcal{B}_T}$. □

2.2.3 Isomorphism

The objective of this section is to define *isomorphism*, which was introduced only informally in Section 2.1.1 (see Figure 2.1.2). In this informal explanation, we mentioned

relabeling. On a graph level, it means to reorder leaves, and based on that also adjust the definitions of non-leaves.

Definition (Graph Redefinition). Let G be an ordered graph with vertices $v_1, \ldots, v_m$, where the first n of them are leaves, and let σ be a permutation on $\{1, \ldots, n\}$. We define the *redefinition of G by σ* as an ordered graph with vertices $v'_1, \ldots, v'_m$ given by the following inductive construction:

1. The vertices $v'_1, \ldots, v'_n$ are equal to $v_{\sigma(1)}, \ldots, v_{\sigma(n)}$.
2. Assume that vertices $v'_1, \ldots, v'_i$ are already defined, where $n \leq i < m$. If vertex v_{i+1} is incident with ordered vertices $v_{a_1}, \ldots, v_{a_p}$, then the vertex v'_{i+1} will be incident with ordered vertices $v'_{a_1}, \ldots, v'_{a_p}$.

The fact that G' is a redefinition of G by σ will be denoted by $G \xrightarrow{\sigma} G'$. Also, we will say that a vertex v_i is *redefined to* v'_i.

Remark 1. In the second step, we are assuming that the vertices $v_{a_1}, \ldots, v_{a_p}$ have already been redefined. We can make this assumption based on the definition of an ordered graph, which ensures that the existing edges (v_{i+1}, v_{a_j}) have $a_j < i + 1$.

Remark 2. Note that the redefinition induces a mapping $v_i \mapsto v'_i$ for $1 \leq i \leq m$, which is a graph isomorphism between G and G'.

Example 2.2.10. Consider the left graph on Figure 2.2.6 with ordered vertices A, B, C, D, E, where A, B, C are leaves, D has ordered successors A, B, and E has ordered successors C, D. After performing the redefinition by a permutation $(2, 3, 1)$, the new graph (the right one on Figure 2.2.6) has ordered vertices B, C, A, D', E', where B, C, A are leaves, D' has ordered successors B, C, and E' has ordered successors A, D'.

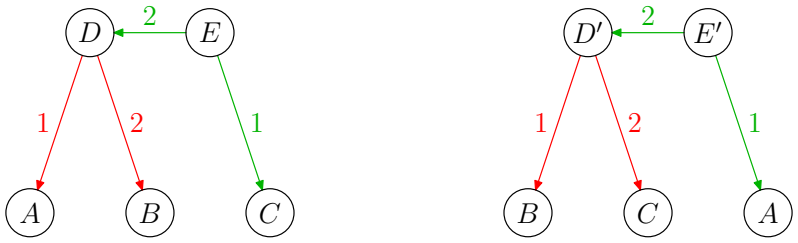


Figure 2.2.6 Graphs from the example from Example 2.2.10.

Lemma 2.2.5. Let G_1, G_2, G_3 be ordered graphs with n leaves, and σ_1, σ_2 be permutations on $\{1, \ldots, n\}$. Then the following claims are true:

a) $G_1 \xrightarrow{i} G_1$, where i is the identity permutation on $\{1, \ldots, n\}$.

b) If $G_1 \xrightarrow{\sigma_1} G_2$ and $G_2 \xrightarrow{\sigma_2} G_3$, then $G_1 \xrightarrow{\sigma_2 \circ \sigma_1} G_3$.

c) If $G_1 \xrightarrow{\sigma_1} G_2$, then $G_2 \xrightarrow{\sigma_1^{-1}} G_1$.

Proof. To realize these properties, notice that the redefinition of a graph is determined by the reordering of the leaves. Therefore:

a) Redefining the leaves by i does not affect them, which proves a).

b) Redefining the leaves by σ_1 and subsequently by σ_2 produces the same order of theirs as the redefinition by $\sigma_2 \circ \sigma_1$, which proves b).

c) Redefining the leaves by σ_1 and subsequently by σ_1^{-1} produces the same order of theirs as the redefinition by $\sigma_1^{-1} \circ \sigma_1 = i$, which proves c). □

Remark. Notice that this lemma means that the redefinition is a group action on the set of ordered graphs by the group of permutations on the leaf indices. Generally, the third property follows from the previous two.

The most important concept of isomorphism is applying a graph redefinition to a configuration, as it essentially is an ordered graph with labels. Since the leaves represent objects of a certain geometry object base, we cannot reorder them arbitrarily, but only according to its invariant permutation.

Definition (Configuration Redefinition). If $\mathcal{C}_{\mathcal{B}_T}$ is a configuration and σ is an invariant permutation of $\mathcal{B}_T$, then we define the *redefinition of* $\mathcal{C}_{\mathcal{B}_T}$ *by* σ as a configuration $\mathcal{C}'_{\mathcal{B}_T}$ such that $\mathcal{C}_{\mathcal{B}_T} \xrightarrow{\sigma} \mathcal{C}'_{\mathcal{B}_T}$ (as ordered graphs), and the labels are preserved under the induced graph isomorphism.

Example 2.2.11. Let $\mathcal{C}_{\mathcal{B}_T}$ be a configuration with $\mathcal{B}_T = \mathsf{RightTriangle}$, having ordered objects A, B, C, M, N, where M is the midpoint of AB, and N is the reflection of M in C (see the left graph on Figure 2.2.7). Its redefinition by an invariant permutation $(1, 3, 2)$ yields a configuration $\mathcal{C}'_{\mathcal{B}_T}$ with ordered objects A, C, B, M', N', where M' is the midpoint of BC, and N' is the reflection of M' in C (see the right graph on Figure 2.2.7).

Figure 2.2.7 Graphs of the configurations from Example 2.2.11.

Lemma 2.2.6. If $\mathcal{C}^1_{\mathcal{B}_T}, \mathcal{C}^2_{\mathcal{B}_T}, \mathcal{C}^3_{\mathcal{B}_T}$ are configurations and σ_1, σ_2 are invariant permutations of $\mathcal{B}_T$, then the following claims are true:

a) $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{i} \mathcal{C}^1_{\mathcal{B}_T}$, where i is the identity invariant permutation of $\mathcal{B}_T$.

b) If $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma_1} \mathcal{C}^2_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T} \xrightarrow{\sigma_2} \mathcal{C}^3_{\mathcal{B}_T}$, then $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma_2 \circ \sigma_1} \mathcal{C}^3_{\mathcal{B}_T}$.

c) If $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma} \mathcal{C}^2_{\mathcal{B}_T}$, then $\mathcal{C}^2_{\mathcal{B}_T} \xrightarrow{\sigma^{-1}} \mathcal{C}^1_{\mathcal{B}_T}$.

Proof. According to Theorem 2.2.1, if σ_1 and σ_2 are invariant permutations of $\mathcal{B}_T$, then σ_1^{-1} and $\sigma_2 \circ \sigma_1$ are also. That makes the redefinitions in the lemma correct. The claims are clearly true by Lemma 2.2.5 and the fact that redefinition does not affect labels. □

The following lemma shows, informally speaking, that a redefinition preserves object equivalence. This intuitively follows from the fact that every object is essentially defined via the base objects (through its successors), and the base objects are the same in equivalent configurations.

Lemma 2.2.7. Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be configurations with the same set of base objects. Assume that o_1 and o_2 are equivalent objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively. If we redefine both these configurations by the same invariant permutation of $\mathcal{B}_T$, then o_1 and o_2 will be redefined to equivalent objects.

Proof. Let $o^1_1, \ldots, o^1_{m_1}$ and $o^2_1, \ldots, o^2_{m_2}$ be the respective objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, where the first n of them are leaves, and let $o^{1'}_1, \ldots, o^{1'}_{m_1}$ and $o^{2'}_1, \ldots, o^{2'}_{m_2}$ be the respective objects of the configurations obtained by redefining $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ by an invariant permutation of $\mathcal{B}_T$. Also let $o_1 = o^1_a$ and $o_2 = o^2_b$.

We will prove the following claim: If $o^1_i \sim o^2_j$, for some $1 \leq i \leq a$ and $1 \leq j \leq b$, then $o^{1'}_i \sim o^{2'}_j$. The proof will be done by induction on $\max\{i, j\}$.

If $\max\{i, j\} \leq n$, then both objects o^1_i and o^2_j must be base ones, i.e. their equivalence implies their equality, and so after the redefinition, they will also be equal.

Assume that we have proven the statement for every o^1_i and o^2_j such that $\max\{i, j\} \leq t$, where $t \geq n$. We will now prove it for such i and j that $\max\{i, j\} = t+1$. Suppose that $o^1_i \sim o^2_j$. Since $\max\{i, j\} = t + 1$, either $i = t + 1 > n$, or $j = t + 1 > n$, which means that both objects o^1_i and o^2_j must be constructed objects with the same construction c. Let $s^1_1, \ldots, s^1_p$ and $s^2_1, \ldots, s^2_p$ be their respective ordered successors. Then there exists an invariant permutation σ of c such that $s^1_k \sim s^2_{\sigma(k)}$ for $1 \leq k \leq p$.

Clearly, $\{s^1_1, \ldots, s^1_p\} \subset \{o^1_1, \ldots, o^1_{i-1}\}$ and $\{s^2_1, \ldots, s^2_p\} \subset \{o^2_1, \ldots, o^2_{j-1}\}$. Also, we have $\max\{i - 1, j - 1\} = t$, therefore, by the induction hypothesis we have that equivalent objects s^1_k and $s^2_{\sigma(k)}$ are redefined to equivalent objects $s^{1'}_k$ and $s^{2'}_{\sigma(k)}$, for $1 \leq k \leq p$, and so the redefined objects $o^{1'}_i$ and $o^{2'}_j$ have their respective ordered successors $s^{1'}_1, \ldots, s^{1'}_p$ and $s^{2'}_1, \ldots, s^{2'}_p$ such that $s^{1'}_k \sim s^{2'}_{\sigma(k)}$ for $1 \leq k \leq p$, which concludes the proof of $o^1_i \sim o^2_j$, hence concludes the proof by induction. □

Lemma 2.2.8. A redefinition of a correct configuration is a correct configuration.

Proof. Assume that $\mathcal{C}_{\mathcal{B}_T}$ is a configuration, and $\mathcal{C}'_{\mathcal{B}_T}$ is its definition by a symmetric permutation σ of $\mathcal{B}_T$. We will show that if $\mathcal{C}_{\mathcal{B}_T}$ is correct, then $\mathcal{C}'_{\mathcal{B}_T}$ is correct, by contradiction. Assume that $\mathcal{C}'_{\mathcal{B}_T}$ is not correct, i.e. it contains two distinct equivalent objects o_1 and o_2. By Lemma 2.2.6c, $\mathcal{C}'_{\mathcal{B}_T} \xrightarrow{\sigma^{-1}} \mathcal{C}_{\mathcal{B}_T}$, and by Lemma 2.2.7, objects o_1 and o_2 are redefined to trivially distinct equivalent objects of $\mathcal{C}_{\mathcal{B}_T}$, which contradicts its correctness. □

Lemma 2.2.9. Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be equivalent configurations. A redefinition of these configurations by the same invariant permutation of $\mathcal{B}_T$ yields equivalent configurations.

Proof. Since $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are equivalent, there exists a bijection f mapping the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ to objects of $\mathcal{C}^2_{\mathcal{B}_T}$ equivalent to them. Therefore, configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ have the same number of objects, denote them by $o^1_1, \ldots, o^1_m$ and $o^2_1, \ldots, o^2_m$, respectively. After redefining them by the same invariant permutation, we obtain configurations $\mathcal{C}^{1'}_{\mathcal{B}_T}$ and $\mathcal{C}^{2'}_{\mathcal{B}_T}$ with respective objects $o^{1'}_1, \ldots, o^{1'}_m$ and $o^{2'}_1, \ldots, o^{2'}_m$.

Define the bijection f' between the objects of $\mathcal{C}^{1'}_{\mathcal{B}_T}$ and $\mathcal{C}^{2'}_{\mathcal{B}_T}$ such that if $f(o^1_i) = o^2_j$ for some $1 \leq i, j \leq m$, then $f'(o^{1'}_i) = o^{2'}_j$. It is easy to show that f' maps equivalent objects, since $f(o^1_i) = o^2_j$ gives $o^1_i \sim o^2_j$, and Lemma 2.2.7 means $o^{1'}_i \sim o^{2'}_j$, i.e. $o^{1'}_i \sim f'(o^{1'}_i)$. This concludes the proof of $\mathcal{C}^{1'}_{\mathcal{B}_T} \equiv \mathcal{C}^{2'}_{\mathcal{B}_T}$. □

Configuration isomorphism was informally portrayed as: Two configurations are isomorphic if we can relabel vertices of the first one to obtain a configuration same as the second one. Formally, relabeling will correspond to redefining and sameness to equivalence.

Definition (Isomorphic Configurations). We say that two configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are *isomorphic*, denoted by $\mathcal{C}^1_{\mathcal{B}_T} \simeq \mathcal{C}^2_{\mathcal{B}_T}$, if there exists an invariant permutation σ of $\mathcal{B}_T$ such that redefining $\mathcal{C}^1_{\mathcal{B}_T}$ by σ yields a configuration equivalent to $\mathcal{C}^2_{\mathcal{B}_T}$.

Remark 1. Observe that isomorphic configurations must have the same sets of base objects.

Remark 2. Notice that equivalent configurations are naturally isomorphic, because we may take σ as the identity invariant permutation.

Example 2.2.12. Consider configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ such that: Configuration $\mathcal{C}^1_{\mathcal{B}_T}$ has ordered objects A, C, B, O, H, M, where O is the circumcenter of BAC, H is the orthocenter of OBA, M is the midpoint of AC (see the left graph of Figure 2.2.8). Configuration $\mathcal{C}^2_{\mathcal{B}_T}$ has ordered objects B, C, A, M', O', H', where M' is the midpoint of BC, O' is the circumcenter of BCA, H' is the orthocenter of $CO'A$ (see the right graph of Figure 2.2.8).

If we redefine $\mathcal{C}^1_{\mathcal{B}_T}$ by an invariant permutation $\sigma = (2, 3, 1)$, then we will get a configuration $\mathcal{C}^3_{\mathcal{B}_T}$ with ordered objects C, B, A, O'', H'', M'', where O'' is the circumcenter of ACB, H'' is the orthocenter of $O''AC$, M'' is the midpoint of CB. Clearly, configurations $\mathcal{C}^3_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are equivalent, due to the bijection $f\colon (A, B, C, O'', H'', M'') \mapsto (A, B, C, O', M', N')$ mapping pairs of equivalent objects.

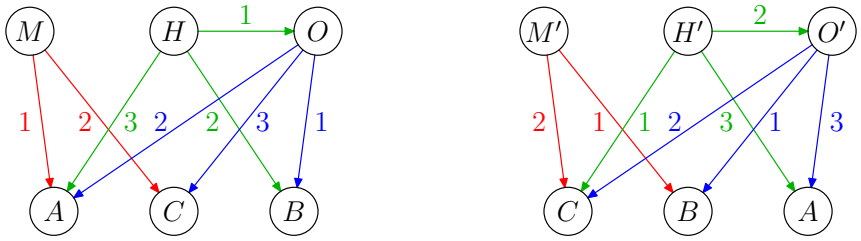


Figure 2.2.8 Graphs of the configurations from Example 2.2.12.

Theorem 2.2.4. Configuration isomorphism is an equivalence relation.

Proof. Let us verify the needed axioms:

- (Reflexivity) According to Lemma 2.2.6a, for any configuration $\mathcal{C}_{\mathcal{B}_T}$ it holds that $\mathcal{C}_{\mathcal{B}_T} \xrightarrow{i} \mathcal{C}_{\mathcal{B}_T}$, where i is the identity invariant permutation of $\mathcal{B}_T$, therefore we have $\mathcal{C}_{\mathcal{B}_T} \simeq \mathcal{C}_{\mathcal{B}_T}$.
- (Symmetry) Assume that $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are isomorphic configurations, i.e. there exists an invariant permutation σ of $\mathcal{B}_T$ such that $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma} \mathcal{C}^{2'}_{\mathcal{B}_T}$, where $\mathcal{C}^{2'}_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$. By Lemma 2.2.6c, $\mathcal{C}^{2'}_{\mathcal{B}_T} \xrightarrow{\sigma^{-1}} \mathcal{C}^1_{\mathcal{B}_T}$, and by Lemma 2.2.9, also $\mathcal{C}^2_{\mathcal{B}_T} \xrightarrow{\sigma^{-1}} \mathcal{C}^{1'}_{\mathcal{B}_T}$, where $\mathcal{C}^{1'}_{\mathcal{B}_T} \equiv \mathcal{C}^1_{\mathcal{B}_T}$, which means $\mathcal{C}^2_{\mathcal{B}_T} \simeq \mathcal{C}^1_{\mathcal{B}_T}$.
- (Transitivity) Assume that $\mathcal{C}^1_{\mathcal{B}_T} \simeq \mathcal{C}^2_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T} \simeq \mathcal{C}^3_{\mathcal{B}_T}$. Then there are invariant permutations σ_1 and σ_2 of $\mathcal{B}_T$ such that $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma_1} \mathcal{C}^{2'}_{\mathcal{B}_T}$, where $\mathcal{C}^{2'}_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$, and $\mathcal{C}^2_{\mathcal{B}_T} \xrightarrow{\sigma_2} \mathcal{C}^{3'}_{\mathcal{B}_T}$, where $\mathcal{C}^{3'}_{\mathcal{B}_T} \equiv \mathcal{C}^3_{\mathcal{B}_T}$. By Lemma 2.2.9, we have $\mathcal{C}^{2'}_{\mathcal{B}_T} \xrightarrow{\sigma_2} \mathcal{C}^{3''}_{\mathcal{B}_T}$, where $\mathcal{C}^{3''}_{\mathcal{B}_T} \equiv \mathcal{C}^{3'}_{\mathcal{B}_T}$. By Theorem 2.2.3, this means $\mathcal{C}^{3''}_{\mathcal{B}_T} \equiv \mathcal{C}^3_{\mathcal{B}_T}$, and by Lemma 2.2.6b also $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma_2 \circ \sigma_1} \mathcal{C}^{3''}_{\mathcal{B}_T}$, which together gives that $\mathcal{C}^1_{\mathcal{B}_T} \simeq \mathcal{C}^3_{\mathcal{B}_T}$. □

Theorem 2.2.5. If two configurations are isomorphic configurations and one of them is correct, then the other is correct too.

Proof. Assume that $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are isomorphic configurations and $\mathcal{C}^1_{\mathcal{B}_T}$ is correct. By the definition of configuration isomorphism, there exists an invariant permutation of $\mathcal{B}_T$ such that $\mathcal{C}^1_{\mathcal{B}_T} \xrightarrow{\sigma} \mathcal{C}^{2'}_{\mathcal{B}_T}$, where $\mathcal{C}^{2'}_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$. Since $\mathcal{C}^1_{\mathcal{B}_T}$ is correct, by Lemma 2.2.8 also $\mathcal{C}^{2'}_{\mathcal{B}_T}$ is correct. We will prove that $\mathcal{C}^2_{\mathcal{B}_T}$ is also correct, by contradiction.

If $\mathcal{C}^2_{\mathcal{B}_T}$ were not correct, then it would contain two distinct objects o_1 and o_2 such that $o_1 \sim o_2$. Since $\mathcal{C}^{2'}_{\mathcal{B}_T} \equiv \mathcal{C}^2_{\mathcal{B}_T}$, by Theorem 2.2.3 also $\mathcal{C}^2_{\mathcal{B}_T} \equiv \mathcal{C}^{2'}_{\mathcal{B}_T}$, therefore configuration $\mathcal{C}^{2'}_{\mathcal{B}_T}$ contains objects o'_1 and o'_2 such that $o_1 \sim o'_1$ and $o_2 \sim o'_2$. By Lemma 2.2.2, $o_1 \sim o'_1$ implies $o'_1 \sim o_1$. By Lemma 2.2.3, $o_1 \sim o_2$ and $o_2 \sim o'_2$ give $o_1 \sim o'_2$. By the same lemma, $o'_1 \sim o_1$ and $o_1 \sim o'_2$ give $o'_1 \sim o'_2$. Together with the obvious fact that o'_1 and o'_2 are distinct, this is is a contradiction with $\mathcal{C}^{2'}_{\mathcal{B}_T}$ being correct. □

2.3 Problem generation

In Section 2.1 we established the basic separation of problem generation into configuration generation and theorem finding. We claimed to have developed a generation algorithm of non-isomorphic configurations with sophisticated memory handling, which will finally be described in Section 2.3.1. The theorem finding methods are then explained in Section 2.3.2.

2.3.1 Configuration generation

The developed algorithm builds on the basic generation idea presented at the beginning of Section 2.1.1. We will still generate the graph in Figure 2.1.1, but in a thought-out order. Also, in the process we will dismiss several configurations that would lead to generating isomorphic ones.

Configuration extension algorithm. To perform generation, we need an algorithm to generate new configurations from a given one. As already established, this algorithm is applied to the initial configuration and also subsequently generated configurations.

Note that the algorithm works with a finite set of constructions fixed in advance. In the entire thesis, we use the constructions from Table 2.1.1.

The algorithm takes a configuration $\mathcal{C}_{\mathcal{B}_T}$ with objects $o_1, \ldots, o_m$ and:

1. For every construction c with input types $t_1, \ldots, t_n$ and every n-tuple of distinct objects $o'_1, \ldots, o'_n$ from $\{o_1, \ldots, o_m\}$ with corresponding types $t_1, \ldots, t_n$ defines a configuration $\mathcal{C}'_{\mathcal{B}_T}$, whose objects are $o_1, \ldots, o_{m+1}$, where o_{m+1} has ordered direct successors $o'_1, \ldots, o'_n$, and labels c and the output type of c.
2. From the generated configurations, only those that are *correct* are taken.
3. The remaining configurations are grouped into the sets of equivalent ones. From each set, one arbitrary configuration is taken.

Dismissal of isomorphic configurations. During the generation, it may happen that two configurations isomorphic to each other are generated. As highlighted in Section 2.1.2, this is not desired. We will present an algorithm that prevents this.

The main idea is to dismiss several configurations along the generation. In other words, define an *exclusion algorithm* that will decide whether a given generated configuration should be excluded or not. This algorithm will ensure that no two isomorphic configurations are generated, and also that unexcluded configurations are generable from the initial one.

Encoding function. The key idea of the exclusion algorithm is encoding objects and configurations as strings and using lexicographical order. We will gradually explain this technique. The first step is to define an *encoding function* that will be able to capture

equivalence of two objects, i.e. two objects will be equivalent if and only if they are encoded to the same string.

In the encoding, we will use the letters of the English alphabet and 3 additional special characters: left and right round brackets and a comma.

We will assume that the available constructions $c_1, \ldots, c_t$ have arbitrary distinct respective codes $c'_1, \ldots, c'_t$ not containing these special characters.

Let $\mathcal{C}_{\mathcal{B}_T}$ be a configuration with objects $o_1, \ldots, o_m$, where the first n of them are base objects. Then, we will recursively encode their objects as follows:

1. The base objects $o_1, \ldots, o_n$ have arbitrary distinct respective codes $o'_1, \ldots, o'_n$ not containing the special characters.
2. If objects $o_1, \ldots, o_i$ are already encoded, where $i \geq n$, then we will encode (constructed) object o_{i+1} as follows: Let c_j be its construction and $s_1, \ldots, s_k$ the codes of its direct successors. Consider the set of strings

$$S^\star = \{\ (s_{\sigma(1)}, \cdots, s_{\sigma(k)}) \quad | \quad \sigma \text{ is an invariant permutation of } c_j\ \},$$

where the round brackets and commas are literal characters. If σ' is the permutation realizing the lexicographically smallest element of $S^\star$, the code of the object o_{i+1} will be equal to:

$$c'_j(s_{\sigma'(1)}, \cdots, s_{\sigma'(k)})$$

Example 2.3.1. Consider a configuration $\mathcal{C}_{\mathcal{B}_T}$ with $\mathcal{B}_T = \mathsf{Triangle}$ and objects A, B, C, D, E such that D is the circumcenter of BAC and E is the midpoint of DA (see Figure 2.3.1). One way to encode these objects by the encoding function is as follows:

1. A, B, C have respective codes `A`, `B`, `C`.
2. D has code `Circumcenter(A,B,C)`.
3. E has code `Midpoint(A,Circumcenter(A,B,C))`.

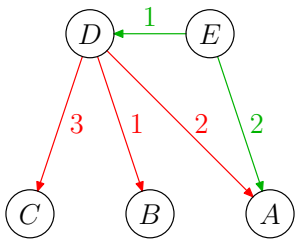


Figure 2.3.1 A graph of the configuration from Example 2.3.1.

Theorem 2.3.6. Two objects are equivalent if and only if they are encoded to the same string.

Proof. Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be configurations with the same base objects. Denote by $o^1_1, \ldots, o^1_{m_1}$ and $o^2_1, \ldots, o^2_{m_2}$ the objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively.

We will prove for $1 \leq i \leq m_1$ and $1 \leq j \leq m_2$ that $o_i^1 \sim o_j^2$ if and only the codes of o_i^1 and o_j^2 are the same. The proof will be done by induction on $\max\{i, j\}$.

If $\max\{i, j\} \leq n$, then o_i^1 and o_j^2 are base objects, which gives that their equivalence is the same as their equality, which is the same as the equality of their codes (since these codes are assumed to be distinct), thereby the claim is trivial.

Assume that we have proven the statement for every o_i^1 and o_j^2 such that $\max\{i, j\} \leq t$, where $t \geq n$. We will now prove it for such i and j that $\max\{i, j\} = t + 1$.

First, suppose that $o_i^1 \sim o_j^2$. We will prove that they must be encoded to the same string. Since $\max\{i, j\} = t + 1$, either $i = t + 1 > n$, or $j = t + 1 > n$, which means that both objects o_i^1 and o_j^2 must be constructed objects with the same construction c. Let $s_1^1, \ldots, s_p^1$ and $s_1^2, \ldots, s_p^2$ be their respective ordered successors. Since $o_i^1 \sim o_j^2$, there exists an invariant permutation σ of c such that $s_k^1 \sim s_{\sigma(k)}^2$ for $1 \leq k \leq p$.

Clearly, $\{s_1^1, \ldots, s_p^1\} \subset \{o_1^1, \ldots, o_{i-1}^1\}$ and $\{s_1^2, \ldots, s_p^2\} \subset \{o_1^2, \ldots, o_{j-1}^2\}$. Also we have $\max\{i - 1, j - 1\} = t$, therefore by the induction hypothesis we have that equivalent objects s_k^1 and $s_{\sigma(k)}^2$ have the same codes $s_k^{1'}$ and $s_{\sigma(k)}^{2'}$, for $1 \leq k \leq p$. Therefore, the sets $\{s_1'^1, \ldots, s_p'^1\}$ and $\{s_1'^2, \ldots, s_p'^2\}$ are equal and therefore also the sets $S^\star$ from step 2 corresponding to the objects $s_1^1, \ldots, s_p^1$ and $s_1^2, \ldots, s_p^2$ are the same, hence their lexicographically smallest elements are the same. Together with the fact that the objects o_i^1 and o_j^2 have the same construction, it is now clear that they must be encoded to the same string.

On the other hand, assume that objects o_i^1 and o_j^2 are encoded to the same string. We will prove that they are equivalent. The fact that either $i > n$ or $j > n$ means that at least one of them is constructed. Codes of constructed objects contain round brackets, whereas codes of base objects do not. Therefore, both objects o_i^1 and o_j^2 are constructed, i.e. their code is in form $c(s)$. The string c necessarily represents an encoded construction, because construction codes do not contain round brackets. Since distinct constructions have distinct codes, the objects o_i^1 and o_j^2 must have the same construction c.

The string s contains comma-separated values of object codes. Notice that these commas can be identified unambiguously as those commas that are not enclosed in round brackets. Denote these object codes by $s_1, \ldots, s_p$, and also denote by $s_1^1, \ldots, s_p^1$ and $s_1^2, \ldots, s_p^2$ the ordered successors of o_i^1 and o_j^2, respectively.

By the way the codes $s_1, \ldots, s_p$ were created, there are invariant permutations σ_1 and σ_2 of c such that the sequences $s_{\sigma_1(1)}^1, \ldots, s_{\sigma_1(p)}^1$ and $s_{\sigma_2(1)}^2, \ldots, s_{\sigma_2(p)}^2$ are encoded as $s_1, \ldots, s_p$, therefore for $1 \leq k \leq p$, it holds that the objects $s_{\sigma_1(k)}^1$ and $s_{\sigma_2(k)}^2$ are encoded to the same string. Clearly, it holds that $\{s_1^1, \ldots, s_p^1\} \subset \{o_1^1, \ldots, o_{i-1}^1\}$ and $\{s_1^2, \ldots, s_p^2\} \subset \{o_1^2, \ldots, o_{j-1}^2\}$. Also we have $\max\{i-1, j-1\} = t$, therefore by the induction hypothesis we have that $s_{\sigma_1(k)}^1 \sim s_{\sigma_2(k)}^2$, for $1 \leq k \leq p$. This gives $s_k^1 \sim s_{(\sigma_2 \circ \sigma_1^{-1})(k)}^2$,

for $1 \leq k \leq p$, which finally concludes the proof of $o_i^1 \sim o_j^2$, therefore the entire proof by induction. □

Exclusion algorithm. Let $\mathcal{I}_{\mathcal{B}_T}$ be an *initial configuration*. We are ready to define the algorithm that will find out whether a given configuration $\mathcal{C}_{\mathcal{B}_T}$ with objects $o_1, \ldots, o_n$ having $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration should be excluded during the generation. The exclusion algorithm goes as follows:

1. For every invariant permutation σ of $\mathcal{B}_T$, perform the redefinition of $\mathcal{C}_{\mathcal{B}_T}$ by σ. Let the obtained configurations be $\mathcal{C}^1_{\mathcal{B}_T}, \ldots, \mathcal{C}^t_{\mathcal{B}_T}$. For every $1 \leq k \leq t$, let S^k be the set of configurations equivalent to $\mathcal{C}^k_{\mathcal{B}_T}$ such that $\mathcal{I}_{\mathcal{B}_T}$ is their subconfiguration. Also let $S = S^1 \cup \cdots \cup S^t$.
2. The encoding function is applied to the objects of each configuration from S, to obtain string sequences $(s'_1, \ldots, s'_n)$. Let S' be the set of these sequences.
3. Apply the encoding function to the objects $o_1, \ldots, o_n$ to find the string sequence $(s_1, \ldots, s_n)$. The configuration $\mathcal{C}_{\mathcal{B}_T}$ is excluded if and only if $(s_1, \ldots, s_n)$ is not equal to the lexicographically smallest element of S'.

Remark. Note that the last step of the algorithm would not be correct if S' were empty. However, it can be easily seen that it cannot happen, due to the following lemma:

Lemma 2.3.1. If the exclusion algorithm with an initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on a configuration $\mathcal{C}_{\mathcal{B}_T}$ having $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration, then $\mathcal{C}_{\mathcal{B}_T}$ is an element of the set S from step 1.

Proof. Note that if in step 1 we take σ as the identity invariant permutation of $\mathcal{B}_T$, then the configuration obtained by the redefinition will be $\mathcal{C}_{\mathcal{B}_T}$. By the assumption, $\mathcal{C}_{\mathcal{B}_T}$ has $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. Therefore, there will be an index j such that $\mathcal{C}_{\mathcal{B}_T} \in S^j$. Since $S^j \subset S$, also $\mathcal{C}_{\mathcal{B}_T} \in S$. □

The following lemma reveals the actual intention behind step 1, which is generating all relevant isomorphic configurations:

Lemma 2.3.2. If the exclusion algorithm with an initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on two isomorphic configurations, then the corresponding sets S from step 1 are the same.

Proof. Assume that the exclusion algorithm with an initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on a configuration $\mathcal{C}_{\mathcal{B}_T}$ having $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. Denote by $\overline{S}$ the set that the algorithm would generate in step 1 without the condition of configurations having $\mathcal{I}_{\mathcal{B}_T}$ as their subconfiguration. By the definition of configuration isomorphism, $\overline{S}$ contains every configuration isomorphic to $\mathcal{C}_{\mathcal{B}_T}$. By Theorem 2.2.4, if the sets $\overline{S}$ are constructed from two isomorphic configurations, then these sets will be the same. Therefore, after excluding configurations not having $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration, the yielded sets S will still be the same. □

Configurations that are not excluded by the exclusion algorithm will be ones that are actually on the output. In the following parts, we will prove their important properties.

Definition (Minimal Configuration). Let $\mathcal{I}_{\mathcal{B}_T}$ a configuration and let $\mathcal{C}_{\mathcal{B}_T}$ be a configuration with $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. Suppose that the exclusion algorithm with an initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on $\mathcal{C}_{\mathcal{B}_T}$. If $\mathcal{C}_{\mathcal{B}_T}$ is not excluded by the algorithm, then we will say that $\mathcal{C}_{\mathcal{B}_T}$ is *minimal for* $\mathcal{I}_{\mathcal{B}_T}$.

Remark. Notice that there might be more minimal configurations that are distinct, but equivalent to each other: Assume that $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are configurations with objects $o^1_1, \ldots, o^1_m$ and $o^2_1, \ldots, o^2_m$ such that $o^1_i \sim o^2_i$ for all $1 \leq i \leq m$. Then if $\mathcal{C}^1_{\mathcal{B}_T}$ is minimal (for $\mathcal{I}_{\mathcal{B}_T}$), then also $\mathcal{C}^2_{\mathcal{B}_T}$ is minimal (for $\mathcal{I}_{\mathcal{B}_T}$), due to equivalence of their objects, the sequences of their encoded objects are the same (due to Theorem 2.3.6).

The following lemma calls attention to the important property of the exclusion algorithm, specifically that it does not exclude everything in relevant cases:

Lemma 2.3.3. If the exclusion algorithm with an initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on a configuration $\mathcal{C}_{\mathcal{B}_T}$ having $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration, then any configuration $\mathcal{C}'_{\mathcal{B}_T}$ from S corresponding to the smallest element of the set S' from step 2 is minimal for $\mathcal{I}_{\mathcal{B}_T}$.

Proof. Suppose that the exclusion algorithm (with the initial configuration $\mathcal{I}_{\mathcal{B}_T}$) was called on $\mathcal{C}'_{\mathcal{B}_T}$. Clearly $\mathcal{C}_{\mathcal{B}_T} \simeq \mathcal{C}'_{\mathcal{B}_T}$, therefore by Lemma 2.3.2 we have that the sets S from step 1 will be the same, and so the corresponding sets S' will be the same too. Hence their smallest elements are the same. By the assumption, one of them corresponds to $\mathcal{C}'_{\mathcal{B}_T}$, therefore it was not excluded, hence it is minimal for $\mathcal{I}_{\mathcal{B}_T}$. □

The next key lemma shows the importance of lexicographical order in the algorithm:

Lemma 2.3.4. Let $\mathcal{C}_{\mathcal{B}_T}$ be a minimal configuration for $\mathcal{I}_{\mathcal{B}_T}$ with at least $m+1$ objects, where m is the number of objects of $\mathcal{I}_{\mathcal{B}_T}$. Then the configuration $\mathcal{C}^-_{\mathcal{B}_T}$ defined as $\mathcal{C}_{\mathcal{B}_T}$ without the last object is also minimal for $\mathcal{I}_{\mathcal{B}_T}$.

Proof. Let $o_1, \ldots, o_n$ be the objects of $\mathcal{C}_{\mathcal{B}_T}$ and let $(s_1, \ldots, s_n)$ be the sequence of their codes. Since $\mathcal{C}_{\mathcal{B}_T}$ is minimal for $\mathcal{I}_{\mathcal{B}_T}$, it must have $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. Clearly due to $n \geq m+1$, configuration $\mathcal{C}^-_{\mathcal{B}_T}$ also has $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. Therefore, when the exclusion algorithm with the initial configuration $\mathcal{I}_{\mathcal{B}_T}$ is called on $\mathcal{C}^-_{\mathcal{B}_T}$, configuration $\mathcal{C}^-_{\mathcal{B}_T}$ will be contained in the set S from step 2, by Lemma 2.3.1.

To show that configuration $\mathcal{C}^-_{\mathcal{B}_T}$ is minimal (for $\mathcal{I}_{\mathcal{B}_T}$), it remains to prove that the sequence $(s_1, \ldots, s_{n-1})$ of its object codes will be the lexicographically smallest element of the set S' from step 2. We will do this by contradiction.

Assume that $(s'_1, \ldots, s'_{n-1})$ is the smallest element of S' distinct from $(s_1, \ldots, s_{n-1})$, corresponding to a minimal configuration $\mathcal{C}'^-_{\mathcal{B}_T}$ (for $\mathcal{I}_{\mathcal{B}_T}$) with objects $o'_1, \ldots, o'_{n-1}$. Clearly, we have $\mathcal{C}^-_{\mathcal{B}_T} \simeq \mathcal{C}'^-_{\mathcal{B}_T}$, i.e. there exists an invariant permutation σ of $\mathcal{B}_T$ such that $\mathcal{C}^-_{\mathcal{B}_T} \xrightarrow{\sigma} \mathcal{C}''^-_{\mathcal{B}_T}$, where $\mathcal{C}''^-_{\mathcal{B}_T} \equiv \mathcal{C}'^-_{\mathcal{B}_T}$. Let o'_n be the last object of the configuration that we

would obtain after redefining $\mathcal{C}_{\mathcal{B}_T}$ by σ. Note that the configuration $\mathcal{C}'_{\mathcal{B}_T}$ with objects $o'_1, \ldots, o'_n$ is isomorphic to $\mathcal{C}_{\mathcal{B}_T}$.

Configuration $\mathcal{C}'_{\mathcal{B}_T}$ has $\mathcal{C}'^{-}_{\mathcal{B}_T}$ as a subconfiguration. $\mathcal{C}'^{-}_{\mathcal{B}_T}$ is minimal for $\mathcal{I}_{\mathcal{B}_T}$, and therefore has $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. By Lemma 2.2.4, these two facts give that $\mathcal{C}'_{\mathcal{B}_T}$ also has $\mathcal{I}_{\mathcal{B}_T}$ as a subconfiguration. This fact and also $\mathcal{C}_{\mathcal{B}_T} \simeq \mathcal{C}'_{\mathcal{B}_T}$ mean that $\mathcal{C}'_{\mathcal{B}_T}$ belongs to the set S corresponding to $\mathcal{C}_{\mathcal{B}_T}$ in step 1. Therefore, if $(s'_1, \ldots, s'_n)$ is the sequence of the encoded objects of $\mathcal{C}'_{\mathcal{B}_T}$, it belongs to the set S' from step 2 corresponding to $\mathcal{C}_{\mathcal{B}_T}$.

By the assumption that configuration $\mathcal{C}_{\mathcal{B}_T}$ is minimal (for $\mathcal{I}_{\mathcal{B}_T}$), we must have that $(s_1, \ldots, s_n)$ is lexicographically smaller than or equal to $(s'_1, \ldots, s'_n)$. However, this means that $(s_1, \ldots, s_{n-1})$ is lexicographically smaller than or equal to $(s'_1, \ldots, s'_{n-1})$, which contradicts the assumption that $(s'_1, \ldots, s'_{n-1})$ is the smallest element of S' distinct from $(s_1, \ldots, s_{n-1})$. □

After gaining a good understanding of minimal configurations, it is time to precisely define the nucleus of the thesis, the generation algorithm.

Generation algorithm. Let $\mathcal{I}_{\mathcal{B}_T}$ be a configuration, called the *initial configuration* and n a positive integer, called the *number of iterations*. Define sets $G^0, \ldots, G^n$, where $G^0 = \{\, \mathcal{I}_{\mathcal{B}_T} \,\}$, recursively as follows: For $1 \leq i \leq n$ and any configuration $\mathcal{C}_{\mathcal{B}_T}$ from G^{i-1}, the set G^i contains every configuration produced by the configuration extension algorithm called on $\mathcal{C}_{\mathcal{B}_T}$ which is not excluded by the exclusion algorithm with the initial configuration $\mathcal{I}_{\mathcal{B}_T}$. We say that the algorithm *generates* configurations from the set $G^0 \cup \cdots \cup G^n$.

Remark. Notice that the algorithm generates configurations that are either $\mathcal{I}_{\mathcal{B}_T}$, or minimal for $\mathcal{I}_{\mathcal{B}_T}$.

The core of this section is to prove that the generation algorithm, informally speaking, generates what it is supposed to, and nothing else. To do that, let us define what *being generable* precisely means.

Definition (Generable Configuration). Let $\mathcal{I}_{\mathcal{B}_T}$ be a configuration with m objects. We say a configuration $\mathcal{C}_{\mathcal{B}_T}$ with $n \geq m$ objects is *generable from* $\mathcal{I}_{\mathcal{B}_T}$, if it is an element of the set G^{n-m} from the generation algorithm with the initial configuration $\mathcal{I}_{\mathcal{B}_T}$ and the number of iterations n.

Let us connect generable configurations with minimal ones. In fact, the prime objective of all their proven properties is the following crucial lemma:

Lemma 2.3.5. Let $\mathcal{I}_{\mathcal{B}_T}$ be a configuration with m objects. Every configuration $\mathcal{C}_{\mathcal{B}_T}$, that is either $\mathcal{I}_{\mathcal{B}_T}$, or a configuration minimal for $\mathcal{I}_{\mathcal{B}_T}$ with at least $m+1$ objects, is generable from $\mathcal{I}_{\mathcal{B}_T}$.

Proof. We will prove the claim by induction on the number n of the objects of configuration $\mathcal{C}_{\mathcal{B}_T}$. If $n = m$, necessarily $\mathcal{C}_{\mathcal{B}_T} = \mathcal{I}_{\mathcal{B}_T}$, which is generable by definition.

Assume that the claim is true for some $n \geq m$. Let $\mathcal{C}_{\mathcal{B}_T}$ be a minimal configuration (for $\mathcal{I}_{\mathcal{B}_T}$) with $n+1$ objects. Let $\mathcal{C}^-_{\mathcal{B}_T}$ be the configuration $\mathcal{C}_{\mathcal{B}_T}$ without its last object. By Lemma 2.3.4, configuration $\mathcal{C}^-_{\mathcal{B}_T}$ is minimal (for $\mathcal{I}_{\mathcal{B}_T}$). Since it has n objects, by the induction hypothesis it is generable from $\mathcal{I}_{\mathcal{B}_T}$. Clearly, the configuration extension algorithm applied to $\mathcal{C}^-_{\mathcal{B}_T}$ yields a configuration equivalent to $\mathcal{C}_{\mathcal{B}_T}$, which finishes the proof of $\mathcal{C}_{\mathcal{B}_T}$ being generable, hence the entire proof. □

We are finally equipped to prove the first most important theorem of the section, stating that the generation algorithm does not omit important configurations:

Theorem 2.3.7. Let $\mathcal{I}_{\mathcal{B}_T}$ be a configuration. For any correct configuration $\mathcal{C}_{\mathcal{B}_T}$ with a subconfiguration $\mathcal{I}_{\mathcal{B}_T}$ there exists a correct configuration generable from $\mathcal{I}_{\mathcal{B}_T}$ isomorphic to $\mathcal{C}_{\mathcal{B}_T}$.

Proof. By Lemma 2.3.1, the exclusion algorithm with the initial configuration $\mathcal{I}_{\mathcal{B}_T}$ called on $\mathcal{C}_{\mathcal{B}_T}$ will create a non-empty set S from step 1. By Lemma 2.3.3, the set S contains a minimal configuration $\mathcal{C}'_{\mathcal{B}_T}$ (for $\mathcal{I}_{\mathcal{B}_T}$). Clearly $\mathcal{C}_{\mathcal{B}_T} \simeq \mathcal{C}'_{\mathcal{B}_T}$. Since $\mathcal{C}_{\mathcal{B}_T}$ is correct, by Theorem 2.2.5, configuration $\mathcal{C}'_{\mathcal{B}_T}$ is also correct. Finally, by Lemma 2.3.5, configuration $\mathcal{C}'_{\mathcal{B}_T}$ is generable from $\mathcal{I}_{\mathcal{B}_T}$. □

The main purpose of the complexity of the algorithm was to avoid generating isomorphic configurations. The following lemma is a start of the proof that it has been accomplished. It shows the significance of the fact that the same encoded strings necessarily represent equivalent objects.

Lemma 2.3.6. Let $\mathcal{I}_{\mathcal{B}_T}$ be a configuration and let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be two isomorphic configurations generable from $\mathcal{I}_{\mathcal{B}_T}$ with respective objects $o^1_1, \ldots, o^1_m$ and $o^2_1, \ldots, o^2_m$. Then $o^1_i \sim o^2_i$ for every $1 \leq i \leq n$.

Proof. The fact that $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are generable from $\mathcal{I}_{\mathcal{B}_T}$ means that they are either both $\mathcal{I}_{\mathcal{B}_T}$, or minimal for $\mathcal{I}_{\mathcal{B}_T}$. In the first case, the theorem is trivial. In the second case, by Lemma 2.3.2, the corresponding sets S from step 1 of the exclusion algorithm are the same. Subsequently, the corresponding sets S' from step 2 are also the same. The fact that the configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ are minimal (for $\mathcal{I}_{\mathcal{B}_T}$) means that the string sequences $s^1_1, \ldots, s^1_m$ and $s^2_1, \ldots, s^2_m$ of the encoded objects of $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$, respectively, are both equal to the lexicographically smallest sequence of the same set S. However, by Theorem 2.3.6, $s^1_i = s^2_i$ means that $o^1_i \sim o^2_i$ for every $1 \leq i \leq m$. □

Finally, we can prove that the generation algorithm correctly internally identifies isomorphic configurations and does not allow for generating two of them in the same generation.

Theorem 2.3.8. The generation algorithm never generates two distinct isomorphic configurations.

Proof. Let $\mathcal{I}_{\mathcal{B}_T}$ be the initial configuration of the generation algorithm and let n be the number of iterations. Clearly, two isomorphic configurations have the same number of

objects. Therefore, it is enough to show that for each $0 \leq i \leq n$, the set G^i from the algorithm does not contain two isomorphic configurations.

We will prove this by the induction on i. For $i = 0$, the claim is obvious. Assume that it is true for some $i \geq 0$ and we will prove it for $i + 1$, by contradiction.

Let $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ be two isomorphic configurations from the set G^{i+1}. Let $o^1_1, \ldots, o^1_m$ and $o^2_1, \ldots, o^2_m$ be their respective objects. By Lemma 2.3.6 we have $o^1_j \sim o^2_j$ for $1 \leq j \leq m$. Let $\mathcal{C}^{1-}_{\mathcal{B}_T}$ and $\mathcal{C}^{2-}_{\mathcal{B}_T}$ be the configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ without their last object, respectively. Since they have at least m objects, they are clearly generable.

The configurations $\mathcal{C}^{1-}_{\mathcal{B}_T}$ and $\mathcal{C}^{2-}_{\mathcal{B}_T}$ are equivalent, and so also isomorphic. By the induction hypothesis, this means they are identical. Therefore, the configurations $\mathcal{C}^1_{\mathcal{B}_T}$ and $\mathcal{C}^2_{\mathcal{B}_T}$ must have been generated in the same call of the configuration extension algorithm. That is a contradiction, given that the configuration extension algorithm never generates equivalent configurations. □

DFS generation. In the generation algorithm, we need to remember previously generated configurations. We can achieve having their number upper-bounded by the number of iterations. This is accomplished by emulating the order in which a well-known *depth-first search algorithm* [9, p. 603] would traverse the generated graph (see Figure 2.3.2).

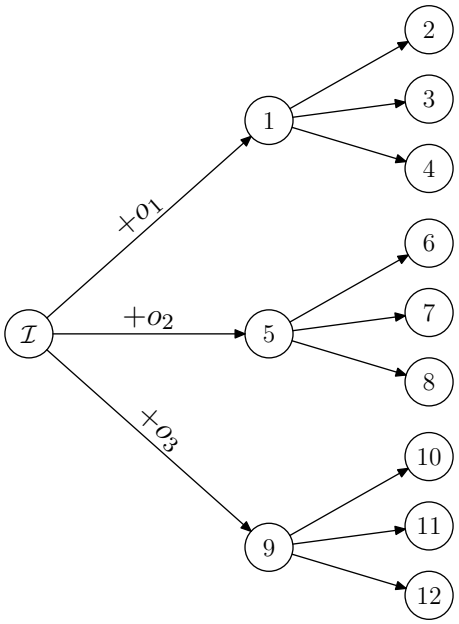


Figure 2.3.2 An illustration of the DFS order, in which the generation algorithm generates geometric configurations, starting with an initial configuration $\mathcal{I}$. With this approach, the maximal number of configurations that are needed to be in memory at the same time is linearly bounded by the depth of the generation graph, which is at most 5 in practice.

Memory usage. The memory usage of a DFS generation is generally linear in the maximal search depth. In our case, the depth is equal to the number of iterations n. The other parts of the algorithm, i.e. the configuration extension algorithm and the exclusion algorithm, do not require having knowledge of more than a constant number of generated configurations. In conclusion, the number of configurations that are needed to be in memory, is linear in n. Practically, this is at most 5, and the algorithm requires at most a few hundreds of megabytes regardless of the run-time.

2.3.2 Theorem finding

We slightly introduced theorem finding in Section 2.1.3, where we established the types of sought theorems in Table 2.1.3 and illustrated their amounts in Table 2.1.4. In this section, we will explain the algorithm for discovering them.

Due to their immense count, a method for finding theorems must be fast and to a high extent reliable. For these reasons, we choose a randomized numeric method. With careful implementation, a 64-bit floating-point arithmetic rarely fails to find the theorems correctly. Similar approaches have also been used in [14, 16].

Simple algorithm. We aim to discover all theorems in a given configuration. One way to achieve this is as follows:

1. Make every hypothesis, for instance conjecture every triple of points to be collinear.
2. Verify each hypothesis numerically using analytic geometry in several randomly generated figures (3 are enough in practice).
3. The theorems that hold in all the figures are considered correct.

This algorithm is applied to the initial configuration from which we start a generation.

Caching algorithm. For a configuration generated from a configuration C by adding an object O we can adopt the following slightly faster version:

1. Make every hypothesis that contains the object O.
2. Verify these theorems as in the previous algorithm.
3. Merge these results with the cached theorems of the configuration C.

The caching algorithm is used for every configuration except for the initial one, where it provides approximately two times speedup against the simple algorithm.

2.4 Filtering problems

The most important part of filtering is the exclusion of simple problems. This is performed by means of *theorem proving*. We will describe its capabilities regarding our situation in Section 2.4.1.

Two other filtering approaches are then discussed in Section 2.4.2. These assume that we already have interesting problems and we want to intelligently reduce their final count.

2.4.1 Theorem proving

In the ideal scenario, we would have a theorem prover that can solve every problem with the estimation of difficulty and guaranteed maximal runtime speed. The state of the art of geometry theorem proving does not provide a solution like that (see Section 1.2).

The ability to estimate the problem's difficulty requires human-like proofs. However, the provers that provide such proofs have very limited capabilities regarding the number of theorems that they can prove.

Furthermore, the more theorems a prover can prove, the slower it is. This is, however, in line with their design, because they were not meant to analyze the magnitude of theorems that we encounter (see Table 2.1.4). This represents the main reason for us not using them.

Other important issues of existing provers in regard to our scenario are:

- They rarely make use of proving theorems in bulk. In our case, this is an important feature, because the prover receives a configuration and *all* of its theorems.
- They seldom allow for assuming some theorems being proven ahead. This is crucial in our case, because when our prover receives a configuration and its theorems, it does not need to prove all of them, but only those that do not appear in the previously generated configurations.

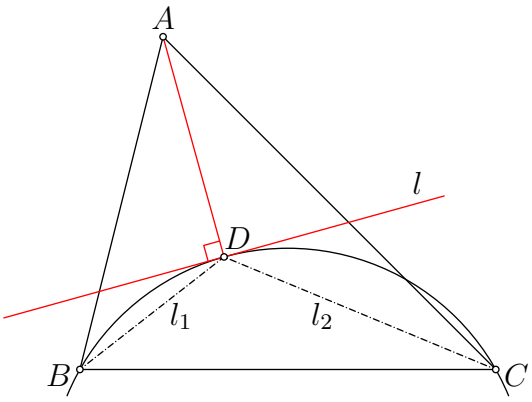
Goal. For the aforementioned reasons, we designed our own theorem prover more suitable in our situation. Instead of building a universal theorem prover that can prove everything and estimate difficulty, we decided to use the prover as a filter, i.e. provable theorems are those that are not interesting. The ultimate goal was to meet the following criteria:

1. The prover must be capable of proving simple theorems.
2. The runtime speed must be reasonably upper-bounded.

Demonstration example. To get a better overview of how our prover works, let us show its proof of the problem in Figure 2.4.1.

This problem is far beyond the scope of the high-school curriculum, but still not intriguing enough for the olympiad. Our prover can recognize this by performing the following steps in under 0.2 seconds.

1. Trivially, from the construction of D, it holds that $D \in l_1$ and $D \in l_2$.



Let ABC be a triangle with its incenter D. The internal angle bisectors l_1 and l_2 of $\angle CBA$ and $\angle ACB$, respectively, meet at D. Let l be the the line tangent to the circumcircle of BDC at D.

Prove that $AD \perp l$.

Figure 2.4.1 An example of a nontrivial, but still not an olympiad problem. The problem is used to illustrate the capabilities of the developed theorem prover (Section 2.4.1). The prover found the demonstrated nontrivial proof of this theorem in under 0.2 seconds.

2. As D lies on both internal angle bisectors l_1 and l_2, it must be equal to the incenter I of ABC.
3. The presence of the discovered incenter I indicates explicit introduction of the internal angle bisector l' of $\angle BAC$.
4. On the other hand, the presence of the tangent line l to the circumcircle of BIC suggests adding the center O of the circle (see Figure 2.4.2 for illustration).
5. There is a well-known lemma stating that this point is equal to the midpoint M of the arc opposite to BAC of the circumcircle of ABC.
6. Another well-known lemma states that $M \in l'$, therefore $O \in l'$.
7. Obviously, A lies on l'.
8. Since I is the incenter, it lies on l' too.
9. The last three facts together prove that the points A, I, O are collinear.
10. A basic property of a tangent line indicates that $OI \perp l$.
11. Since $OI \perp l$ and points A, I, O are collinear, we have the conclusion $AI \perp l$.

In the upcoming parts, we will discuss in depth this proof and how it was discovered by the prover.

Inference rules. The nucleus of our prover is a knowledge database. It is represented in the form of manually written inference rules. This idea was developed independently but has a resemblance to [7].

To give a better overview what inference rules are, we will give a few introductory examples. Before doing so, note that all referred objects in all inference rules are meant to represent distinct objects, unless appearing in an equality.

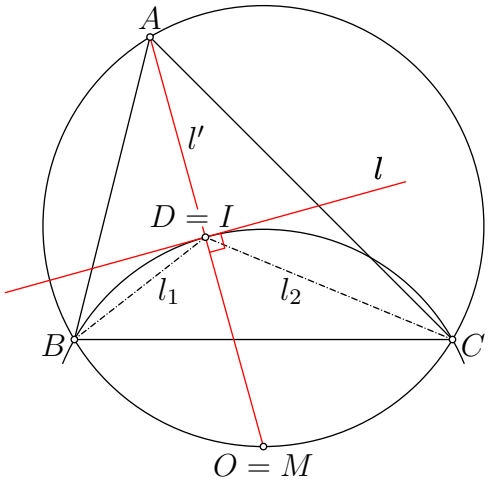


Figure 2.4.2 An illustration for the nontrivial automatically found proof of the problem from Figure 2.4.1. The prover introduced the circumcenter O of triangle BIC, which is key to the solution.

1. If k, l, m are lines, then $k \parallel l$ and $l \parallel m$ imply $k \parallel m$.
2. If A, B, C are points, then supposing they are collinear and $BA = BC$ imply $B = \mathsf{Midpoint}(A, C)$.
3. If l is a line and A, B, C, I are points, then $l = \mathsf{InternalAngleBisector}(B, A, C)$ and $I = \mathsf{Incenter}(A, B, C)$ imply $I \in l$.
4. If A, B are points and k, l are lines, then $A \in k$, $B \in k$, $A \in l$, $B \in l$ imply $k = l$.

Generally, each inference rule consists of:

1. *Object variables*, e.g. lines k, l, m in the first example.
2. *Assumption clauses*, each of one of the two types:
   a) *Standard clause.* This is in a form of a standard theorem of one of the types from Table 2.1.3, for example $BA = BC$ in the second example.
   b) *Object equality clause.* An equality of an object variable and an object constructed from the variables, e.g. $I = \mathsf{Incenter}(A, B, C)$ in the third example.

   In rarer cases, assumption clauses can be negative, i.e. assuming that something is not true. This will be shown later.
3. *Conclusion clause*, which again can be:
   a) *Standard clause.*
   b) *Object equality clause.*
   c) *General equality clause.* An equality of two object variables, e.g. $k = l$ in the fourth example.

Note that the third rule in the example can be contracted to a more readable form: If A, B, C are points, then $\mathsf{Incenter}(A, B, C) \in \mathsf{InternalAngleBisector}(B, A, C)$. In the upcoming examples, we will generally prefer readability over strictly following the prescribed structure.

In the forthcoming algorithm, it will be useful to distinguish between inference rules based on their structure. The following types are considered:

1. *General rules.* These are rules that do not contain any object equality clauses in their assumptions. There are two sub-types of these rules:
   a) *Non-equality rules.* The conclusion clause is a standard clause, e.g. the first example.
   b) *Equality rules.* The conclusion clause is either an object equality clause (the second example), or a general equality clause (the forth example).
2. *Object rules.* At least one of the assumption clauses is an object equality clause, e.g. the third example.

Examples of inference rules. Let us have a look at a few more elaborate examples, including ones that were used in the demonstration example. In the following examples, we will assume that all mentioned objects are distinct.

(i) A very useful *general non-equality rule* is the following one: If A, B, C are points and l is a line, then

$$(A \in l) \wedge (B \in l) \wedge (C \in l) \Rightarrow (A,\ B,\ C \text{ are collinear}).$$

This rule was used in step 9 of the demonstration proof, for points A, I, O and l as the internal angle bisector of $\angle BAC$.

The last step of the demonstration problem actually used a general rule too: If A, B, C are points and l is a line, then

$$(AB \perp l) \wedge (A,\ B,\ C \text{ are collinear}) \Rightarrow (AC \perp l).$$

(ii) As an example of a nontrivial *general non-equality rule* with negative assumptions, assume that A, B, C, D, E, F are points. Then, the radical axis theorem [10, p. 67] can be stated as follows (Figure 2.4.3):

$$\begin{gathered}(A,\ B,\ C,\ D \text{ are concyclic}) \wedge (C,\ D,\ E,\ F \text{ are concyclic}) \wedge \\ \wedge (E,\ F,\ A,\ B \text{ are concyclic}) \wedge (A,\ B,\ C,\ E \text{ are } \textbf{not} \text{ concyclic}) \wedge \\ \wedge (AB \text{ and } CD \text{ are } \textbf{not} \text{ parallel}) \Rightarrow (AB,\ CD,\ EF \text{ are concurrent}).\end{gathered}$$

A few words how negative assumptions are handled. Proving that something is not true is algorithmically challenging, and even in olympiad proofs often omitted. We

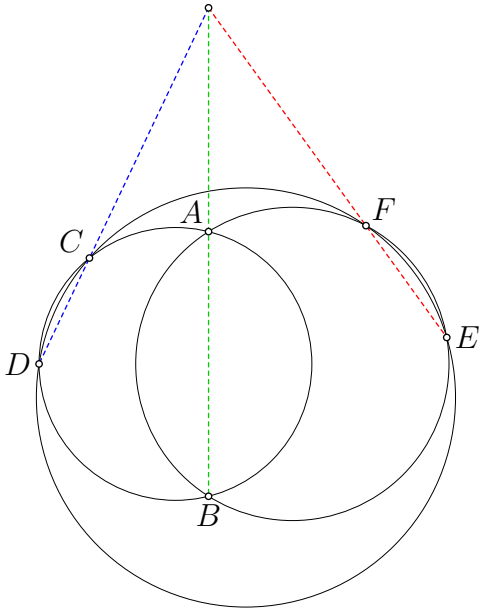


Figure 2.4.3 The well-known radical axis theorem stating that if quadruples (A, B, C, D), (C, D, E, F) (E, F, C, D) consists of concyclic points and the six points are not all concyclic, then lines AB, CD, EF are either parallel or concurrent. The figure illustrates the latter.

will follow this approach and merely verify these statements using the numerical model built for finding theorems.

(iii) As a nontrivial *general equality rule*, notice that if points A, B, C, H are points, then

$$(HB \perp AC) \wedge (HC \perp AB) \Rightarrow (H = \mathsf{Orthocenter}(A, B, C)).$$

Instead of saying that H is the orthocenter, we could have had $HA \perp BC$. However, that would be less powerful, because after discovering that an object is an orthocenter, we can use this fact to apply object rules specific to an orthocenter, like its famous property that it lies on the line with the centroid and circumcenter of the triangle (Euler line [10, p. 18]).

(iv) When it comes to *object* rules, they were used widely throughout the demonstration example. The most nontrivial one was used in step 5 and could be formally written as: If we have points A, B, C, I, O, then

$$(I = \mathsf{Incenter}(A, B, C)) \wedge (O = \mathsf{Circumcenter}(B, I, C)) \Rightarrow$$
$$\Rightarrow (O = \mathsf{MidpointOfOppositeArc}(B, A, C)).$$

Notice that prior to using this rule, we had to verify that I was the incenter. This happened in step 2 by applying the rule

$$(I \in \mathsf{InternalAngleBisector}(A, B, C)) \wedge (I \in \mathsf{InternalAngleBisector}(A, C, B)) \Rightarrow$$
$$\Rightarrow (I = \mathsf{Incenter}(A, B, C)).$$

When it comes to other used object rules, in step 6 we used

$$\mathsf{MidpointOfOppositeArc}(B, A, C) \in \mathsf{InternalAngleBisector}(B, A, C),$$

whereas in step 8 it was

$$\mathsf{Incenter}(A, B, C) \in \mathsf{InternalAngleBisector}(B, A, C),$$

and finally, the following rule was used in step 10, where by $\mathsf{TangentLine}(A, B, C)$ we mean the tangent line to the circumcircle of ABC at A:

$$(O = \mathsf{Circumcenter}(A, B, C)) \Rightarrow (\mathsf{TangentLine}(A, B, C) \perp AO).$$

(v) Notice we have not covered four steps of the proof. In steps 3 and 4, no theorem was inferred, we merely introduced a new object to the figure. On the other hand, in steps 1 and 7, we claim that A lies on the internal angle bisector of $\angle BAC$. This, however, is a construction-related theorem (described in Section 2.1.4, see Figure 2.1.4). As we will see, these are automatically found whenever we introduce a new object, or realize that an existing object can be defined differently (as in (iii), where we can infer the orthocenter H from $HB \perp AC$ and $HC \perp AB$, and then have $HA \perp BC$ as a construction-related theorem).

Now that we have explained all the rules used in the demonstration example, let us visually summarize the proof in Figure 2.4.4.

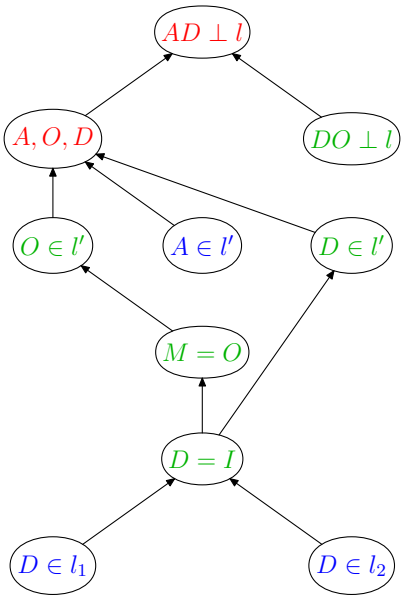


Figure 2.4.4 A graph of the proof of the demonstration example (see Figure 2.4.1) from the beginning of Section 2.4.1. Note that by A, O, D, we refer to the collinearity of these points. Individual nodes representing claims were concluded based on the described inference rules. Colorwise, red theorems were inferred by general rules, green ones by object rules, and blue ones are construction-related theorems.

Database of inference rules. We selected rules that entail common geometry knowledge including a few more complex lemmas known to olympiad solvers. At the final stages, we had 183 rules, 22 of them were general equality ones, 60 general non-equality ones, and 102 were object rules. All the rules are available on the GitHub page[1] of the project.

Object introduction rules. Even simple theorems sometimes require an introduction of a new object. Doing this unwisely would lead to an extreme slowdown of the prover. For that reason, object introduction techniques must be designed very carefully.

We decided to introduce objects based on two simple methods:

1. *Construction-based introduction.* The presence of an object $o = f(o_1, \ldots, o_n)$ constructed via a certain construction f and arguments $o_1, \ldots, o_n$ might indicate introduction of an object constructed from $o, o_1, \ldots, o_n$, e.g. if D is the reflection of a point A in a line l, then the introduced object might be the midpoint of AD.
2. *Theorem-based introduction.* Theorems of certain types indicate what could be introduced, specifically:
   a) If we are trying to prove that two lines are perpendicular, we introduce their intersection point.
   b) If it is being proven that three lines are concurrent, we introduce the intersection points of some two of them (i.e. 3 options).

The construction-based introduction is founded on arbitrarily chosen rules. We choose 36 of them. All these rules can be found on the GitHub[2] page of the project.

Theorem-based introduction example. Construction-based introduction was already shown in the demonstration example, precisely in steps 2 and 3. To illustrate theorem-based introduction, assume that we have the rules about points A, B, C, O and lines l_1, l_2, l_3 saying:

$$\text{(i)} \quad (C \in \mathsf{PerpendicularBisector}(A, B)) \Rightarrow (CA = CB)$$
$$\text{(ii)} \quad (CA = CB) \Rightarrow (C \in \mathsf{PerpendicularBisector}(A, B))$$
$$\text{(iii)} \quad (OA = OB) \wedge (OB = OC) \Rightarrow (OA = OC)$$
$$\text{(iv)} \quad (O \in l_1) \wedge (O \in l_2) \wedge (O \in l_3) \Rightarrow (l_1, l_2, l_3 \text{ are concurrent})$$

With this, the prover can easily show that the perpendicular bisectors l_1, l_2, l_3 of sides AB, BC, AC, respectively, of a triangle ABC are concurrent (see Figure 2.4.5):

1. First it introduces the intersection point T of l_1 and l_2.
2. Trivially, $T \in l_1$ and $T \in l_2$.

[1] See https://github.com/PatrikBak/GeoGen/tree/2abcbb48eb8bc74b601ea722f1e33a9cc4018711#inference-rules.
[2] See https://github.com/PatrikBak/GeoGen/tree/2abcbb48eb8bc74b601ea722f1e33a9cc4018711#object-introduction-rules.

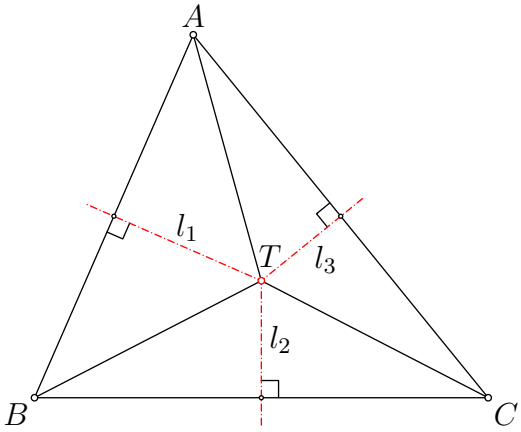


Figure 2.4.5 The problem stating that the perpendicular bisectors of the sides of a triangle are concurrent. This problem can be proved automatically by the theorem prover by means of theorem-based introduction technique, described in Section 2.4.1.

3. By (i), it proves $TA = TB$ and $TB = TC$.
4. By (ii), it proves $TA = TC$.
5. By (iii), it proves $T \in l_3$.
6. By (iv), it proves that l_1, l_2 and l_3 are concurrent.

Proving algorithm. After having discussed inference and object introduction rules, we are able to use them in the core proving algorithm.

As an input, this algorithm takes a configuration and all its theorems. Its goal is to prove those of them that cannot be stated without the last object of the configuration. These theorems are deemed *interesting* and passed to complementary methods for further processing.

An important part of the algorithm is choosing which inference rules should be applied. We want to apply them iteratively, until the desired theorems are concluded. Trying every rule in every iteration would be inefficient. Therefore, we will use *scheduling* algorithms to handle this. They will ensure that appropriate inference rules are prepared (scheduled) for applying.

Since the algorithm is complex, we will first give a high-level overview of its steps. In the upcoming parts, we will then describe the individual complex steps in depth. The algorithm goes as follows:

1. Establish theorems that are automatically considered proven, i.e. *axioms*.
2. Prepare inferences rules that will be scheduled for usage. This is done via the *initial scheduling algorithm*.
3. Run the *main loop*. In each iteration, we will try to apply one inference rule.

a) Obtain the current scheduled inference rule.

b) If there is no remaining one, perform the *object introduction algorithm*. It will finish in either of the two ways:

   (i) It internally ensures that there is a new scheduled inference rule, therefore, the main loop can continue.

   (ii) There is no new scheduled inference rule. In that case, the entire algorithm is over.

c) In this step, there is an inference rule to be applied. This is done by matching already proven theorems (including axioms) with the assumptions of the rule, and also potentially verifying negative assumptions. If this matching can be done, then we have proven the conclusion.

d) For every new proven theorem, the following steps are performed:

   (i) If it found out that some object can be constructed differently, then its construction-related theorems of the alternative constructed version are found.

   (ii) The proven theorem, and potentially the found construction-related theorems, are used by the *schedule after proving algorithm*.

e) If there is nothing left to be proven, we are done. Otherwise, we continue with the main loop.

Axioms. To make inferences, we need a set of basic propositions that will serve as axioms. As hinted in the analysis of the demonstration problem, we will use *construction-related theorems*. These are easily discoverable solely from the object's definition.

In the demonstration example, the prover was able to solve the problem only with construction-related axioms. However, when the prover is being run as part of a generation, it is more efficient to regard proven every theorem that is true in a previous configuration, as well as every new theorem that is *simplifiable* (described in Section 2.1.4, see Figure 2.1.5). This might seem like an overly strong assumption, the reasons for doing it this way are as follows:

- The theorems of the previous configuration have been examined previously, therefore their exclusion (by assuming them proven) is not a loss.
- Similarly with simplifiable theorems, which, by their definition, already appeared in a smaller configuration (which we assume was generated or contains too few objects to be interesting).
- Most importantly, assume that some of these previous or simplifiable theorems is considered proven, but in reality it is very difficult. Then, its difficulty lies in a smaller configuration. Therefore, if we use this difficult theorem as an assumption and infer some other theorem, than the inferred theorem also has its difficulty in the

smaller configuration. Hence, assuming difficult previous or simplifiable theorems proven will not cause a loss of interesting problems.

Scheduling algorithms. In the proving algorithm, we mentioned two needed operations concerning rule scheduling, specifically *initial scheduling algorithm* in step 2 and *schedule after proving algorithm* in step 3d (ii). These are parts of the general scheduling principle that shall be described now.

The main idea of the scheduler is to maintain two separate queues of scheduled inference rules, one for general and one for object rules. When the next rule is requested by the main algorithm, then a general one will be preferred, if it is available, otherwise an object one, else none. The idea behind this decision is that general rules are more likely to prove theorems.

Another important part is *premapping*. When a rule is scheduled, it is possible to premap what conclusion it is supposed to prove, or what assumption it should use, or in case of an object rule, which concrete object should be used.

Initial scheduling algorithm. As an input, there are theorems needed to be proven, and the configuration in which they hold. With this we will schedule rules in the following way:

1. For every theorem t to be proven, and every general rule r that concludes a theorem of the same type as t, schedule r with a premapped conclusion t.
2. Schedule every general equality rule.
3. For every object o of the configuration, and every object rule r that contains an object with the same construction as o, schedule r with a premapped object o.

Schedule after proving algorithm. When a theorem is proven, the scheduler should react on it by scheduling rules that might use this fact as an assumption. The algorithm is as follows:

1. If the proven theorem t is an equality, then it means that t is about a constructed object o (geometrically equal to an object of the configuration). Therefore, object rules that contain an object with the same construction as o should be scheduled with premapped o.
2. If the proven theorem t is not an equality, then:
   (i) For every rule r that contains an assumption of the same type as t, schedule r with a premapped assumption t.
   (ii) For every object rule r that was scheduled previously and contains an assumption of the same type as t, schedule r with a premapped assumption t (and keep the premapped object).

Object introduction algorithm. When the algorithm runs out of scheduled rules, it tries to introduce a new object to the situation. We already presented the two methods used to accomplish so, theorem-related and object-related introduction. The following algorithm puts them into the context of the main proving algorithm.

The most dangerous part of introducing new objects is a potential time explosion, as one of the important aspects of the prover is its speed. Introducing every object would result into a practically unusable prover. This is the reason why a compromise is needed. The following statements are its outcome:

1. At all times there should be at most one newly introduced point.
2. Lines and circles can be introduced freely.
3. All introduced objects must be definable in the examined configuration.

The primary motivation behind the first decision is a logical observation that points are what causes time complexity. On the other hand, lines and circles are usually already present implicitly by points, hence they should not cause a slowdown.

The order in which objects are being introduced is the following one:

1. Construction-based lines and circles.
2. Theorem-based objects.
3. Construction-based points.

The actual object introduction algorithm uses this order as follows:

1. Remove the last introduced object, if it was a point, and did not turn out to be equal to an object of the examined configuration.
2. Introduce the next object.
3. If there is none left, then the algorithm is over.
4. Otherwise there is a new object o. Find all construction-related theorems of o.
5. For each call the *schedule after proving* algorithm.
6. Schedule object rules that contain an object with the same construction as o with premapped o.

Results. To illustrate the prover's quality, we will refer to the analysis portrayed in Section 2.1.4, specially Table 2.1.7. Running the prover on the 11,718 symmetric theorems of the performed test case experiment (Section 2.1.2) took under 3 minutes, and the prover filtered up to 95.6% of these theorems.

2.4.2 Complementary methods

After the initial filtering by theorem proving (Section 2.4.1), numerous problems still remain. To provide better final results, we further introduced the following methods:

1. Asymmetric theorems are excluded, as drafted in Section 2.1.4.
2. It is possible that a theorem appears in more problems in different forms, for instance see Figure 2.4.6. Such a situation is detected numerically.

   We will only choose one of these problems. The problem with fewer objects is preferred, for it being more elegant and less obfuscated. If there is equality, then the problem with fewer points is chosen. If the tie remains, then we use ranking, or else select the problem at random.

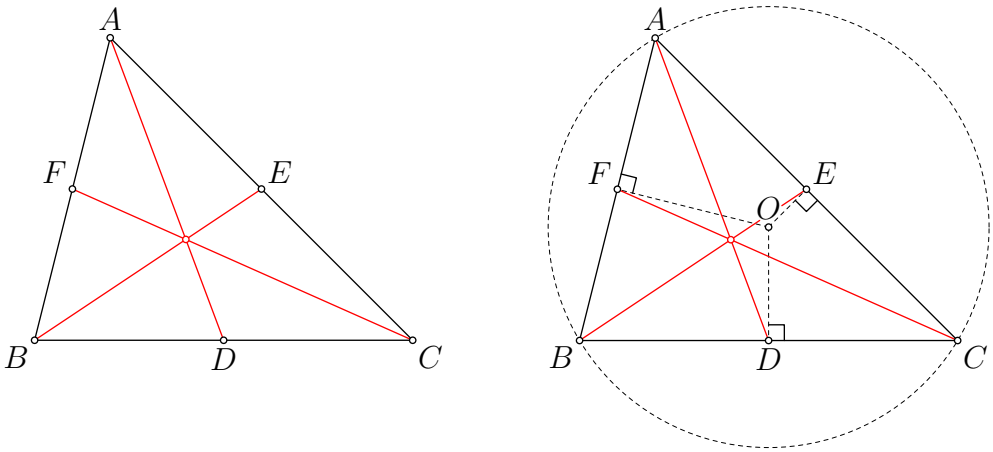


Figure 2.4.6 A median concurrency theorem in two versions, where the right one uses the circumcenter O of ABC to define the midpoints of the sides. Generally, a simpler version is the "cleaner" one, hence the right one will be excluded as not interesting. Recognizing such cases is a part of complementary filtering (Section 2.4.2).

Results. The impact of the asymmetric theorem exclusion was widely examined in Section 2.1.4. Specifically, Table 2.1.6 suggests that there are about 19 times more asymmetric than symmetric theorems.

Regarding the influence of the second method, we will refer to the 512 theorems (Table 2.1.7) of the test case experiment (Section 2.1.2) not filtered out by the theorem prover (Section 2.4.1). These 512 theorems are reduced to 250, which represents a reduction by approximately 51.1%.

2.5 Problem ranking

In section Section 2.1.4, we highlighted that there are still hundreds of theorems even after filtering (from Section 2.4). Therefore, it is favorable to have them sorted by their estimated suitability for the mathematical olympiad. This way their subsequent manual inspection will become much simpler. For this reason, we designed a *ranking* system.

The key idea is to choose various relevant aspects and rate them individually. This way we obtain ratings $r_1, \ldots, r_n$. Using manually configured weights $w_1, \ldots, w_n$, we calculate the final rating of the problem as a linear combination

$$w_1 r_1 + \cdots + w_n r_n.$$

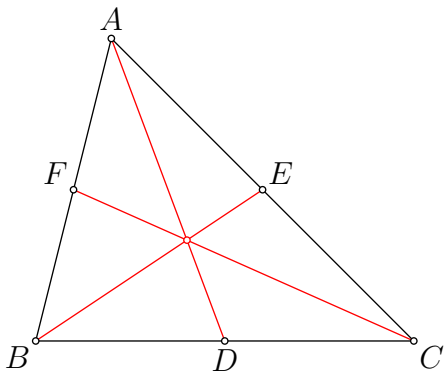
Olympiad problems are required to be both engaging and challenging. Taking this into consideration, let us present the proposed rated aspects. Later in Section 2.5.4, we will address their overall importance and set up their weights.

2.5.1 Symmetry rating

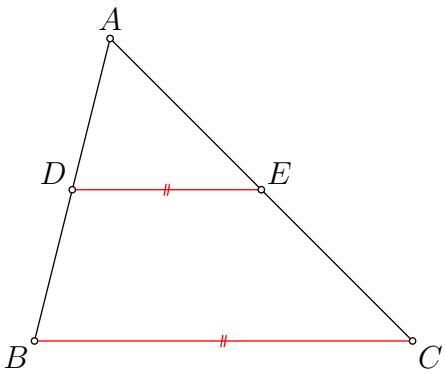
We introduced symmetry in Section 2.1.4 (see Figure 2.1.3) and in Section 2.4.2 established that we would exclude asymmetric problems. However, there are more degrees of symmetry and they can be rated. More symmetric problems are generally more elegant.

Let us recall that a triangle problem is said to be symmetric if there is a relabeling of two of the triangle's vertices that represents the same problem. We will refer to such relabelings as *symmetry relabelings*. Clearly, a triangle has at most 3 symmetry relabelings.

Similarly, for a quadrilateral, a relabeling, where we interchange at least one pair of points while preserving the problem, is called a symmetry relabeling (for a quadrilateral). One can check there are at most 9 such relabelings.[1]



(a) A symmetric theorem, in which there exist three symmetric relabelings (exchanging some two vertices).



(b) A symmetric theorem, in which there exists exactly one symmetric relabeling, $(A, B, C) \mapsto (A, C, B)$.

Figure 2.5.1 An example of two levels of triangle symmetry described via *symmetric relabelings* defined in Section 2.5.1. Generally, problems with a higher degree of symmetry are more appealing.

Based on the number of symmetry relabelings we can distinguish various levels of symmetry (see Figure 2.5.1). More precisely, the symmetry rating is calculated by dividing

[1] This number is equal to the number of permutations of 4 elements containing a 2-cycle.

the number of actual symmetry relabelings by the number of potential symmetry relabelings.

For example, in Figure 2.5.1a, all the three potential symmetry relabelings actually preserve the problem, i.e. the rating of symmetry is 1, whereas in Figure 2.5.1b only $(A, B, C) \mapsto (A, C, B)$ preserves the problem, i.e. the rating is $\frac{1}{3}$.

2.5.2 Level rating

Most interesting problems often appear simple. One reason for problems seeming complicated is that their objects have long definitions. For example, a point A is the midpoint of BC, where B is the circumcenter of DEF, where D...

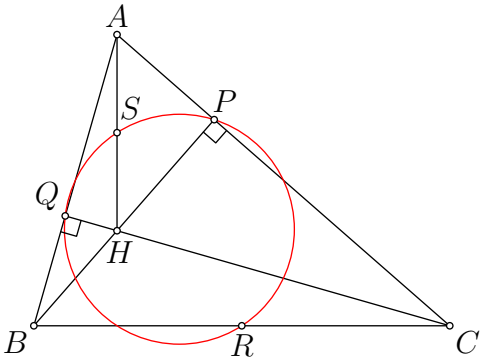
Ideally, the best problems should have simpler definitions that are easier to imagine and understand. On top of that, this will affect the difficulty too, because short definitions often hide the relations between the objects needed for the solution.

To capture the definition's complexity, we will assign a natural number to every object, called its *level*. Intuitively, it will be equal to the depth of the object's definition. It is calculated as follows:

(i) Initial objects have a level of 0.

(ii) If the maximal level among objects $o_1, \ldots, o_n$ is l and f is a construction, then the level of the object $f(o_1, \ldots, o_n)$ is $l + 1$.

As an example, see Figure 2.5.2.



Let ABC be an acute triangle. Denote by H its orthocenter. Let P and Q be the projections of B and C on AC and AB, respectively. Furthermore, let R and S be the midpoints of BC and AH, respectively.

Prove that points P, Q, R, S are concyclic.

Figure 2.5.2 A problem to illustrate object levels defined in Section 2.5.2. In the problem, points A, B and C have levels of 0, points P, Q, R, and H of 1, and finally the level of S is 2.

After having the levels, the rating is calculated as follows: Assume that $l_1, \ldots, l_n$ are the levels of the problem's constructed objects (i.e. not initial ones).

1. We combine the levels by using the sum of their squares. The reason for squaring levels is to penalize higher ones more. It also provides a wider range of attainable values.

2. The sum of their squares is then normalized to be in the interval $[0, 1]$. Clearly,

$$n = 1^2 + \cdots + 1^2 \leq l_1^2 + \cdots + l_n^2 \leq 1^2 + \cdots + n^2 = \frac{1}{6}n(n+1)(2n+1),$$

therefore for $n > 1$ we have

$$0 \leq \frac{(l_1^2 + \cdots + l_n^2) - n}{\frac{1}{6}n(n+1)(2n+1) - n} \leq 1.$$

Cases $n = 0$ and $n = 1$ are not interesting, because for $n = 0$ we have only initial objects, and for $n = 1$ we have only construction-related theorems.

3. To achieve that better problems have higher ratings, we replace a rating r with $1 - r$. After an algebraic simplification, the final formula is

$$L(l_1, \ldots, l_n) = 1 - 6 \cdot \frac{(l_1^2 + \cdots + l_n^2) - n}{n(n-1)(2n+5)}.$$

For example, in the problem on Figure 2.5.2, the levels of P, Q, R, H, S are 1, 1, 1, 1, 2, respectively, therefore the level rating of the problem is equal to

$$L(1, 1, 1, 1, 2) = 1 - 6 \cdot \frac{(1^2 + 1^2 + 1^2 + 1^2 + 2^2) - 5}{5 \cdot 4 \cdot 15} = \frac{47}{50}.$$

Notice that this aspect does not take the theorem into account, but merely the configuration where it holds.

2.5.3 Complementary rating

In addition to symmetry and level rating, we also considered two supplementary aspects, whose aim is to correlate with the problem's difficulty.

Number of theorems. More theorems in a geometry problem usually indicate that there are more ways to make inferences among them, which means the problem is easier. For that reason, their count provides an appropriate evaluation of the problem's difficulty.

This aspect will have assigned a negative weight, i.e. we can view a theorem as a penalty. If there are n theorems in the rated problem, then the rating will be $n - 1$. The idea behind the subtraction of 1 is to exclude the problem we are currently rating. Hence, an ideal problem would have this rating 0.

Number of cyclic quadrilaterals. Cyclic quadrilaterals have a variety of powerful properties. The more they occur in a problem, the easier it is to exploit them to solve the problem. Therefore, we include their count into the rating.

Analogous to the number of theorems, this aspect will also have a negative weight. If there are n concyclic point theorems, then the rating will be either n, if the rated theorem is not one of them, otherwise $n - 1$. Again, an ideal problem would have this rating 0.

2.5.4 Configuring weights

We have described the four rated aspects, (1) symmetry, (2) level, (3) number of theorems, (4) number of cyclic quadrilaterals. Assume that we have a problem whose ratings for these aspects are s, l, t, c, respectively. To get the total rating, we need to know the values of the corresponding weights w_s, w_l, w_l, w_c, yielding the total rating equal to

$$w_s \cdot s + w_l \cdot l + w_t \cdot t + w_c \cdot c.$$

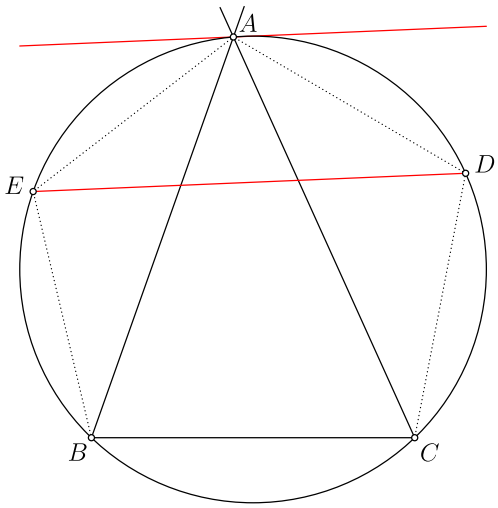
Before revealing the actual chosen values, let us point a few heuristic observations:

1. Problems with higher levels of symmetry are generally rare, which itself makes them more intriguing. Therefore, we will set the value w_s very high.
2. As discussed in Section 2.5.2, the level is the most universal aspect, which entails both beauty and difficulty. There will be thousands of problems within the same symmetry category, therefore a high value of w_l will distinguish them.
3. There are still hundreds of problems with the same symmetry and level rating. The ratings t and c are designed to provide their order, and generally a better dispersion of results. As said earlier, the weights w_t and w_c will be negative. Cyclic quadrilaterals are usually way more useful to make implications than other theorems, therefore we will have $|w_c| > |w_t|$.

The final setup of the weights is

$$10000s + 1000l - t - 10c$$

For an example see Figure 2.5.3.



Let ABC be a triangle. Let l be the external angle bisector of $\angle BAC$. Let D be the midpoint of opposite arc ABC. Let E be the midpoint of opposite arc ACB.

Show that $DE \parallel l$.

Type	Rating	Weight	Contribution
Symmetry	0.33	10000	3333.33
Level	1	1000	1000
Number of theorems	8	−1	−8
Cyclic quadrilaterals	5	−10	−50

Total rating: 4275.33

Figure 2.5.3 An automatically drawn and stated problem with ratings calculated via the rating methods described in Section 2.5. This simple-looking problem is recognized as interesting and well-rated by the ranking system.

Chapter 3

Experiments and results

To demonstrate the functionality of our solution, we have conducted both large-scale (Section 3.2) and local generation experiments. These experiments resulted in many high-quality problems, as presented in Section 3.3.

3.1 Implementation

The system has been implemented in the programming language C# using the .NET Core 3.1 framework. The fully documented source code is provided on the GitHub page of the project `https://github.com/PatrikBak/GeoGen`, commit 2abcbb4.

3.1.1 Visualization

The program generates problems in a text form. To improve the user experience, we developed a drawer that can automatically produce a human-readable problem text alongside METAPOST figures. The drawer can handle aesthetic details including:

- Objects are drawn according to their importance, e.g. an auxiliary line would appear dotted whereas a point from the formulation of a theorem would be colored red.
- Point labels are intelligently placed to prevent them from crossing other objects.
- Large circles are clipped to ensure the figure is not unnecessarily large.
- The point positions are generated to avoid having points too close or too far from each other.
- Triangles and quadrilaterals are drawn with respect to their symmetry, which is a standard way of drawing olympiad figures.

For examples of automatically drawn figures, see Appendix and Figure 2.5.3.

3.2 Large-scale experiments

The main idea of the large-scale experiments was to run a vast number of independent generations in parallel and merge the results afterward. Throughout the experiments, we used a fixed set of constructions, the ones from Table 2.1.1. The experiments consisted of generating problems of two types:

(i) triangle problems with 8 objects in total, i.e. 5 additional objects;

(ii) quadrilateral problems with 8 objects in total, i.e. 4 additional objects.

As initial configurations, we generated a triangle with 2 additional objects, which yielded 1,355 configurations, and a quadrilateral with 2 additional objects, giving the next 3,187 configurations. This allowed us to use parallelization and conduct 4,524 configurations independent generations.

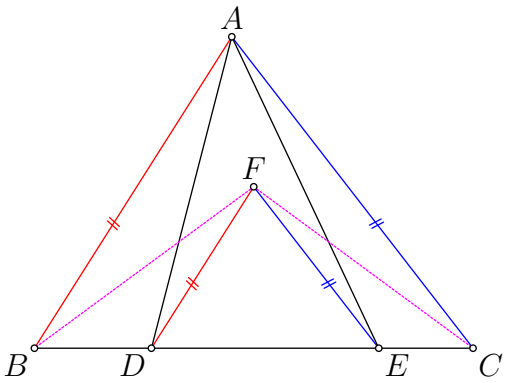
3.2.1 Run-time

The experiments were conducted on the cluster in Max Planck Institute for Intelligent Systems[1]. The entire run-time was around 40,000 CPU hours altogether, with a maximal run-time of approximately 40 hours. After merging the results, the experiments produced more than 100,000 problems, sorted by their estimated quality using the ranking methods (see Section 2.5). Nine of these results are shown in Appendix, with automatically generated figures (see Section 3.1.1).

3.3 Results

One type of result are problems discovered during the development and myriads of diverse local generation experiments. Many of these problems are in the process of submission to various national and international contests. Two of them have already been accepted to the Czech-Slovak mathematical olympiad, specifically Figure 3.3.1 and Figure 3.3.2.

Other significant results are certainly the problems generated in the large-scale experiments (Section 3.2). Five of these problems have been proposed to the International Mathematical Olympiad 2020. At the time of writing this thesis, their status remains undecided. These five proposed problems are only a small portion of reasonably looking experiment results ready to be proposed in the future to the Internal Mathematical Olympiad and other competitions.

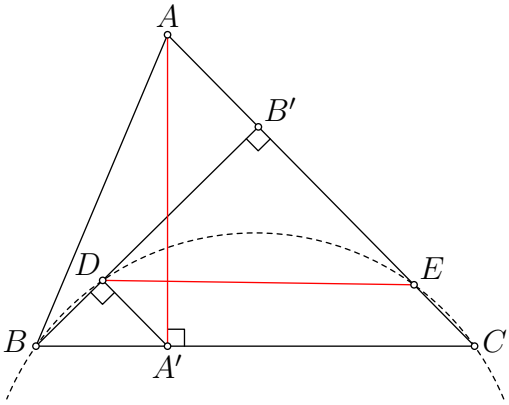


Let ABC be an acute triangle. Suppose that points D and E lie on the side BC such that D is between B and E, $AD = CD$, and $AE = BE$. Point F is a point satisfying $FD \parallel AB$ and $FE \parallel AC$.

Prove that $FB = FC$.

Figure 3.3.1 A generated problem that has been accepted to the home round of the Czech-Slovak mathematical olympiad 2020/2021, category A (highest).

[1] https://is.mpg.de/



Let ABC be an acute triangle with its altitudes AA' and BB'. Let D be the perpendicular projection of A' on the altitude BB'. Suppose that the circle through points B, C, D intersects the segment AC at its inner point E.

Prove that $DE = AA'$.

Figure 3.3.2 A generated problem that has been accepted to the home round of the Czech-Slovak mathematical olympiad 2020/2021, category B (second highest).

Conclusion

We have demonstrated that even such a creativity demanding task as the writing of olympiad geometry problems can eminently benefit from *automation*. The experiments show that our developed system can generate large numbers of quality problems, offering geometers a seemingly unlimited resource for their problem writing endeavors.

The five problems proposed to the International Mathematical Olympiad, the two problems already accepted into the Czech-Slovak mathematical contests, and the nine problems in Appendix are merely the tip of the iceberg regarding the capabilities of our system in planar geometry olympiad problem generation.

In the discussion section of [18] the following question has been posed: *What is the role of software in the process of problem posing for competitions?* Traditionally, dynamic geometry software such as GeoGebra [12] provide tremendous aid in examining geometric situations given in advance. We take a step further and provide a tool that suggests these problems. To the best of our knowledge, our designed system is the first answer to the challenge of automated geometry olympiad problem creation.

Opportunities for future work fall into two main categories:

1. Improving filtering methods from Section 2.4, especially enhancing the theorem prover (Section 2.4.1) by including other proving methods such as the full-angle method [6, 5, 23]. This would result in the filtering of many difficult problems, but the remaining ones would be even more challenging.
2. Exploring options of AI methods, mainly with regards to the selection of the best generated results (Section 2.5). This could introduce other unprecedented challenges, such as building an annotated database of olympiad problems.

Resumé

Geometrické úlohy sú odjakživa neoddeliteľnou súčasťou matematických súťaží ako je Medzinárodná Matematická Olympiáda. Tradične tieto úlohy vymýšľajú skúsení geometri, pričom táto činnosť si vyžaduje popri vysokej odbornosti aj roky tréningu. V tejto diplomovej práci ukazujeme, že aj takáto kreatívna a nesmierne časovo náročná činnosť dokáže neobyčajne profitovať z automatizácie. Predstavujeme náš systém automatického generovania rovinných geometrických úloh pre Matematickú olympiádu.

V Kapitole 1 sa venujeme prieskumu aktuálneho stavu v relevantných oblastiach. Ukazuje sa, že ohľadom generovania úloh bolo urobené len veľmi málo. Existujúce riešenia nevykazujú možnosti na generovanie úloh vyššej obtiažnosti pre Matematickú olympiádu.

V predmetnej oblasti sa venuje väčšia pozornosť automatizovanému dokazovaniu geometrických viet. Túto sféru sme taktiež preskúmali, keďže jednou zo súčastí nášho riešenia je dokazovací systém. Neskôr je vysvetlené, prečo tieto riešenia nie sú v našom prípade vhodné.

V Kapitole 2 predstavujeme naše riešenie. Jeho základnou myšlienku je rozdelenie problému generovania úloh na generovanie geometrických konfigurácií a hľadanie a spracovanie geometrických viet.

V podkapitole 2.1 zľahka uvádzame naše riešenie analyzovaním naivných prístupov. Výsledkom sú tieto problémy:

1. Priamočiare generovanie vedie ku generovaniu veľkého počtu konfigurácií, z ktorých mnohé sú izomorfné (predstavujúce tú istú geometrickú situáciu zadanú iným spôsobom).
2. Vo vygenerovaných konfiguráciách platí veľmi veľa viet, ľudskými silami len veľmi ťažko spracovateľných. Navyše, mnohé z nich nevyhovujú kritériám olympiádnych úloh.

To nás vedie k tomu, že potrebujeme:

1. netriviálny algoritmus generovania konfigurácií, ktorý rozpoznáva izomorfné konfigurácie (podkapitola 2.3),
2. sofistikovaný systém filtrovania jednoduchých úloh (podkapitola 2.4),
3. systém hodnotenia kvality úloh, pomocou ktorého sa usporiadajú neodfiltrované úlohy (podkapitola 2.5).

V podkapitole 2.2 sme popísali formálny matematický model, ktorý nám neskôr umožnil v podkapitole 2.3 exaktne popísať a dokázať správnosť algoritmu generovania. Základom tohto modelu je reprezentácia geometrických konfigurácií pomocou teórie grafov, konkrétne orientovaných grafov s usporiadanými vrcholmi a hranami.

V podkapitole 2.3 sme sa okrem exaktného popisu algoritmu generovania a dôkazu jeho korektnosti zaoberali aj otázkou jeho pamäťovej zložitosti. Zámerom bolo totiž použiť tento algoritmus na dlhotrvajúce generovania (desiatky hodín). Bolo teda potrebné, aby množstvo pamäte spotrebovanej algoritmom bolo z hľadiska praxe rozumne ohraničené. To sa podarilo dosiahnuť. Náš algoritmus si počas generovania potrebuje pamätať počet konfigurácií, ktorý je lineárne závislý od hĺbky generovaného grafu, čo je v praxi nanajvýš 5. V praxi tento algoritmus potrebuje nanajvýš niekoľko stovák MB pamäte RAM.

V podkapitole 2.4 sme popísali náš systém filtrovania ľahkých úloh. Jeho hlavný komponent je založený na dokazovaní viet. Hlavná myšlienka je použiť dokazovač ako filter. Dokazovač je teda prispôsobený tomu, aby vedel rýchlo odhaliť jednoduché úlohy. Základom je využitie známej metódy deduktívnej databázy, avšak vyvinutej nezávisle. Dokazovač bol otestovaný na ukážkovom experimente a bol schopný odhaliť 95.6% ľahkých viet za menej než 3 minúty.

V podkapitole 2.5 sa venujeme systému hodnotenia úloh. Tento systém predpokladá, že dostáva na hodnotenie zaujímavé úlohy, a snaží sa z nich vybrať tie najzaujímavejšie. To robí tak, že individuálne ohodnotí štyri zvolené aspekty a výsledky lineárne skombinuje. Výsledky sú potom usporiadané podľa tohto finálneho hodnotenia.

V Kapitole 3 popisujeme vykonané experimenty a uvádzame výsledky. Okrem nespočetného množstva malých lokálnych experimentov priebežne vykonávaných počas vývoja sme vykonali aj experimenty vo veľkom meradle. Tieto experimenty bežali paralelne na približne 4 500 počítačoch v Inštitúte Maxa Plancka v Nemecku, pričom spotrebovali približne 40 000 CPU hodín.

Hlavnými výsledkami práce sú:

1. dve úlohy zverejnené v domácich kolách Česko-Slovenskej matematickej olympiády 2020/2021, vygenerovaných počas vývoja,
2. päť úloh navrhnutých na Medzinárodnú Matematickú Olympiádu 2020, vygenerovaných vo veľkých experimentoch,
3. tisíce ďalších kvalitných úloh vygenerovaných vo veľkých experimentoch, ktoré budú priebežne navrhované na rôzne národné a medzinárodné súťaže.

V prílohe uvádzame ukážky netriviálnych úloh vygenerovaných počas veľkých experimentov. Obrázky spolu so zadaniami sú produktom nášho automatizovaného systému na kreslenie obrázkov a generovanie textových zadaní pre vygenerované úlohy.

Bibliography

[1] Rajiv Bagai, Vasant Shanbhogue, Jan M. Żytkow, and Shang-Ching Chou. Automatic theorem generation in plane geometry. In: International Symposium on Methodologies for Intelligent Systems. 1993. 415–424.

[2] Bruno Buchberger. Applications of Gröbner bases in non-linear computational geometry. 1988.

[3] Shang-Ching Chou. An introduction to Wu's method for mechanical theorem proving in geometry. Journal of Automated Reasoning. 1988, 4 (3), 237–267.

[4] Shang-Ching Chou. Mechanical geometry theorem proving. Springer Science & Business Media, 1988.

[5] Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. A collection of 110 geometry theorems and their machine produced proofs using full-angles. Washington State University, Washington. 1994,

[6] Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. Automated generation of readable proofs with geometric invariants. II. Theorem proving with full-angles. Journal of Automated Reasoning. 1996, 17 (3), 349–370.

[7] Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. A deductive database approach to automated geometry theorem proving and discovering. Journal of Automated Reasoning. 2000, 25 (3), 219–246.

[8] Shang-Ching Chou, Xiao-Shan Gao, and Jingzhong Zhang. Machine proofs in geometry: Automated production of readable proofs for geometry theorems. World Scientific, 1994.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to Algorithms, 3rd-edition. The MIT Press. 2009,

[10] Harold Scott Macdonald Coxeter, and Samuel L Greitzer. Geometry revisited. Maa, 1967.

[11] Xiao-Shan Gao, and Qiang Lin. MMP/geometer–a software package for automated geometric reasoning. In: International Workshop on Automated Deduction in Geometry. 2002. 44–66.

[12] M. Hohenwarter, M. Borcherds, G. Ancsin, B. Bencze, M. Blossier, J. Éliás, K. Frank, L. Gál, A. Hofstätter, F. Jordan, Z. Konečný, Z. Kovács, E. Lettner,

S. Lizelfelner, B. Parisse, C. Solyom-Gecse, C. Stadlbauer, and M. Tomaschko. GeoGebra 6.0.507.0. 2018. http://www.geogebra.org.

[13] Predrag Janicic, Julien Narboux, and Pedro Quaresma. The area method: a recapitulation. 2012,

[14] Lars Erik Johnson. Automated elementary geometry theorem discovery via inductive diagram manipulation. Ph.D. Thesis, Massachusetts Institute of Technology. 2015.

[15] Deepak Kapur. Using Gröbner bases to reason about geometry problems. Journal of Symbolic Computation. 1986, 2 (4), 399–408.

[16] Ulrich Kortenkamp, and JURGEN Richter-Gebert. The interactive geometry software Cinderella. In: Mathematical Software: Proceedings of the First International Congress of Mathematical Software: Beijing, China, 17-19 August 2002. 2002. 208.

[17] Noboru Matsuda, and Kurt Vanlehn. Gramy: A geometry theorem prover capable of construction. Journal of Automated Reasoning. 2004, 32 (1), 3–33.

[18] Andreas Poulos. A research on the creation of problems for mathematical competitions. Teaching of Mathematics. 2017, 20 (1),

[19] Rahul Singhal, Martin Henz, and Kevin McGee. Automated generation of high school geometric questions involving implicit construction. In: CSEDU (1). 2014. 467–472.

[20] Wen-Dun TUN. On the decision problem and the mechanization of theorem-proving in elementary geometry. Scientia Sinica. 1978, 21 (2), 159–172.

[21] Dongming Wang. Geother 1.1: Handling and proving geometric theorems automatically. In: International Workshop on Automated Deduction in Geometry. 2002. 194–215.

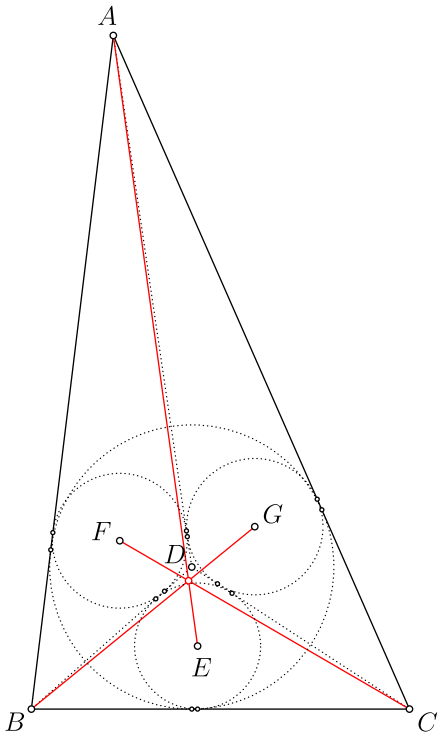
[22] Ke Wang, and Zhendong Su. Automated geometry theorem proving for human-readable proofs. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[23] Sean Wilson, and Jacques D Fleuriot. Combining dynamic geometry, automated geometry theorem proving and diagrammatic proofs. In: Workshop on User Interfaces for Theorem Provers (UITP). 2005.

[24] Zheng Ye, Shang-Ching Chou, and Xiao-Shan Gao. An introduction to java geometry expert. In: International Workshop on Automated Deduction in Geometry. 2008. 189–195.

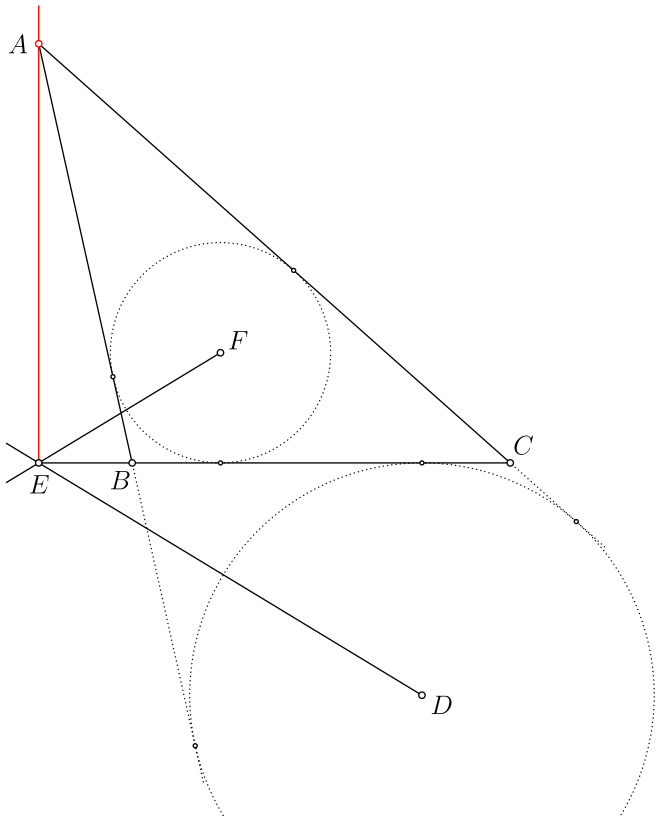
Appendix

The following are nine difficult problems, each for every sought theorem type (Table 2.1.3), generated in the large-scale generation experiments (see Section 3.2), automatically drawn and stated by our visualisation tool (see Section 3.1.1).



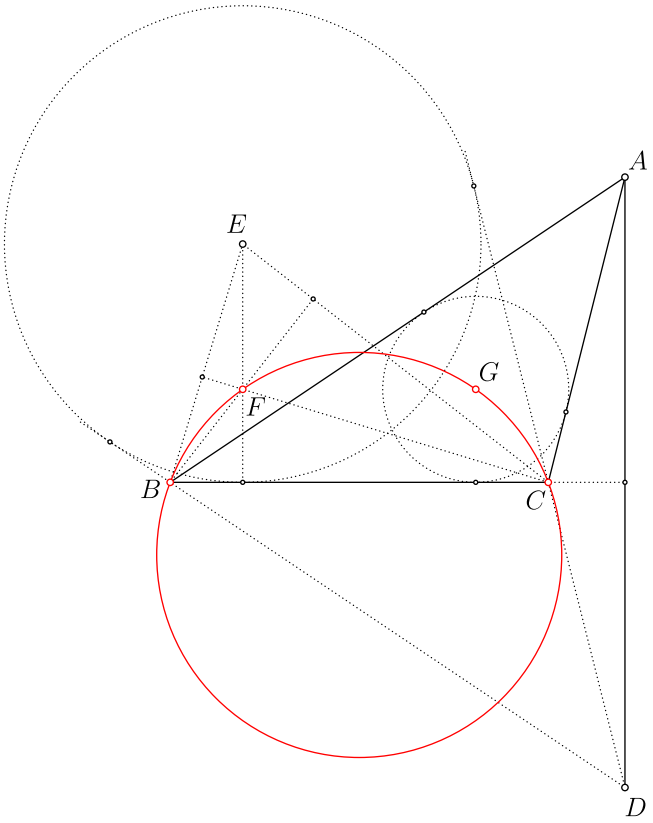
Let ABC be a triangle. Let D be the incenter of ABC. Let E be the incenter of BCD. Let F be the incenter of ABD. Let G be the incenter of ACD.

Show that lines AE, BG, CF are concurrent.



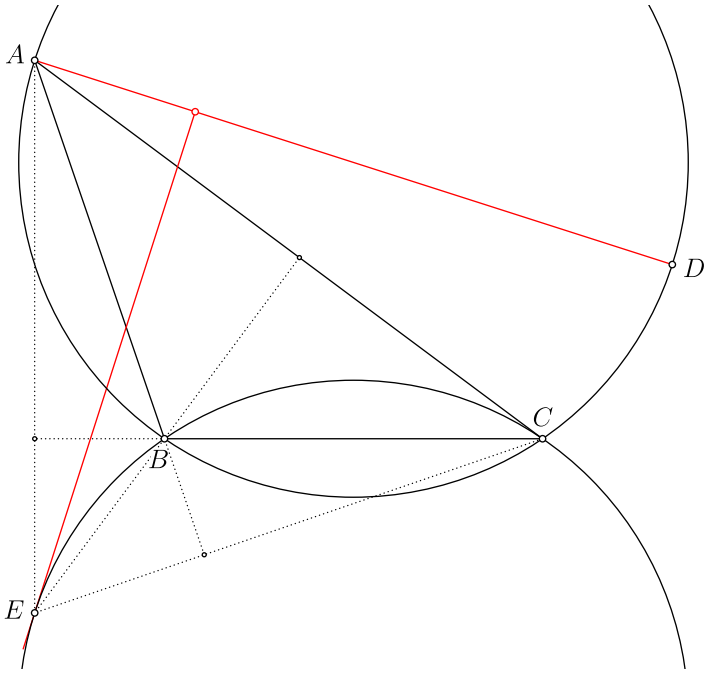
Let ABC be a triangle. Let D be the A-excenter of ABC. Let E be the projection of A on BC. Let F be the incenter of ABC. Let l be the external angle bisector of $\angle DEF$.

Show that A lies on l.



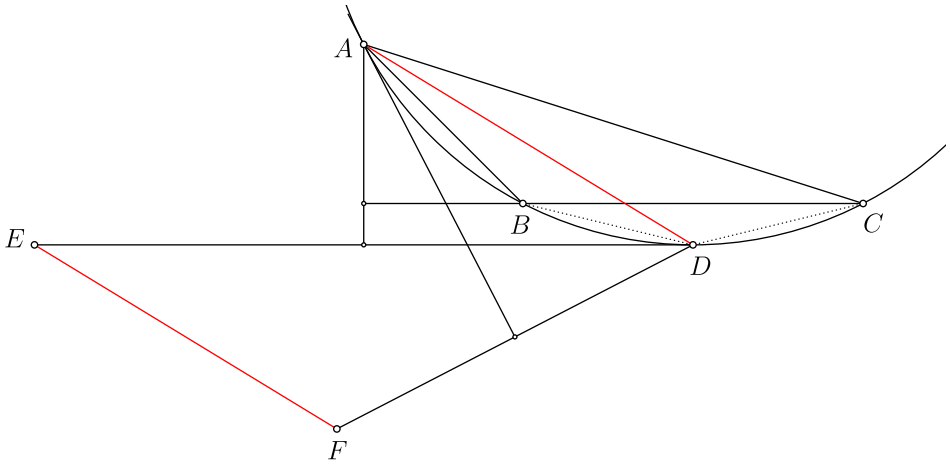
Let ABC be a triangle. Let D be the reflection of A in BC. Let E be the D-excenter of DBC. Let F be the orthocenter of BCE. Let G be the incenter of ABC.

Show that points B, C, F, G are concyclic.



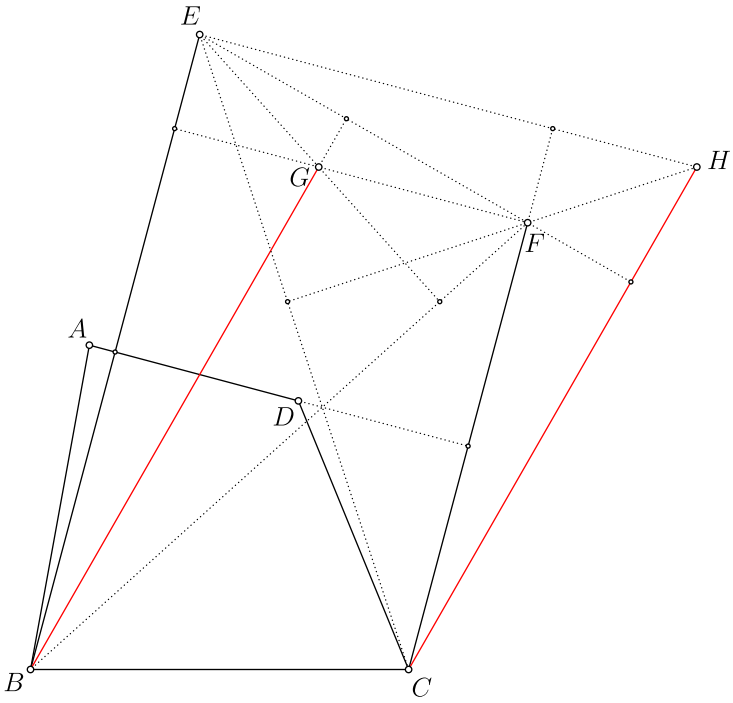
Let ABC be a triangle. Let D be the point opposite to A on circle BAC. Let E be the orthocenter of ABC. Let l be the line tanget to circle EBC at E.

Show that $AD \perp l$.



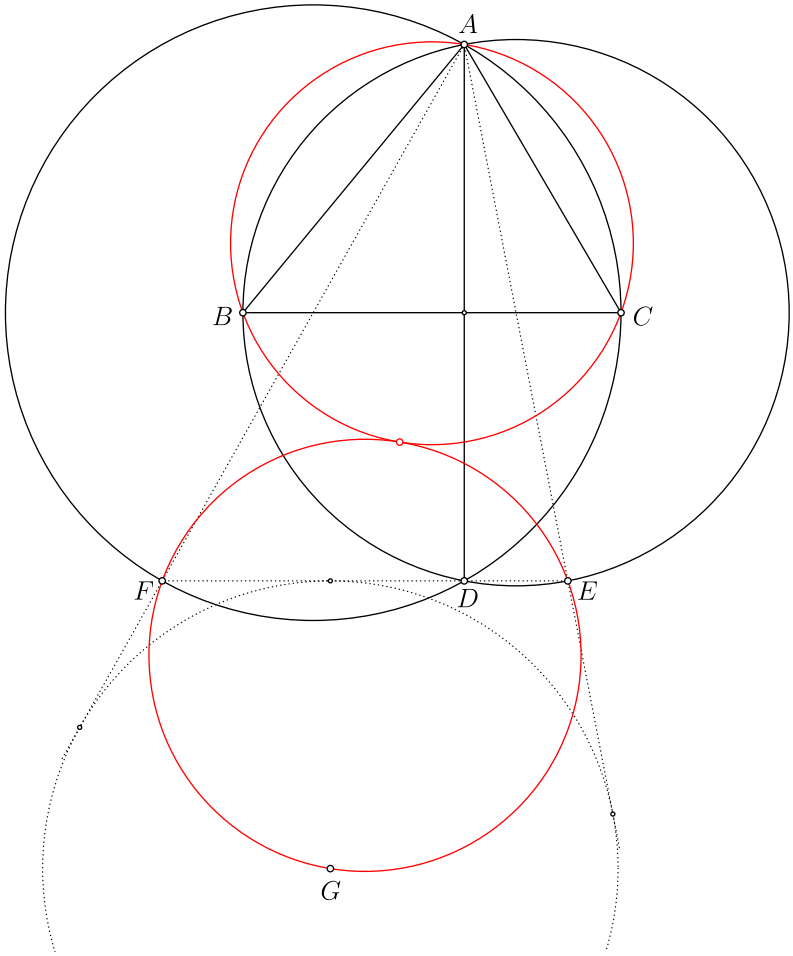
Let ABC be a triangle. Let l_1 be the line through A perpendicular to BC. Let l_2 be the line tangent to circle ABC at A. Let D be the midpoint of opposite arc BAC. Let E be the reflection of D in l_1. Let F be the reflection of D in l_2.

Show that $AD \parallel EF$.



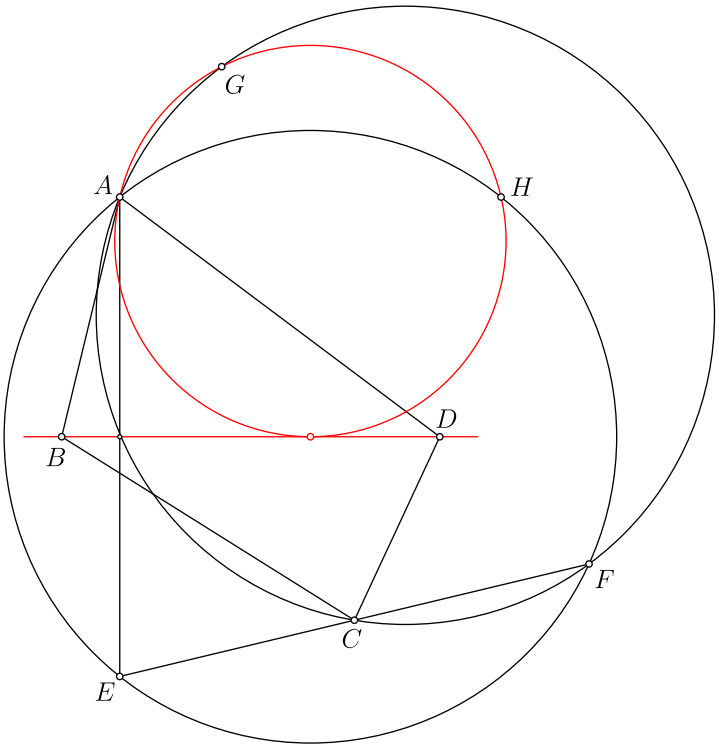
Let $ABCD$ be a convex quadrilateral. Let E be the reflection of B in AD. Let F be the reflection of C in AD. Let G be the orthocenter of BEF. Let H be the orthocenter of CEF.

Show that $BG = CH$.



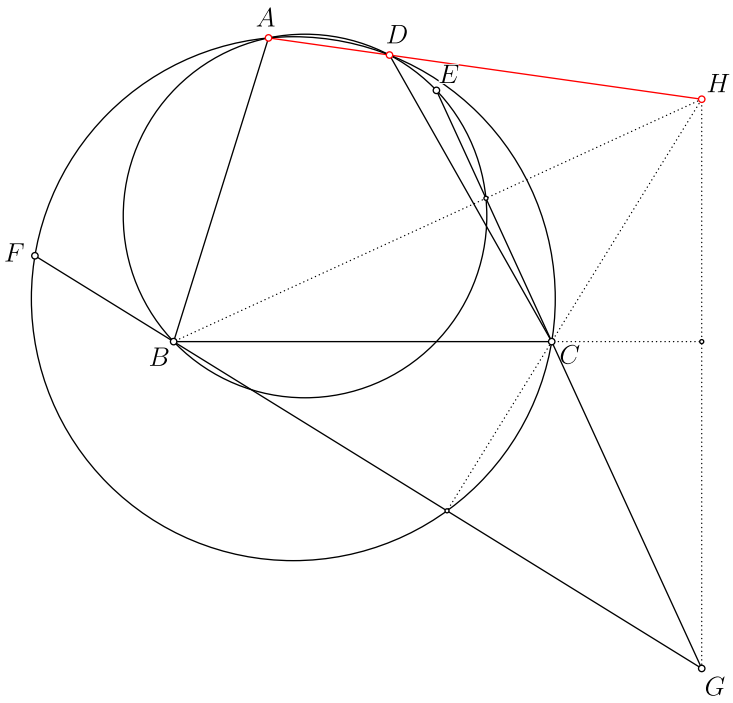
Let ABC be a triangle. Let D be the reflection of A in BC. Let E be the point opposite to A on circle BAD. Let F be the point opposite to A on circle CAD. Let G be the A-excenter of AEF.

Show that circles ABC and EFG are tangent to each other.



Let $ABCD$ be a convex quadrilateral. Let E be the reflection of A in BD. Let F be the reflection of E in C. Let G be the point opposite to F on circle AFC. Let H be the point opposite to E on circle AEF.

Show that line BD and circle AGH are tangent to each other.



Let $ABCD$ be a convex quadrilateral. Let E be the point opposite to B on circle ABD. Let F be the point opposite to C on circle ACD. Let G be the intersection point of BF and CE. Let H be the orthocenter of BCG.

Show that points A, D, H are collinear.