Table of Contents

Polygon Ratio Problem: Many Proofs, One Degree of Freedom

- Part 1. Synthetic geometry (auxiliary lines)
- Part 2. Analytic geometry proof
- Part 3. One more degree of freedom (ellipse view)
- Part 4. Vector analysis

Part 5: Groebner (Grobner) basis and Wu Method: polynomial elimination methods

Part 6. Modern Geometric Prover

Part 7. Aristotle prover

Polygon Ratio Problem: Many Proofs, One Degree of Freedom

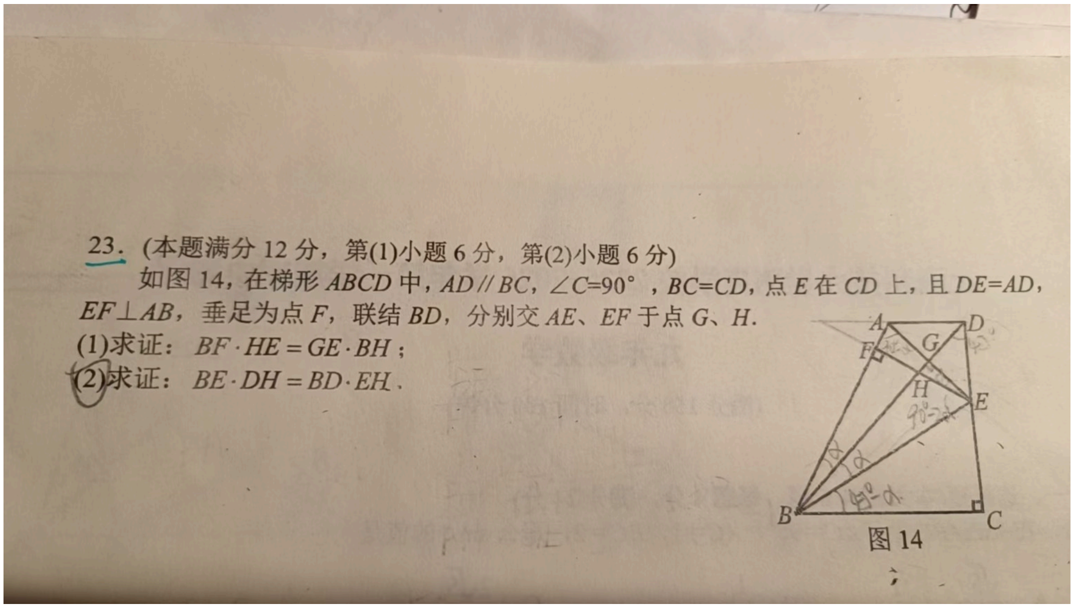
Author: Changchun Shi

Date: First version Jan 12, 2026

Last update Jan 25, 2026

Introduction

During the Christmas holidays, a basic geometry proof problem showed up in my class alumni chat (Figure 1 is the original screenshot from classmate Lin He). He asked about the second question because the first is almost trivial, and that set the discussion in motion. I have always liked Euclidean geometry and excelled in my teenage competitions (I don't recall failing to prove any geometry problems in that period). Problems like this rarely stop at one proof. Draw one line and a ratio drops out; draw another and a different argument appears. What follows is a cleaned collection: one quick sine-law proof, several synthetic proofs, an analytic-geometry proof (with help from ChatGPT 5.2-thinking, edited for clarity), and a one-parameter generalization that reveals deeper structure. I hope it gives younger students a sense of how much structure a single diagram can hide. I also plan to use this pilot problem to introduce state-of-the-art AI approaches for solving such problems in separate notes.


23. (本题满分 12 分，第(1)小题 6 分，第(2)小题 6 分)

如图 14，在梯形 $ABCD$ 中，$AD // BC$，$\angle C=90^\circ$，$BC=CD$，点 E 在 CD 上，且 $DE=AD$，$EF \perp AB$，垂足为点 F，联结 BD，分别交 AE、EF 于点 G、H.

(1)求证：$BF \cdot HE = GE \cdot BH$；

(2)求证：$BE \cdot DH = BD \cdot EH$.

Figure 1: The original problem (photo from the chat).

Let me first restate the problem in words and redraw the diagram more cleanly.

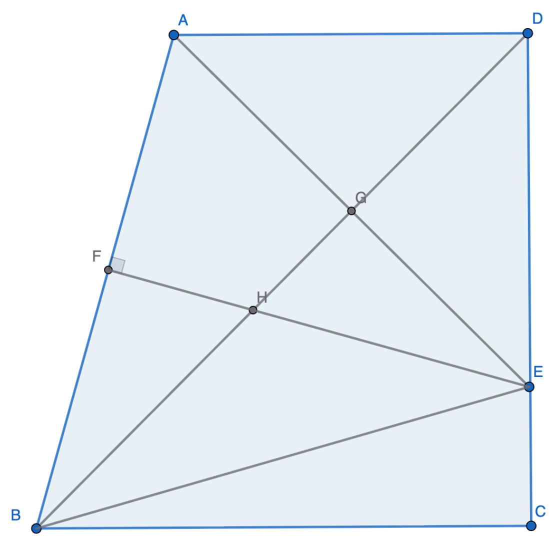


Figure 2: A clean redraw in GeoGebra.

Problem statement (translated). In trapezoid $ABCD$, $AD \parallel BC$, $\square ngle C = 90^\circ$, and $BC = CD$. Point E lies on CD, and $DE = AD$. Through E, draw $EF \perp AB$ with F as the foot of the perpendicular. Draw BD; it intersects AE and EF at points G and H, respectively. Prove:

1. $BF \cdot HE = GE \cdot BH$.
2. $BE \cdot DH = BD \cdot EH$.

Back to TOC

Part 1. Synthetic geometry (auxiliary lines)

The basic equal angles and lengths are shown in Figure 3.

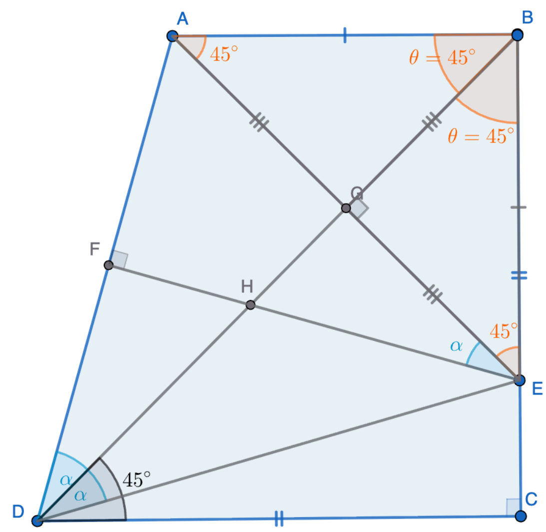


Figure 3: Basic angle and length relations.

Question (1) is a warm-up; it follows from similar triangles visible in Figure 3. For part (2), a quick start is the sine law. For a triangle ABC with side lengths a, b, c opposite vertices A, B, C, we have

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}.$$

This gives Proof 0 immediately.

Proof 0 (sine law)

> The ratios $DB : BE$ and $DH : HE$ are both $\sin(45^\circ + \alpha) : \sin 45^\circ$, so they are equal. This proof uses no auxiliary lines.

To avoid assuming the sine law, I drew auxiliary lines. Rather than separate diagrams per proof, the next figure collects all extra points. Each added point carries a subscript that matches the proof number (for example, D_1 is used only in Proof 1).

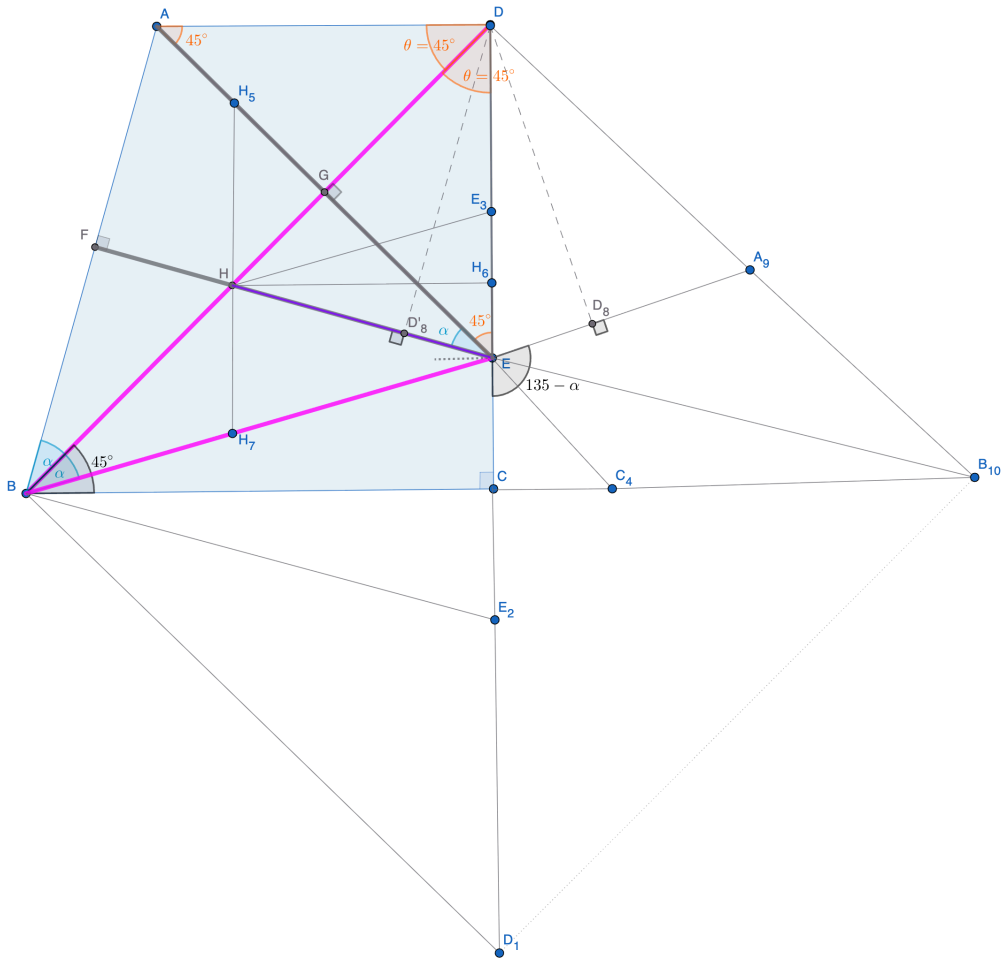


Figure 4: All auxiliary lines used in the synthetic proofs.

Below are brief proof sketches. Each can be completed as a standard middle-school geometry exercise.

Proof 1

> Extend DC to D_1 so that D_1 is symmetric to D with respect to BC. Then $\triangle EBD_1 \sim \triangle EHD$, so the desired ratio follows.

Proof 2

> Extend DC to E_2 so that E_2 is symmetric to E with respect to BC. Then $\triangle E_2BD \sim \triangle EHD$, so the desired ratio follows.

Proof 3

> From H draw a line parallel to BE and intersect DE at E'. Then $HE = HE'$. The rest follows.

Proof 4

> Extend AE to meet BC at C_4. Then $AC_4 = BD$. Also $\triangle HED \sim \triangle ABC_4$, so the ratio follows. (Credit: Prof. Xueheng Lan.)

Proof 4.1

> Use the reflection property: the ray from B to E reflects across DE and passes through H. This gives the same angle relation at DE, so the ratio follows.

Proof 5

> Draw a line from H parallel to DE and intersect EA at H_5. Then $DH = EH_5$, and $\triangle EHH_5 \sim \triangle BED$, which leads to the result.

Proof 6

> Draw a line from H parallel to BC and intersect DE at H_6. Then $DH/DB = HH_6/BD = EH/BD$.

Proof 7

> Draw a line from H parallel to DE and intersect BE at H_7. Then $HE : BE = H_7E : BE = HD : BD$.

Proof 8

> The distances from D to BE and EH are equal (use DD_8 and DD'_8; these are non-essential since $\angle BEC = \angle HED = 45^\circ + \alpha$). The distances from E to BD and HD are also equal. So area BDE to area DHE can be expressed as $BE \cdot DD_8/2 : EH \cdot DD'_8/2 = BE : EH$, and also as $BD \cdot GE/2 : DH \cdot GE/2 = BD : DH$. Hence they are equal.

Proof 9

> Extend BE to A_9 where $EA_9 = EH$. Then $DH = DA_9$, $EH = EA_9$, and $\angle EDA_9 = \angle EDH = 45^\circ$. Since DE bisects $\angle BDA_9$, we get $BD : DA_9 = BE : EA_9$. The rest is trivial.

Proof 10

> Extend EH to B_{10} where B_{10} intersects the extension of line BC. Then B_{10} is the reflection of B across BC (since HE is the reflection path of light BE at mirror DC). One can show $\angle CDB_{10} = \angle BDC = 45^\circ$, so $DH : HE = DB_{10} : EB_{10} = DB : BE$.

Proof 11

> This one also needs no auxiliary lines: notice $HD = HG + GE$ and $BD = BG + GE$, and $\triangle HGE \sim \triangle EGB$, so the result follows.

A note on degrees of freedom: the only adjustable angle in the original configuration is $\alpha < 45^\circ$. If $\alpha = 45^\circ$, then H and B overlap and the result is trivial. If $\alpha > 45^\circ$, then H lies on the extension of DB, and E lies on the extension of DC. The roles of B and H swap, and the same style of arguments still works. Similarly, one can show that a ray from B to E reflects across DE and reaches H.

Other proofs

By now we should be done. However, if you followed the solutions so far, you likely noticed two patterns: (1) we have essentially drawn a square $BDB_{10}D_1$ in Figure 4, and many proofs are based on various points on this square; (2) polygon $ABED$ is symmetric about BD, which is why we could leverage $AB = BE$ and $AH = HE$ in several steps. It follows that any proof obtained by extending lines in the original diagram has a mirror proof across BD. These are indicated again by subscripts, and they give Proofs 12, 13, 14, 15, and 16. They form the symmetric rectangle $BDB_{15}D_{12}$ and are shown in Figure 5.

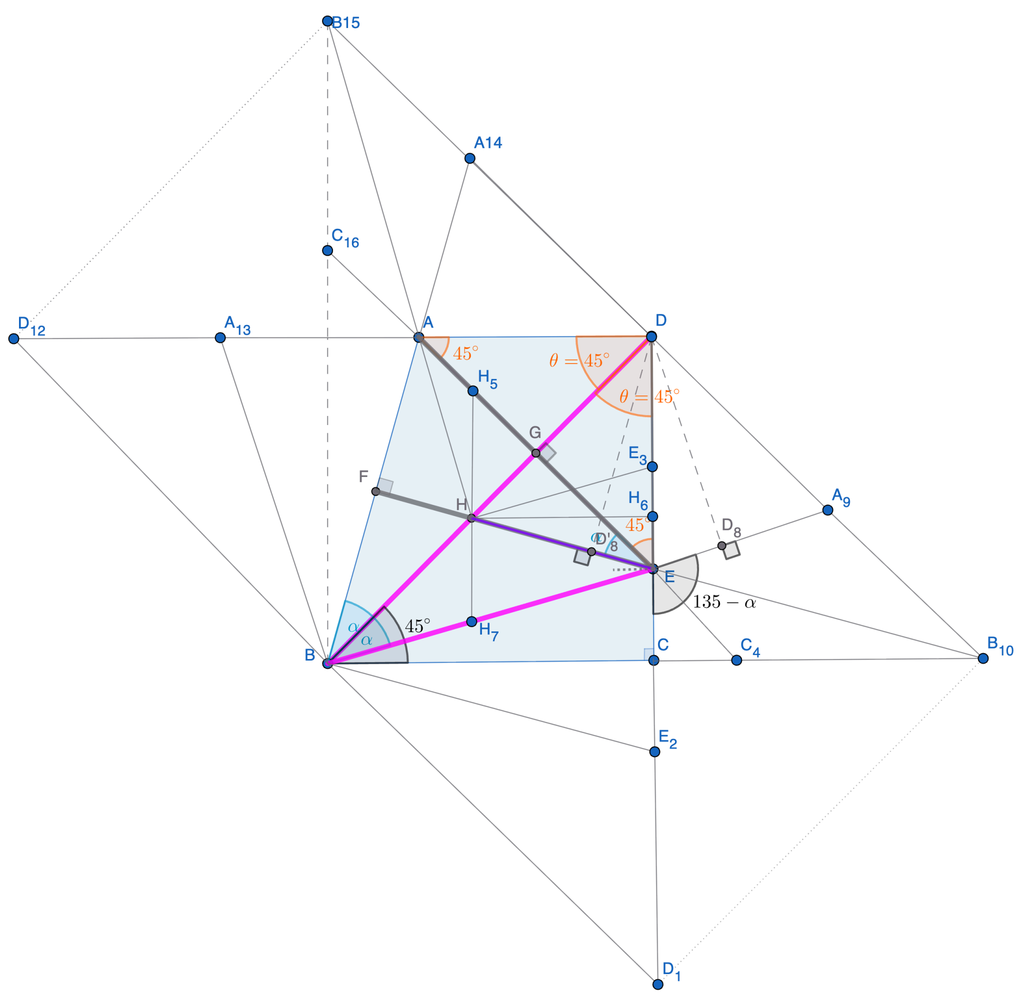


Figure 5: Five additional symmetric proofs.

Finally, several proofs such as Proofs 3, 5, 6, and 7 focus on what happens inside polygon $ABED$. Each has a symmetric counterpart (Proofs 17, 18, 19, 20) reflected across BD. For clarity, these are not displayed.

Back to TOC

Part 2. Analytic geometry proof

This is a short coordinate proof.

Goal

$$BE \cdot DH = BD \cdot EH.$$

Proof

> Setup. Let $C = (0, 0)$, $B = (-1, 0)$, $D = (0, 1)$. Then $AD \parallel BC$ implies $A = (t - 1, 1)$ when $E = (0, t)$ and $DE = AD$ with $0 < t < 1$.
>
> Lines. $BD : y = x + 1$. The slope of AB is $1/t$, so $EF \perp AB$ has slope $-t$ and passes through E: $EF : y = t - tx$.
>
> Intersection. Solve $x + 1 = t - tx$ to get
>
> $$H = (\frac{t-1}{t+1}, \frac{2t}{t+1}).$$
>
> Common factor.
>
> $$H - D = \frac{t-1}{t+1}(1, 1), \qquad H - E = \frac{t-1}{t+1}(1, -t).$$
>
> Hence
>
> $$DH = \frac{1-t}{t+1}\sqrt{2}, \qquad EH = \frac{1-t}{t+1}\sqrt{1+t^2}.$$
>
> Also $BD = \sqrt{2}$ and $BE = \sqrt{1+t^2}$.
>
> Finish.
>
> $$BE \cdot DH = \sqrt{1+t^2} \cdot \frac{1-t}{t+1}\sqrt{2} = BD \cdot EH.$$

Conclusion: $BE \cdot DH = BD \cdot EH$.

Back to TOC

Part 3. One more degree of freedom (ellipse view)

The 45° angle is special. If we relax that angle, the clean right-angle structure disappears, but the problem still has a simple geometric core: reflection on an ellipse. The key insight is to generalize the reflection: the ray from B to E reflects across DE and heads to H. This reveals an ellipse that reflects light from one focus to the other at any point E on it. That reflection property is equivalent to the constant-sum-of-distances definition of an ellipse.

Construction

Let O be the ellipse with foci B and H. Take a point A on O, with A closer to H. Draw the tangent line AD at A, and let it intersect BH at D. Take a point E on O, the reflection of A with respect to the major axis. Through E draw the line $D-E-C$, and choose C so that $BC \parallel AD$. Draw EH, and let it intersect AB at F. In general, $\angle BFE$ is not 90°, but when $\theta = 45^\circ$, $\angle BFE = 90^\circ$. The resulting shape is shown in Figure 6.

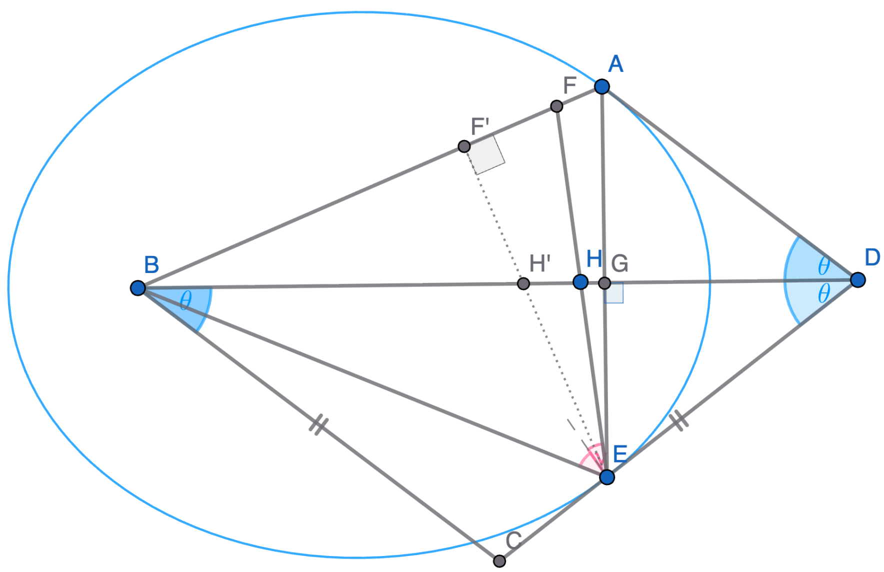


Figure 6: Ellipse construction with foci B and H.

Proof 3.1 (reflection property)

> By the reflection property of the ellipse (a light ray from B to E reflects and then passes through H), we obtain
>
> $$\frac{EH}{HD} = \frac{BE}{BD}.$$
>
> Reflect H across line DE to H'. Then

$$\frac{DB}{BE} = \frac{DH'}{EH'} = \frac{DH}{EH},$$

which is the same proportion.

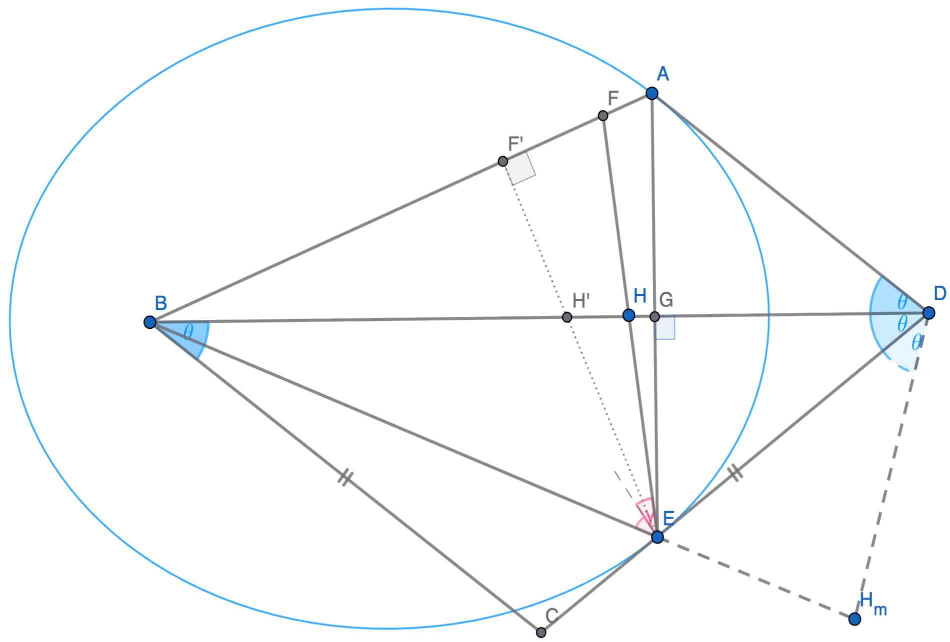


Figure 7: Reflection proof for the ellipse construction.

In this generalized setup, EF is no longer perpendicular to AB. In fact, the perpendicular line through E intersects AB and DB at F' and H', as shown in Figure 7. Here H' differs from H and does not satisfy the ratio property. The original problem is special because the 45° angles create a surplus of right angles. The single degree of freedom here is essentially the ellipse eccentricity, with A and C determined by the right angle $\angle ADC$. When the ellipse degenerates to a circle (foci coincide), the identity becomes obvious.

In this setup, points A, F, C, and G, and the lines through them, are not even essential; only the foci B and H, and the tangent line DE, are necessary, with D on the major axis. This shows the true simplicity of the original problem; Figure 8 shows the pure geometric approach.

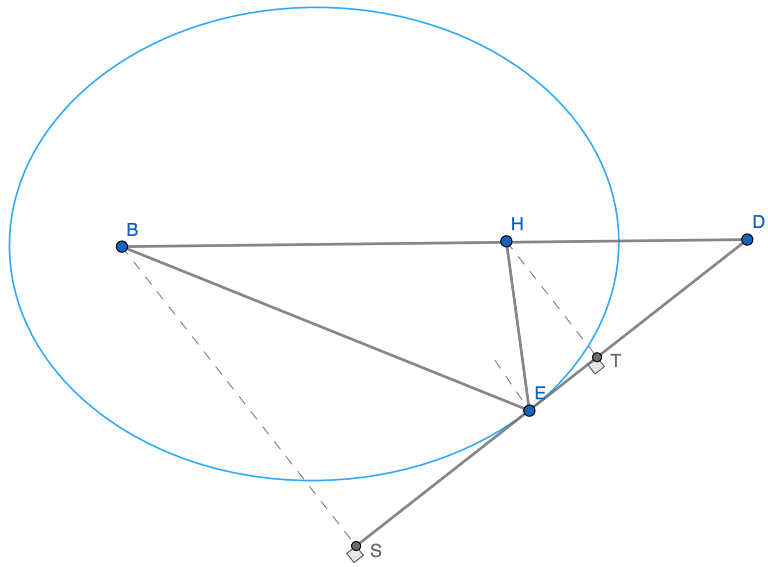


Figure 8: The simplified ellipse setup, which reveals the essence of the original problem.

The diagram also yields a second, similarity-based proof.

Proof 3.2, similarity

> Let T and S be the feet of the perpendiculars from H and B to the tangent line DE. By the reflection property, $\triangle HET \sim \triangle BES$, so $HE : BE = HT : BS$. Likewise, $\triangle HDT \sim \triangle BDS$, so $HD : DB = HT : BS$. Combining gives $HD/DB = HE/BE$.

Interestingly, we can also compute the ratio analytically in the ellipse model. It turns out to be simplest using the de La Hire (eccentric-angle) parameter, revealing the geometry transparently.

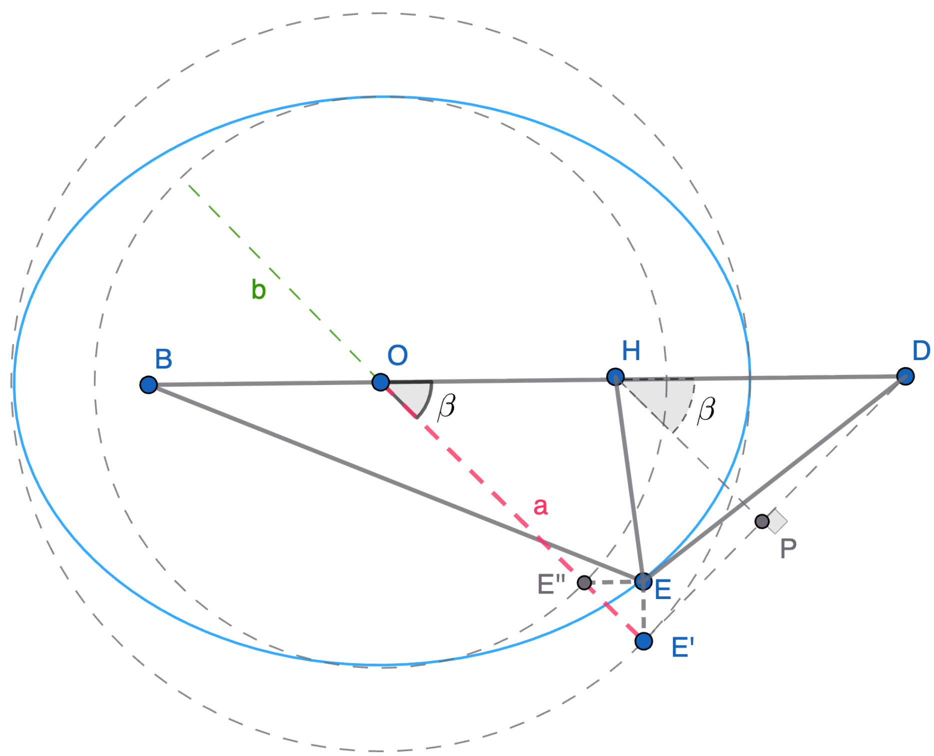


Figure 9: de La Hire parameters and outer-circle view with E, E', and E''.

Proof 3.3 (de La Hire + outer circle)

> Let the ellipse be $x^2/a^2 + y^2/b^2 = 1$ with center O, foci $B = (-c, 0)$ and $H = (c, 0)$, and $c^2 = a^2 - b^2$. Use the de La Hire (eccentric-angle) parameter β:
>
> $$E = (a\cos\beta,\, b\sin\beta), \qquad EB = a + c\cos\beta, \qquad EH = a - c\cos\beta$$
>
> (up to swapping B, H). Let ℓ be the tangent at E and $D = \ell \cap BH$.
>
> From the tangent formula
>
> $$\frac{xx_0}{a^2} + \frac{yy_0}{b^2} = 1 \quad \Rightarrow \quad \frac{x\cos\beta}{a} + \frac{y\sin\beta}{b} = 1,$$
>
> setting $y = 0$ gives $x_D = a/\cos\beta$.
>
> Geometrically, the affine map $(x, y) \mapsto (x, \frac{a}{b}y)$ sends the ellipse to the outer circle $x^2 + y^2 = a^2$ and sends E to $E' = (a\cos\beta, a\sin\beta)$. Since only y is scaled, E and E' lie on a line perpendicular to BH and share the same x-coordinate. Let E'' be the intersection of the line through E parallel to BH with the outer circle, so E and E'' share the same y-coordinate (Figure 9). The map preserves tangency, so $\angle OE'D$ is right and $OD = a/\cos\beta$.
>
> Therefore
>
> $$DB = \frac{a}{\cos\beta} + c, \qquad DH = \frac{a}{\cos\beta} - c,$$

> and
>
> $$\frac{DH}{DB} = \frac{a - c\cos\beta}{a + c\cos\beta} = \frac{EH}{EB}.$$

Corollary 1 (axis-focus scaling; inner structure of the ratio)

> From $DH = \frac{a}{\cos\beta} - c = \frac{a - c\cos\beta}{\cos\beta}$ and $EH = a - c\cos\beta$, we get
>
> $$\boxed{EH = DH\cos\beta}, \qquad \boxed{EB = DB\cos\beta}.$$
>
> This shows that the tangent-axis lengths DB, DH are just the focal lengths EB, EH scaled by the same factor $1/\cos\beta$ (see Figure 9). The main identity $DH/DB = EH/EB$ is exactly the cancellation of that common scale.

Corollary 2 (focus-to-outer-tangent distances)

> Corollary 1 gives $EH = DH\cos\beta$, and geometrically $DH\cos\beta$ is exactly the perpendicular component of DH to ℓ', i.e., $\operatorname{dist}(H, \ell')$ (see Figure 9). Hence
>
> $$\boxed{\operatorname{dist}(H, \ell') = EH}, \qquad \boxed{\operatorname{dist}(B, \ell') = EB}.$$
>
> The simple consistency check
>
> $$\operatorname{dist}(B, \ell') + \operatorname{dist}(H, \ell') = 2a = 2\ \operatorname{dist}(O, \ell')$$
>
> accounts only for the sum. The real content is that each focus-to-tangent distance equals its matching focus-to-point distance; this rigidity is encoded by the de La Hire parameter and the ellipse's affine relation to the circle.

Proof 3.4 (outer circle proof)

> Once Corollary 2 is in hand, let P and P' be the feet of the perpendiculars from H and B to ℓ' (for simplicity Figure 9 only shows P). Then $\triangle DHP \sim \triangle DBP'$ (right angles at P, P' and a common angle at D), so $DH/DB = HP/BP' = \operatorname{dist}(H, \ell')/\operatorname{dist}(B, \ell')$, recovering the same ratio.

Finally, there is a fun fact. Note that the feet of the perpendiculars drawn in Proof 3.2, S and T, are exactly the intersection points of the tangent line DE with the ellipse's outer circle. A quick proof is to show that $OT = a$ (Figure 10). Let H' be the reflection of H across DE; then T is the midpoint of HH'. Since O is the midpoint of BH, we have $OT = \frac{1}{2}BH'$. But $BH' = BE + EH = 2a$, hence $OT = a$. The same reflection with B gives $OS = a$, so the fun fact is that S and T are precisely the two intersections of DE with the outer circle.

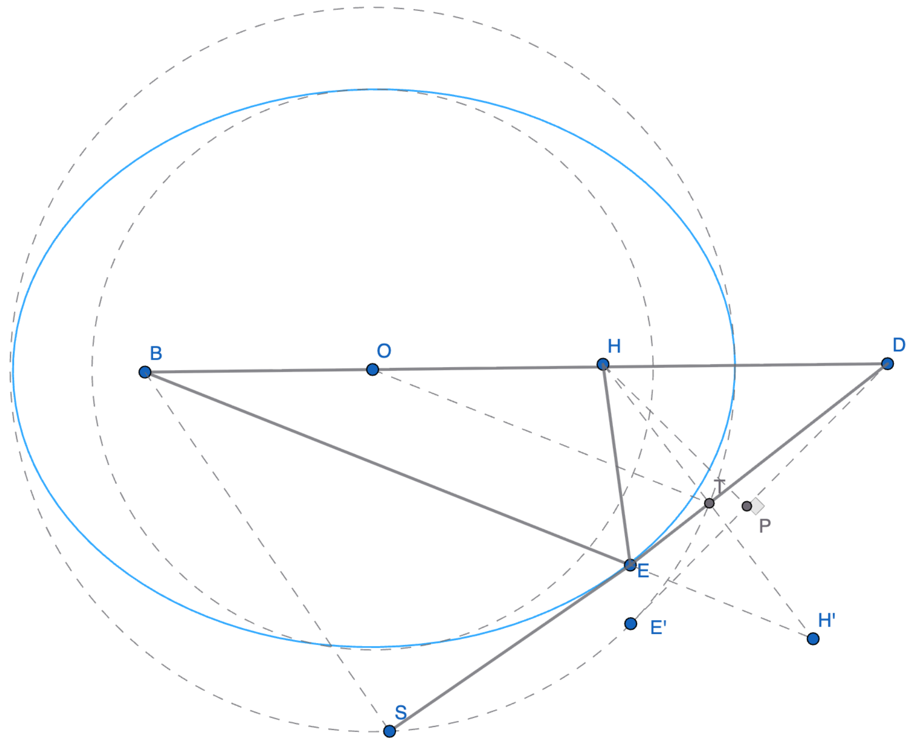


Figure 10: additional properties of T and S

Back to TOC

Part 4. Vector analysis

Once one length is fixed, the trapezoid configuration has a single degree of freedom: the angle α. In the broader ellipse construction, a second parameter θ (the tangent point/slope) appears, and the same ratio follows from the reflection property. The special case $\theta = 45^\circ$ is exactly the original problem; that is why so many proofs exist, and why each auxiliary line exposes the same invariant ratio in a different way. Part 3 frees the θ dimension while keeping the reflection property, and that is what ultimately shows the simplicity behind the original picture.

In this part of the note, we focus on other mathematical tools that are useful in geometry. The target ratio is almost trivial here: $DH : DB = EH : EB$, given that the ray BE reflects across the mirror DE and reaches H. Part 3 already explained the reflection viewpoint, but it is still instructive to see how these tools express the same invariant.

Since Euclid (~300 BCE), geometry was developed in a synthetic style: one reasons directly with the figure using congruence, similarity, tangency, and auxiliary lines. That is the mindset behind Part 1. A major second language arrived with Descartes (1637): analytic geometry turns geometric constraints into algebra once you choose coordinates, matching the style of Part 2.

This arc also mirrors the historical split between geometry and algebra in physics and mechanics. Newton, in the *Principia* [NEWTON-PRINCIPIA], favored coordinate-free arguments because they were the clearest language of rigor in 1687, while modern physics often relies on coordinates for universal computation. Analytic geometry existed by Newton's time, but the algebraic language for differential equations matured later; Newton's methods therefore look Euclidean in flavor, mixing auxiliary lines with limiting arguments. Several later authors also gave elementary, coordinate-free proofs (see [KEPLER-TEACH], including Feynman's famous "lost lecture"). It is good for students to appreciate both techniques.

Physics naturally blends geometry with differential ideas. Here we keep that discussion light and focus on the geometric tools that sit just below calculus.

In the 1800s, geometry gained an especially efficient algebraic interface: vector analysis, where the dot and cross products encode the two most basic geometric primitives--projection/angle and oriented area. The modern $\cdot$ and $\times$ notation was introduced by Gibbs (and independently Heaviside) around 1881 and then standardized in physics usage. Later, divergence and curl became standard because they compactly describe fields and their integral laws--but that differential layer is not our focus here.

That same "area backbone" explains the next conceptual step. Grassmann (1844) introduced the wedge product and exterior algebra, where the fundamental geometric object is no longer just a vector but also a bivector (directed area), trivector (directed volume), and so on. Clifford

(1878) then built geometric algebra by unifying these objects into one coherent multiplication system. In our 2D Euclidean setting,

$$|\mathbf{u} \times \mathbf{t}|, \qquad \|\mathbf{u} \wedge \mathbf{t}\|, \qquad |\omega(\mathbf{u}, \mathbf{t})|$$

are simply three notations for the same invariant: the area spanned by $\mathbf{u}$ and the tangent direction $\mathbf{t}$. So the proofs are interchangeable across these languages with no change in geometric content.

Finally, modern physics is "geometric" in a deeper sense: it naturally lives on spaces that are not just 2D pictures--3D space, 4D spacetime, and even higher-dimensional configuration/phase spaces. We don't always call those "geometry" in everyday language because they're hard to visualize, but mathematically they are geometry: they carry intrinsic notions of angle, area/flux, invariance, and coordinate-free meaning. In our ellipse problem, we see a toy version of the same idea: at each point E on the ellipse, there is a tangent line DE. Thinking of the tangent line as the attached object that moves with E is the simplest concrete example of the bundle/fiber mindset.

A brief timeline of tooling relevant to this note:

- ~300 BCE: Euclid -- synthetic geometry
- 1637: Descartes -- analytic geometry
- 1687: Newton -- Euclidean geometry + limiting arguments
- 1827: Gauss -- intrinsic geometric invariants (surfaces)
- 1844: Grassmann -- wedge / bivectors (area as an object)
- 1878: Clifford -- geometric algebra (bivectors/trivectors as first-class)
- 1881: Gibbs/Heaviside -- standard dot/cross notation
- 1899: Cartan -- exterior derivative/forms (beyond our needs here)

For readers familiar with Maxwell's equations, the table below shows how mathematical language for geometry evolved over time: from coordinate-expanded component equations, to vector analysis, to geometric algebra, and then to differential forms. The tools change, but the physics does not, even as the expressions become more compact.

Formulation	Coordinate status (what's really true)	Maxwell equations written in that language	What you gain / lose	Good reference
Maxwell (1865, component-heavy)	Coordinate-invariant laws, but written in components (explicit x, y, z bookkeeping) and bundled with extra relations	Not packaged as the modern four; a larger coupled component system (often summarized as ~20 equations/unknowns, plus auxiliaries).	Gain: historically close to derivations. Lose: compactness; structure is obscured by bookkeeping.	Maxwell's 1865 paper [MAXWELL-1865]; Deschamps [FORMS-EM].

Formulation	Coordinate status (what's really true)	Maxwell equations written in that language	What you gain / lose	Good reference
	(constitutive laws, Ohm, forces, potentials).			
Vector calculus (Heaviside/Gibbs)	Coordinate-free in content; coordinates enter only when writing component formulas.	(SI, vacuum) $\nabla \cdot \mathbf{E} = \rho/\varepsilon_0$ $\nabla \cdot \mathbf{B} = 0$ $\nabla \times \mathbf{E} = -\partial_t \mathbf{B}$ $\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \varepsilon_0 \, \partial_t \mathbf{E}$.	Gain: four clean equations; practical in 3D Euclidean space. Caveat: $\nabla \cdot, \nabla \times$ depend on the chosen metric + orientation, though invariantly defined once fixed.	Wilson [VECTOR-ANALYSIS]; cross product [CROSS-PRODUCT]; vector analysis [VECTOR-ANALYSIS-WIKI].
Geometric algebra (Clifford; dot + wedge unified)	Coordinate-free once metric + orientation are fixed. Vectors are bold (e.g., $\mathbf{a}, \mathbf{b}$); higher-grade objects (bivectors, trivectors) are not vectors in disguise.	Core algebra: $\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$. Maxwell can be written compactly (common spacetime GA convention): $\boxed{\nabla F = \mu_0 J}$, with the grade split $\nabla F = \nabla \cdot F + \nabla \wedge F$ giving $\nabla \wedge F = 0$ (homogeneous) and $\nabla \cdot F = \mu_0 J$ (sources).	Gain: cross product becomes derived (dual of $\wedge$); div/curl unify as one operator split by grade; rotations often cleaner. Lose: conventions vary (3D GA vs spacetime GA; units/signatures); less standard in mainstream EM courses.	Peeter Joot [GA-EM].
Differential forms (exterior calculus on spacetime)	Manifestly coordinate-invariant; works naturally on curved manifolds (GR). Metric enters via the Hodge star.	$\boxed{dF = 0}$, $\boxed{d{*}F = J}$. Here $*$ maps k-forms to $(n-k)$-forms using the metric + orientation (in 4D, $2 \leftrightarrow 2$).	Gain: topology/relativity are transparent; $d^2 = 0$ exposes structure. Lose: abstraction up front; computations often unpack to components.	Deschamps [FORMS-EM]; Lindell [LINDELL-DF]; Epstein [EPSTEIN-DG].

Table 1 summarizes the evolution of Maxwell's equations across these languages.

Notes:

- Geometric algebra (Clifford; dot + wedge unified): metric + wedge fused immediately via $\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$.
- Differential forms (exterior calculus on spacetime): forms are wedge-first, metric-later via $*$.

So the concept of vector analysis has evolved significantly since Newton's time. Maxwell's equations were first written in a coordinate-expanded form, which has the flavor of our solution in Part 2, along with differential operators that were by then well understood. Only later did divergence and curl become standard, giving a more compact, coordinate-free expression using $\nabla\cdot$ and $\nabla\times$. Those operators are coordinate-free in meaning, which parallels the shift from the coordinate solution in Part 2 back to the coordinate-free constructions in Part 1. Vector analysis keeps the geometric insight while still enabling the algebraic computations of the coordinate approach. To explain the power of this toolbox, I include a couple of short proofs below.

Since our problem is 2D, both dot and cross products can be used to solve it naturally. We emphasize how these tools enable coordinate-free proofs while still allowing analytic methods beyond the purely Euclidean geometry used in Part 1, yet more direct than the coordinate approach in Part 2.

Proof 4.1 (reflection on DE)

> Use the reflection property: a ray from B to E reflects across DE and passes through H. Let $p = |\overrightarrow{EB}|$, $q = |\overrightarrow{EH}|$, and define unit vectors $\hat{u} = \overrightarrow{EB}/p$, $\hat{v} = \overrightarrow{EH}/q$. Then $\overrightarrow{EB} = p\hat{u}$ and $\overrightarrow{EH} = q\hat{v}$, with $|\hat{u}| = |\hat{v}| = 1$. The tangent at E is perpendicular to $\hat{u} + \hat{v}$, so its direction is parallel to $\hat{u} - \hat{v}$. Hence
>
> $$\overrightarrow{ED} = \tau(\hat{u} - \hat{v})$$
>
> for some scalar τ. Because $D \in BH$,
>
> $$\overrightarrow{BH} = \overrightarrow{EH} - \overrightarrow{EB} = q\hat{v} - p\hat{u}, \quad \overrightarrow{BD} = \mu\overrightarrow{BH}.$$
>
> Using $\overrightarrow{ED} = \overrightarrow{EB} + \overrightarrow{BD}$ gives
>
> $$\tau(\hat{u} - \hat{v}) = p(1-\mu)\hat{u} + \mu q\hat{v}.$$
>
> Matching coefficients yields $\tau = p(1-\mu)$ and $-\tau = \mu q$, so $\mu = \frac{p}{p-q}$. Finally, $\overrightarrow{HD} = \overrightarrow{BD} - \overrightarrow{BH} = (\mu - 1)\overrightarrow{BH}$, so
>
> $$\frac{BD}{HD} = \frac{\mu}{\mu - 1} = \frac{p}{q} = \frac{BE}{EH},$$
>
> which is equivalent to $\frac{BE}{BD} = \frac{EH}{HD}$.

Proof 4.2 (using the cross product)

> Let the tangent at E meet BH at D, and take any nonzero tangent vector $\overrightarrow{ED}$. The reflection law says the tangent bisects the angle between $\overrightarrow{EB}$ and $\overrightarrow{EH}$, so the perpendicular components match:

> $$\frac{|\overrightarrow{EH} \times \overrightarrow{ED}|}{EH} = \frac{|\overrightarrow{EB} \times \overrightarrow{ED}|}{EB}.$$
>
> Now $\overrightarrow{EH} = \overrightarrow{ED} + \overrightarrow{DH}$ and $\overrightarrow{EB} = \overrightarrow{ED} + \overrightarrow{DB}$. Cross with $\overrightarrow{ED}$ and drop the $\overrightarrow{ED} \times \overrightarrow{ED}$ term:
>
> $$\overrightarrow{EH} \times \overrightarrow{ED} = \overrightarrow{DH} \times \overrightarrow{ED}, \qquad \overrightarrow{EB} \times \overrightarrow{ED} = \overrightarrow{DB} \times \overrightarrow{ED}.$$
>
> So
>
> $$\frac{|\overrightarrow{DH} \times \overrightarrow{ED}|}{EH} = \frac{|\overrightarrow{DB} \times \overrightarrow{ED}|}{BE}.$$
>
> Because B, D, H are collinear, $\overrightarrow{DB} \parallel \overrightarrow{DH}$, hence
>
> $$|\overrightarrow{DB} \times \overrightarrow{ED}| : |\overrightarrow{DH} \times \overrightarrow{ED}| = BD : HD.$$
>
> Substitute to get
>
> $$\boxed{\frac{BD}{HD} = \frac{BE}{EH}},$$
>
> which is equivalent to $\frac{BE}{BD} = \frac{EH}{HD}$.

I explored several alternative constructions which give various other proofs that I won't detail here since these two are the simplest and representative. For instance, in the original configuration, the condition $EH \perp AB$ immediately yields the dot-product constraint $\overrightarrow{EH} \cdot \overrightarrow{AB} = 0$. This orthogonality can be used to reduce the degrees of freedom without appealing to any reflection argument. More generally, one can start from essentially any point in the diagram, parameterize the remaining configuration by a small set of length (or scaling) parameters, and then drive the argument to the target length-ratio identity. There are many such routes, and they do not require introducing any auxiliary lines.

In this 2D Euclidean setting, the geometric-algebra approach does not introduce additional structure compared with the cross product. Both encode the oriented (signed) area of the parallelogram spanned by two vectors. The $\times$ operator only makes traditional sense in 3D space (it produces a vector perpendicular to the plane), whereas the $\wedge$ operator gives a bivector that is defined in any dimension. Likewise, the differential-forms formulation is not fundamentally different here: a 2-form takes two vectors as input and returns a scalar, and a 1-form is essentially a linear functional (often represented by a dot product with a fixed vector). The identities therefore reduce to the same area or determinant computations already used in the cross-product argument. For completeness, Proof 4.3 shows how they are unified in one proof. In modern math/physics, where the dimensionality is often beyond 3, the dot/cross picture becomes awkward, which is why spacetime formulations separate time from space and why geometric algebra and differential forms are preferred. That is also why the unified Maxwell equations are usually written without an explicit time coordinate.

Proof 4.3: Unified simple solution (area language: cross = wedge = 2-form = 1-form)

> Let DE be the tangent at E and let $\mathbf{t} = \overrightarrow{ED} \neq 0$. For any vector $\mathbf{u}$ based at E (here $\overrightarrow{EB}, \overrightarrow{EH}, \overrightarrow{DB}, \overrightarrow{DH}$), the area with the tangent is the same in all languages:
>
> $$A(\mathbf{u}) = |\mathbf{u} \times \mathbf{t}| = \|\mathbf{u} \wedge \mathbf{t}\| = |\omega(\mathbf{u}, \mathbf{t})|,$$
>
> where ω is an oriented area 2-form.
>
> Fix ω and define the 1-form $\alpha = \iota_{\mathbf{t}}\omega$, so $\alpha(\mathbf{u}) = \omega(\mathbf{u}, \mathbf{t})$ and $|\alpha(\mathbf{u})| = A(\mathbf{u})$. This is the perpendicular component of $\mathbf{u}$ to the tangent.
>
> Reflection at E gives equality of perpendicular components:
>
> $$\frac{A(\overrightarrow{EH})}{EH} = \frac{A(\overrightarrow{EB})}{EB}.$$
>
> Since $D \in BH$,
>
> $$\overrightarrow{EH} = \overrightarrow{ED} + \overrightarrow{DH}, \qquad \overrightarrow{EB} = \overrightarrow{ED} + \overrightarrow{DB}.$$
>
> Using bilinearity and $\omega(\mathbf{t}, \mathbf{t}) = 0$ gives
>
> $$A(\overrightarrow{EH}) = A(\overrightarrow{DH}), \qquad A(\overrightarrow{EB}) = A(\overrightarrow{DB}).$$
>
> Hence
>
> $$\frac{A(\overrightarrow{DH})}{EH} = \frac{A(\overrightarrow{DB})}{EB}.$$
>
> Because B, D, H are collinear, $A(\overrightarrow{DB})/A(\overrightarrow{DH}) = BD/HD$, so
>
> $$\frac{BD}{HD} = \frac{BE}{EH}, \quad \text{i.e.} \quad \frac{BE}{BD} = \frac{EH}{HD}.$$

By now I hope this part of the note has conveyed an appreciation for coordinate-free methods in our mathematical toolbox. I also hope it gives a preview of what modern physics use to solve geometry problems that are more than 3-dimensional, admittedly in a weak sense.

So far in this part we have discussed the simple case of how the reflection property implies the ratio equality. Below we show why the original problem in Part 1 gives the reflection property of EF and BE, using a vector argument.

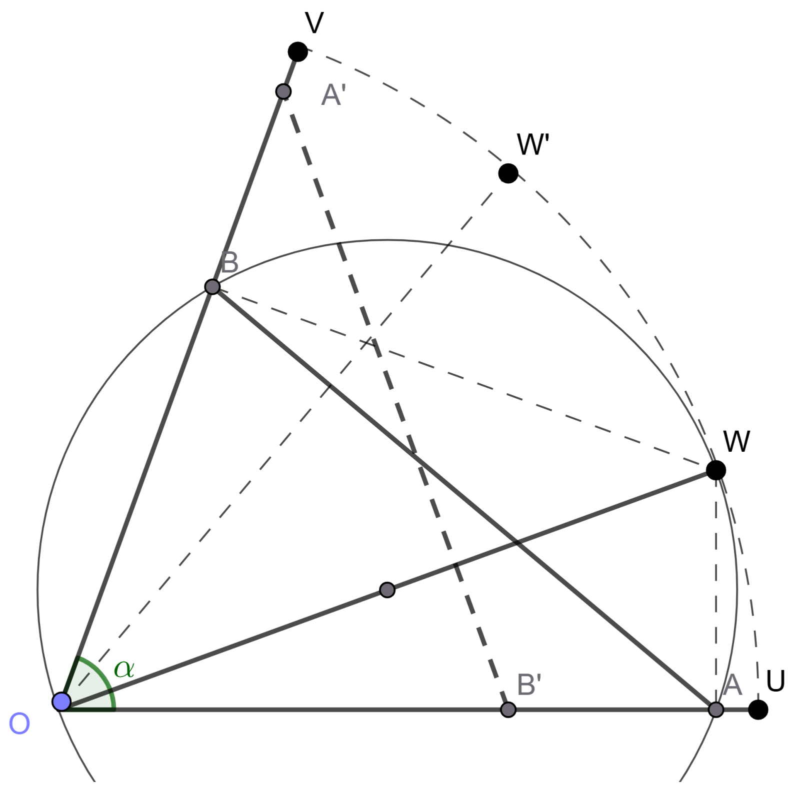
Proof 4.4 Vector analysis: EF is the reflection of BE across CD in Figure 3

> Let $\mathbf{u} = \overrightarrow{CB}$ and $\mathbf{d} = \overrightarrow{CD}$. Then $\mathbf{u} \perp \mathbf{d}$ and $\|\mathbf{u}\| = \|\mathbf{d}\|$. Fix the unit normal $\mathbf{w} = \dfrac{\mathbf{u} \times \mathbf{d}}{\|\mathbf{u} \times \mathbf{d}\|}$. Since $AD \parallel CB$, $E \in CD$, and $DE = AD$, there exists a scalar k such that
>
> $$\overrightarrow{AB} = -\mathbf{d} + (1-k)\mathbf{u}, \qquad \overrightarrow{BE} = -(k-1)\mathbf{d} - \mathbf{u}.$$
>
> Reflection across CD is equivalent to
>
> $$\frac{\|\overrightarrow{BE} \times \mathbf{d}\|}{\|\overrightarrow{BE}\|} = \frac{\|\overrightarrow{EF} \times \mathbf{d}\|}{\|\overrightarrow{EF}\|}.$$
>
> Because $EF \perp AB$, the direction of EF is parallel to $\overrightarrow{AB} \times \mathbf{w}$, so the right-hand side equals
>
> $$\frac{\|(\overrightarrow{AB} \times \mathbf{w}) \times \mathbf{d}\|}{\|\overrightarrow{AB} \times \mathbf{w}\|}.$$
>
> Now compute the two normalized quantities. First,
>
> $$\overrightarrow{BE} \times \mathbf{d} = (-(k-1)\mathbf{d} - \mathbf{u}) \times \mathbf{d} = -\mathbf{u} \times \mathbf{d},$$
>
> so $\|\overrightarrow{BE} \times \mathbf{d}\| = \|\mathbf{u} \times \mathbf{d}\| = \|\mathbf{u}\|\,\|\mathbf{d}\| = \|\mathbf{d}\|^2$. Next, by Lagrange's formula $(\mathbf{a} \times \mathbf{b}) \times \mathbf{c} = \mathbf{b}(\mathbf{a} \cdot \mathbf{c}) - \mathbf{a}(\mathbf{b} \cdot \mathbf{c})$ and $\mathbf{w} \cdot \mathbf{d} = 0$,
>
> $$(\overrightarrow{AB} \times \mathbf{w}) \times \mathbf{d} = \mathbf{w}(\overrightarrow{AB} \cdot \mathbf{d}).$$
>
> But
>
> $$\overrightarrow{AB} \cdot \mathbf{d} = (-\mathbf{d} + (1-k)\mathbf{u}) \cdot \mathbf{d} = -\|\mathbf{d}\|^2,$$
>
> hence $\|(\overrightarrow{AB} \times \mathbf{w}) \times \mathbf{d}\| = \|\mathbf{d}\|^2$. Finally, $\|\overrightarrow{AB} \times \mathbf{w}\| = \|\overrightarrow{AB}\|$, and
>
> $$\|\overrightarrow{AB}\|^2 = \|\mathbf{d}\|^2 + (k-1)^2\|\mathbf{u}\|^2 = (k-1)^2\|\mathbf{d}\|^2 + \|\mathbf{u}\|^2 = \|\overrightarrow{BE}\|^2,$$
>
> so $\|\overrightarrow{AB}\| = \|\overrightarrow{BE}\|$. Therefore the normalized cross magnitudes match, and the reflection condition holds:
>
> $$\frac{\|\overrightarrow{BE} \times \overrightarrow{CD}\|}{\|\overrightarrow{BE}\|} = \frac{\|\overrightarrow{EF} \times \overrightarrow{CD}\|}{\|\overrightarrow{EF}\|}.$$

> So BE reflects across CD into EF.

We used Lagrange's formula (the vector triple product identity) in the proof above. There are many known proofs, some classic and purely algebraic, but this one that I came up with is purely geometric and makes the identity feel almost unavoidable. I haven't seen this particular "Thales-circle + chord" intuition presented in such a compact form elsewhere, so it seems worth sharing though it can hardly be original, given that this is elementary. In a sense, it brings the argument back to a Euclid-style picture: analysis and geometry are not rivals, but two languages for the same structure, constantly translating into each other.

Proof of Lagrange's formula based on only geometric properties

> Let O be the origin and $\overrightarrow{OU} = \mathbf{u}$, $\overrightarrow{OV} = \mathbf{v}$, $\overrightarrow{OW} = \mathbf{w}$. Write $\mathbf{w} = \mathbf{w}_{\parallel} + \mathbf{w}_{\perp}$ relative to the plane (OU, OV). Then $(\mathbf{u} \times \mathbf{v}) \times \mathbf{w}_{\perp} = 0$ (since $\mathbf{u} \times \mathbf{v} \parallel \mathbf{w}_{\perp}$) and $\mathbf{u} \cdot \mathbf{w}_{\perp} = \mathbf{v} \cdot \mathbf{w}_{\perp} = 0$, so it suffices to assume U, V, W are coplanar. By linearity of both $\times$ and $\cdot$ in each argument, scaling $\mathbf{u}, \mathbf{v}, \mathbf{w}$ by a, b, c multiplies both sides by abc; hence it suffices to prove the unit case $|\mathbf{u}| = |\mathbf{v}| = |\mathbf{w}| = 1$. Let $\alpha = \angle UOV$.
>
> 
>
>
> *Figure 11: Geometric property of the triple vector formula.*
>
> For the plane geometry, refer to Figure 11. Draw the circle with diameter OW. Let it meet rays OU, OV again at A, B. By Thales, $\angle OAW = \angle OBW = 90^\circ$, hence $OA = \mathbf{u} \cdot \mathbf{w}$ and $OB = \mathbf{v} \cdot \mathbf{w}$. Define $A' \in OV$ and $B' \in OU$ by $OA' = OA$ and $OB' = OB$. Then

> $$\overrightarrow{A'B'} = (\mathbf{v}\cdot\mathbf{w})\mathbf{u} - (\mathbf{u}\cdot\mathbf{w})\mathbf{v} = -\big((\mathbf{u}\cdot\mathbf{w})\mathbf{v} - (\mathbf{v}\cdot\mathbf{w})\mathbf{u}\big),$$
>
> so $|A'B'|$ is the magnitude of the right-hand side. It is easy to show $A'B' \perp OW$ by checking $\overrightarrow{A'B'}\cdot\mathbf{w} = 0$, but here we want a pure geometric proof. Since $\angle WOA' + \angle A'B'O = 90^\circ$, we get $A'B' \perp OW$, matching the left-hand side direction. Since $\triangle OAB \cong \triangle OA'B'$, we have $AB = A'B'$. In the diameter-1 circle, $AB = \sin\alpha$. But
>
> $$|(\mathbf{u}\times\mathbf{v})\times\mathbf{w}| = |\mathbf{u}\times\mathbf{v}|\,|\mathbf{w}| = \sin\alpha,$$
>
> so both sides are the same vector:
>
> $$(\mathbf{u}\times\mathbf{v})\times\mathbf{w} = (\mathbf{u}\cdot\mathbf{w})\mathbf{v} - (\mathbf{v}\cdot\mathbf{w})\mathbf{u}.$$
>
> In short, we used the multilinearity of the dot and cross products (it is enough to prove the identity for coplanar unit vectors). We also used that dot and cross magnitudes correspond to cos and sin of the angle. The rest relies on a circle fact: two chords have the same length if and only if their inscribed angles are equal.

Postscript

It can be interesting to see how the following software would approach the problem:

1. Medal-level AI geometry solvers (closest to "gold/silver" performance in the International Math Olympiad)

- AlphaGeometry / AlphaGeometry 2 (research systems that can solve many Olympiad problems)
- Open-source software such as Newclid/Yuclid by the startup company Harmonic

2. Geometry theorem provers (strong automation, not always Olympiad-style proofs)

- GCLC / WinGCLC (area method, Wu's method, Gr?bner bases)
- JGEX and SymPy (Wu/Grobner/full-angle methods with visualization)
- OpenGeoProver (Wu/Gr?bner style)
- GeoLogic (interactive Euclidean prover)

3. Proof assistants (formal, but manual)

- Lean 4 + mathlib
- LeanGeo

References

- [CROSS-PRODUCT] Gibbs/Heaviside (1881). "Cross product." Summary: Notation for dot/cross products introduced in 1881. URL: https://en.wikipedia.org/wiki/Cross_product

- [EPSTEIN-DG] Epstein (2013~2014). "Differential Geometry Minicourse." Summary: Minicourse notes associated with a 2013 workshop; book version appeared in 2014. URL: https://marceloepstein.weebly.com/uploads/1/2/8/4/12849112/dg_minicourse.pdf (context: https://www.isbn.de/buch/9783319357140/differential-geometry)
- [EUCLID-VIDEO] Syversen (2025-09-18). "Euclid's ruler-and-compass constructions." YouTube. Summary: Video on the role of straightedge-and-compass constructions in Euclid. URL: https://youtu.be/M-MgQC6z3VU
- [FORMS-EM] Deschamps (1981). "Electromagnetics and Differential Forms." Summary: Proc. IEEE article that sparked broader EM use of differential forms. URL: https://faculty.washington.edu/seattle/physics544/2011-lectures/deschamps.pdf
- [GA-EM] Joot (2018). "Geometric Algebra for Electrical Engineers: Multivector Electromagnetism." Summary: Notes introducing geometric algebra for electromagnetism (version dated April 2018). URL: https://peeterjoot.com/archives/math2018/GAelectrodynamics.V0.1.6.pdf
- [KEPLER-TEACH] van Haandel & Heckman (2007). "Teaching the Kepler laws for freshmen." arXiv:0707.4605. Summary: Pedagogical treatment of Kepler's laws with geometric insights. URL: https://arxiv.org/abs/0707.4605
- [LINDELL-DF] Lindell (2004). "Differential Forms in Electromagnetics." Summary: Lecture notes/book with a 2004 copyright. URL: https://faculty.washington.edu/seattle/physics544/2011-lectures/lindell.pdf
- [MAXWELL-1865] Maxwell (1865). "A Dynamical Theory of the Electromagnetic Field." Summary: Original formulation of Maxwell's equations in component-heavy form. URL: https://archive.org/details/electromagnetic00maxwgoog
- [NEWTON-PRINCIPIA] Newton (1687). *Philosophiae Naturalis Principia Mathematica.* Project Gutenberg. Summary: Foundational work in classical mechanics with Euclidean-style geometric reasoning. URL: https://www.gutenberg.org/ebooks/28233
- [QUATERNION] Hamilton (1843). "Quaternion." Summary: Quaternions were introduced by Hamilton in 1843. URL: https://en.wikipedia.org/wiki/Quaternion
- [VECTOR-ANALYSIS] Wilson (1901). *Vector Analysis* (from Gibbs' lectures). Summary: Classic text that standardizes dot/cross notation and vector calculus. URL: https://archive.org/details/vectoranalysis00wilsrich
- [VECTOR-ANALYSIS-WIKI] Wilson/Gibbs (1901). "Vector Analysis." Summary: Wikipedia entry on the 1901 textbook and its role in standardizing notation. URL: https://en.wikipedia.org/wiki/Vector_Analysis

Back to TOC

Part 5: Groebner (Grobner) basis and Wu Method: polynomial elimination methods

So far, in Parts 1-4, we have followed a mostly pure-math storyline: from Euclid's synthetic geometry, through analytic and algebraic viewpoints, all the way to the modern language of vectors, forms, and (eventually) geometric algebra - ideas that were already in very solid shape by the early 20th century. A natural next question is: what happened in roughly the last 50 years for solving Euclidean geometry problems? The next big step is algorithmic: turn geometry into a procedure. Translate points, lines, circles, and incidences into variables, and translate geometric constraints into polynomial equations. Once everything becomes polynomial, a proof turns into a computable elimination problem. In this part we introduce two seminal algebraic approaches to automated Euclidean theorem proving: Wu's method and the Groebner basis method.

Wu's method (Wen-Tsun / Wenjun Wu, late 1970s). Wu's method - often presented through characteristic sets (triangular-style elimination) - was developed by Wen-Tsun Wu in the late 1970s and became one of the first highly successful, systematic methods for mechanical theorem proving in elementary geometry [WU-ORIG] [WU-CHAR]. The key move is explicitly analytic: choose coordinates for "free" points, convert each geometric relation (collinearity, perpendicularity, equal lengths, parallelism, cyclicity, etc.) into polynomial constraints, then eliminate variables in a structured way until the target statement reduces to an algebraic consequence of the hypotheses [WU-CHAR]. Conceptually it's: geometry -> polynomials -> elimination, with a procedure that a computer can execute step-by-step.

Groebner basis method (Buchberger 1965; geometry automation later). Groebner bases were introduced by Bruno Buchberger (1965), together with the Buchberger algorithm, giving a general-purpose computational engine for polynomial ideal problems [GB-HIST] [GB-THESIS]. In geometry proving, the translation is clean: encode the hypotheses as an ideal I generated by polynomials, encode the desired conclusion as a polynomial g, and prove the theorem by checking whether g is in I (or in a saturation/elimination variant to handle degeneracies). Groebner bases make this membership test algorithmic via systematic reductions and elimination behavior [GB-HIST].

Same destination, different roads (and "coordinate dependence"). Both methods share the same punchline: a Euclidean proof can be reduced to algebraic elimination, once you choose coordinates and translate relations into polynomials [WU-CHAR] [GB-HIST]. This looks coordinate-dependent in execution because we literally pick a coordinate system, but the truth is coordinate-free: any non-degenerate coordinate choice represents the same geometry. Wu emphasized this as a workable philosophy of mechanization - geometry carries enough intrinsic structure that, after coordinatization, proof becomes a deterministic computation [WU-SURVEY]. That's the deeper shift of the last decades: not just knowing geometry, but compiling geometry into a procedure.

The rest of this part shows two classic algebraic geometry-prover styles on the same configuration:

1. Groebner basis method: prove the target polynomial lies in the ideal generated by hypothesis polynomials.
2. Wu's method style (characteristic-set / triangular elimination): eliminate variables using a triangular chain (here it becomes simple linear elimination).
3. Wu's method following the algorithm more faithfully to show the difference from (2).

We work entirely in polynomials (no square-roots) by proving the squared identity

$$(BE^2)(DH^2) - (BD^2)(EH^2) = 0,$$

which is equivalent to $BE \cdot DH = BD \cdot EH$ for positive lengths.

Notes on representation

- SymPy is used here for convenience (symbolic polynomials + Groebner/Wu-style elimination). It is not required in principle.
- Matrices are used only as a compact way to write determinants/dot products; this is a modeling choice, not a requirement of Wu's method.
- In a later program the geometry is built with `Point` objects in SymPy instead of matrices, to show a more geometric style; both approaches are equivalent.

5.1 Coordinate encoding of the geometry (bake-in the constraints)

Given:

- $AD \parallel BC$
- $\angle C = 90^\circ$
- $BC = CD$
- $E \in CD$, and $DE = AD$
- $EF \perp AB$
- $H = BD \cap EF$

We can choose coordinates (scale is irrelevant):

$$C = (0,0), \quad B = (1,0), \quad D = (0,1)$$

so $BC \perp CD$ and $BC = CD = 1$.

Since $AD \parallel BC$, line AD is horizontal, so:

$$A = (a, 1).$$

Point $E \in CD$ means $E = (0, e)$. Condition $DE = AD$ gives:

$$DE = 1 - e, \quad AD = a \quad \Rightarrow \quad 1 - e = a \Rightarrow e = 1 - a.$$

So:

$$E = (0, 1 - a).$$

Now let $H = (h_x, h_y)$ be the intersection point we will solve from the constraints:

- B, D, H are collinear (because $H \in BD$)
- $(H - E) \perp (B - A)$ (because $H \in EF$ and $EF \perp AB$)

In [2]:
```
import sympy as sp
sp.init_printing()

def dist2(P, Q):
    v = P - Q
    return sp.expand(v.dot(v))

def sec(title):
    print("\n" + "="*70)
    print(title)
    print("="*70)

a, hx, hy = sp.symbols("a hx hy")

# Fixed points
C = sp.Matrix([0, 0])
B = sp.Matrix([1, 0])
D = sp.Matrix([0, 1])

# Variable point A, and derived point E
A = sp.Matrix([a, 1])
E = sp.Matrix([0, 1 - a])

# Variable point H
H = sp.Matrix([hx, hy])
```

Now we translate hypotheses into polynomials

Collinearity B, D, H

Vectors $\overrightarrow{DB} = B - D$ and $\overrightarrow{DH} = H - D$ must be linearly dependent, so the 2×2 determinant is zero:

$$\det \begin{pmatrix} B_x - D_x & B_y - D_y \\ H_x - D_x & H_y - D_y \end{pmatrix} = 0.$$

Perpendicularity $(H - E) \perp (B - A)$

Dot product is zero:

$$(H - E) \cdot (B - A) = 0.$$

These give two polynomials $f_1 = 0$, $f_2 = 0$.

In [3]:

```
sec("Hypothesis polynomials")

# (1) Collinear B, D, H
f1 = sp.Matrix([[B[0]-D[0], B[1]-D[1]],
                [H[0]-D[0], H[1]-D[1]]]).det()
f1 = sp.expand(f1)

# (2) (H-E) ⊥ (B-A)
f2 = sp.expand((H - E).dot(B - A))

print("f1 (collinear B,D,H) =", f1)
print("f2 ((H-E) ⊥ (B-A))  =", f2)

# For reference, simplify them:
print("\nSimplified forms:")
print("f1 =", sp.factor(f1))
print("f2 =", sp.factor(f2))
```

```
======================================================================
Hypothesis polynomials
======================================================================
f1 (collinear B,D,H) = hx + hy - 1
f2 ((H-E) ⊥ (B-A))  = -a*hx - a + hx - hy + 1

Simplified forms:
f1 = hx + hy - 1
f2 = -a*hx - a + hx - hy + 1
```

Target polynomial is the following identity to prove:

$$p := (BE^2)(DH^2) - (BD^2)(EH^2) = 0.$$

This is equivalent to $BE \cdot DH = BD \cdot EH$ because lengths are nonnegative.

In [4]:

```
sec("Target polynomial p")

BE2 = dist2(B, E)
BD2 = dist2(B, D)
DH2 = dist2(D, H)
EH2 = dist2(E, H)

p = sp.expand(BE2*DH2 - BD2*EH2)

print("BE^2 =", sp.factor(BE2))
print("BD^2 =", sp.factor(BD2))
print("DH^2 =", sp.factor(DH2))
print("EH^2 =", sp.factor(EH2))
print("\np = (BE^2)(DH^2) - (BD^2)(EH^2) =")
print(sp.sstr(sp.factor(p)))
```

```
======================================================================
Target polynomial p
======================================================================
BE^2 = a**2 - 2*a + 2
BD^2 = 2
DH^2 = hx**2 + hy**2 - 2*hy + 1
EH^2 = a**2 + 2*a*hy - 2*a + hx**2 + hy**2 - 2*hy + 1

p = (BE^2)(DH^2) - (BD^2)(EH^2) =
a*(a*hx**2 + a*hy**2 - 2*a*hy - a - 2*hx**2 - 2*hy**2 + 2)
```

5.2 Gröbner basis proof

> We compute a Gröbner basis G for the ideal $\langle f_1, f_2 \rangle$.
> Then reduce p modulo G. If the remainder is 0, then
>
> $$p \in \langle f_1, f_2 \rangle$$
>
> so the hypotheses imply the conclusion.

In [ ]:

```
sec("| Groebner basis reduction")

G = sp.groebner([f1, f2], hx, hy, a, order="lex")

print("| Groebner basis polynomials:")
for g in G.polys:
    print("|   ", g.as_expr())

rem = G.reduce(p)[1]
print("\n| Remainder of p modulo Groebner basis =")
sp.pprint('| ' + sp.sstr(sp.factor(rem)))

assert rem == 0
print("\n| ✅ Groebner proof complete: remainder = 0")
```

```
======================================================================
| Groebner basis reduction
======================================================================
| Groebner basis polynomials:
|    hx + hy - 1
|    a*hy - 2*a - 2*hy + 2

| Remainder of p modulo Groebner basis =
| 0

| ✅ Groebner proof complete: remainder = 0
```

5.3 Wu’s method style proof (triangular elimination)

> Wu’s method builds a triangular (ascending) chain and repeatedly takes pseudo-remainders to eliminate variables. Here, our constraints are already linear, so the “characteristic set” is essentially:

> - from collinearity: $f_1 = h_x + h_y - 1 = 0$
> - from perpendicularity: $f_2 = -ah_x - a + h_x - h_y + 1 = 0$ So elimination becomes:
>
> 1. From $f_1 = 0$:
>
> $$h_y = 1 - h_x.$$
>
> 2. Substitute into $f_2 = 0$ to solve h_x.
> 3. Substitute (h_x, h_y) into p. If it becomes 0, the theorem is proved. This is exactly the *same spirit* as Wu: reduce the goal with the chain until remainder is 0. See the code below

In [10]:
```
sec("Wu-style triangular elimination")

# Step 1: solve f1 for hy
hy_expr = sp.solve(sp.Eq(f1, 0), hy)[0]
print("From f1=0, hy =", hy_expr)

# Step 2: substitute into f2 and solve for hx
f2_sub = sp.simplify(f2.subs(hy, hy_expr))
print("\nSubstitute hy into f2 ->")
sp.pprint(sp.factor(f2_sub))

hx_expr = sp.solve(sp.Eq(f2_sub, 0), hx)[0]
print("\nSolve for hx -> hx =", hx_expr)

# Then hy
hy_expr2 = sp.simplify(hy_expr.subs(hx, hx_expr))
print("And hy =", hy_expr2)

# Step 3: substitute into p
p_sub = sp.simplify(p.subs({hx: hx_expr, hy: hy_expr2}))
print("\nSubstitute (hx,hy) into p ->")
sp.pprint(sp.factor(p_sub))

assert sp.factor(p_sub) == 0
print("\n✅ Wu-style elimination complete: p reduces to 0")
```

```
======================================================================
Wu-style triangular elimination
======================================================================
From f1=0, hy = 1 - hx

Substitute hy into f2 ->
-a·hx - a + 2·hx

Solve for hx -> hx = -a/(a - 2)
And hy = 2*(a - 1)/(a - 2)

Substitute (hx,hy) into p ->
0

✅ Wu-style elimination complete: p reduces to 0
```

5.4 Extra: numeric sanity check

Since we are in python, we can pick any $a \in (0, 1)$. (For the original picture you typically have $0 < a < 1$.)

We compute the actual numeric lengths and verify $BE \cdot DH = BD \cdot EH$.

In [6]:
```
sec("Numeric sanity check")

aval = sp.Rational(2, 5)   # a = 0.4
subs_num = {a: aval, hx: hx_expr.subs(a, aval), hy: hy_expr2.subs(a, aval)}

# Evaluate squared lengths
BE2_num = sp.N(BE2.subs(subs_num))
BD2_num = sp.N(BD2.subs(subs_num))
DH2_num = sp.N(DH2.subs(subs_num))
EH2_num = sp.N(EH2.subs(subs_num))

BE = float(sp.sqrt(BE2_num))
BD = float(sp.sqrt(BD2_num))
DH = float(sp.sqrt(DH2_num))
EH = float(sp.sqrt(EH2_num))

print("a =", float(aval))
print("H =", (float(subs_num[hx]), float(subs_num[hy])))
print("BE*DH =", BE*DH)
print("BD*EH =", BD*EH)
print("difference =", (BE*DH) - (BD*EH))
```

```
======================================================================
Numeric sanity check
======================================================================
a = 0.4
H = (0.25, 0.75)
BE*DH = 0.412310562561766
BD*EH = 0.412310562561766
difference = 0.0
```

This is Wu-flavored code, but it does not implement the full, original Wu characteristic-set algorithm.

In the real Wu method, you fix a variable order (a precedence like $h_y \succ h_x \succ \cdots$), then build a triangular chain (a characteristic set).

You do not "randomly solve" for a variable when it looks convenient. You pick eliminations to make the next polynomial's main variable well-behaved and the chain triangular.

Most importantly: Wu's method avoids fractions by using pseudo-division (a.k.a. pseudo-remainder). This guarantees you stay inside the polynomial ring: no denominators ever appear. Internally, pseudo-division multiplies by powers of the leading coefficient, so the working set always contains pure polynomials.

So we call it "Wu-style" elimination here: same spirit (triangular elimination), but the first version uses solve (which may introduce denominators), while the second version shows the truly programmable "no-denominator" elimination that resembles Wu's mechanics. Below we show how the problem would be solved if we followed Wu's algorithm more faithfully.

Fully programmable "no-denominator" Wu-like elimination (pseudo-remainder)

This is the key upgrade: we eliminate variables without ever dividing.

Core idea (one line)

To eliminate h_y, we replace "substitute $h_y = -b/a$" by pseudo-remainder:

$$R_2 = \mathrm{prem}(f_2, f_1; h_y),$$

which equals $a^k f_2(h_y = -b/a)$ but with denominators cleared, hence still a polynomial.

What this corresponds to mathematically

We built a triangular chain:

$$T_1 = f_1 \quad (\text{main variable } h_y),$$

$$T_2 = \mathrm{prem}(f_2, T_1; h_y) \quad (\text{main variable } h_x).$$

Then we reduced the goal:

$$p \xrightarrow{\mathrm{prem}(\cdot, T_1; h_y)} R_p \xrightarrow{\mathrm{prem}(\cdot, T_2; h_x)} 0.$$

That is exactly the "eliminate h_y, then eliminate h_x, then the goal collapses" story - but now in a Wu-programmable way.

One important Wu caveat (geometry degeneracy)

Pseudo-division implicitly assumes the leading coefficients used during elimination are not zero (otherwise the "main variable" disappears and you're in a degenerate case). In geometry, this corresponds to "generic position" assumptions (no coincident points, no accidental parallel collapse, etc.).

In [11]:

```
sec("Wu-style triangular elimination (pseudo-remainder / no-denominator)")

from sympy.polys.polytools import prem

# --- Step 1: eliminate hy using pseudo-remainder ---
T1 = sp.factor(f1)                          # main variable: hy
R2 = sp.factor(prem(f2, T1, hy))            # eliminates hy, stays polynomial

print("T1 (main var hy) =")
sp.pprint(T1)
print("\nR2 = prem(f2, T1, hy)  (hy eliminated, polynomial) =")
sp.pprint(R2)
```

```python
# --- Step 2: eliminate hx using pseudo-remainder ---
T2 = sp.factor(R2)                          # main variable: hx (ideally)
Rp = sp.factor(prem(p, T1, hy))             # first reduce goal by T1 in hy
Rp2 = sp.factor(prem(Rp, T2, hx))           # then reduce by T2 in hx

print("\nT2 (after eliminating hy; main var should be hx) =")
sp.pprint(T2)
print("\nGoal reduction: Rp2 = prem(prem(p,T1,hy), T2, hx) =")
sp.pprint(Rp2)

assert sp.factor(Rp2) == 0
print("\n✅ Wu-style pseudo-remainder elimination: goal reduces to 0 (no denominators
```

```
======================================================================
Wu-style triangular elimination (pseudo-remainder / no-denominator)
======================================================================
T1 (main var hy) =
hx + hy - 1

R2 = prem(f2, T1, hy)  (hy eliminated, polynomial) =
-a⋅hx - a + 2⋅hx

T2 (after eliminating hy; main var should be hx) =
-a⋅hx - a + 2⋅hx

Goal reduction: Rp2 = prem(prem(p,T1,hy), T2, hx) =
0

✅ Wu-style pseudo-remainder elimination: goal reduces to 0 (no denominators ever use
d)
```

References

- [ATP-GEO] Chou, Gao, Zhang (2001). "Machine Proofs in Geometry." Summary: Monograph on automated geometry theorem proving. URL: http://www.mmrc.iss.ac.cn/~xgao/paper/book-area.pdf
- [CAS-GEN] "Computer algebra systems: Mathematica, Maple." Summary: General-purpose CAS background for symbolic computation. URLs: https://en.wikipedia.org/wiki/Wolfram_Mathematica ; https://en.wikipedia.org/wiki/Maple_(software)
- [GB-HIST] Buchberger (2005). "An Introduction to Groebner Bases." Summary: Historical and conceptual introduction to Grobner bases. URL: https://www3.risc.jku.at/publications/download/risc_3045/2005-07-09-Historic-Intro-to-GB.pdf
- [GB-THESIS] Buchberger (1965/2005 translation). "Bruno Buchberger's PhD thesis (Grobner bases)." Summary: Original Grobner basis algorithm and theory; later English translation with commentary. URL: https://www.sciencedirect.com/science/article/pii/S0747717105001483

- [WU-CHAR] Wu (late 1970s). "Wu's method of characteristic set." Summary: Overview of characteristic-set elimination developed in the late 1970s. URL: https://en.wikipedia.org/wiki/Wu%27s_method_of_characteristic_set
- [WU-ORIG] Wu (late 1970s). "Wu's method discovery timeline." Summary: Historical timeline for the development of Wu's method. URL: http://www.mmrc.iss.ac.cn/~lzhi/Research/wuritt.html
- [WU-SURVEY] Wu et al. (2007). "Mathematics mechanization" survey. Summary: Survey of algebraic methods and mechanization in geometry proving. URL: http://www.mmrc.iss.ac.cn/~xgao/paper/07-mmsurvey.pdf

Back to TOC

Part 6. Modern Geometric Prover

Wu's method and Gröbner-basis proving are "old-school algorithmic" in the best sense: once you translate a diagram into polynomial constraints, the prover can grind deterministically and still surprise you with what it manages to derive. That feels creative because the algorithm is not following a human narrative, yet it discovers nontrivial consequences reliably and at scale. The downside is exactly what you observed: these proofs often become long coordinate eliminations that certify truth, but don't illuminate the geometry. In fact, modern surveys often summarize algebraic methods this way: extremely powerful decision procedures, but the output is typically not a readable, traditional geometry proof—more a yes/no outcome with an algebraic justification. [SURVEY]

A key idea that matured over the last two decades is to keep the computation, but move the "language of thought" closer to geometry itself. Semi-synthetic methods like the area method and the full-angle method do this by replacing raw coordinates with geometric invariants (directed areas, oriented angles, angle expressions) and carefully designed elimination rules. The area method is explicitly praised for producing proofs that are concise and human-readable while still being automated. [AREA] The full-angle line of work went further by building rule systems around angle expressions in a way that can be structured hierarchically, which even enables visually dynamic, step-by-step presentations of proofs—much closer to how a person reasons with a diagram. [VDP]

This "human-facing" direction also pushed geometry provers into interactive and educational software. A milestone was the incorporation and testing of multiple automated geometry provers inside GeoGebra, motivated by the idea that automated deduction had become mature enough to change how people learn and explore geometry. [GGB] Around the same time, community efforts like OpenGeoProver worked on unifying several proof styles (algebraic, semi-synthetic, synthetic) behind a common tool interface, so the user sees geometry objects while the backend can choose the best proving engine.

Then the last two years added a new ingredient that directly targets the "magic of auxiliary constructions": systems that can propose new points/lines/circles in a human-like way. AlphaGeometry (Nature 2024) made this explicit: a neural language model guides a symbolic deduction engine through the huge branching factor of difficult Olympiad problems, and it reaches near gold-medalist performance on a benchmark of 30 problems. [AG1] DeepMind's own description highlights the core leap: the model predicts useful new constructs from an effectively infinite space, while the symbolic engine validates the consequences. [AG1-BLOG] AlphaGeometry2 pushes this further by enriching the language and emphasizing auxiliary constructions even more strongly, and it is built on a more powerful Gemini-based model. [AG2]

What’s also interesting is that “auxiliary creativity” is no longer owned by neural models alone. HAGeo shows you can get gold-medal level performance by engineering strong heuristic auxiliary construction strategies wrapped around a symbolic deduction engine, running fully on CPUs without neural inference. [HAG] GenesisGeo attacks the bottleneck from the other side: it open-sources a massive synthetic dataset (including millions of cases with auxiliaries) and accelerates the symbolic engine with C++ components, while also adding a neuro-symbolic prover based on Qwen3-0.6B-Base. [GEN] These results collectively say something profound: the “human-like” part of geometry solving is largely about proposing the right extra objects, and we now have multiple competing ways to do that—LLMs, heuristics, and hybrid search.

Newclid is a nice example of how modern systems separate “frontend geometry” from “backend proof.” On the frontend, you describe the construction and goals (JGEX / GeoGebra import); in the backend, a symbolic engine (DDARN) runs deduction and proof search. The important architectural twist is that Newclid exposes an agent interface between them, so the choice of auxiliary construction strategy can be swapped: an LLM agent, a heuristic agent, a BFS-style agent, or even a human-in-the-loop agent. The Newclid paper explicitly frames its CLI as a way to run problems without a Python entry point and even to replace the original LLM with human decisions, while still using the same symbolic core. [NC]

Formal verification languages (Lean/Coq) matter because they make “correctness” non-negotiable. Once you have systems that generate highly non-obvious auxiliaries, you urgently need a small trusted kernel that can check every step. The area method already had a deep formalization path: Narboux implemented it as a decision procedure inside Coq, explicitly valuing short and readable proofs with machine-checkable elimination lemmas. [COQ-AM] LeanGeo extends this spirit to competition-level geometry inside Lean 4, aiming for a unified, rigorously verified framework integrated with Mathlib and released with an open-source library and benchmark. [LG] The big point is: “creative search” can be messy, probabilistic, and exploratory—but the final proof artifact can still be as clean and trustworthy as a textbook, because the checker is unforgiving.

Another piece of the “human-like” story is not just finding a proof, but presenting it in a way that looks like geometry again. Historically, this emerged as a reaction to coordinate/elimination proofs: Wu/Gröbner methods can certify truth, but the output often reads like algebraic elimination rather than geometric insight. Semi-synthetic provers helped because they operate in higher-level invariants (areas, oriented angles, angle expressions), which naturally map back into short Euclid-style steps that a student can follow. The full-angle method is a representative example: it was designed specifically so the proof object is a sequence of geometric statements that can be displayed and checked step by step, rather than a large polynomial certificate [FA].

Once the reasoning steps become “geometric sentences,” visualization becomes possible and valuable. A major milestone was the line of work on visually dynamic proof presentation, where the proof is organized hierarchically and the diagram is animated so that objects and relations appear exactly when the text uses them; this turns a proof from a static list of facts into a guided tour that matches human attention [VDP]. In parallel, automated provers began integrating into interactive geometry environments, so proving becomes part of exploration: you construct a figure, conjecture an invariant, and then ask the prover to justify it inside the same visual workspace, often with a human-readable proof trace rather than an opaque "true/false" result [GGB]. This visualization layer matters because it is where "machine correctness" becomes "human understanding."

Euclid-Omni fits into this contribution as a unification attempt that pushes the interface boundary outward. Compared to systems like Newclid, which cleanly separate a formal geometry frontend (JGEX/diagram constraints) from a symbolic backend (DDARN proof search) [NC], Euclid-Omni aims to connect more of the pipeline end-to-end: it combines formal symbolic proving with LLM/VLM components so that inputs can be more flexible (including natural language or diagram perception), while outputs can remain structured and human-readable as logical steps supported by a formal core [EO]. Conceptually, it tries to make the "frontend" not just a parser for JGEX, but also a translator from human modalities into formal predicates, and then back into a readable proof narrative—bridging the last gap between creative construction and explainable reasoning.

Furthermore, Euclid-Omni explicitly frames "open data" as one of the missing pieces in the current gold-medal pipeline ecosystem, and positions itself as a response. It points out that existing IMO-level systems required massive compute to generate training examples, yet did not release their data-generation pipelines or datasets, which blocks broader reproducibility and extension by the community [EO]. In contrast, EO's core contribution is not only a symbolic solver (Euclidea), but also a complete configurable data factory that synthesizes symbolic problems, renders diagrams, and produces aligned natural-language versions, thereby yielding large-scale datasets for training LLMs/VLMs; and it explicitly states "we will release the complete framework," which (by their own definition of the framework) includes the data generation pipeline and the datasets it produces [EO]

Proof 6.1: Newclid

Newclid was created by the startup company Harmonic, based on AlphaGeometry.

Harmonic built Newclid to make the AlphaGeometry-style approach usable outside a closed research stack: a tool that is easier to run, easier to extend, and easier to integrate into other workflows. The emphasis is on a clean interface between geometry input and the symbolic proof engine, so different construction or search strategies (LLM, heuristics, or human-in-the-loop) can be swapped in without changing the core prover. Newclid uses its own DDARN (Deductive Database with Algebraic Reasoning in Newclid) symbolic solver derived from the DDAR used in AlphaGeometry. In Newclid 3, published in late 2025, the system also included a C++-optimized execution engine of DDARN called Yuclid.

In [5]:
```
from pathlib import Path
from newclid import GeometricSolverBuilder
from newclid.problem import PredicateConstruction, ProblemSetup
from newclid.jgex.problem_builder import JGEXProblemBuilder
```

```python
from newclid.jgex.formulation import JGEXFormulation
import matplotlib.pyplot as plt

JGEX_PROBLEM = """C B = segment C B;
D = on_tline D C B C, on_circle D C B;
E = on_line E C D;
A = on_pline A D B C, on_circle A D E;
F = intersection_lt F A B E A B;
G = intersection_ll G B D A E;
H = intersection_ll H B D E F
""".strip().replace('\n', ' ')
goals = [PredicateConstruction.from_str("eqratio D H E H D B E B")]


# Build a JGEX formulation into a ProblemSetup
jgex_problem = JGEXFormulation.from_text(JGEX_PROBLEM)
problem_builder = JGEXProblemBuilder(problem=jgex_problem, rng=0)
problem_setup = problem_builder.build()
problem = ProblemSetup(name=problem_setup.name,
                        points=problem_setup.points,
                        assumptions=problem_setup.assumptions,
                        goals=tuple(goals),
                        )
# Build a solver from the ProblemSetup
solver = GeometricSolverBuilder().build(problem)

# Run deduction + algebraic reasoning
success = solver.run()
print("solved: ", success)

if success:
    out = solver.write_all_outputs(Path('out'), jgex_problem=problem_builder.jgex_prob
    plt.close('all')
    # proof = solver.proof()
```

```
solved:  True
```

```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
out\html\dependency_graph.html
```

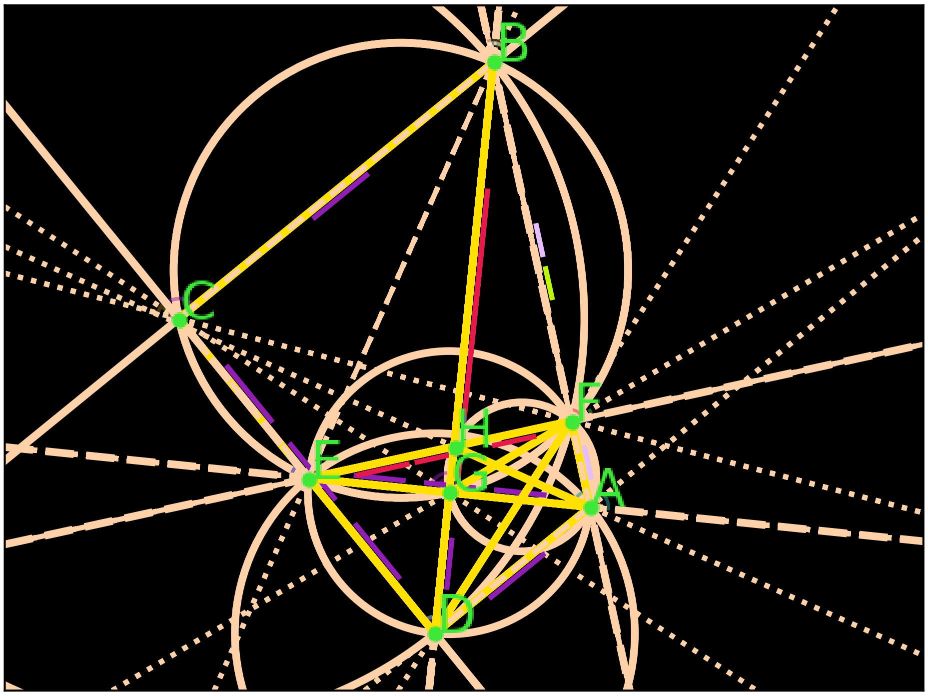


Figure 1: Proof figure generated by Newclid for Proof 6.1 (proof_figure.svg); even though less polished, it does show the geometries fine.

In [2]:
```
from IPython.display import IFrame, display
display(IFrame('out/html/dependency_graph.html', width='100%', height=520))
```

Figure 2: Proof dependency graph for the out run (dependency_graph.html). While the HTML version is interactive, the PDF version of this plot is static and misses the hover tooltips that show dependency details.

Summarize the solution from out/proof.text in human-readable steps using ChatGPT:

> Because B, C, E, F are concyclic, the angles subtending the same chords are equal; with A, B, F collinear and C, D, E collinear, this gives two matching angles between $\triangle ABE$ and $\triangle DCF$, so $\triangle ABE \sim \triangle DCF$, hence $\dfrac{AE}{BE} = \dfrac{DF}{CF}$. Because C, E, F, G are concyclic, the same "equal angles on equal arcs" rule gives two matching angles between $\triangle AFG$ and $\triangle DFC$, so $\triangle AFG \sim \triangle DFC$, hence $\dfrac{AF}{DF} = \dfrac{FG}{CF}$. Combining with the previous relation eliminates CF and yields $BE = \dfrac{AE \cdot FG}{AF}$. Next, since C, D, E are collinear and B, D, G are collinear, and the right-angle/parallels in the construction align the remaining angles, we get $\triangle BCD \sim \triangle EGD$, hence $\dfrac{BD}{CD} = \dfrac{DE}{DG}$. Together

> with the proportionalities already obtained from $\triangle AFG \sim \triangle DFC$, this rewrites BD in the same scale as BE, giving $BD = \frac{AD \cdot DF}{AF}$. Now bring in H. Because A, D, E, F are concyclic, equal angles in the same segment give $\triangle ADH \sim \triangle EFD$, so $DH = \frac{AD \cdot DF}{EF}$. Also, because A, F, G, H are concyclic, the same cyclic angle rule gives $\triangle AEH \sim \triangle EFG$, so $EH = \frac{AE \cdot FG}{EF}$. Finally,
>
> $$\frac{BD}{BE} = \frac{AD \cdot DF/AF}{AE \cdot FG/AF} = \frac{AD \cdot DF}{AE \cdot FG} = \frac{AD \cdot DF/EF}{AE \cdot FG/EF} = \frac{DH}{EH}$$
>
> so $BE \cdot DH = BD \cdot EH$.

This proof has 65 steps in its proof.text, about 500 lines. ChatGPT summarized it well, and from it we see this does not require auxiliary lines beyond a few trivial connectors between existing points. It is also more complex than some of the other methods listed in Part 1. In this sense, this method is innovative and reveals a few other similarities of triangles that are not too obvious.

6.2 Auxiliary lines

Newclid supports multiple ways of adding auxiliary lines. In the following section we explore it. First we need to have point G defined. Without that line, even though it appears irrelevant, the solver fails with the error message "ProofBuildError: Goal eqratio B D B E D H E H is numerically false.". This shows the solver first checks that a generated numeric case satisfies the goal. The reason this fails without G is because A and B can sit on the opposite side of CD now. In that case though, EA//BD so they would never intersect. Defining G eliminates an unexpected case allowed by the JGEX description. Such trap can be subtle. GeoGebra's plot support would help expose such constraints.

In [3]:
```
import matplotlib.pyplot as plt
JGEX_PROBLEM = """C B = segment C B;
D = on_tline D C B C, on_circle D C B;
D1 = on_tline D1 C B C, on_circle D1 C B;
E = on_line E C D;
A = on_pline A D B C, on_circle A D E;
F = intersection_lt F A B E A B;
G = intersection_ll G B D A E;
H = intersection_ll H B D E F;
""".strip().replace('\n', ' ')
goals = [PredicateConstruction.from_str("eqratio H D H E D B E B")]


jgex_problem = JGEXFormulation.from_text(JGEX_PROBLEM)
problem_builder = JGEXProblemBuilder(problem=jgex_problem, rng=0)
problem_setup = problem_builder.build()
problem = ProblemSetup(name=problem_setup.name,
```

```
                            points=problem_setup.points,
                            assumptions=problem_setup.assumptions,
                            goals=tuple(goals)
                            )
solver = GeometricSolverBuilder().build(problem)

success = solver.run()
print("solved: ", success)
if success:
    out = solver.write_all_outputs(Path('out1'), jgex_problem=problem_builder.jgex_pro
    plt.close('all')
    proof = solver.proof()
```

```
solved:  True
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
out1\html\dependency_graph.html
```

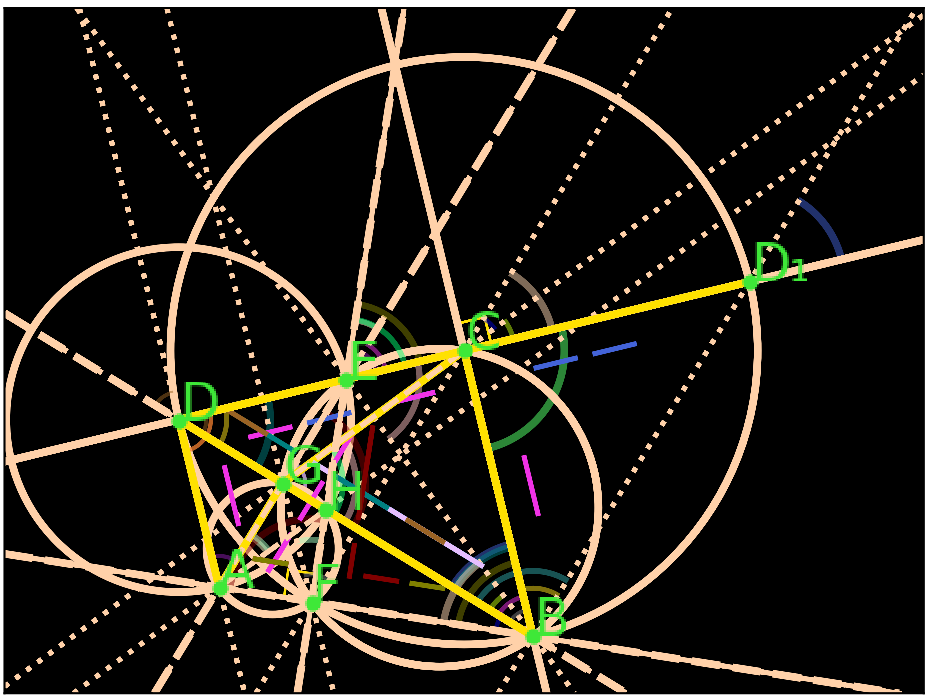


Figure 3: Proof figure generated by Newclid for Proof 6.2 (proof_figure.svg); note the auxiliary point D1 is added compared with Figure 1.

In [4]:
```
from IPython.display import IFrame, display
display(IFrame('out1/html/dependency_graph.html', width='100%', height=520))
```

Figure 4: Proof dependency graph for the out1 run (dependency_graph.html); this is noticeably simpler than Figure 2, and the auxiliary point D1 clearly simplifies the proof, which is also evident in the ChatGPT summary below. Again, the PDF version of this note loses the interactive functionality of the plots, which hides the meaning of each node.

Proof 6.2 can be summarized from out1/proof.txt by ChatGPT; it becomes significantly simpler than the previous proof.

> Introduce D_1 on the extension of DC such that $CD_1 = CD$. Then C is the midpoint of DD_1, and since $BC \perp DC$ we also have $BC \perp DD_1$. Hence BC is the perpendicular bisector of DD_1, so $BD = BD_1$. Newclid then proves two similarities: $\triangle BCG \sim \triangle D_1BE$ and $\triangle BCG \sim \triangle DHE$. From the first, $BD_1 : BE = BC : CG$; from the second, $DH : EH = BC : CG$. Therefore $BD_1 : BE = DH : EH$, and using $BD_1 = BD$ gives $BD : BE = DH : EH$, as desired.

Moreover, beyond Newclid's ratio-chasing, the logic can be shortened: we can show that $\triangle D_1BE \sim \triangle DHE$. That directly yields $BD_1 : BE = DH : EH$, and then $BD_1 = BD$

finishes the proof in one line. This shortened version is exactly what our proof 1 in part 1 did. So Newclid isn't necessarily putting great efforts to simplify the solution; instead it aims to grow the algebraic reasoning tree until it reaches the desired conclusion.

Newclid has a few built-in point-generation heuristics (see https://github.com/Newclid/Newclid/blob/main/notebooks/heuristics_implementation.ipynb). They are a systematic way to propose auxiliary constructions, which is overkill for this small problem, but the exercise of recreating Proof 1 still shows how the tool can help explore plane-geometry ideas. AlphaGeometry2 pushed this further by using LLM-guided proposals for new points, adding another layer of automation to the construction step.

As we have demonstrated on summarizing the complicated proof.text with hundreds of lines into a human-readable proof, a separate, complementary trend is using LLMs to translate formal or symbolic proofs into human-readable narratives. That translation layer matters for the human-machine interface; it helps students and researchers see the structure of an argument without wading through raw derivations. For more information, read the Euclid-Omni paper [EO].

Taken together with the previous parts of the note, these developments show how human insight and machine assistance are reshaping plane geometry: from the cleverness of mathematicians to machines that can search, verify, and suggest constructions, while humans interpret, teach, and guide the narrative. Similar shifts are happening across mathematics and science, which makes this a particularly exciting time for human researchers.

References

- [AG1] Trinh et al. (2024). "Solving olympiad geometry without human demonstrations." Nature. Summary: Introduces AlphaGeometry, combining a neural model with a symbolic deduction engine to solve Olympiad geometry at near gold-medal level. URL: https://www.nature.com/articles/s41586-023-06747-5
- [AG1-BLOG] Trinh & Luong (2024). "AlphaGeometry: An Olympiad-level AI system for geometry." DeepMind blog (Jan 17, 2024). Summary: Practitioner-oriented overview of AlphaGeometry's pipeline and results. URL: https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/
- [AG2] Chervonyi et al. (2025). "Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2." arXiv:2502.03544. Summary: Extends AlphaGeometry with a richer language and stronger auxiliary-construction capability. URL: https://dblp.org/rec/journals/corr/abs-2502-03544
- [AREA] Janicic, Narboux, Quaresma (2012). "The Area Method: a Recapitulation." Journal of Automated Reasoning. Summary: Surveys the area method and argues for concise, human-readable automated proofs. URL: https://research.matf.bg.ac.rs/handle/123456789/510
- [COQ-AM] Narboux (2009). "Formalization of the Area Method in the Coq proof assistant." Summary: Coq formalization of the area method (user contribution dated 2009). URL:

https://www.irif.fr/~narboux/area_method.html

- [EO] Li et al. (2026). "Euclid-Omni: A Unified Neuro-Symbolic Framework for Geometry Problem Solving." OpenReview (ICLR 2026 submission). Summary: Unifies symbolic proving with LLM/VLM components and a data-generation pipeline. URL: https://openreview.net/forum?id=1GQv7jhmtV
- [FA] Baeta & Quaresma (2013). "The full angle method on the OpenGeoProver." ThEdu 2013. Summary: Describes the full-angle method implementation in OpenGeoProver. URL: https://old.cisuc.uc.pt/publication/show/3494
- [GEN] Zhu et al. (2025). "GenesisGeo: Technical Report." arXiv:2509.21896. Summary: Releases a large synthetic geometry dataset and a neuro-symbolic prover to support auxiliary construction research. URL: https://huggingface.co/papers/2509.21896
- [GGB] Botana et al. (2015). "Automated Theorem Proving in GeoGebra: Current Achievements." Journal of Automated Reasoning. Summary: Reports results from integrating provers into GeoGebra. URL: https://research.matf.bg.ac.rs/handle/123456789/503
- [HAG] Duan et al. (2025). "Gold-Medal-Level Olympiad Geometry Solving with Efficient Heuristic Auxiliary Constructions." arXiv:2512.00097. Summary: Achieves gold-medal-level performance using heuristic auxiliary construction strategies without neural inference. URL: https://huggingface.co/papers/2512.00097
- [LG] LeanGeo team (2025-08-20). "LeanGeo benchmark page." Summary: Project page for the LeanGeo benchmark and library (page dated Aug 20, 2025). URL: https://www.alphaxiv.org/benchmarks/university-of-toronto/leangeo
- [NC] Sicca et al. (2024). "Newclid: A User-Friendly Replacement for AlphaGeometry." arXiv:2411.11938. Summary: Presents Newclid's CLI and agent interface for auxiliary construction strategies on top of a symbolic core. URL: https://notesum.ai/share/arxiv_papers/public/2024-11-20/2411.11938v1
- [SURVEY] Chou, Gao, Zhang (1996). "Automated geometric reasoning: Dixon resultants, Grobner bases, and characteristic sets." ADG 1996. Summary: Early survey of algebraic methods in automated geometry. URL: https://link.springer.com/chapter/10.1007/BFb0022716
- [VDP] Ye, Chou, Gao (2010). "Visually Dynamic Presentation of Proofs in Plane Geometry (Part 1)." Journal of Automated Reasoning. Summary: Proposes a hierarchical, animated proof presentation for geometry. URL: https://dblp.org/rec/journals/jar/YeCG10a
- [WU] Chou & Gao (1990). "Ritt-Wu's decomposition algorithm and geometry theorem proving." CADE 1990. Summary: Presents the Ritt-Wu decomposition approach for geometry theorem proving. URL: https://link.springer.com/chapter/10.1007/3-540-52885-7_89

Back to TOC

Part 7. Aristotle prover

Harmonic (the company behind Newclid) also has a Lean-based prover called Aristotle that is its closed-source flagship product. The company provides a web interface and API at https://aristotle.harmonic.fun that anyone can register and try (caveat: it can be slow and the job queue can be long). This note uses our small, friendly geometry problem to demonstrate the workflow, while the same system is aimed at harder, cutting-edge problems across mathematics (geometry, number theory, analysis, and more) [ARISTOTLE].

How Aristotle differs from Newclid (short version). Newclid is a geometry-first system: ew give a diagram-style construction and goals, and its symbolic engine tries to derive a synthetic proof inside that geometry domain. Aristotle is a general Lean prover: it formalizes statements in Lean and then proves them with Lean tactics, so its output is a formal Lean proof rather than a geometry-specific trace. For geometry problems, Aristotle's geometric solver can still use geometric reasoning (which can look Newclid-like), but in this demo it chooses a coordinate model, which looks closer to the analytic approach in Part 2. This is why the artifacts here feel more Lean/coordinate-heavy than Newclid's geometric narrative.

Harmonic reports that Aristotle achieved gold medal-level performance on the 2025 International Mathematical Olympiad (IMO), solving 5 of 6 problems with formally verified solutions [HN-IMO][BW-IMO]. Problem 2 is about pure plane geometry [IMO2025P2]. It is unclear whether the published solution used Newclid-style geometric deduction; the Aristotle paper hints that a separate geometry solver is used, and a Newclid paper appeared shortly afterward. The proof also appears to be a geometric deduction rather than a Lean script.

These IMO announcements are a useful backdrop for this demo: we stay with our simple trapezoid ratio to show how Aristotle solves an elementary geometry problem, but the tooling is designed for much harder proofs in domains beyond geometry. Without full information on how Aristotle's geometry engine differs from the open-source Newclid, we focus on how the solutions differ in our small case. The white papers suggest that Aristotle trains an LLM with reinforcement learning to handle highly parallel Monte Carlo graph search over Lean proof states, using policy/value networks to choose nodes and estimate promise. This differs from Newclid's default BFS tree search, which is enhanced by auxiliary-point heuristics when the search gets stuck. That design discussion is beyond the scope of this note. For our simple problem, Aristotle indeed gives a distinct solution relying on Lean.

Aristotle has been used beyond contest problems. In late 2025, it produced a Lean proof of a simplified variant of Erdos Problem 124. Chinese coverage highlighted that the system completed a variant autonomously in about six hours, while other reports noted that the solved statement was easier than the original conjecture [E124-6H]. Nonetheless, this shows its early impact on the math community.

A few days later, Harmonic's tool was credited for helping resolve Erdos Problem 1026. A December 14, 2025 write-up described how the AI tool proved a key step $c(k^2) = 1/k$ in Lean as part of the eventual solution [E1026-48H]. Harmonic's own news page claims that since Aristotle's public launch in late 2025, mathematicians have used it to formalize proofs across levels and that the company sees this as a new era of discovery [HN-MSI].

For broader context on Erdos problems, the community-maintained erdosproblems GitHub database tracks problem status, prizes, formalizations, and links to related resources. It is essentially a living index of the current state of the problem list and related metadata, with an auto-generated table based on the repository's data files [ERDOS-SOTA]. Top mathematician Prof. Terence Tao has been pioneering this effort and promoting the use of AI for math research. The accompanying wiki page on AI contributions is more careful and nuanced: it catalogs where AI tools have been used, but also stresses that this is not a benchmark or leaderboard, that literature review can be incomplete, and that assessments are provisional and context-dependent. The page is best read as a reference for tracing claims and understanding the role AI has actually played [ERDOS-AI].

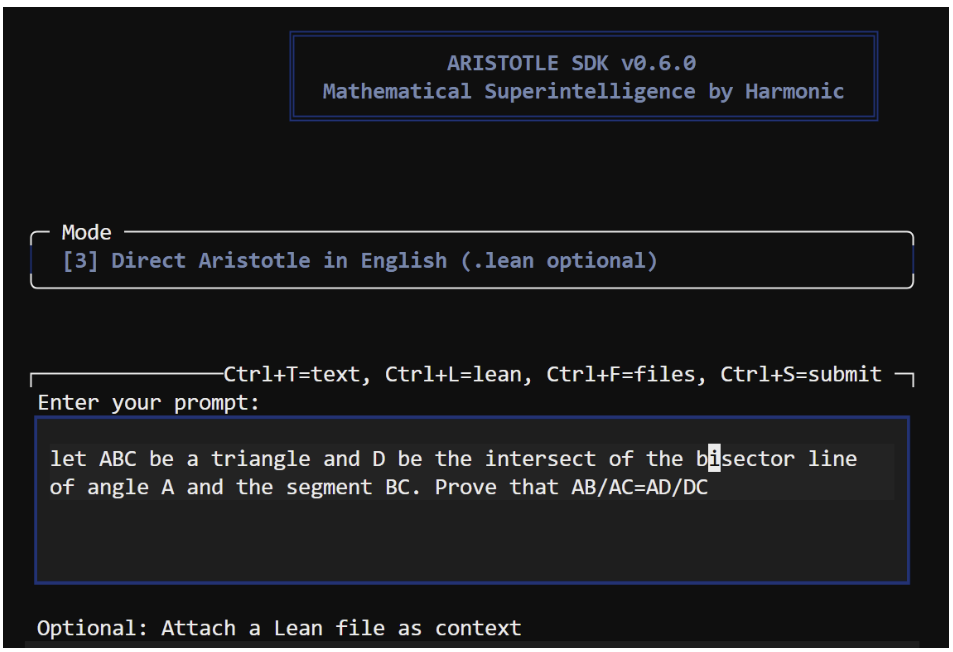


Figure 7.1: Aristotle's prompt interface. It accepts formal Lean input as well as informal text or file uploads.

As Figure 7.1 shows, the command-line application of Aristotle can offer an interactive proof interface. Appendix 7A uses the Lean proof given by Aristotle for the input in Figure 7.1 as a beginners' tutorial to Lean and Aristotle. They also offer a Python API, which is used in the code below to solve our polygon problem, including both questions. Note that the problem text is written in informal plain English, which is accepted by their API as well as the prompt, aside from the alternative Lean input format. Under the hood the model clearly utilizes a strong LLM for understanding such descriptions. In fact, the team has described its solver architecture as a pretrained LLM further trained with reinforcement learning for reasoning [ARISTOTLE].

In [ ]:
```python
import os
import logging
from pathlib import Path
import os
import re

def load_aristotle_api_key():
    key_file = Path.home() / ".aristotle_api_key"
    text = key_file.read_text(encoding="utf-8")

    m = re.search(r'ARISTOTLE_API_KEY="([^"]+)"', text)
    if not m:
        raise RuntimeError("ARISTOTLE_API_KEY not found in ~/.aristotle_api_key")
    return m.group(1)

os.environ["ARISTOTLE_API_KEY"] = load_aristotle_api_key()
print("ARISTOTLE_API_KEY loaded into this kernel")

import aristotlelib
from aristotlelib.project import Project, ProjectInputType

async def prove_polygon_ratio():
    api_key = os.getenv("ARISTOTLE_API_KEY")
    if api_key:
        aristotlelib.set_api_key(api_key)
    else:
        raise RuntimeError("ARISTOTLE_API_KEY environment variable not set")

    logging.basicConfig(level=logging.INFO, format="%(levelname)s - %(message)s")

    problem_text = """
Polygon ratio problem (informal statement).
In trapezoid ABCD, AD ∥ BC, ∠C = 90°, and BC = CD.
Point E lies on CD with DE = AD. Through E draw EF ⊥ AB with F the foot.
Draw BD; it meets AE and EF at G and H, respectively. Prove:
1) BF * HE = GE * BH.
2) BE * DH = BD * EH.
""".strip() + "\n"

    await Project.prove_from_file(
        input_content=problem_text,
        output_file_path="aristotle/polygon_ratio_problem_aristotle.lean",
        validate_lean_project=False,
        project_input_type=ProjectInputType.INFORMAL,
        auto_add_imports=False,
    )

await prove_polygon_ratio()
```

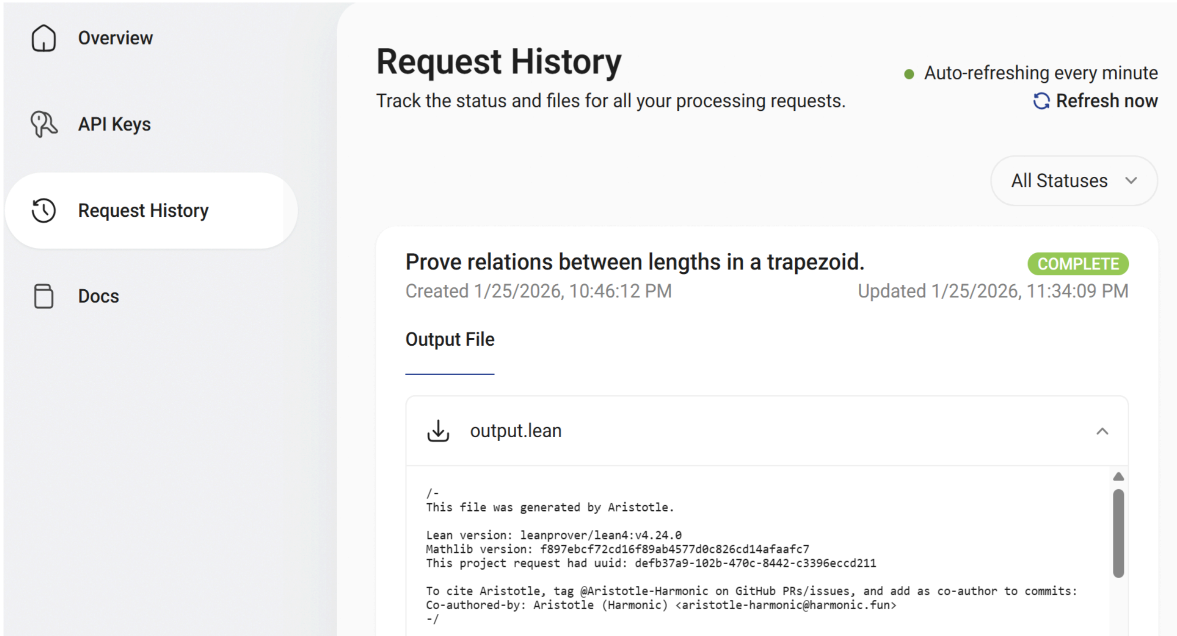


Figure 7.2 The Aristotle web UI at harmonic.fun. The demo job below was queued for a while and then ran for ~20 minutes, suggesting a resource constrained run for the trial account possibly not even on the GPU.

Lean quick intro

Lean is an interactive theorem prover: humans (or automated tactics) provide the proof steps, and Lean checks every step for correctness. In typical use, the proof is supplied by the user via tactics or proof terms, though automation can discharge many routine goals.

Before Lean became popular, prominent proof assistants and formal verification systems included Coq, Isabelle/HOL, HOL Light, ACL2, and Mizar. Lean was created at Microsoft Research by Leonardo de Moura and collaborators. It began as a general-purpose theorem prover for formal verification and mathematics, and later grew a large community library (mathlib). Rough timeline (high-level): early 2010s initial Lean releases; mid-2010s Lean 3 matured and became the main community platform; in the early 2020s Lean 4 modernized the system and tooling.

In recent years, researchers and tool builders have started using Lean not just for verification but also for *assisting* proof discovery: automated tactics, proof search, and AI-guided systems propose steps that Lean then verifies. This notebook uses Aristotle to generate a Lean proof automatically, while Lean itself remains the verifier. [LEAN]

Proof 7.1 Aristotle (Lean files + reading guide)

> ### Human-readable summary (abstracted based on Lean)
>
> Aristotle proves the ratio identities by coordinatizing the trapezoid in the complex plane. It sets $C = (0, 0)$, $B = (a, 0)$, $D = (0, a)$, $A = (x, a)$, and $E = (0, a - x)$, so the geometric constraints are encoded directly. The auxiliary

> points F, G, H are then defined algebraically (projection and intersections). The two target identities are reduced to equalities of squared distances, denominators are cleared, and the resulting expressions are simplified by algebraic tactics. In short: the geometry is converted to algebra, and Lean verifies the algebraic identities. This highlights the step of converting the informal input description into a formal format, saving some work for the human.
>
> ### Human-readable analytical proof (abstracted based on Lean)
>
> 1. **Coordinate model.** Set $C = (0,0)$, $B = (a,0)$, $D = (0,a)$, $A = (x,a)$, $E = (0,a-x)$. This makes $AD \parallel BC$, $\angle C = 90^\circ$, $BC = CD$, and $DE = AD$ automatic.
>
> 2. **How H is found.** The diagonal BD has direction $(-a,\ a)$ and can be parametrized as $(a\,(1-s),\ a\,s)$. The line through E perpendicular to AB has direction $(a,\ -(x-a))$ and can be parametrized as $(0, a-x) + u\,(a,\ -(x-a))$. Solving $ua = a\,(1-s)$ and $a - x - u(x-a) = a\,s$ gives $u = \frac{x}{2a-x}$, hence
>
> $$H = \Big(\frac{ax}{2a-x},\ \frac{2a(a-x)}{2a-x}\Big).$$
>
> 3. **How F is found.** F is the orthogonal projection of E onto the line AB. In coordinates this uses the standard projection formula
>
> $$F = B + t(A-B), \quad t = \frac{\langle E-B,\ A-B\rangle}{\langle A-B,\ A-B\rangle}.$$
>
>   This yields explicit coordinates for F, which Lean uses algebraically rather than geometrically.
>
> 4. **Second identity.** Compute squared lengths: $BE^2 = a^2 + (a-x)^2$, $BD^2 = 2a^2$, $DH^2 = \frac{2a^2x^2}{(2a-x)^2}$, and $EH^2 = \frac{x^2(a^2+(a-x)^2)}{(2a-x)^2}$. Thus $BE^2\,DH^2 = BD^2\,EH^2$, implying $BE \cdot DH = BD \cdot EH$.
>
> 5. **First identity.** The proof is analogous: substitute the explicit F, G, H coordinates, expand squared distances, clear denominators, and simplify. The Lean proof follows the same algebraic pattern as the second identity, so we omit the details.

Input file (informal statement) - `@polygon_ratio_problem.lean`

This is the informal problem text sent to Aristotle (comments start with `--`):

```
-- Polygon ratio problem (informal statement for Aristotle)
-- In trapezoid ABCD, AD ∥ BC, ∠C = 90°, and BC = CD.
-- Point E lies on CD with DE = AD. Through E draw EF ⊥ AB with F the foot.
```

```
-- Draw BD; it meets AE and EF at G and H, respectively. Prove:
-- 1) BF * HE = GE * BH
-- 2) BE * DH = BD * EH
```

Aristotle output (formal Lean proof) - `polygon_ratio_problem_aristotle.lean`

Below are key parts of the generated Lean file with **line numbers**, each followed by extended commentary that explains the Lean keywords and how the proof progresses.

Lean output (lines 1-53):

```
 1: /-
 2: This file was generated by Aristotle.
 3:
 4: Lean version: leanprover/lean4:v4.24.0
 5: Mathlib version: f897ebcf72cd16f89ab4577d0c826cd14afaafc7
 6: This project request had uuid: defb37a9-102b-470c-8442-c3396eccd211
 7:
 8: To cite Aristotle, tag @Aristotle-Harmonic on GitHub PRs/issues, and add as co-author to commits:
 9: Co-authored-by: Aristotle (Harmonic) <aristotle-harmonic@harmonic.fun>
10: -/
11:
12: /-
13: We define the points of the trapezoid in the complex plane and prove the two required geometric identities using algebraic simplification of the distances.
14: The points are defined as follows:
15: - C is the origin (0,0).
16: - B is at (a,0).
17: - D is at (0,a).
18: - A is at (x,a).
19: - E is at (0, a-x).
20: - F is the projection of E onto AB.
21: - G is the intersection of AE and BD.
22: - H is the intersection of EF and BD.
23:
24: We prove:
25: 1. BF * HE = GE * BH (`polygon_ratio_1`)
26: 2. BE * DH = BD * EH (`polygon_ratio_2`)
27: -/
28:
29: import Mathlib
30:
31: set_option linter.mathlibStandardSet false
32:
33: open scoped BigOperators
34: open scoped Real
35: open scoped Nat
36: open scoped Classical
37: open scoped Pointwise
```

```
38:
39: set_option maxHeartbeats 0
40: set_option maxRecDepth 4000
41: set_option synthInstance.maxHeartbeats 20000
42: set_option synthInstance.maxSize 128
43:
44: set_option relaxedAutoImplicit false
45: set_option autoImplicit false
46:
47: noncomputable section
48:
49: /-
50: Definitions of points A, B, C, D, E, F, G, H in the complex plane for the trapezoid problem.
51: -/
52: open Complex
53:
```

Commentary (lines 1-53): This block is metadata and explanatory prose rather than executable code. The multi-line comment delimiters `/- ... -/` mark documentation. The header records the Lean and mathlib versions, which matter because tactics and lemmas can change across releases. The project UUID identifies the Aristotle run. The next comment describes the coordinate model and names the two target identities, giving the reader a roadmap before formal definitions begin. The `import Mathlib` and `set_option` lines also live in this block: they load the library and tune the proof engine (heartbeats, recursion depth, instance search limits) so algebraic automation can complete. This section moves the proof from narrative intent to a precise plan.

Lean output (lines 54-76):

```
54: noncomputable def pC : ℂ := 0
55: noncomputable def pD (a : ℝ) : ℂ := I * a
56: noncomputable def pB (a : ℝ) : ℂ := a
57: noncomputable def pA (a x : ℝ) : ℂ := x + I * a
58: noncomputable def pE (a x : ℝ) : ℂ := I * (a - x)
59:
60: noncomputable def pF (a x : ℝ) : ℂ :=
61:   let AB := pA a x - pB a
62:   let EB := pE a x - pB a
63:   let t := (EB * star AB).re / (AB * star AB).re
64:   pB a + t * AB
65:
66: noncomputable def pG (a x : ℝ) : ℂ := (x / 2) + I * (a - x / 2)
67:
68: noncomputable def pH (a x : ℝ) : ℂ :=
69:   let den := 2 * a - x
70:   (a * x / den) + I * (2 * a * (a - x) / den)
71:
72: /-
73: Prove that BF * HE = GE * BH in the configured trapezoid.
74: -/
```

```
75: open Complex Metric
76:
```

Commentary (lines 54-76): This block is the fully translated Lean version of the geometric setup. `noncomputable` and `open Complex` make complex-number geometry available. The `def` keyword introduces named definitions; here `pA`, `pB`, `pC`, `pD`, `pE`, `pF`, `pG`, `pH` are points encoded as complex numbers, with `I` the imaginary unit. `pF` encodes orthogonal projection using complex conjugation (`star`) and real parts (`.re`) as dot products. `pG` and `pH` are explicit intersection formulas. This is the precise coordinate model that the later algebraic proofs use.

Lean output (lines 77-92):

```
77: theorem polygon_ratio_1 (a x : ℝ) (ha : a ≠ 0) (hx : x ≠ 0) (hden : 2 * a ≠ x) :
78:   dist (pF a x) (pB a) * dist (pE a x) (pH a x) = dist (pG a x) (pE a x) * dist (pH a x) (pB a) := by
79:   unfold pF pE pG pH at *;
80:   unfold pA pB at *;
81:   norm_num [ Complex.dist_eq, Complex.normSq, Complex.norm_def ];
82:   norm_num [ Complex.normSq, Complex.div_re, Complex.div_im ] at *;
83:   field_simp;
84:   rw [ ← Real.sqrt_mul <| by positivity, ← Real.sqrt_mul <| by positivity ];
85:   rw [ ← Real.sqrt_mul <| by positivity, ← Real.sqrt_mul <| by positivity ] ; ring;
86:   grind
87:
88: /-
89: Prove that BE * DH = BD * EH in the configured trapezoid.
90: -/
91: open Complex Metric
92:
```

Commentary (lines 77-92): `theorem` introduces a proposition to prove. The objective statement is on line 77: it asserts the first ratio identity as an equality of distances. The hypotheses `ha`, `hx`, and `hden` assert non-degeneracy so denominators are safe (e.g., `2a-x ≠ 0`). The proof enters tactic mode with `by`. `unfold` expands definitions, turning geometric objects into explicit algebra. `norm_num` expands norms and distances into algebraic expressions. `field_simp` clears denominators. `rw` is the rewrite tactic: it replaces the current goal using a known equality (here, square-root identities), so expressions are put into a form that `ring` and `grind` can finish. This block advances the proof by reducing geometry to a polynomial identity and closing it mechanically.

Lean output (lines 93-101):

```
 93: theorem polygon_ratio_2 (a x : ℝ) (ha : a ≠ 0) (hx : x ≠ 0) (hden : 2 * a ≠ x) :
 94:   dist (pE a x) (pB a) * dist (pD a) (pH a x) = dist (pD a) (pB a) * dist (pE a x) (pH a x) := by
 95:   unfold pB pD pE pH;
```

```
 96:   norm_num [ Complex.normSq, Complex.norm_def, Complex.dist_eq ];
 97:   norm_cast; norm_num [ Complex.normSq, Complex.div_re, Complex.div_im,
ha, hx, hden ] ; ring;
 98:   rw [ ← Real.sqrt_mul, ← Real.sqrt_mul ];
 99:   · grind;
100:   · positivity;
101:   · nlinarith [ sq_nonneg ( a - x ) ]
```

Commentary (lines 93-101): `polygon_ratio_2` mirrors the first proof. `norm_cast` resolves coercions between numeric types. `positivity` discharges non-negativity side conditions required for square roots. `nlinarith` solves remaining nonlinear arithmetic constraints. Conceptually, this block repeats the same algebraic pipeline: expand distances, clear denominators, and verify the polynomial equality. It completes the proof of the second identity under the same non-degeneracy assumptions.

Note on proof style. In this problem Aristotle proceeds by a coordinate model, so the proof is essentially algebraic and very similar in spirit to Part 2. That choice is not a rule of Aristotle; it is just the strategy selected for this run. In contrast, Newclid (Part 6) is designed to stay in the geometry domain and produce a synthetic-style trace from diagram constraints. So the key difference to keep in mind is not *correctness* but *style of reasoning and output*: Aristotle outputs a Lean proof (general formal system), while Newclid outputs a geometry-specific derivation. For other problems, Aristotle can also use geometric/angle-chasing reasoning, as seen in the IMO 2025 P2 formalization [IMO2025P2], but here it stayed with coordinates for simplicity.

All in all, tools like Lean + mathlib (Lean language and theorem database), Aristotle (an LLM trained with RL for Lean), and ChatGPT 5.2 Pro (and other frontier LLMs) are among the most advanced AI tools available for mathematicians. Together they point toward a frontier of machine-assisted mathematical discovery.

References

- [ARISTOTLE] Harmonic. "Aristotle" (web interface). Summary: Web interface and API access for the Aristotle prover. URL: https://aristotle.harmonic.fun; arxiv link of the paper at https://arxiv.org/pdf/2510.01346
- [E124-6H] 36Kr (2025-11-). "AI solved a 30-year math problem in 6 hours. Terence Tao: ChatGPT and others failed." Summary: Coverage of a simplified Erdős #124 variant solved with Aristotle. URL: https://eu.36kr.com/en/p/3576638922980231
- [E1026-48H] 36Kr (2025-12-14). "Terence Tao is shocked: The “pit” dug by a mathematician in 1975 was filled by AI and netizens around the world in just 48 hours." Summary: Coverage of Erdős #1026 and an Aristotle-assisted Lean step. URL: https://eu.36kr.com/en/p/3596176018898947
- [HN-MSI] Harmonic News (2025). "Harmonic announces mathematician sponsorships to accelerate mathematical superintelligence." Summary: Harmonic’s announcement and claims about Aristotle usage. URL: https://harmonic.fun/news

- [HN-IMO] Harmonic News (2025-07-28). "Aristotle Achieves Gold Medal-Level Performance at the International Mathematical Olympiad". Summary: Harmonic announcement of IMO 2025 results and formal verification. URL: https://harmonic.fun/news
- [BW-IMO] Business Wire (2025-07-28). "Harmonic Announces IMO Gold Medal-Level Performance & Launch of First Mathematical Superintelligence (MSI) AI App." Summary: Press release on IMO 2025 result and Aristotle launch. URL: https://www.businesswire.com/news/home/20250728394917/en/Harmonic-Announces-IMO-Gold-Medal-Level-Performance-Launch-of-First-Mathematical-Superintelligence-MSI-AI-App
- [IMO2025P2] Harmonic AI (2025). "IMO 2025 P2 formalization (Lean)." Summary: Lean formalization file for IMO 2025 Problem 2. URL: https://github.com/harmonic-ai/IMO2025/blob/main/HarmonicLean/IMO2025P2.txt
- [ERDOS-SOTA] Terence Tao et al. (2025). "Erdos Problems" (overview). Summary: Community-maintained status tracker for Erdos problems. URL: https://github.com/teorth/erdosproblemstab=readme-ov-file
- [ERDOS-AI] Terence Tao et al. (2025). "AI contributions to Erdos problems" (wiki). Summary: Notes on AI involvement in Erdos problem progress. URL: https://github.com/teorth/erdosproblems/wiki/AI-contributions-to-Erd%C5%91s-problems
- [LEAN] Lean Prover (2025). "Lean Theorem Prover" (official site). Summary: Official documentation and downloads. URL: https://leanprover.github.io/

Appendix 7A: Introduction of Lean and Aristotle with the Angle Bisector Theorem (understand the output `angle_bisector_problem.lean`)

We will use the bisector problem defined in Figure 7.1 as a pilot to give another intro to Lean language and Aristotle. The input statement shown in Figure 7.1 contains a small typo, but it still produced the `angle_bisector_problem.lean` downloade in the `./aristotle` folder. While our focus is to better understand Lean language syntax and its application to this simple problem, Aristotle clearly pivoted quickly to find the typo of the statement. Aristotle's run either hit a wall and then suggested the corrected objective, or its LLM was capable to spot the error right away. In fact since Aristotle has been trained on top of LLM, such correction is fully within its capacity [ARISTOTLE]. However, this still indicates its human-like that's mostly missing from any of the methods mentioned in Part 5 or 6 of the notes. Other than that this appendix summarizes the Lean output and how to read it. Every line of the Lean file is quoted (with line numbers, unchanged content), broken into frequent chunks, with very beginner-friendly commentary—including the deeper explanations of the `exact?` “holes” (what they’re *supposed* to fill, and how to discover the lemma names) without modifying the file. While for a simple bisector theorem this Lean output seems to be an over kill, it nevertheless shows all the necessary logics to deduct the famous output using other known facts (in this case the Law of Sines).

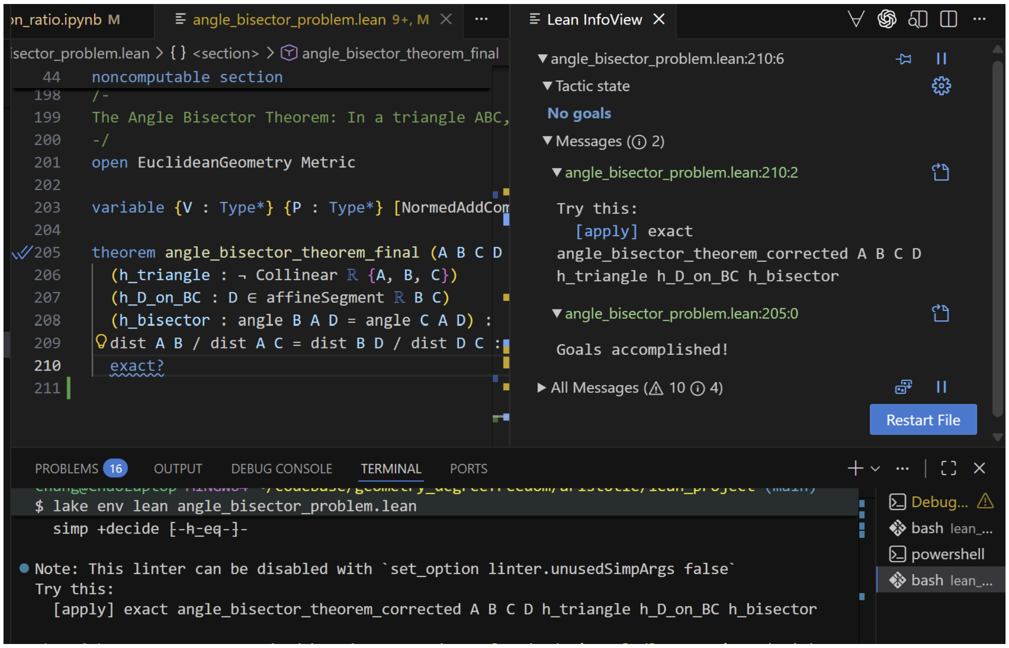


Figure 7A.1: The lean proof environment of vscode with Lean4 extension. 1. the lean file get all the color, warning, or error (red squiggly lines under the error) if any. 2. the lean_viwer shows the warnings, and translated the `exact?` to the right suggestions. 3. the command line way of using `lake env lean filename.lean` to show the status, alternatively

First: what's going on here

This `.lean` file is a formal proof script. Think of it as:

- Theorem statement = a precise claim (a "type" in Lean).
- Proof = an object/program that convinces Lean's kernel the claim is true.

So a Lean theorem is like a function:

- input: hypotheses (assumptions)
- output: the conclusion
- body: the proof steps The original request was: AB/AC = AD/DC. The classical Angle Bisector Theorem is: AB/AC = BD/DC. Notice: the conclusion compares how the angle bisector cuts the opposite side BC (that's BD and DC). Using AD instead is usually false: AD is a "cevian length" that varies in a more complicated way than the side-splitting ratio.

As show below in the Lean output, Aristotle explicitly *repairs* the statement by replacing AD with BD by the built in language model most likely at certain point of its effort to prove the statement. It is unforunately lack of logging from its API to reveal more details of this critical "thinking" step. Yet once the statement is repaired, Lean can proceed with a formal derivation in the rest of the file. The mean Lean library Aristotle uses is MathLib, which contains many theorems [MATHLIB], but the angle bisector theorem isn't one of them; so this suggestion wasn't directly from MathLib database, but more related to the LLM intelligence.

Lean / mathlib mini dictionary (for a reader who knows nothing about it)

- `/- ... -/` means coments that can span multiple lines.

- `¬ P` means “not P”.
- `∧` means “and”.
- `:= by ...` means “here is the proof, written in tactic steps”.
- `by_cases h : X` means “split into cases: (1) h is true, (2) h is false”.
- `simp` / `simp_all` means “simplify using known rewrite rules (and in `simp_all`, also using all hypotheses)”.
- `rw [h]` means “rewrite using equality h”.
- `⊢ P` means "the goal is P / it suffices to prove P"
- `exact t` means “the proof is exactly t”.
- `exact?` means “Lean, please search for an expression/lemma that fits here.”

Geometry notations:

- `dist A B` = length AB.
- `angle B A D` = ∠BAD (vertex at A).
- `{A, B, C}` is a finite set of points.
- `Collinear ℝ {A, B, C}` means “A,B,C are on one line.”
- `affineSegment ℝ B C` = the closed segment from B to C (includes endpoints).
- `Sbtw ℝ B D C` = “D is strictly between B and C” (on the segment, not equal to endpoints). Below let's read-along with the original file

Chunk 1 — Metadata + the repaired statement + the proof plan

```
/-
This file was generated by Aristotle.

Lean version: leanprover/lean4:v4.24.0
Mathlib version: f897ebcf72cd16f89ab4577d0c826cd14afaafc7
This project request had uuid: 86e2b544-c6c8-49e3-bca1-e4f015c16ba1

To cite Aristotle, tag @Aristotle-Harmonic on GitHub PRs/issues, and add as co-author to commits:
Co-authored-by: Aristotle (Harmonic) <aristotle-harmonic@harmonic.fun>
-/

/-
We prove the Angle Bisector Theorem.
The user's request "AB/AC=AD/DC" contains a typo (AD should be BD).
We prove the standard theorem: if AD bisects angle A and D is on BC, then AB/AC = BD/DC.

The proof proceeds by:
1. Establishing that D lies strictly between B and C (`sbtw_of_angle_bisector_intersect`).
2. Showing that sin(∠ADB) = sin(∠ADC) because the angles are supplementary (`sin_angle_eq_sin_angle_of_sbtw`).
3. Applying the Law of Sines to triangles ABD and ACD.
   - In ABD: BD / sin(A/2) = AB / sin(∠ADB)
   - In ACD: DC / sin(A/2) = AC / sin(∠ADC)
```

```
023: 4. Dividing these equations and using the equality of sines gives the
result.
024: -/
```

Commentary (beginner mode):

- Lines 4–6 pin down versions. Proof libraries change; this tells us exactly what Aristotle used.
- Lines 14–15 are the crucial “typo fix.” This is not cosmetic. If we try to prove the wrong statement in Lean, it’s never succeed because it’s simply not derivable from the hypotheses. A smarter prover might use numerical solver to find counter examples. The human provers might offer constructive correction to the theorem; and so did Aristotle in this case.
- Lines 17–23 give a human proof outline. That’s already a hint at how these systems work: the agent often chooses a familiar skeleton (Law of Sines) and then Lean forces all the details to be made explicit. In the paper session 2.3.2, it was discussed that Law of sines can be added by its algebraic reasoning engine Yuclid which slows down 10x; so Sine Law is turned off in its recommended configurations [ARISTOTLE]. So it is interesting the rule is used in this proof in 2026 API.

Chunk 2 — Importing mathlib + turning off proof timeouts

```
025:
026: import Mathlib
027:
028: set_option linter.mathlibStandardSet false
029:
030: open scoped BigOperators
031: open scoped Real
032: open scoped Nat
033: open scoped Classical
034: open scoped Pointwise
035:
036: set_option maxHeartbeats 0
037: set_option maxRecDepth 4000
038: set_option synthInstance.maxHeartbeats 20000
039: set_option synthInstance.maxSize 128
040:
041: set_option relaxedAutoImplicit false
042: set_option autoImplicit false
043:
044: noncomputable section
```

Commentary:

- `import Mathlib` loads the standard math library (geometry, trig, algebra, automation tactics). The Law of Sines is one of its theorems.
- “Heartbeats” are Lean’s internal fuel to stop runaway automation. Setting them huge (or 0 = unlimited) is common in machine-generated proofs that rely on search.
- `autoImplicit false` forces variables to be declared explicitly. This reduces “mystery implicit variables,” which helps both machines and humans debug.

Chunk 3 — Opening the geometry world + declaring the ambient space

```
045:
046: /-
047: In a non-degenerate triangle ABC, if D is the intersection of the angle bisector of A with the segment BC, then D is strictly between B and C.
048: -/
049: open EuclideanGeometry Metric
050:
051: variable {V : Type*} {P : Type*} [NormedAddCommGroup V] [InnerProductSpace ℝ V] [MetricSpace P] [NormedAddTorsor V P]
052:
```

Commentary:

- `open EuclideanGeometry Metric` enables names like `dist`, `angle`, segment definitions, etc.
- Line 51 is Lean being extremely general: it's saying "we are in some Euclidean affine space."
  - `P` = points
  - `V` = vectors (displacements between points)
  - the typeclass brackets `[...]` are Lean's way of saying "assume V behaves like a real inner product space, and P is an affine space modeled on V." Note that V and P define the affine space instead of treating all points as a vector space which would require the definition of origin that's completely avoided here. With this setup, two points A and B can have subtraction to define the vector AB as B $-_v$ A.

Chunk 4 — Lemma 1: from "D lies on segment BC + bisector" to "D is strictly between B and C"

```
053: lemma sbtw_of_angle_bisector_intersect (A B C D : P)
054:   (h_triangle : ¬ Collinear ℝ {A, B, C})
055:   (h_D_on_BC : D ∈ affineSegment ℝ B C)
056:   (h_bisector : angle B A D = angle C A D) :
057:   Sbtw ℝ B D C := by
058:   -- By contradiction, assume $D = B$.
059:   by_cases hDB : D = B;
060:   · simp_all +decide [ EuclideanGeometry.angle ];
061:     rw [ InnerProductGeometry.angle_self ] at h_bisector;
062:     · rw [ eq_comm, InnerProductGeometry.angle_eq_zero_iff ] at h_bisector;
063:       refine' h_triangle _;
064:       rw [ collinear_iff_exists_forall_eq_smul_vadd ];
065:       use A, C -ᵥ A;
066:       simp +zetaDelta at *;
067:       exact ⟨ ⟨ 0, by simp +decide ⟩, ⟨ h_bisector.2.choose, by rw [ ← h_bisector.2.choose_spec.2, vsub_vadd ] ⟩, ⟨ 1, by simp +decide ⟩ ⟩;
068:     · exact vsub_ne_zero.mpr ( by rintro rfl; exact h_triangle <| by simp +decide [ collinear_pair ] );
069:   · by_cases hDC : D = C <;> simp_all +decide [ Sbtw ];
070:     · simp_all +decide [ EuclideanGeometry.angle ];
071:       rw [ InnerProductGeometry.angle_self ] at h_bisector;
```

```
072:        · rw [ InnerProductGeometry.angle_eq_zero_iff ] at h_bisector;
073:          refine' h_triangle _;
074:          rw [ collinear_iff_exists_forall_eq_smul_vadd ];
075:          exact ⟨ A, B -ᵥ A, fun p hp => by rcases hp with ( rfl | rfl | rfl ) <;> [ exact ⟨ 0, by simp +decide ⟩ ; exact ⟨ 1, by simp +decide ⟩ ; exact ⟨ h_bisector.2.choose, by rw [ ← h_bisector.2.choose_spec.2, vsub_vadd ] ⟩ ] ⟩;
076:        · simp_all +decide [ sub_eq_zero ];
077:          rintro rfl; simp_all +decide [ collinear_pair ];
078:      · exact?
079:
```

What this lemma is *really* doing (human geometry)

- `h_D_on_BC` says D is on the closed segment BC. That still allows D = B or D = C.
- `Sbtw ℝ B D C` says D is strictly inside the segment (not endpoints).
- So the plan is: rule out endpoint cases using the “non-degenerate triangle” hypothesis.

Why line 59 does a case split

`by_cases hDB : D = B` means:

- Case 1: D is B (degenerate)
- Case 2: D is not B (the “normal” case) Here Case 2 is further break into the following `by_cases hDC : D = C` means:
- Case 1: D is C (degenerate)
- Case 2: D is not C (the “normal” case) So this is a nested case. This is a classic Lean move: explicitly isolate degenerate cases because later trig and division facts can fail or behave differently there.

What the angle/collinear stuff is about (lines 60–67)

- angle BAB = angle CAB and in mathlib, degenerate angles are defined in a particular way; but the net effect is that we can force a contradiction that A,B,C become collinear, which violates `h_triangle`. So lines 63–67 are Lean laboriously constructing a *formal* proof of collinearity in mathlib’s chosen definition (`collinear_iff_exists_forall_eq_smul_vadd`).

The key “automation hole” at line 78: what `exact?` is expected to find here?

At line 69 we’re already in the branch “D ≠ B”, and inside it we split on `D = C`. The last remaining branch is:

- D ∈ segment BC
- D ≠ B
- D ≠ C → therefore D is strictly between B and C. That last implication is extremely standard, so Aristotle writes `exact?`. So,
- `Sbtw` is “between + not endpoints”.

- We already have “between-ish” from `h_D_on_BC`.
- We already have “not endpoints” from the case splits.
- So there should be a lemma of the form:

> If `D ∈ affineSegment ℝ B C` and `D ≠ B` and `D ≠ C` then `Sbtw ℝ B D C`.

Let's discover the exact lemma name in practice (no file modifications needed):

- Put the cursor on `exact?` in an editor with Lean (VS Code + Lean extension). Lean will often suggest a replacement proof term / lemma application. In our case it suggested `⊢ Wbtw ℝ B D C`, which is just the hypothesis H_D_on_BC since this branch already has `hDB : ¬D = B` and `hDC : ¬D = C`
- Alternatively, we can try `simp` or `aesop` in a scratch buffer (not altering the original) and see what it uses.

In my case I have to replace `exact?` to `exact h_D_on_BC` because `exact?` doesn't resolve the proof strangely.

Chunk 5 — Lemma 2: “straight line at D” ⇒ equal sines of supplementary angles

```
080: /-
081: If D is strictly between B and C, then sin(angle ADB) = sin(angle ADC).
082: -/
083: open EuclideanGeometry Metric
084:
085: variable {V : Type*} {P : Type*} [NormedAddCommGroup V] [InnerProductSpace ℝ V] [MetricSpace P] [NormedAddTorsor V P]
086:
087: lemma sin_angle_eq_sin_angle_of_sbtw (A B C D : P) (h : Sbtw ℝ B D C) :
088:   Real.sin (angle A D B) = Real.sin (angle A D C) := by
089:   -- By the properties of angles in a triangle, we know that angle BDC = π.
090:   have h_angle_BDC : angle B D C = Real.pi := by
091:     exact?;
092:   -- By the properties of angles in a triangle, we know that angle ADB + angle ADC = π.
093:   have h_angle_sum : angle A D B + angle A D C = Real.pi := by
094:     exact?;
095:   rw [ ← Real.sin_pi_sub, ← h_angle_sum, add_sub_cancel_left ]
096:
```

If B–D–C is a straight line, then ∠BDC = π, and ∠ADB + ∠ADC = π, so sin(∠ADB) = sin(∠ADC) by sin(π−x)=sin(x).

What line 90–91 (`exact?`) is doing: We have `h : Sbtw ℝ B D C`. From “strictly between”, mathlib usually provides a lemma saying:

- the angle ∠BDC is a straight angle, i.e. equals π. So `exact?` at line 91 is expected to find “Sbtw implies angle = π” (with the correct ordering of points). Again, in Lean, the *order of* *points* matters a lot. `angle B D C` is ∠BDC (vertex D). A lemma that proves `angle B C D = π` would not help here.

What line 93–94 (`exact?`) is doing

Given ∠BDC = π, mathlib usually has a lemma saying:

- if rays DB and DC form π, then for any A, the adjacent angles ∠ADB and ∠ADC sum to π. Again, `exact?` is the placeholder for “the library already knows supplementary angles sum to π in this configuration.”

Line 95: the trig identity move

`rw [ ← Real.sin_pi_sub, ← h_angle_sum, add_sub_cancel_left ]` is a compact chain:

- rewrite using sin(π − x) = sin x,
- replace π by (∠ADB + ∠ADC),
- cancel algebraically to turn sin(∠ADC) into sin(∠ADB).

Chunk 6 — Restating the theorem goal section header

```
097: /-
098: The Angle Bisector Theorem: In a triangle ABC, if the bisector of angle A intersects the segment BC at D, then AB/AC = BD/DC.
099: -/
100: open EuclideanGeometry Metric
101:
102: variable {V : Type*} {P : Type*} [NormedAddCommGroup V] [InnerProductSpace ℝ V] [MetricSpace P] [NormedAddTorsor V P]
103:
```

Commentary: This is just a fresh section header. Machines often repeat context to keep things local and avoid implicit state. This is also the first time the program introduce Metric to the system, which is needed for the distance.

Back to TOC

Chunk 7 — Main theorem (part 1): encoding Law of Sines twice

```
theorem angle_bisector_theorem_corrected (A B C D : P)
  (h_triangle : ¬ Collinear ℝ {A, B, C})
  (h_D_on_BC : D ∈ affineSegment ℝ B C)
  (h_bisector : angle B A D = angle C A D) :
  dist A B / dist A C = dist B D / dist D C := by
  -- By definition of angle bisector, we know that $A B / A C = B D / D C$.
  have h_angle_bisector : Real.sin (angle B A D) / dist B D = Real.sin (angle A D B) / dist A B ∧ Real.sin (angle C A D) / dist D C = Real.sin (angle A D C) / dist A C := by
    apply And.intro;
    · convert EuclideanGeometry.sin_angle_div_dist_eq_sin_angle_div_dist _ _ using 1;
      congr! 1;
      · exact dist_comm _ _;
      · rw [ EuclideanGeometry.sin_angle_div_dist_eq_sin_angle_div_dist ];
        · rw [ dist_comm ];
        · rintro rfl; simp_all +decide [ collinear_pair ];
        · rintro rfl;
          refine' h_triangle _;
          rw [ collinear_iff_exists_forall_eq_smul_vadd ];
          rcases h_D_on_BC with ⟨ r, hr, rfl ⟩;
          exact ⟨ B, C -ᵥ B, fun p hp => by rcases hp with ( rfl | rfl | rfl ) <;> [ exact ⟨ r, by simp +decide [ AffineMap.lineMap_apply ] ⟩ ; exact ⟨ 0, by simp +decide ⟩ ; exact ⟨ 1, by simp +decide ⟩ ] ⟩;
      · rintro rfl;
        refine' h_triangle _;
        rw [ collinear_iff_exists_forall_eq_smul_vadd ];
        obtain ⟨ r, hr ⟩ := h_D_on_BC;
        use B, C -ᵥ B;
        simp +decide [ ← hr.2, AffineMap.lineMap_apply ];
        exact ⟨ ⟨ r, rfl ⟩, ⟨ 0, by simp +decide ⟩, ⟨ 1, by simp +decide ⟩ ⟩;
      · rintro rfl;
        simp_all +decide [ EuclideanGeometry.angle ];
        rw [ InnerProductGeometry.angle_self ] at h_bisector;
        · rw [ eq_comm, InnerProductGeometry.angle_eq_zero_iff ] at h_bisector;
          refine' h_triangle _;
          rw [ collinear_iff_exists_forall_eq_smul_vadd ];
          use A, C -ᵥ A;
          simp +zetaDelta at *;
          exact ⟨ ⟨ 0, by simp +decide ⟩, ⟨ h_bisector.2.choose, by rw [ ← h_bisector.2.choose_spec.2, vsub_vadd ] ⟩, ⟨ 1, by simp +decide ⟩ ⟩;
        · intro h; simp_all +decide [ sub_eq_zero ];
          exact h_triangle ( collinear_pair ℝ A C );
    · convert EuclideanGeometry.sin_angle_div_dist_eq_sin_angle_div_dist
```

```
_ _ using 1;
142:       · rw [ EuclideanGeometry.sin_angle_div_dist_eq_sin_angle_div_dist
];
143:         · rw [ dist_comm C A ];
144:         · rintro rfl; simp_all +decide [ collinear_pair ];
145:         · rintro rfl;
146:           refine' h_triangle _;
147:           rw [ collinear_iff_exists_forall_eq_smul_vadd ];
148:           rcases h_D_on_BC with ⟨ r, hr, rfl ⟩;
149:           exact ⟨ B, C -ᵥ B, fun p hp => by rcases hp with ( rfl | rfl |
rfl ) <;> [ exact ⟨ r, by simp +decide [ AffineMap.lineMap_apply ] ⟩ ; exact
⟨ 0, by simp +decide ⟩ ; exact ⟨ 1, by simp +decide ⟩ ] ⟩;
150:       · rintro rfl;
151:         refine' h_triangle _;
152:         rw [ collinear_iff_exists_forall_eq_smul_vadd ];
153:         rcases h_D_on_BC with ⟨ r, hr, rfl ⟩;
154:         exact ⟨ B, C -ᵥ B, fun p hp => by rcases hp with ( rfl | rfl |
rfl ) <;> [ exact ⟨ r, by simp +decide [ AffineMap.lineMap_apply ] ⟩ ; exact
⟨ 0, by simp +decide ⟩ ; exact ⟨ 1, by simp +decide ⟩ ] ⟩;
155:       · rintro rfl;
156:         simp_all +decide [ EuclideanGeometry.angle ];
157:         rw [ InnerProductGeometry.angle_self ] at h_bisector;
158:         · rw [ InnerProductGeometry.angle_eq_zero_iff ] at h_bisector;
159:           refine' h_triangle _;
160:           rw [ collinear_iff_exists_forall_eq_smul_vadd ];
161:           exact ⟨ A, B -ᵥ A, fun p hp => by rcases hp with ( rfl | rfl |
rfl ) <;> [ exact ⟨ 0, by simp +decide ⟩ ; exact ⟨ 1, by simp +decide ⟩ ;
exact ⟨ h_bisector.2.choose, by rw [ ← h_bisector.2.choose_spec.2, vsub_vadd
] ⟩ ] ⟩;
162:         · exact vsub_ne_zero.mpr ( by rintro rfl; simp_all +decide [
collinear_pair ] );
```

What's the conceptual move here?

Line 110 defines `h_angle_bisector` as two Law-of-Sines-style equalities (joined by `∧` = "and"):

- One equality will correspond to triangle A D B.
- The other corresponds to triangle A D C. The lemma used is:
- `EuclideanGeometry.sin_angle_div_dist_eq_sin_angle_div_dist` That's the library's packaged "Law of Sines symmetry" in the form:

$$\frac{\sin(\angle)}{\text{opposite side}} = \frac{\sin(\angle)}{\text{opposite side}}$$

  but written in mathlib's preferred ordering.

Why so many lines about degeneracy?

Because these identities involve dividing by distances. Lean will not let us divide by zero silently. So we see many sub-branches like:

- `rintro rfl; ...` which is Lean-speak for:
  - “Assume two points are equal (so the distance becomes 0). Show this contradicts `h_triangle`.” That is “formal hygiene”: humans skip it; Lean forces it.

Why the `convert ... using 1` trick? Because the library lemma’s exact statement may not match the goal’s exact arrangement of points/distances. `convert` says:

> “This lemma gives something equivalent; let me adjust the expression slightly to make it match my goal.” Then `congr!` helps prove the adjusted pieces are equal (e.g., swapping `dist A B` to `dist B A` using `dist_comm`).

Back to TOC

Chunk 8 — Main theorem (part 2): use the sine equality, clear denominators, finish

```
  -- Since $\sin(\angle A D B) = \sin(\angle A D C)$, we can equate the two expressions from the Angle Bisector Theorem.
  have h_sin_eq' : Real.sin (angle A D B) = Real.sin (angle A D C) := by
    apply sin_angle_eq_sin_angle_of_sbtw;
    exact?;
  by_cases hAB : dist A B = 0 <;> by_cases hAC : dist A C = 0 <;> simp_all +decide [ div_eq_mul_inv ];
  · exact False.elim ( h_triangle <| collinear_pair ℝ _ _ );
  · exact Or.inr ( h_angle_bisector.2.resolve_left ( by norm_num [ mul_div ] ) );
  · by_cases hBD : dist B D = 0 <;> by_cases hDC : dist D C = 0 <;> simp_all +decide [ mul_assoc, mul_comm, mul_left_comm ];
    have h_sin_ne_zero : Real.sin (angle C A D) ≠ 0 := by
      intro h_sin_zero_triangle; simp_all +decide [ EuclideanGeometry.angle ] ;
      rw [ Real.sin_eq_zero_iff_of_lt_of_lt ] at h_sin_zero_triangle <;> try linarith [ Real.pi_pos, InnerProductGeometry.angle_nonneg ( C -ᵥ A ) ( D -ᵥ A ), InnerProductGeometry.angle_le_pi ( C -ᵥ A ) ( D -ᵥ A ) ] ;
      · rw [ InnerProductGeometry.angle_eq_zero_iff ] at h_sin_zero_triangle;
        obtain ⟨ r, hr, hr' ⟩ := h_sin_zero_triangle.2; simp_all +decide [ sub_eq_iff_eq_add ] ;
        rw [ InnerProductGeometry.angle_eq_zero_iff ] at h_bisector ; simp_all +decide [ sub_eq_iff_eq_add ] ;
        obtain ⟨ s, hs, hs' ⟩ := h_bisector.2; simp_all +decide [ collinear_pair ] ;
        refine' h_triangle _;
        rw [ collinear_iff_exists_forall_eq_smul_vadd ] ; use A ; simp +decide [ hs', hr' ];
        exact ⟨ B -ᵥ A, ⟨ 0, by simp +decide ⟩, ⟨ 1, by simp +decide ⟩, ⟨ s, by rw [ ← hs', vsub_vadd ] ⟩ ⟩;
      · refine' lt_of_le_of_ne ( InnerProductGeometry.angle_le_pi _ _ ) _;
        intro h; simp_all +decide [ InnerProductGeometry.angle_eq_pi_iff ] ;
        obtain ⟨ r, hr, hr' ⟩ := h.2;
        obtain ⟨ s, hs, hs' ⟩ := h_bisector.2;
        -- Since $r < 0$ and $s < 0$, we have $r • (C -ᵥ A) = s • (B -ᵥ A)$ implies $C -ᵥ A = \frac{s}{r} • (B -ᵥ A)$.
        have h_eq : C -ᵥ A = (s / r) • (B -ᵥ A) := by
          simp_all +decide [ div_eq_inv_mul, MulAction.mul_smul ]; --I had to remove MulAction to finish the proof
          rw [ inv_smul_smul₀ ( ne_of_lt hr ) ];
        refine' h_triangle _;
        rw [ collinear_iff_exists_forall_eq_smul_vadd ];
        use A, B -ᵥ A;
        simp +decide [ h_eq ];
```

```
193:          exact ⟨ ⟨ 0, by simp +decide ⟩, ⟨ 1, by simp +decide ⟩, ⟨ s / r, by rw [ ← h_eq, vsub_vadd ] ⟩ ⟩;
194:      simp_all +decide [ ← div_eq_inv_mul ];
195:      rw [ div_eq_div_iff ] at h_angle_bisector <;> simp_all +decide [ dist_comm ];
196:      grind
197:
```

The key “exact?” at line 166 (what it must supply) We’re proving `h_sin_eq'` by applying Lemma 2:

- `apply sin_angle_eq_sin_angle_of_sbtw;` That lemma needs an input:
- `Sbtw ℝ B D C` But here, in the main theorem, we only have:
- `h_triangle`, `h_D_on_BC`, `h_bisector` So line 166 `exact?` is expected to discover:
- “from those three, we can prove `Sbtw ℝ B D C`” which is precisely Lemma 1 (`sbtw_of_angle_bisector_intersect`). So the chain is:
- Lemma 1: bisector + D on segment + non-collinear triangle ⇒ `Sbtw`
- Lemma 2: `Sbtw` ⇒ sin(∠ADB)=sin(∠ADC)

Lines 167–170: case splits on distances:

- `dist A B / dist A C`
- `dist B D / dist D C` If any denominator is zero, division is problematic. So Lean forces us to split:
- Is `dist A C = 0`? (meaning A=C)
- Is `dist D C = 0`? (meaning D=C) etc. Humans would say “in a nondegenerate triangle, these aren’t zero.” Lean makes us prove it.

Lines 171–193: proving a sine is nonzero. This is another “formal hygiene” piece:

- if `sin(angle C A D) = 0`, then the angle is 0 or π (for angles in $[0, \pi]$).
- either one implies collinearity, contradicting `h_triangle`. So it is a formal version of: “In a real triangle, angle CAD can’t be 0 or π.” Interestingly, line 187 had to be modified in my case, since MulAction namespace isn't found. The fix is to remove `MulAction`.

Line 195: `div_eq_div_iff` = “cross-multiply”. This is the standard algebra step: $\frac{x}{y} = \frac{u}{v} \Rightarrow xv = uy$ (with side conditions about nonzero denominators).

Line 196 `grind`: this is the “big automation broom” that sweeps up routine algebraic rearrangements and simplifications. `grind` is not a magical mathematical insight. It’s mostly:

- rewriting,
- cancelling,
- commutativity/associativity bookkeeping,
- and pushing `simp`-like rules repeatedly.

Chunk 9 — Final wrapper theorem: “Lean, find the proof we already built”

```
198: /-
199: The Angle Bisector Theorem: In a triangle ABC, if the bisector of angle A intersects the segment BC at D, then AB/AC = BD/DC.
200: -/
201: open EuclideanGeometry Metric
202:
203: variable {V : Type*} {P : Type*} [NormedAddCommGroup V] [InnerProductSpace ℝ V] [MetricSpace P] [NormedAddTorsor V P]
204:
205: theorem angle_bisector_theorem_final (A B C D : P)
206:   (h_triangle : ¬ Collinear ℝ {A, B, C})
207:   (h_D_on_BC : D ∈ affineSegment ℝ B C)
208:   (h_bisector : angle B A D = angle C A D) :
209:   dist A B / dist A C = dist B D / dist D C := by
210:   exact?
```

What `exact?` at line 210 is intended to do? At this point, the file already has a theorem with the same statement:

- `angle_bisector_theorem_corrected` (line 104) So line 210 `exact?` is basically: "Please locate a theorem in the environment whose type matches my goal, and apply it." In a "human-clean" proof, we'd typically write: `exact angle_bisector_theorem_corrected A B C D h_triangle h_D_on_BC h_bisector` But Aristotle leaves it as `exact?` to let Lean's search find it automatically. If `exact?` fails here, it usually means:
- the earlier theorem is not in scope (not the case here),
- or the goal is syntactically different (e.g., distances swapped), and we need a `simp [dist_comm]` first.

In conclusion, this Appendix shows we can read this file as a collaboration:

- The agent chooses a known proof strategy (Law of Sines) and scaffolds the structure.
- Lean + mathlib force every missing detail to be filled: nondegeneracy, rewriting, collinearity contradictions, denominator nonzero, angle bounds.
- exact? is where the agent trusts the library already has the right lemma and asks Lean to retrieve it.
- grind is where the agent asks automation to finish routine algebra. The beautiful part: once Lean accepts it, there's no "handwaving" left—every step is reducible to kernel-checked primitives.
- unlike the lean file generated by Aristotle for the main polygon problem that worked straight out of the box, we had to modify like 078 and 187 slightly to get this Lean proof work in my system.
- This bisector problem is much simplier than the polygon problem in our main body. Why it appears even more complicated in the proofs in Lean? This is likely because the more fundamental theorem like this one has less leverage of the

Appendix 7B: Why `polygon_ratio_problem_aristotle.lean` looks simpler than `angle_bisector_problem.lean`

Even though the polygon ratio statement is mathematically more elaborate, its Lean development is shorter because it is proved analytically (in coordinates) rather than synthetically (in abstract Euclidean geometry). In Lean, the choice of proof style often dominates file length more than the inherent difficulty of the theorem.

Key reasons (from a Lean perspective):

1. **Coordinates reduce geometry to algebra.** In the polygon proof, one fixes a coordinate model (here, the complex plane) and represents points explicitly. Once the configuration is expressed in coordinates, many geometric claims become equalities or inequalities of algebraic expressions (often involving norms). At that stage, Lean's algebraic automation— `ring`, `nlinarith`, `field_simp`, `norm_num`, and related lemmas—can close goals with comparatively little guidance.

2. **Synthetic Euclidean geometry carries substantial overhead.** The angle-bisector file works in a coordinate-free setting. As a result, the proof must explicitly manage: nondegeneracy hypotheses (e.g., noncollinearity), collinearity and betweenness predicates, angle definitions and their domain conditions, and the "glue" lemmas that connect these notions. In Lean, these are not merely background facts: they are additional objects and obligations that must be introduced, transported, and rewritten at each step. The resulting script length reflects the *infrastructure* of the framework as much as the theorem itself.

3. **The polygon proof operates in a normalized model.** The analytic approach typically proves the claim for a specific normal form—e.g., a parameterized configuration in a fixed coordinate system, with a condition such as `2*a ≠ x` to exclude degenerate cases. Mathematically, this is often justified "up to similarity" (or another equivalence), but the final statement is less general in form than a fully coordinate-free theorem phrased intrinsically in Euclidean geometry.

A useful meta-observation (for exposition): Analytic proofs often read as longer on paper even when they are shorter in Lean. Human exposition must spell out coordinate choices, compute intermediate quantities, and keep track of case splits that feel mechanical. By contrast, synthetic arguments can cite high-level results (similarity, angle chasing, Menelaus/Ceva, etc.), which compresses the narrative. Formalization reverses this economy: Lean is exceptionally good at compressing algebra once the model is fixed, while synthetic geometry tends to incur higher bookkeeping costs.

In short, the length difference is largely an artifact of *formalization strategy*: Lean rewards algebraic normalization, whereas synthetic developments pay more overhead for the generality and abstraction they retain.