```
In[ ]:= (* Before evaluating , you'll need to load the AreaMethod .wl package *)
```

Area Method Tutorial

Overview

Using the area method in the Wolfram Language is a three step process. First, one defines a geometric construction via a sequence of Elementary Construction Steps. Next, one formulates a precise conjecture of the form $E_1 = E_2$ (inequalities of the form $\leq, <, \geq, >, \neq$ are also acceptable), where E_1, E_2 are polynomials in geometric quantities of points defined in the geometric construction. Finally, one uses VerifyConjecture to verify whether the stated conjecture is a theorem within the geometric construction.

Stating Conjectures

In the context of the AM, a *conjecture* is a polynomial equation or inequality in geometric quantities of points defined within a construction, or a logical combination of such equations or inequalities.

Geometric Quantities

Primitive Objects

The AM makes use of one type of primitive object, called *points*. These are typically identified with points on the Euclidean plane. The set of all points is denoted $\mathbb{P}$.

There is also a primitive binary function $\overline{\bullet\bullet} : \mathbb{P}^2 \to \mathbb{R}$, called the *signed distance*, which has the following properties:

- $\overline{xy} = 0 \iff x = y$;
- $\overline{xy} = -\overline{yx}$.

The final primitive object is a ternary function $\mathcal{S}_{\bullet\bullet\bullet} : \mathbb{P}^3 \to \mathbb{R}$, called the *signed area,* with the following properties:

- $\mathcal{S}_{abc} = \mathcal{S}_{bca}$;
- $\mathcal{S}_{abc} = -\mathcal{S}_{acb}$;
- $\mathcal{S}_{aac} = 0$.

Derived Functions

For any three points a, b, $c \in \mathbb{P}$ we define their *Pythagorean difference* $\mathcal{P}_{xyz} = \overline{xy}^2 + \overline{yz}^2 - \overline{xz}^2$. This definition has many useful properties, in particular:

- $\mathcal{P}_{xyz} = 0 \iff \measuredangle xyz = \frac{\pi}{2}$;

- $\mathcal{P}_{xyx} = \mathcal{P}_{yxy} = 2 \times \overline{xy}^2$;
- $\mathcal{P}_{xxy} = \mathcal{P}_{xyy} = 0$.

For notational convenience the following two functions are also introduced:

- $\mathcal{P}_{vxyz} := \mathcal{P}_{vxz} - \mathcal{P}_{yxz}$;
- $\mathcal{S}_{vxyz} := \mathcal{S}_{vxy} + \mathcal{S}_{vzy}$.

Geometric Quantities

By a *geometric quantity* we mean either of the following:

- a ratio of signed distances $\frac{\overline{ab}}{\overline{xy}}$, subject to the constraints that the points x and y are distinct and the lines ab and xy are parallel;
- a signed area or Pythagorean difference.

Additional Predicates

The following predefined predicates can be useful for stating particular conjectures.

EqualPoints

equalPoints [x, y] means that points x and y are identical.

```
In[ ]:= EqualPoints[x, y] // TraditionalForm
```

```
Out[ ]//TraditionalForm=
```

$$\text{EqualPoints}\,(x,\ y)$$

Collinear

collinear [x, y, z] means that the points x, y, and z are collinear.

```
In[ ]:= Collinear[x, y, z] // TraditionalForm
```

```
Out[ ]//TraditionalForm=
```

$$\text{Collinear}\,(x,\ y,\ z)$$

Parallel

parallel [v, x, y, z] means that vx is parallel to yz or $v = x$ or $y = z$.

```
In[ ]:= Parallel[v, x, y, z] // TraditionalForm
```

```
Out[ ]//TraditionalForm=
```

$$\text{Parallel}\,(v,\ x,\ y,\ z)$$

Perp

perp [v, x, y, z] means that vx is perpendicular to yz.

```
In[ ]:= Perp[v, x, y, z] // TraditionalForm
Out[ ]//TraditionalForm=
```

$$\text{Perp}(v,\ x,\ y,\ z)$$

OnFoot

onFoot [y, p, u, v] means that y is on uv and yp is perpendicular to uv.

```
In[ ]:= OnFoot[y, p, u, v] // TraditionalForm
Out[ ]//TraditionalForm=
```

$$\text{OnFoot}(y,\ p,\ u,\ v)$$

EqDistance

eqDistance [a, b, c, d] means that the unsigned distances from a to b and from c to d are equal.

```
In[ ]:= EqDistance[a, b, c, d] // TraditionalForm
Out[ ]//TraditionalForm=
```

$$\text{EqDistance}(a,\ b,\ c,\ d)$$

EqAngle

eqAngle [a, b, c, d, e, f] means that the angles $\angle abc$ and $\angle def$ have equal measure.

```
In[ ]:= EqAngle[a, b, c, d, e, f] // TraditionalForm
Out[ ]//TraditionalForm=
```

$$\text{EqAngle}(a,\ b,\ c,\ d,\ e,\ f)$$

Geometric Constructions

A geometric construction is a (finite) list $C = (C_1, C_2, \ldots, C_n)$ where each C_i, $1 \le i \le n$ is an Elementary Construction Step. For each C_i the points used in that construction step must already by introduced by some construction step appearing earlier in the list. The point introduced by step i is said to have order i within the construction.

Elementary Construction Steps

There are 5 Elementary Construction Steps (ECS's) utilised by the area method. For a construction step to be well-defined certain conditions, called non-degeneracy conditions, may be required. Some ECS's also require a parameter to be provided as an argument- this may be a real number or a symbolic parameter r. Combined, these steps can be used to reproduce a large subset of classical straightedge and compass constructions. Additionally, some non-classical constructions are also possible thanks to the fact that the parameter r can be any real number. This makes it possible to, for instance, square the circle.

Elementary Construction Step 1

Constructs an arbitrary point U, denoted ECS1[U]. This is the only ECS which can be invoked without first defining any other points, and as such is used to initialize any and all geometric constructions. Points introduced by this step are called “free points”.

ndg-conditions: none

formula: none

In[ ]:=
```
ECS1[u]
```

Out[ ]= ECS1[u]

It is possible to define many free points at once:

In[ ]:=
```
ECS1[x,y,z]
```

Out[ ]= ECS1[x, y, z]

All free points in a construction are assumed to be distinct.

Elementary Construction Step 2

Constructs a point Y such that it is the intersection of (LINE U V) and (LINE P Q), denoted by ECS2[y, u, v, p, q].

ndg-conditions: UV ∦ PQ ∧ $U \neq V \wedge P \neq Q$

formula: UV ∦ PQ ∧ $U \neq V \wedge P \neq Q \wedge \mathcal{S}_{UVY} = 0 \wedge \mathcal{S}_{PQY} = 0$

In[ ]:=
```
ECS2[y,u,v,p,q]@ECS1[u,v,p,q]
```

Out[ ]= ECS2[y, u, v, p, q][ECS1[u, v, p, q]]

Elementary Construction Step 3

Constructs a point Y such that it is the foot from a given point P to (LINE U V), denoted ECS3[y, p, u, v],

ndg-conditions: $U \neq V$

formula: $U \neq V \wedge$ PY ⊥ UV ∧ $\mathcal{S}_{UVY} = 0$

In[ ]:=
```
ECS3[y,p,u,v]@ECS1[p,u,v]
```

Out[ ]= ECS3[y, p, u, v][ECS1[p, u, v]]

Elementary Construction Step 4

Constructs a point Y on the line passing through the point W and parallel to (LINE U V) such that $\overline{WY} = r\,\overline{UV}$, denoted ECS4[y, w, u, v, r]. Note that r can be a real number, a geometric quantity, or a

variable.

ndg-conditions: $U \neq V$ and if r is a rational expression in geometric quantities then its denominator should be nonzero

formula: $U \neq V \wedge \mathrm{WY} \parallel \mathrm{UV} \wedge \frac{\overline{\mathrm{WY}}}{\overline{\mathrm{UV}}} = r$

In[ ]:=
```
ECS4[y,w,u,v,r]@ECS1[w,u,v]
```

Out[ ]=
```
ECS4[y, w, u, v, r][ECS1[w, u, v]]
```

Elementary Construction Step 5

Constructs a point Y on the line passing through the point U and perpendicular to (LINE U V) such that $\frac{4\,S_{\mathrm{UVY}}}{\mathcal{P}_{\mathrm{UVU}}} = r$, denoted ECS5[y, u, v, r]. Note that r can be a real number, a geometric quantity, or a variable.

ndg-conditions: $U \neq V$ and if r is a rational expression in geometric quantities then its denominator should be nonzero

formula: $U \neq V \wedge \mathrm{UY} \perp \mathrm{UV} \wedge \frac{4\,S_{\mathrm{UVY}}}{\mathcal{P}_{\mathrm{UVU}}} = r$

In[ ]:=
```
ECS5[y,u,v,r]@ECS1[u,v]
```

Out[ ]=
```
ECS5[y, u, v, r][ECS1[u, v]]
```

Additional Construction Steps

There are many more predefined constructions available, making it easier to set up a particular geometric construction. Some of these may introduce auxiliary points or parameters into the construction.

FreePoint

FreePoint [x] is equivalent to ECS1 [x].

In[ ]:=
```
FreePoint[x]
```

Out[ ]=
```
FreePoint[x]
```

OnLine

OnLine[y, a, b] constructs an arbitrary point y on the line $\overline{\mathrm{ab}}$.

In[ ]:=
```
OnLine[y, a, b]@FreePoint[a, b]
```

Out[ ]=
```
OnLine[y, a, b][FreePoint[a, b]]
```

onLine[y, a, b, r] constructs a point y on the line $\overline{\mathrm{ab}}$ such that $\frac{\overline{\mathrm{ay}}}{\overline{\mathrm{ab}}} = r$.

In[ ]:=
```
OnLine[y, a, b, r]@FreePoint[a, b]
```

Out[ ]=
```
OnLine[y, a, b, r][FreePoint[a, b]]
```

IsIntersection

IsIntersection[y, a, b, c, d] is equivalent to ECS2[y, a, b, c, d] and means that *y* is the intersection of lines *ab* and *cd*.

```
In[ ]:= IsIntersection[y, a, b, c, d]@FreePoint[a, b, c, d]
Out[ ]= IsIntersection[y, a, b, c, d][FreePoint[a, b, c, d]]
```

OnParallel

OnParallel[y, a, b, c] constructs an arbitrary point *y* on the line parallel to *bc* going through *a*.

```
In[ ]:= OnParallel[y, a, b, c]@FreePoint[a, b, c]
Out[ ]= OnParallel[y, a, b, c][FreePoint[a, b, c]]
```

OnParallel[y, a, b, c, r] constructs a point *y* on the line parallel to *bc* going through *a* such that $\frac{\overline{ay}}{\overline{bc}} = r$.

```
In[ ]:= OnParallel[y, a, b, c, r]@FreePoint[a, b, c]
Out[ ]= OnParallel[y, a, b, c, r][FreePoint[a, b, c]]
```

OnInterLineParallel

OnInterLineParallel[y, a, b, p, c, d] means that *y* is the intersection of $\overline{ab}$ and the line through *p* parallel to $\overline{cd}$.

```
In[ ]:= OnInterLineParallel[y, a, b, p, c, d]@FreePoint[a, b, c, d, p]
Out[ ]= OnInterLineParallel[y, a, b, p, c, d][FreePoint[a, b, c, d, p]]
```

OnInterParallelParallel

OnInterParallelParallel[y, p, a, b, q, c, d] means that *y* is the intersection of the line through *p* parallel to $\overline{ab}$ and the line through *q* parallel to $\overline{cd}$.

```
In[ ]:= OnInterParallelParallel[y, p, a, b, q, c, d]@FreePoint[a, b, c, d, p, q]
Out[ ]= OnInterParallelParallel[y, p, a, b, q, c, d][FreePoint[a, b, c, d, p, q]]
```

IsMidpoint

IsMidpoint[y, a, b] constructs a point *y* such that $\frac{\overline{ay}}{\overline{ab}} = 0.5$.

```
In[ ]:= IsMidpoint[y, a, b]@FreePoint[a, b]
Out[ ]= IsMidpoint[y, a, b][FreePoint[a, b]]
```

OnPerp

OnPerp[y, a, b] constructs an arbitrary point *y* on the line through *a* perpendicular to *ab*.

```
In[ ]:= OnPerp[y, a, b]@FreePoint[a, b]
Out[ ]= OnPerp[y, a, b][FreePoint[a, b]]
```

OnPerp[y, a, b, r] is equivalent to ECS5[y, a, b, r] anc constructs a point *y* on the line through *a* perpendicular to *ab* such that $4 \times \mathcal{S}_{aby} = r \times \mathcal{P}_{aba}$.

```
In[ ]:= OnPerp[y, a, b, r]@FreePoint[a, b]
Out[ ]= OnPerp[y, a, b, r][FreePoint[a, b]]
```

OnInterLinePerp

OnInterLinePerp[y, a, b, p, c, d] means *y* is the intersection of *ab* and the line through *p* perpendicular to $\overline{cd}$.

```
In[ ]:= OnInterLinePerp[y, a, b, p, c, d]@FreePoint[a, b, c, d, p]
Out[ ]= OnInterLinePerp[y, a, b, p, c, d][FreePoint[a, b, c, d, p]]
```

IsCircumcenter

IsCircumcenter[o, a, b, c] constructs the circumcenter *o* of the triangle Δabc.

```
In[ ]:= IsCircumcenter[o, a, b, c]@FreePoint[a, b, c]
Out[ ]= IsCircumcenter[o, a, b, c][FreePoint[a, b, c]]
```

IsOrthocenter

IsOrthocenter[h, a, b, c] constructs the orthocenter *h* of the triangle Δabc.

```
In[ ]:= IsOrthocenter[h, a, b, c]@FreePoint[a, b, c]
Out[ ]= IsOrthocenter[h, a, b, c][FreePoint[a, b, c]]
```

IsCentroid

IsCentroid[i, a, b, c] constructs the centroid *i* of the triangle Δabc.

```
In[ ]:= IsCentroid[i, a, b, c]@FreePoint[a, b, c]
Out[ ]= IsCentroid[i, a, b, c][FreePoint[a, b, c]]
```

Disclaimer:

The naming convention for these predicates faithfully follows the names used in the Coq implementation. I am not particularly attached to this convention, but it should make it easier for people already

familiar with the AM to make use of these functions.

Verifying Conjectures

VerifyConjecture[*hypothesis_*, *construction_*, *options*] uses an appropriate sequence of elimination lemmas in order to check whether the provided hypothesis is in fact a theorem in the context of the provided construction.

Elimination Lemmas

Elimination Lemmas are used to eliminate all occurrences of constructed points from the stated conjecture. At each step this elimination process removes the last- with respect to the construction order- constructed point which still occurs in the conjecture. In many cases this procedure will be sufficient to resolve a conjecture. For example, the construction and statement of Heron's Formula only uses geometric quantities of free points, which means that no Elimination Lemmas can be applied. However, we still expect the Theorem to be true and provable.

Area Coordinates

It is entirely possible that after all Elimination Lemmas are applied the resulting reduced conjecture C cannot be resolved. In this case the "UseAreaCoordinates" Option can be set to True to try to further reduce the conjecture by introducing an orthonormal basis into the construction and expressing all remaining geometric quantities in terms of area coordinates. This works in the following way:

- the first free point in the construction is identified with the "origin" of the plane and renamed O;
- the second free point in the construction is renamed X and the distance $\overline{\text{OX}}$ is assumed to be equal to 1;
- a new point Y is added to the construction via ECS5[Y,O,X,1,construction]. In particular, we have $\mathcal{P}_{\text{XOY}} = 0$ and $\mathcal{S}_{\text{OXY}} = \frac{1}{2}$;
- area coordinate substitution lemmas, which are distinct from the normal elimination lemmas, are applied to C resulting in a new statement C';
- Since C' should not contain any dependent quantities, the conjecture should be resolvable using purely algebraic methods;

Checking for parallel ratios

In some cases Elimination Lemmas can be erroneously applied to ratios of non-parallel line segments. This may lead to VerifyConjecture erroneously returning True when a conjecture is in fact false or unprovable. This can happen particularly when the conjecture involves many ratios of signed distances, as in the examples of Ceva's and Menelaus' Theorems. In such cases it is advisable to set the "CheckParallelRatios" Option to True to avoid this issue. Note that this may significantly increase the time necessary to resolve the conjecture.

Proof Notebooks

Setting the "ProofNotebook" Option to True will return a detailed, step-by-step proof or maximally reduced form (in cases where the area method is not sufficient to prove the statement, or when the statement is in general false) of the conjecture.

```
In[ ]:= VerifyConjecture [SignedArea [a,b,c]==SignedArea [a,b,d]+SignedArea [a,d,c]+SignedArea [d,b,c],FreeP
        VerifyConjecture [SignedArea [a,b,c]==SignedArea [a,b,d]+SignedArea [a,d,c]+SignedArea [d,b,c],FreeP
```

```
Out[ ]= VerifyConjecture [
         SignedArea [a, b, c] == SignedArea [a, b, d] + SignedArea [a, d, c] + SignedArea [d, b, c],
         FreePoint [a, b, c, d], ProofNotebook → True]
```

```
Out[ ]= VerifyConjecture [
         SignedArea [a, b, c] == SignedArea [a, b, d] + SignedArea [a, d, c] + SignedArea [d, b, c],
         FreePoint [a, b, c, d], UseAreaCoordinates → True , ProofNotebook → True]
```

Inconsistent Constructions

Each ECS tries to make sure it can be correctly applied to an existing construction.

```
In[ ]:= IsIntersection [y,a,b,c,d]@OnParallel [d,c,a,b]@FreePoint [a,b,c]
```

```
Out[ ]= IsIntersection [y, a, b, c, d][OnParallel [d, c, a, b][FreePoint [a, b, c]]]
```

Furthermore, VerifyConjecture checks that each ECS evaluated correctly. If that's not the case, it returns an "ImpossibleConstruction" Message.

```
In[ ]:= VerifyConjecture [PythagoreanDifference [d,c]==PythagoreanDifference [a,b],IsIntersection [y,a,b,
```

```
Out[ ]= VerifyConjecture [PythagoreanDifference [d, c] == PythagoreanDifference [a, b],
         IsIntersection [y, a, b, c, d][OnParallel [d, c, a, b][FreePoint [a, b, c]]]]
```

Example Constructions and Theorems

A short and entirely noncomprehensive list of famous theorems which can be stated and proved using the AM.

Ceva's Theorem

The Setup:

In[ ]:=
```
(* Ceva's Theorem  Construction  *)
ClearAll[cevasSetup];
cevasSetup =IsIntersection[f,c,p,a,b]@IsIntersection[e,b,p,a,c]@IsIntersection[d,a,p,b,c]@Fr
```

Out[ ]=
```
IsIntersection[f, c, p, a, b][
 IsIntersection[e, b, p, a, c][IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]]
```

The Conjecture:

In[ ]:=
```
cevasTheorem =(SignedDistance[a,f] SignedDistance[b,d] SignedDistance[c,e])/(SignedDistance[f,b] SignedDistance[d,c] SignedDistance[e,a])==1//TraditionalForm
```

Out[ ]//TraditionalForm=

$$\frac{\text{SignedDistance}(a,\,f)\,\text{SignedDistance}(b,\,d)\,\text{SignedDistance}(c,\,e)}{\text{SignedDistance}(e,\,a)\,\text{SignedDistance}(f,\,b)\,\text{SignedDistance}(d,\,c)}=1$$

In[ ]:=
```
AbsoluteTiming @VerifyConjecture[cevasTheorem ,cevasSetup]
AbsoluteTiming @VerifyConjecture[cevasTheorem ,cevasSetup ,"CheckParallelRatios "→True]
```

Out[ ]=
```
{5. × 10^-6,
 VerifyConjecture[SignedDistance(a, f) SignedDistance(b, d) SignedDistance(c, e)/(SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
  IsIntersection[f, c, p, a, b][
   IsIntersection[e, b, p, a, c][IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]]]}
```

Out[ ]=
```
{2. × 10^-6,
 VerifyConjecture[SignedDistance(a, f) SignedDistance(b, d) SignedDistance(c, e)/(SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
  IsIntersection[f, c, p, a, b][IsIntersection[e, b, p, a, c][
    IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]], CheckParallelRatios → True]}
```

In[ ]:=
```
VerifyConjecture[cevasTheorem ,cevasSetup ,"ProofNotebook "→True]
```

Out[ ]=
```
VerifyConjecture[SignedDistance(a, f) SignedDistance(b, d) SignedDistance(c, e)/(SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
 IsIntersection[f, c, p, a, b][IsIntersection[e, b, p, a, c][
   IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]], ProofNotebook → True]
```

A Counterexample :

In[ ]:=
```
cevaCounterexample =(cevasTheorem /. SignedDistance [a,f]→SignedDistance [a,c])
```

Out[ ]//TraditionalForm=

$$\frac{\text{SignedDistance}(a,\,c)\,\text{SignedDistance}(b,\,d)\,\text{SignedDistance}(c,\,e)}{\text{SignedDistance}(e,\,a)\,\text{SignedDistance}(f,\,b)\,\text{SignedDistance}(d,\,c)} = 1$$

In[ ]:=
```
(* With the default value of the "CheckParallelRatios " option VerifyConjecture  is much fas
AbsoluteTiming @VerifyConjecture [cevaCounterexample ,cevasSetup ] (* This incorrectly  returns
AbsoluteTiming @VerifyConjecture [cevaCounterexample ,cevasSetup ,"CheckParallelRatios "→True]
```

Out[ ]=
```
{7. × 10^-6,
 VerifyConjecture[SignedDistance(a, c) SignedDistance(b, d) SignedDistance(c, e) / (SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
  IsIntersection[f, c, p, a, b][
   IsIntersection[e, b, p, a, c][IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]]]}
```

Out[ ]=
```
{1. × 10^-6,
 VerifyConjecture[SignedDistance(a, c) SignedDistance(b, d) SignedDistance(c, e) / (SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
  IsIntersection[f, c, p, a, b][IsIntersection[e, b, p, a, c][
    IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]], CheckParallelRatios → True]}
```

In[ ]:=
```
VerifyConjecture [cevaCounterexample ,cevasSetup ,"ProofNotebook "→True]
VerifyConjecture [cevaCounterexample ,cevasSetup ,"CheckParallelRatios "→True,"ProofNotebook "-
```

Out[ ]=
```
VerifyConjecture[SignedDistance(a, c) SignedDistance(b, d) SignedDistance(c, e) / (SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
 IsIntersection[f, c, p, a, b][IsIntersection[e, b, p, a, c][
   IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]], ProofNotebook → True]
```

Out[ ]=
```
VerifyConjecture[SignedDistance(a, c) SignedDistance(b, d) SignedDistance(c, e) / (SignedDistance(e, a) SignedDistance(f, b) SignedDistance(d, c)) == 1,
 IsIntersection[f, c, p, a, b][
  IsIntersection[e, b, p, a, c][IsIntersection[d, a, p, b, c][FreePoint[a, b, c, p]]]],
 CheckParallelRatios → True, ProofNotebook → True]
```

Desargues Theorem

The Setup:

In[ ]:=
```
ClearAll[desarguesSetup];
desarguesSetup=OnInterLineParallel[c2,a2,x,c,a,c]@OnInterLineParallel[b2,a2,x,b,a,b]@OnLine
```

The Conjecture:

In[ ]:=
```
ClearAll[desarguesTheorem];
desarguesTheorem=Parallel[b,c,b2,c2]//TraditionalForm
```

Out[ ]//TraditionalForm=

$\text{Parallel}(b,\ c,\ \text{b2},\ \text{c2})$

In[ ]:=
```
AbsoluteTiming@VerifyConjecture[desarguesTheorem,desarguesSetup]
AbsoluteTiming@VerifyConjecture[desarguesTheorem,desarguesSetup,"CheckParallelRatios"→Tru
```

Out[ ]=
```
{1.×10^-6,
 VerifyConjecture[Parallel(b, c, b2, c2), OnInterLineParallel[c2, a2, x, c, a, c][
   OnInterLineParallel[b2, a2, x, b, a, b][OnLine[x, a, a2][FreePoint[a, b, c, a2]]]]]}
```

Out[ ]=
```
{1.×10^-6, VerifyConjecture[Parallel(b, c, b2, c2),
  OnInterLineParallel[c2, a2, x, c, a, c][OnInterLineParallel[b2, a2, x, b, a, b][
    OnLine[x, a, a2][FreePoint[a, b, c, a2]]]], CheckParallelRatios → True]}
```

In[ ]:=
```
VerifyConjecture[desarguesTheorem,desarguesSetup,"ProofNotebook"→True]
```

Out[ ]=
```
VerifyConjecture[Parallel(b, c, b2, c2),
 OnInterLineParallel[c2, a2, x, c, a, c][OnInterLineParallel[b2, a2, x, b, a, b][
   OnLine[x, a, a2][FreePoint[a, b, c, a2]]]], ProofNotebook → True]
```

Euler Line Theorem

The Setup:

For any non-degenerate triangle the circumcenter, orthocenter, and Centroid are collinear.

In[ ]:=
```
ClearAll[ELTheoreomSetup];
ELTheoreomSetup=IsCentroid[z,a,b,c]@IsOrthocenter[y,a,b,c]@IsCircumcenter[x,a,b,c]@FreePoint
```

The Conjecture:

In[ ]:=
```
(* The orthocenter , circumcenter , and centroid  are collinear  *)
ClearAll [ELTheorem ];
ELTheorem =Collinear [x,y,z]//TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{Collinear}(x,\ y,\ z)$$

In[ ]:=
```
(* Note that proving  this  theorem  requires  using  area  coordinates  *)
AbsoluteTiming @VerifyConjecture [ELTheorem ,ELTheoreomSetup ]
AbsoluteTiming @VerifyConjecture [ELTheorem ,ELTheoreomSetup ,"UseAreaCoordinates "→True]
```

Out[ ]= {5. × 10^-6, VerifyConjecture [Collinear(x, y, z), IsCentroid[z, a, b, c][
IsOrthocenter[y, a, b, c][IsCircumcenter[x, a, b, c][FreePoint[a, b, c]]]]}

Out[ ]= {2. × 10^-6,
VerifyConjecture [Collinear(x, y, z), IsCentroid[z, a, b, c][IsOrthocenter[y, a, b, c][
IsCircumcenter[x, a, b, c][FreePoint[a, b, c]]]], UseAreaCoordinates → True]}

In[ ]:=
```
VerifyConjecture [ELTheorem ,ELTheoreomSetup ,"UseAreaCoordinates "→True ,"ProofNotebook "→True]
```

Out[ ]= VerifyConjecture [Collinear(x, y, z), IsCentroid[z, a, b, c][
IsOrthocenter[y, a, b, c][IsCircumcenter[x, a, b, c][FreePoint[a, b, c]]]],
UseAreaCoordinates → True, ProofNotebook → True]

Gauss-Newton Line

The Setup:

In a complete quadrangle the midpoints of all three diagonals are collinear.

In[ ]:=
```
ClearAll [gaussNewtonLineSetup ];
gaussNewtonLineSetup =IsMidpoint [m3,x,y]@IsMidpoint [m2,a0,a2]@IsMidpoint [m1,a1,a3]@IsIntersec
```

The Conjecture:

In[ ]:=
```
ClearAll [gaussNewtonLineTheorem ];
gaussNewtonLineTheorem =Collinear [m1,m2,m3]//TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{Collinear}(\text{m1},\ \text{m2},\ \text{m3})$$

In[ ]:=
```
AbsoluteTiming @VerifyConjecture [gaussNewtonLineTheorem  , gaussNewtonLineSetup ]
AbsoluteTiming @VerifyConjecture [gaussNewtonLineTheorem  , gaussNewtonLineSetup  ,"UseAreaCoord
```

Out[ ]=
```
{5. × 10^-6, VerifyConjecture [Collinear (m1, m2, m3), IsMidpoint [m3, x, y][
    IsMidpoint [m2, a0, a2][IsMidpoint [m1, a1, a3][IsIntersection [y, a0, a1, a2, a3][
      IsIntersection [x, a0, a3, a1, a2][FreePoint [a0, a1, a2, a3]]]]]]}
```

Out[ ]=
```
{2. × 10^-6,
 VerifyConjecture [Collinear (m1, m2, m3), IsMidpoint [m3, x, y][IsMidpoint [m2, a0, a2][
   IsMidpoint [m1, a1, a3][IsIntersection [y, a0, a1, a2, a3][IsIntersection [x,
     a0, a3, a1, a2][FreePoint [a0, a1, a2, a3]]]]], UseAreaCoordinates → True]}
```

In[ ]:=
```
VerifyConjecture [gaussNewtonLineTheorem  , gaussNewtonLineSetup  ,"UseAreaCoordinates  "→True ,"F
```

Out[ ]=
```
VerifyConjecture [Collinear (m1, m2, m3), IsMidpoint [m3, x, y][
  IsMidpoint [m2, a0, a2][IsMidpoint [m1, a1, a3][IsIntersection [y, a0, a1, a2, a3][
    IsIntersection [x, a0, a3, a1, a2][FreePoint [a0, a1, a2, a3]]]]],
 UseAreaCoordinates → True, ProofNotebook → True]
```

Heron's Formula

The Setup:

In[ ]:=
```
ClearAll [heroSetup ];
heroSetup =FreePoint [a,b,c];
```

The Conjecture:

In[ ]:=
```
ClearAll [heronsFormula ];
heronsFormula =(16*SignedArea [a,b,c]^2==PythagoreanDifference  [a,c,a]*PythagoreanDifference  [b,a
```

Out[ ]//TraditionalForm=

$$16\ \text{SignedArea}\,(a,\ b,\ c)^2 =$$
$$\text{PythagoreanDifference}\ (b,\ a,\ b)\,\text{PythagoreanDifference}\ (a,\ c,\ a) - \text{PythagoreanDifference}\ (c,\ a,\ b)^2$$

In[ ]:=
```
(* Note that proving this theorem requires using area coordinates *)
AbsoluteTiming @VerifyConjecture [heronsFormula ]@heroSetup
AbsoluteTiming @VerifyConjecture [heronsFormula ,"UseAreaCoordinates "→True]@heroSetup
```

Out[ ]= {4. × 10^-6, VerifyConjecture [16 SignedArea (a, b, c)^2 =
PythagoreanDifference (b, a, b) PythagoreanDifference (a, c, a) -
PythagoreanDifference (c, a, b)^2][FreePoint[a, b, c]]}

Out[ ]= {2. × 10^-6, VerifyConjecture [16 SignedArea (a, b, c)^2 =
PythagoreanDifference (b, a, b) PythagoreanDifference (a, c, a) -
PythagoreanDifference (c, a, b)^2, UseAreaCoordinates → True][FreePoint[a, b, c]]}

In[ ]:=
```
VerifyConjecture [heronsFormula ,heroSetup ,"UseAreaCoordinates "→True ,"ProofNotebook "→True]
```

Out[ ]= VerifyConjecture [
16 SignedArea (a, b, c)^2 = PythagoreanDifference (b, a, b) PythagoreanDifference (a, c, a) -
PythagoreanDifference (c, a, b)^2, FreePoint[a, b, c],
UseAreaCoordinates → True, ProofNotebook → True]

Intercept Theorem

The Setup:

In[ ]:=
```
ClearAll [interceptSetup ];
interceptSetup =IsIntersection [s,a,b,c,d]@OnParallel [d,b,a,c,r]@FreePoint [a,b,c]
```

Out[ ]= IsIntersection [s, a, b, c, d][OnParallel [d, b, a, c, r][FreePoint [a, b, c]]]

The Conjecture:

In[ ]:=
```
ClearAll [interceptTheorem ];
interceptTheorem =(SignedDistance [s,a]/SignedDistance [a,b]==SignedDistance [s,c]/SignedDistance [
```

Out[ ]//TraditionalForm=

$$\frac{\text{SignedDistance}\,(s,\,a)}{\text{SignedDistance}\,(a,\,b)} = \frac{\text{SignedDistance}\,(s,\,c)}{\text{SignedDistance}\,(c,\,d)}$$

In[ ]:=
```
(* This is an example of a theorem which isn't correctly proved without setting the "CheckP
AbsoluteTiming @VerifyConjecture [interceptTheorem ,interceptSetup ]
AbsoluteTiming @VerifyConjecture [interceptTheorem ,interceptSetup ,"CheckParallelRatios "→Tru
```

Out[ ]= $\left\{4. \times 10^{-6}, \text{VerifyConjecture}\left[\frac{\text{SignedDistance}(s, a)}{\text{SignedDistance}(a, b)} = \frac{\text{SignedDistance}(s, c)}{\text{SignedDistance}(c, d)}, \text{IsIntersection}[s, a, b, c, d][\text{OnParallel}[d, b, a, c, r][\text{FreePoint}[a, b, c]]]\right]\right\}$

Out[ ]= $\left\{2. \times 10^{-6}, \text{VerifyConjecture}\left[\frac{\text{SignedDistance}(s, a)}{\text{SignedDistance}(a, b)} = \frac{\text{SignedDistance}(s, c)}{\text{SignedDistance}(c, d)}, \text{IsIntersection}[s, a, b, c, d][\text{OnParallel}[d, b, a, c, r][\text{FreePoint}[a, b, c]]], \text{CheckParallelRatios} \to \text{True}\right]\right\}$

In[ ]:=
```
VerifyConjecture [interceptTheorem ,interceptSetup ,"CheckParallelRatios "→True ,"ProofNotebook
```

Out[ ]= $\text{VerifyConjecture}\left[\frac{\text{SignedDistance}(s, a)}{\text{SignedDistance}(a, b)} = \frac{\text{SignedDistance}(s, c)}{\text{SignedDistance}(c, d)}, \text{IsIntersection}[s, a, b, c, d][\text{OnParallel}[d, b, a, c, r][\text{FreePoint}[a, b, c]]], \text{CheckParallelRatios} \to \text{True}, \text{ProofNotebook} \to \text{True}\right]$

Midpoint Theorem

The Setup:

In[ ]:=
```
(* Midpoint Theorem Construction *)
ClearAll [midpointSetup ];
midpointSetup =IsMidpoint [e,a,c]@IsMidpoint [d,b,c]@FreePoint [a,b,c]
```

Out[ ]= $\text{IsMidpoint}[e, a, c][\text{IsMidpoint}[d, b, c][\text{FreePoint}[a, b, c]]]$

The Conjecture:

In[ ]:=
```
ClearAll [midpointTheorem ];
midpointTheorem =Parallel [a,b,d,e]//TraditionalForm
```

Out[ ]//TraditionalForm=

$\text{Parallel}(a, b, d, e)$

In[ ]:=
```
AbsoluteTiming @VerifyConjecture [midpointTheorem ,midpointSetup ]
```

Out[ ]= $\left\{5. \times 10^{-6}, \text{VerifyConjecture}[\text{Parallel}(a, b, d, e), \text{IsMidpoint}[e, a, c][\text{IsMidpoint}[d, b, c][\text{FreePoint}[a, b, c]]]]\right\}$

```
In[ ]:= VerifyConjecture [midpointTheorem ,midpointSetup ,"ProofNotebook "→True]
```

```
Out[ ]= VerifyConjecture [Parallel(a, b, d, e),
  IsMidpoint[e, a, c][IsMidpoint[d, b, c][FreePoint[a, b, c]]], ProofNotebook → True]
```

A Counterexample:

```
In[ ]:= midpointCounterexample =(midpointTheorem /.SignedArea [a,d,e]→PythagoreanDifference [a,d,e])
```

```
Out[ ]//TraditionalForm=
```

$$\text{Parallel}(a,\ b,\ d,\ e)$$

```
In[ ]:= AbsoluteTiming @VerifyConjecture [midpointCounterexample ,midpointSetup ]
```

```
Out[ ]= {2. × 10^-6, VerifyConjecture [Parallel(a, b, d, e),
    IsMidpoint[e, a, c][IsMidpoint[d, b, c][FreePoint[a, b, c]]]]}
```

```
In[ ]:= VerifyConjecture [midpointCounterexample ,midpointSetup ,"ProofNotebook "→True]
```

```
Out[ ]= VerifyConjecture [Parallel(a, b, d, e),
  IsMidpoint[e, a, c][IsMidpoint[d, b, c][FreePoint[a, b, c]]], ProofNotebook → True]
```

Menelaus' Theorem

The Setup:

Given a triangle ΔABC and a transversal line that crosses BC, AC, and AB at points D, E, and F respectively, with D, E, and F distinct from A, B, and C, then:

$$\frac{\overline{AF}}{\overline{FB}} \times \frac{\overline{BD}}{\overline{DC}} \times \frac{\overline{CE}}{\overline{EA}} = -1$$

```
In[ ]:= (* Menelaus ' Theorem  Construction  *)
ClearAll [menelausSetup ];
menelausSetup =IsIntersection [f,x,y,a,b]@IsIntersection [e,x,y,a,c]@IsIntersection [d,x,y,b,c]@
```

```
Out[ ]= IsIntersection[f, x, y, a, b][
  IsIntersection[e, x, y, a, c][IsIntersection[d, x, y, b, c][FreePoint[a, b, c, x, y]]]]
```

The Conjecture:

In[ ]:=
```
ClearAll [menelausTheorem ];
menelausTheorem =((SignedDistance [a,f]*SignedDistance [b,d]*SignedDistance [c,e])/(SignedDistance [
```

Out[ ]//TraditionalForm=

$$\frac{\text{SignedDistance}\,(a,\,f)\,\text{SignedDistance}\,(b,\,d)\,\text{SignedDistance}\,(c,\,e)}{\text{SignedDistance}\,(e,\,a)\,\text{SignedDistance}\,(f,\,b)\,\text{SignedDistance}\,(d,\,c)} = -1$$

In[ ]:=
```
AbsoluteTiming @VerifyConjecture [menelausTheorem ,menelausSetup ]
AbsoluteTiming @VerifyConjecture [menelausTheorem ,menelausSetup ,"CheckParallelRatios "→True]
```

Out[ ]=
```
{3. × 10^-6,
 VerifyConjecture[SignedDistance (a, f) SignedDistance (b, d) SignedDistance (c, e) / (SignedDistance (e, a) SignedDistance (f, b) SignedDistance (d, c)) == -1,
  IsIntersection[f, x, y, a, b][
   IsIntersection[e, x, y, a, c][IsIntersection[d, x, y, b, c][FreePoint[a, b, c, x, y]]]]]}
```

Out[ ]=
```
{2. × 10^-6,
 VerifyConjecture[SignedDistance (a, f) SignedDistance (b, d) SignedDistance (c, e) / (SignedDistance (e, a) SignedDistance (f, b) SignedDistance (d, c)) == -1,
  IsIntersection[f, x, y, a, b][IsIntersection[e, x, y, a, c][
    IsIntersection[d, x, y, b, c][FreePoint[a, b, c, x, y]]]], CheckParallelRatios → True]}
```

In[ ]:=
```
VerifyConjecture [menelausTheorem ,menelausSetup ,"CheckParallelRatios "→True ,"ProofNotebook "-
```

Out[ ]=
```
VerifyConjecture[SignedDistance (a, f) SignedDistance (b, d) SignedDistance (c, e) / (SignedDistance (e, a) SignedDistance (f, b) SignedDistance (d, c)) == -1,
 IsIntersection[f, x, y, a, b][
  IsIntersection[e, x, y, a, c][IsIntersection[d, x, y, b, c][FreePoint[a, b, c, x, y]]]],
 CheckParallelRatios → True, ProofNotebook → True]
```

Pappus's Line Theorem

The Setup:

In[ ]:=
```
ClearAll [pappusSetup ];
pappusSetup =IsIntersection [z,B1,C2,B2,C1]@IsIntersection [y,A1,C2,A2,C1]@IsIntersection [x,A1
```

The Conjecture:

```
In[ ]:= ClearAll [pappusTheorem ];
        pappusTheorem =Collinear [x,y,z]//TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{Collinear}(x,\ y,\ z)$$

```
In[ ]:= AbsoluteTiming @VerifyConjecture [pappusTheorem ]@pappusSetup
```

```
Out[ ]= {4. × 10^-6, VerifyConjecture [Collinear(x, y, z)][IsIntersection [z, B1, C2, B2, C1][
          IsIntersection [y, A1, C2, A2, C1][IsIntersection [x, A1, B2, A2, B1][
            OnLine[C2, A2, B2][OnLine[C1, A1, B1][FreePoint[A1, A2, B1, B2]]]]]]]}
```

```
In[ ]:= VerifyConjecture [pappusTheorem ,"ProofNotebook "→True]@pappusSetup
```

```
Out[ ]= VerifyConjecture [Collinear(x, y, z), ProofNotebook → True][
         IsIntersection [z, B1, C2, B2, C1][
          IsIntersection [y, A1, C2, A2, C1][IsIntersection [x, A1, B2, A2, B1][
            OnLine[C2, A2, B2][OnLine[C1, A1, B1][FreePoint[A1, A2, B1, B2]]]]]]]
```

Pythagorean Theorem

The Setup:

```
In[ ]:= (* Pythagorean  Theorem  Construction  *)
        ClearAll [pythagoreanSetup ];
        pythagoreanSetup =OnPerp [c,a,b]@FreePoint [a,b];
```

The Conjecture:

```
In[ ]:= ClearAll [pythagorasTheorem ];
        pythagorasTheorem =(PythagoreanDifference [c,a,b]==0)//TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{PythagoreanDifference}\ (c,\ a,\ b) = 0$$

```
In[ ]:= AbsoluteTiming @VerifyConjecture [pythagorasTheorem ,pythagoreanSetup ]
```

```
Out[ ]= {3. × 10^-6,
          VerifyConjecture [PythagoreanDifference (c, a, b) = 0, OnPerp[c, a, b][FreePoint[a, b]]]}
```

In[ ]:=
```
VerifyConjecture [PythagoreanDifference [c,a,b]==0,"ProofNotebook "→True]@pythagoreanSetup
```

Out[ ]=
```
VerifyConjecture [PythagoreanDifference [c, a, b] == 0, ProofNotebook → True][
 OnPerp[c, a, b][FreePoint[a, b]]]
```

Another example:

In[ ]:=
```
ClearAll [pythagoreanSetup2 ];
pythagoreanSetup2 =ECS5[d,c,b,1,ECS5[c,b,a,1,ECS1[a,b]]];
AbsoluteTiming @VerifyConjecture [PythagoreanDifference [b,a,d]==0]@pythagoreanSetup2
```

Out[ ]=
```
{1. × 10^-6, VerifyConjecture [PythagoreanDifference [b, a, d] == 0][
   ECS5[d, c, b, 1, ECS5[c, b, a, 1, ECS1[a, b]]]]}
```

In[ ]:=
```
(* Note that you have to be very careful about the ordering of points and/or signs in your
AbsoluteTiming @VerifyConjecture [SignedArea [a,b,c,d]==SignedDistance [a,b]^2//TraditionalForm ]@
AbsoluteTiming @VerifyConjecture [SignedArea [a,b,c,d]==-SignedDistance [a,b]^2//TraditionalForm ]@
AbsoluteTiming @VerifyConjecture [SignedArea [a,d,c,b]==SignedDistance [a,b]^2//TraditionalForm ]@
```

Out[ ]=
```
{8. × 10^-6, VerifyConjecture [SignedArea (a, b, c, d) = SignedDistance (a, b)^2][
   ECS5[d, c, b, 1, ECS5[c, b, a, 1, ECS1[a, b]]]]}
```

Out[ ]=
```
{0.000015 , VerifyConjecture [SignedArea (a, b, c, d) = -SignedDistance (a, b)^2][
   ECS5[d, c, b, 1, ECS5[c, b, a, 1, ECS1[a, b]]]]}
```

Out[ ]=
```
{3. × 10^-6, VerifyConjecture [SignedArea (a, d, c, b) = SignedDistance (a, b)^2][
   ECS5[d, c, b, 1, ECS5[c, b, a, 1, ECS1[a, b]]]]}
```

Triangle Inequality

The Setup:

In[ ]:=
```
ClearAll [triangleInequalitySetup ];
triangleInequalitySetup =ECS1[a,b,c];
```

The Conjecture:

In[ ]:=
```
ClearAll [triangleInequality ];
triangleInequality =(Sqrt[PythagoreanDifference [a,b,a]]+Sqrt[PythagoreanDifference [b,c,b]]≥Sqrt
```

Out[ ]//TraditionalForm=

$$\sqrt{\text{PythagoreanDifference}\ (a,\ b,\ a)}+\sqrt{\text{PythagoreanDifference}\ (b,\ c,\ b)}\geq\sqrt{\text{PythagoreanDifference}\ (a,\ c,\ a)}$$

```
In[ ]:= AbsoluteTiming @VerifyConjecture [triangleInequality ,triangleInequalitySetup ,"UseAreaCoordina
```

```
Out[ ]= {2. × 10^-6,
 VerifyConjecture [√PythagoreanDifference (a, b, a) + √PythagoreanDifference (b, c, b) ≥
   √PythagoreanDifference (a, c, a) , ECS1[a, b, c], UseAreaCoordinates → True]}
```

```
In[ ]:= VerifyConjecture [triangleInequality ,triangleInequalitySetup ,"UseAreaCoordinates "→True ,"Pro
```

```
Out[ ]= VerifyConjecture [√PythagoreanDifference (a, b, a) + √PythagoreanDifference (b, c, b) ≥
   √PythagoreanDifference (a, c, a) , ECS1[a, b, c],
 UseAreaCoordinates → True, ProofNotebook → True]
```

How Fast Is It?

The first working version of this package took about 15s (on my old, old laptop) to prove the existence of the Euler line, and less 3s for most other theorems. Run the following code to see how fast it is now.

```
In[ ]:= (* If you want a fresh kernel: *)
Quit

(* If you Quit, remember to reload the AreaMethod.wl package. *)
(* Set up some constructions and conjectures: *)
ClearAll[cevasTheorem , desarguesTheorem , ELTheorem ,
  gaussNewtonLineTheorem , heronsFormula , interceptTheorem , midpointTheorem ,
  menelausTheorem , pappusTheorem , pythagorasTheorem , triangleInequality ];
cevasSetup = IsIntersection [...][...] ;
cevasTheorem =
  TraditionalForm [((AreaMethod`SignedDistance [a, f] × AreaMethod`SignedDistance [b, d]) ×
       AreaMethod`SignedDistance [c, e]) /
     ((AreaMethod`SignedDistance [f, b] × AreaMethod`SignedDistance [d, c]) ×
       AreaMethod`SignedDistance [e, a]) == 1];
desarguesSetup = OnInterLineParallel [...][...] ;
desarguesTheorem = Parallel[b, c, b2, c2] // TraditionalForm ;
ELTheoreomSetup = IsCentroid [...][...] ;
ELTheorem = TraditionalForm [AreaMethod`Collinear [x, y, z]];
gaussNewtonLineSetup = IsMidpoint [...][...] ;
gaussNewtonLineTheorem = TraditionalForm [AreaMethod`Collinear [m1, m2, m3]];
heroSetup = FreePoint [...] ;
heronsFormula = TraditionalForm [16 AreaMethod`SignedArea [a, b, c]^2 ==
    AreaMethod`PythagoreanDifference [a, c, a] × AreaMethod`PythagoreanDifference [
      b, a, b] - AreaMethod`PythagoreanDifference [c, a, b]^2];
```

```
interceptSetup = IsIntersection [...][...] +;
interceptTheorem = TraditionalForm [
    AreaMethod`SignedDistance [s, a]/AreaMethod`SignedDistance [a, b] == AreaMethod`SignedDistance [s, c]/AreaMethod`SignedDistance [c, d]];
midpointSetup = IsMidpoint [...][...] +;
midpointTheorem = TraditionalForm [AreaMethod`Parallel [a, b, d, e]];
menelausSetup = IsIntersection [...][...] +;
menelausTheorem =
  TraditionalForm [((AreaMethod`SignedDistance [a, f] × AreaMethod`SignedDistance [b, d]) ×
       AreaMethod`SignedDistance [c, e]) /
     ((AreaMethod`SignedDistance [f, b] × AreaMethod`SignedDistance [d, c]) ×
       AreaMethod`SignedDistance [e, a]) == -1];
pappusSetup = IsIntersection [...][...] +;
pappusTheorem = TraditionalForm [AreaMethod`Collinear [x, y, z]];
pythagoreanSetup = OnPerp [...][...] +;
pythagorasTheorem = (*TraditionalForm [AreaMethod`PythagoreanDifference [c,a,b]==0]*)
  AreaMethod`PythagoreanDifference [c, a, b] == 0;
triangleInequalitySetup = ECS1 [...] +;
triangleInequality = √AreaMethod`PythagoreanDifference [a, b, a] +
    √AreaMethod`PythagoreanDifference [b, c, b] ≥
   √AreaMethod`PythagoreanDifference [c, a, c] ;
```

In[ ]:=
```
(* See how fast the prover currently is *)
Grid[{{"Theorem", "Area Coordinates", "Time", "Result"},
  Flatten@{"Ceva's Theorem", "no", AbsoluteTiming[
     VerifyConjecture[cevasTheorem, cevasSetup(*,"CheckParallelRatios"→True*)]]},
  Flatten@{"Desargues Theorem", "no", AbsoluteTiming[
     VerifyConjecture[desarguesTheorem, desarguesSetup]]},
  Flatten@{"Euler Line", "yes", AbsoluteTiming[
     VerifyConjecture[ELTheorem, ELTheoreomSetup, "UseAreaCoordinates" → True]]},
  Flatten@{"Gauss-Newton Line", "yes", AbsoluteTiming[VerifyConjecture[
      gaussNewtonLineTheorem, gaussNewtonLineSetup, "UseAreaCoordinates" → True]]},
  Flatten@{"Heron's Formula", "yes", AbsoluteTiming[
     VerifyConjecture[heronsFormula, heroSetup, "UseAreaCoordinates" → True]]},
  Flatten@{"Intercept Theorem", "no", AbsoluteTiming[VerifyConjecture[
      interceptTheorem, interceptSetup, "CheckParallelRatios" → True]]},
  Flatten@{"Midpoint Theorem", "no", AbsoluteTiming[VerifyConjecture[
      midpointTheorem, midpointSetup]]}, Flatten@{"Menelaus' Theorem",
    "no", AbsoluteTiming[VerifyConjecture[menelausTheorem, menelausSetup]]},
  Flatten@{"Pappus's Line Theorem", "no",
    AbsoluteTiming[VerifyConjecture[pappusTheorem, pappusSetup]]},
  Flatten@{"Pythagorean Theorem", "no",
    AbsoluteTiming[VerifyConjecture[pythagorasTheorem, pythagoreanSetup]]},
  Flatten@{"Triangle Inequality", "yes", AbsoluteTiming[VerifyConjecture[
      triangleInequality, triangleInequalitySetup, "UseAreaCoordinates" → True]]}},
 Frame → All, Alignment → {{Left, Center, Center, Center}}]
```

Out[ ]=

Theorem	Area Coordinates	Time	Result
Ceva's Theorem	no	0.606582	True
Desargues Theorem	no	0.270391	True
Euler Line	yes	0.619713	True
Gauss-Newton Line	yes	0.25387	True
Heron's Formula	yes	0.175368	True
Intercept Theorem	no	0.540738	True
Midpoint Theorem	no	0.126281	True
Menelaus' Theorem	no	0.689813	True
Pappus's Line Theorem	no	0.292553	True
Pythagorean Theorem	no	0.0770836	True
Triangle Inequality	yes	0.127107	True

Notes and Experiments

For dev purposes only, not part of the tutorial.

Nice Messages

In[ ]:=
```
ECS2[q, a, f, g, h]@ECS2[p, a, b, c, d]@ECS1[a, b, c, d]
```

ECS2 ::UndefinedPoints : The points {f, g, h} have not yet been constructed .

VerifyConjecture ::UndefinedVariable : Points or parameters occuring in the conjecture are not introduced in the construction .

Out[ ]=
```
ECS2[q, a, f, g, h, <| a → <| ECS → 1, points → {}, parameters → {}, order → 1|> ,
  b → <| ECS → 1, points → {}, parameters → {}, order → 2|> ,
  c → <| ECS → 1, points → {}, parameters → {}, order → 3|> ,
  d → <| ECS → 1, points → {}, parameters → {}, order → 4|> ,
  p → <| ECS → 2, points → {a, b, c, d}, parameters → {}, order → 5|> |>]
```

In[ ]:=
```
VerifyConjecture [SignedArea [p, a, b] / SignedArea [p, a, c] ==
  SignedDistance [a, b] / SignedDistance [a, c], ECS4[b, d, a, c, r]@ECS1[a, c, p]]
```

ECS4 ::UndefinedPoint : The point d has not yet been constructed .

VerifyConjecture ::ImpossibleConstruction : Inconsistent sequence of construction steps .

In[ ]:=
```
test = ECS2[g, a, b, c, d]@ECS4[b, p, a, c, r]@ECS1[a, c, p]
```

ECS2 ::UndefinedPoint : The point d has not yet been constructed .

VerifyConjecture ::UndefinedVariable : Points or parameters occuring in the conjecture are not introduced in the construction .

Out[ ]=
```
ECS2[g, a, b, c, d, <| a → <| ECS → 1, points → {}, parameters → {}, order → 1|> ,
  c → <| ECS → 1, points → {}, parameters → {}, order → 2|> ,
  p → <| ECS → 1, points → {}, parameters → {}, order → 3|> ,
  b → <| ECS → 4, points → {p, a, c}, parameters → {r}, order → 4|> |>]
```

In[ ]:=
```
test
```

Out[ ]=
```
ECS2[g, a, b, c, d, <| a → <| ECS → 1, points → {}, parameters → {}, order → 1|> ,
  c → <| ECS → 1, points → {}, parameters → {}, order → 2|> ,
  p → <| ECS → 1, points → {}, parameters → {}, order → 3|> ,
  b → <| ECS → 4, points → {p, a, c}, parameters → {r}, order → 4|> |>]
```

"test" is already the evaluated expression, so we won't get another ECS2 message, but we will always get a warning regardless

In[ ]:=
```
VerifyConjecture [SignedArea [p, a, b] / SignedArea [p, a, c],
  ECS4[b, d, a, c, r]@ECS1[a, c, p]]
```

ECS4 ::UndefinedPoint : The point d has not yet been constructed .

VerifyConjecture ::ImpossibleConstruction : Inconsistent sequence of construction steps .

In[ ]:= **ECS2[p, a, b, c, d]@ECS4[d, c, a, b, r]@ECS1[a, b, c]**

ECS2 ::ImpossibleConstruction : The line through a and b cannot be parallel to the line through c and d.

Out[ ]= ECS2[p, a, b, c, d, ⟨| a → ⟨| ECS → 1, points → {}, parameters → {}, order → 1|⟩ ,
b → ⟨| ECS → 1, points → {}, parameters → {}, order → 2|⟩ ,
c → ⟨| ECS → 1, points → {}, parameters → {}, order → 3|⟩ ,
d → ⟨| ECS → 4, points → {c, a, b}, parameters → {r}, order → 4|⟩ |⟩]