Basic Geometric Properties of Triangles

Manuel Eberl

August 16, 2018

Abstract

In this work, we define angles between vectors and between three points. Building on this, we prove basic geometric properties of triangles, such as the Isosceles Triangle Theorem, the Law of Sines and the Law of Cosines, that the sum of the angles of a triangle is π, and the congruence theorems for triangles.

The definitions and proofs were developed following those by John Harrison in HOL Light. However, due to Isabelle's type class system, all definitions and theorems in the Isabelle formalisation hold for all real inner product spaces.

Contents

1 Definition of angles

theory *Angles*
imports
  HOL−Analysis.Analysis
begin

lemma *collinear-translate-iff*: *collinear* $(((+)\ a)\ `\ A) \longleftrightarrow$ *collinear* A
  by (*auto simp*: *collinear-def*)

definition *vangle* **where**
vangle u v = (*if u* = *0* ∨ *v* = *0 then pi* / *2 else arccos* (*u* · *v* / (*norm u* ∗ *norm v*)))

definition *angle* **where**
angle a b c = *vangle* (*a* − *b*) (*c* − *b*)

lemma *angle-altdef*: *angle a b c* = *arccos* ((*a* − *b*) · (*c* − *b*) / (*dist a b* ∗ *dist c b*))
by (*simp add*: *angle-def vangle-def dist-norm*)

lemma *vangle-0-left* [*simp*]: *vangle 0 v* = *pi* / *2*
and *vangle-0-right* [*simp*]: *vangle u 0* = *pi* / *2*
by (*simp-all add*: *vangle-def*)

lemma *vangle-refl* [*simp*]: *u* ≠ *0* ⟹ *vangle u u* = *0*
by (*simp add*: *vangle-def dot-square-norm power2-eq-square*)

lemma *angle-refl* [*simp*]: *angle a a b* = *pi* / *2 angle a b b* = *pi* / *2*
by (*simp-all add*: *angle-def*)

lemma *angle-refl-mid* [*simp*]: *a* ≠ *b* ⟹ *angle a b a* = *0*
by (*simp add*: *angle-def*)

lemma *cos-vangle*: *cos* (*vangle u v*) = *u* · *v* / (*norm u* ∗ *norm v*)
unfolding *vangle-def* **using** *Cauchy-Schwarz-ineq2*[*of u v*] **by** (*auto simp*: *field-simps*)

lemma *cos-angle*: *cos* (*angle a b c*) = (*a* − *b*) · (*c* − *b*) / (*dist a b* ∗ *dist c b*)
by (*simp add*: *angle-def cos-vangle dist-norm*)

lemma *inner-conv-angle*: (*a* − *b*) · (*c* − *b*) = *dist a b* ∗ *dist c b* ∗ *cos* (*angle a b c*)
by (*simp add*: *cos-angle*)

lemma *vangle-commute*: *vangle u v* = *vangle v u*
by (*simp add*: *vangle-def inner-commute mult.commute*)

lemma *angle-commute*: *angle a b c* = *angle c b a*
by (*simp add*: *angle-def vangle-commute*)

lemma *vangle-nonneg*: *vangle u v* ≥ *0* **and** *vangle-le-pi*: *vangle u v* ≤ *pi*
using *Cauchy-Schwarz-ineq2*[*of u v*]
by (*auto simp*: *vangle-def field-simps intro*!: *arccos-lbound arccos-ubound*)

lemmas *vangle-bounds* = *vangle-nonneg vangle-le-pi*

lemma *angle-nonneg*: *angle a b c* ≥ *0* **and** *angle-le-pi*: *angle a b c* ≤ *pi*
using *vangle-bounds* **unfolding** *angle-def* **by** *blast*+

lemmas *angle-bounds* = *angle-nonneg angle-le-pi*

lemma *sin-vangle-nonneg*: *sin* (*vangle u v*) $\geq$ *0*
using *vangle-bounds* **by** (*rule sin-ge-zero*)

lemma *sin-angle-nonneg*: *sin* (*angle a b c*) $\geq$ *0*
using *angle-bounds* **by** (*rule sin-ge-zero*)

lemma *vangle-eq-0D*:
assumes *vangle u v* = *0*
shows *norm u* $*_R$ *v* = *norm v* $*_R$ *u*
proof −
from *assms* **have** *u* $\cdot$ *v* = *norm u* $*$ *norm v*
using *arccos-eq-iff*[*of* (*u* $\cdot$ *v*) / (*norm u* $*$ *norm v*) *1*] *Cauchy-Schwarz-ineq2*[*of u v*]
by (*fastforce simp*: *vangle-def split*: *if-split-asm*)
thus *?thesis* **by** (*subst* (*asm*) *norm-cauchy-schwarz-eq*) *simp-all*
qed

lemma *vangle-eq-piD*:
assumes *vangle u v* = *pi*
shows *norm u* $*_R$ *v* + *norm v* $*_R$ *u* = *0*
proof −
from *assms* **have** ($-u$) $\cdot$ *v* = *norm* ($-u$) $*$ *norm v*
using *arccos-eq-iff*[*of* (*u* $\cdot$ *v*) / (*norm u* $*$ *norm v*) -1] *Cauchy-Schwarz-ineq2*[*of u v*]
by (*simp add*: *field-simps vangle-def split*: *if-split-asm*)
thus *?thesis* **by** (*subst* (*asm*) *norm-cauchy-schwarz-eq*) *simp-all*
qed

lemma *dist-triangle-eq*:
fixes *a b c* :: *'a* :: *real-inner*
shows (*dist a c* = *dist a b* + *dist b c*) $\longleftrightarrow$ *dist a b* $*_R$ (*c* − *b*) + *dist b c* $*_R$ (*a* − *b*) = *0*
using *norm-triangle-eq*[*of b* − *a c* − *b*]
by (*simp add*: *dist-norm norm-minus-commute algebra-simps*)

lemma *angle-eq-pi-imp-dist-additive*:
assumes *angle a b c* = *pi*
shows *dist a c* = *dist a b* + *dist b c*
using *vangle-eq-piD*[*OF assms*[*unfolded angle-def*]]
by (*subst dist-triangle-eq*) (*simp add*: *dist-norm norm-minus-commute*)

lemma *orthogonal-iff-vangle*: *orthogonal u v* $\longleftrightarrow$ *vangle u v* = *pi* / *2*
using *arccos-eq-iff*[*of u* $\cdot$ *v* / (*norm u* $*$ *norm v*) *0*] *Cauchy-Schwarz-ineq2*[*of u v*]

by (*auto simp*: *vangle-def orthogonal-def*)

lemma *cos-minus1-imp-pi*:
assumes $cos\ x = -1\ x \geq 0\ x < 3 * pi$
shows $x = pi$
proof −
have $cos\ (x - pi) = 1$ **by** (*simp add*: *assms*)
then obtain n :: *int* **where** n: $of\text{-}int\ n = (x / pi - 1) / 2$
by (*subst* (*asm*) *cos-one-2pi-int*) (*auto simp*: *field-simps*)
also from *assms* **have** $\ldots \in \{-1<..<1\}$ **by** (*auto simp*: *field-simps*)
finally have $n = 0$ **by** *simp*
with n **show** *?thesis* **by** *simp*
qed

lemma *vangle-eqI*:
assumes $u \neq 0\ v \neq 0\ w \neq 0\ x \neq 0$
assumes $(u \cdot v) * norm\ w * norm\ x = (w \cdot x) * norm\ u * norm\ v$
shows $vangle\ u\ v = vangle\ w\ x$
using *assms Cauchy-Schwarz-ineq2*[*of u v*] *Cauchy-Schwarz-ineq2*[*of w x*]
unfolding *vangle-def* **by** (*auto simp*: *arccos-eq-iff field-simps*)

lemma *angle-eqI*:
assumes $a \neq b\ a \neq c\ d \neq e\ d \neq f$
assumes $((b-a) \cdot (c-a)) * dist\ d\ e * dist\ d\ f = ((e-d) \cdot (f-d)) * dist\ a\ b * dist\ a\ c$
shows $angle\ b\ a\ c = angle\ e\ d\ f$
using *assms* **unfolding** *angle-def*
by (*intro vangle-eqI*) (*simp-all add*: *dist-norm norm-minus-commute*)

lemma *cos-vangle-eqD*: $cos\ (vangle\ u\ v) = cos\ (vangle\ w\ x) \Longrightarrow vangle\ u\ v = vangle\ w\ x$
by (*rule cos-inj-pi*) (*simp-all add*: *vangle-bounds*)

lemma *cos-angle-eqD*: $cos\ (angle\ a\ b\ c) = cos\ (angle\ d\ e\ f) \Longrightarrow angle\ a\ b\ c = angle\ d\ e\ f$
unfolding *angle-def* **by** (*rule cos-vangle-eqD*)

lemma *sin-vangle-zero-iff*: $sin\ (vangle\ u\ v) = 0 \longleftrightarrow vangle\ u\ v \in \{0,\ pi\}$
proof
assume $sin\ (vangle\ u\ v) = 0$
then obtain n :: *int* **where** n: $of\text{-}int\ n = vangle\ u\ v / pi$
by (*subst* (*asm*) *sin-zero-iff-int2*) *auto*
also have $\ldots \in \{0..1\}$ **using** *vangle-bounds* **by** (*auto simp*: *field-simps*)
finally have $n \in \{0,1\}$ **by** *auto*
thus $vangle\ u\ v \in \{0,pi\}$ **using** n **by** (*auto simp*: *field-simps*)
qed *auto*

lemma *sin-angle-zero-iff*: $sin\ (angle\ a\ b\ c) = 0 \longleftrightarrow angle\ a\ b\ c \in \{0,\ pi\}$

unfolding *angle-def* **by** (*simp only*: *sin-vangle-zero-iff*)

lemma *vangle-collinear*: *vangle u v* $\in$ {*0*, *pi*} $\Longrightarrow$ *collinear* {*0*, *u*, *v*}
apply (*subst norm-cauchy-schwarz-equal* [*symmetric*])
apply (*subst norm-cauchy-schwarz-abs-eq*)
apply (*auto dest*!: *vangle-eq-0D vangle-eq-piD simp*: *eq-neg-iff-add-eq-0*)
done

lemma *angle-collinear*: *angle a b c* $\in$ {*0*, *pi*} $\Longrightarrow$ *collinear* {*a*, *b*, *c*}
apply (*unfold angle-def*, *drule vangle-collinear*)
apply (*subst collinear-translate-iff* [*symmetric*, *of* - $-b$])
apply (*auto simp*: *insert-commute*)
done

lemma *not-collinear-vangle*: $\neg$*collinear* {*0*,*u*,*v*} $\Longrightarrow$ *vangle u v* $\in$ {*0*<..<*pi*}
  using *vangle-bounds*[*of u v*] *vangle-collinear*[*of u v*]
  by (*cases vangle u v* = *0* $\vee$ *vangle u v* = *pi*) *auto*

lemma *not-collinear-angle*: $\neg$*collinear* {*a*,*b*,*c*} $\Longrightarrow$ *angle a b c* $\in$ {*0*<..<*pi*}
  using *angle-bounds*[*of a b c*] *angle-collinear*[*of a b c*]
  by (*cases angle a b c* = *0* $\vee$ *angle a b c* = *pi*) *auto*

1.1 Contributions from Lukas Bulwahn

lemma *vangle-scales*:
  assumes $0 < c$
  shows *vangle* $(c *_R v_1)$ v_2 = *vangle* v_1 v_2
using *assms* **unfolding** *vangle-def* **by** *auto*

lemma *vangle-inverse*:
  vangle $(- v_1)$ v_2 = *pi* $-$ *vangle* v_1 v_2
proof $-$
  have $|v_1 \cdot v_2 / (norm\ v_1 * norm\ v_2)| \leq 1$
  proof *cases*
    assume $v_1 \neq 0 \wedge v_2 \neq 0$
    from *this* **show** *?thesis* **by** (*simp add*: *Cauchy-Schwarz-ineq2*)
  next
    assume $\neg$ $(v_1 \neq 0 \wedge v_2 \neq 0)$
    from *this* **show** *?thesis* **by** *auto*
  qed
  from *this* **show** *?thesis*
    unfolding *vangle-def*
    by (*simp add*: *arccos-minus-abs*)
qed

lemma *orthogonal-iff-angle*:
  shows *orthogonal* $(A - B)$ $(C - B)$ $\longleftrightarrow$ *angle A B C* = *pi* / *2*
unfolding *angle-def* **by** (*auto simp only*: *orthogonal-iff-vangle*)

```
lemma angle-inverse:
  assumes between (A, C) B
  assumes A ≠ B B ≠ C
  shows angle A B D = pi − angle C B D
proof −
  from ‹between (A, C) B› obtain u where u: u ≥ 0 u ≤ 1
    and X: B = u *_R A + (1 − u) *_R C
    by (metis add.commute betweenE between-commute)
  from ‹A ≠ B› ‹B ≠ C› X have u ≠ 0 u ≠ 1 by auto
  have 0 < ((1 − u) / u)
    using ‹u ≠ 0› ‹u ≠ 1› ‹u ≥ 0› ‹u ≤ 1› by simp
  from X have A − B = − (1 − u) *_R (C − A)
   by (simp add: real-vector.scale-right-diff-distrib real-vector.scale-left-diff-distrib)
  moreover from X have C − B = u *_R (C − A)
    by (simp add: scaleR-diff-left real-vector.scale-right-diff-distrib)
  ultimately have A − B = − (((1 − u) / u) *_R (C − B))
    using ‹u ≠ 0› by simp (metis minus-diff-eq real-vector.scale-minus-left)
  from this have vangle (A − B) (D − B) = pi − vangle (C − B) (D − B)
    using ‹0 < (1 − u) / u› by (simp add: vangle-inverse vangle-scales)
  from this show ?thesis
    unfolding angle-def by simp
qed

lemma strictly-between-implies-angle-eq-pi:
  assumes between (A, C) B
  assumes A ≠ B B ≠ C
  shows angle A B C = pi
proof −
  from ‹between (A, C) B› obtain u where u: u ≥ 0 u ≤ 1
    and X: B = u *_R A + (1 − u) *_R C
    by (metis add.commute betweenE between-commute)
  from ‹A ≠ B› ‹B ≠ C› X have u ≠ 0 u ≠ 1 by auto
  from ‹A ≠ B› ‹B ≠ C› ‹between (A, C) B› have A ≠ C by auto
  from X have A − B = − (1 − u) *_R (C − A)
   by (simp add: real-vector.scale-right-diff-distrib real-vector.scale-left-diff-distrib)
  moreover from this have dist A B = norm ((1 − u) *_R (C − A))
    using ‹u ≥ 0› ‹u ≤ 1› by (simp add: dist-norm)
  moreover from X have C − B = u *_R (C − A)
    by (simp add: scaleR-diff-left real-vector.scale-right-diff-distrib)
  moreover from this have dist C B = norm (u *_R (C − A))
    by (simp add: dist-norm)
  ultimately have (A − B) · (C − B) / (dist A B * dist C B) = u * (u − 1) /
(|1 − u| * |u|)
    using ‹A ≠ C› by (simp add: dot-square-norm power2-eq-square)
  also have ... = − 1
    using ‹u ≠ 0› ‹u ≠ 1› ‹u ≥ 0› ‹u ≤ 1› by (simp add: divide-eq-minus-1-iff)
  finally show ?thesis
    unfolding angle-altdef by simp
qed
```

end

2 Basic Properties of Triangles

theory *Triangle*
imports
Complex-Main
HOL−Analysis.Topology-Euclidean-Space
Angles
begin

We prove a number of basic geometric properties of triangles. All theorems hold in any real inner product space.

2.1 Thales' theorem

theorem *thales*:
fixes *A B C* :: *'a* :: *real-inner*
assumes *dist B* (*midpoint A C*) = *dist A C* / *2*
shows *orthogonal* (*A* − *B*) (*C* − *B*)
proof −
have *dist A C* ˆ *2* = *dist B* (*midpoint A C*) ˆ *2* ∗ *4*
by (*subst assms*) (*simp add*: *field-simps power2-eq-square*)
thus *?thesis*
by (*auto simp*: *orthogonal-def dist-norm power2-norm-eq-inner midpoint-def algebra-simps inner-commute*)
qed

2.2 Sine and cosine laws

The proof of the Law of Cosines follows trivially from the definition of the angle, the definition of the norm in vector spaces with an inner product and the bilinearity of the inner product.

lemma *cosine-law-vector*:
norm (*u* − *v*) ˆ *2* = *norm u* ˆ *2* + *norm v* ˆ *2* − *2* ∗ *norm u* ∗ *norm v* ∗ *cos* (*vangle u v*)
by (*simp add*: *power2-norm-eq-inner cos-vangle algebra-simps inner-commute*)

lemma *cosine-law-triangle*:
dist b c ˆ *2* = *dist a b* ˆ *2* + *dist a c* ˆ *2* − *2* ∗ *dist a b* ∗ *dist a c* ∗ *cos* (*angle b a c*)
using *cosine-law-vector*[*of b* − *a c* − *a*]
by (*simp add*: *dist-norm angle-def vangle-commute norm-minus-commute*)

According to our definition, angles are always between 0 and π and therefore, the sign of an angle is always non-negative. We can therefore look at $\sin(\alpha)^2$, which we can express in terms of $\cos(\alpha)$ using the identity $\sin(\alpha)^2+\cos(\alpha)^2 = 1$. The remaining proof is then a trivial consequence of the definitions.

```
lemma sine-law-triangle:
  sin (angle a b c) * dist b c = sin (angle b a c) * dist a c (is ?A = ?B)
proof (cases a = b)
  assume neq: a ≠ b
  show ?thesis
  proof (rule power2-eq-imp-eq)
    from neq have (sin (angle a b c) * dist b c) ^ 2 * dist a b ^ 2 =
                  dist a b ^ 2 * dist b c ^ 2 − ((a − b) · (c − b)) ^ 2
      by (simp add: sin-squared-eq cos-angle dist-commute field-simps)
    also have ... = dist a b ^ 2 * dist a c ^ 2 − ((b − a) · (c − a)) ^ 2
      by (simp only: dist-norm power2-norm-eq-inner)
         (simp add: power2-eq-square algebra-simps inner-commute)
    also from neq have ... = (sin (angle b a c) * dist a c) ^ 2 * dist a b ^ 2
      by (simp add: sin-squared-eq cos-angle dist-commute field-simps)
     finally show ?A^2 = ?B^2 using neq by (subst (asm) mult-cancel-right)
simp-all
  qed (auto intro!: mult-nonneg-nonneg sin-angle-nonneg)
qed simp-all
```

The following forms of the Law of Sines/Cosines are more convenient for eliminating sines/cosines from a goal completely.

```
lemma cosine-law-triangle':
  2 * dist a b * dist a c * cos (angle b a c) = (dist a b ^ 2 + dist a c ^ 2 − dist b
c ^ 2)
  using cosine-law-triangle[of b c a] by simp

lemma cosine-law-triangle'':
  cos (angle b a c) = (dist a b ^ 2 + dist a c ^ 2 − dist b c ^ 2) / (2 * dist a b *
dist a c)
  using cosine-law-triangle[of b c a] by simp

lemma sine-law-triangle':
  b ≠ c ⟹ sin (angle a b c) = sin (angle b a c) * dist a c / dist b c
  using sine-law-triangle[of a b c] by (simp add: divide-simps)

lemma sine-law-triangle'':
  b ≠ c ⟹ sin (angle c b a) = sin (angle b a c) * dist a c / dist b c
  using sine-law-triangle[of a b c] by (simp add: divide-simps angle-commute)
```

2.3 Sum of angles

```
context
begin

private lemma gather-squares: a * (a * b) = a^2 * (b :: real)
  by (simp-all add: power2-eq-square)
```

```
private lemma eval-power: x ^ numeral n = x * x ^ pred-numeral n
  by (subst numeral-eq-Suc, subst power-Suc) simp
```

The proof that the sum of the angles in a triangle is π is somewhat more involved. Following the HOL Light proof by John Harrison, we first prove that $\cos(\alpha + \beta + \gamma) = -1$ and $\alpha + \beta + \gamma \in [0; 3\pi)$, which then implies the theorem.

The main work is proving $\cos(\alpha + \beta + \gamma)$. This is done using the addition theorems for the sine and cosine, then using the Laws of Sines to eliminate all sin terms save $\sin(\gamma)^2$, which only appears squared in the remaining goal. We then use $\sin(\gamma)^2 = 1 - \cos(\gamma)^2$ to eliminate this term and apply the law of cosines to eliminate this term as well.

The remaining goal is a non-linear equation containing only the length of the sides of the triangle. It can be shown by simple algebraic rewriting.

```
lemma angle-sum-triangle:
  assumes a ≠ b ∨ b ≠ c ∨ a ≠ c
  shows   angle c a b + angle a b c + angle b c a = pi
proof (rule cos-minus1-imp-pi)
  show cos (angle c a b + angle a b c + angle b c a) = − 1
  proof (cases a ≠ b)
    case True
    thus cos (angle c a b + angle a b c + angle b c a) = −1
      apply (simp add: cos-add sin-add cosine-law-triangle'' field-simps
                  sine-law-triangle''[of a b c] sine-law-triangle''[of b a c]
                  angle-commute dist-commute gather-squares sin-squared-eq)
      apply (simp add: eval-power algebra-simps dist-commute)
      done
  qed (insert assms, auto)

  show angle c a b + angle a b c + angle b c a < 3 * pi
  proof (rule ccontr)
    assume ¬(angle c a b + angle a b c + angle b c a < 3 * pi)
    with angle-le-pi[of c a b] angle-le-pi[of a b c] angle-le-pi[of b c a]
      have A: angle c a b = pi angle a b c = pi by simp-all
    thus False using angle-eq-pi-imp-dist-additive[of c a b]
              angle-eq-pi-imp-dist-additive[of a b c] by (simp add: dist-commute)
  qed
qed (auto intro!: add-nonneg-nonneg angle-nonneg)

end
```

2.4 Congruence Theorems

If two triangles agree on two angles at a non-degenerate side, the third angle must also be equal.

```
lemma similar-triangle-aa:
  assumes b1 ≠ c1 b2 ≠ c2
```

assumes *angle a1 b1 c1* = *angle a2 b2 c2*
assumes *angle b1 c1 a1* = *angle b2 c2 a2*
shows *angle b1 a1 c1* = *angle b2 a2 c2*
proof −
from *assms angle-sum-triangle*[*of a1 b1 c1*] *angle-sum-triangle*[*of a2 b2 c2*, *symmetric*]
show *?thesis* **by** (*auto simp*: *algebra-simps angle-commute*)
qed

A triangle is defined by its three angles and the lengths of three sides up to congruence. Two triangles are congruent if they have their angles are the same and their sides have the same length.

locale *congruent-triangle* =
fixes *a1 b1 c1* :: *'a* :: *real-inner* **and** *a2 b2 c2* :: *'b* :: *real-inner*
assumes *sides'*: *dist a1 b1* = *dist a2 b2 dist a1 c1* = *dist a2 c2 dist b1 c1* = *dist b2 c2*
and *angles'*: *angle b1 a1 c1* = *angle b2 a2 c2 angle a1 b1 c1* = *angle a2 b2 c2 angle a1 c1 b1* = *angle a2 c2 b2*
begin

lemma *sides*:
dist a1 b1 = *dist a2 b2 dist a1 c1* = *dist a2 c2 dist b1 c1* = *dist b2 c2*
dist b1 a1 = *dist a2 b2 dist c1 a1* = *dist a2 c2 dist c1 b1* = *dist b2 c2*
dist a1 b1 = *dist b2 a2 dist a1 c1* = *dist c2 a2 dist b1 c1* = *dist c2 b2*
dist b1 a1 = *dist b2 a2 dist c1 a1* = *dist c2 a2 dist c1 b1* = *dist c2 b2*
using *sides'* **by** (*simp-all add*: *dist-commute*)

lemma *angles*:
angle b1 a1 c1 = *angle b2 a2 c2 angle a1 b1 c1* = *angle a2 b2 c2 angle a1 c1 b1* = *angle a2 c2 b2*
angle c1 a1 b1 = *angle b2 a2 c2 angle c1 b1 a1* = *angle a2 b2 c2 angle b1 c1 a1* = *angle a2 c2 b2*
angle b1 a1 c1 = *angle c2 a2 b2 angle a1 b1 c1* = *angle c2 b2 a2 angle a1 c1 b1* = *angle b2 c2 a2*
angle c1 a1 b1 = *angle c2 a2 b2 angle c1 b1 a1* = *angle c2 b2 a2 angle b1 c1 a1* = *angle b2 c2 a2*
using *angles'* **by** (*simp-all add*: *angle-commute*)

end

lemmas *congruent-triangleD* = *congruent-triangle.sides congruent-triangle.angles*

Given two triangles that agree on a subset of its side lengths and angles that are sufficient to define a triangle uniquely up to congruence, one can conclude that they must also agree on all remaining quantities, i.e. that they are congruent.

The following four congruence theorems state what constitutes such a uniquely-defining subset of quantities. Each theorem states in its name which quantities are required and in which order (clockwise or counter-clockwise): an "s" stands for a side, an "a" stands for an angle.

The lemma "congruent-triangleI-sas, for example, requires that two adjacent sides and the angle inbetween are the same in both triangles.

```
lemma congruent-triangleI-sss:
  fixes a1 b1 c1 :: 'a :: real-inner and a2 b2 c2 :: 'b :: real-inner
  assumes dist a1 b1 = dist a2 b2
  assumes dist b1 c1 = dist b2 c2
  assumes dist a1 c1 = dist a2 c2
  shows   congruent-triangle a1 b1 c1 a2 b2 c2
proof −
  have A: angle a1 b1 c1 = angle a2 b2 c2
    if dist a1 b1 = dist a2 b2 dist b1 c1 = dist b2 c2 dist a1 c1 = dist a2 c2
    for a1 b1 c1 :: 'a and a2 b2 c2 :: 'b
  proof −
    from that cosine-law-triangle''[of a1 b1 c1] cosine-law-triangle''[of a2 b2 c2]
      show ?thesis by (intro cos-angle-eqD) (simp add: dist-commute)
  qed
  from assms show ?thesis by unfold-locales (auto intro!: A simp: dist-commute)
qed

lemmas congruent-triangle-sss = congruent-triangleD[OF congruent-triangleI-sss]

lemma congruent-triangleI-sas:
  assumes dist a1 b1 = dist a2 b2
  assumes dist b1 c1 = dist b2 c2
  assumes angle a1 b1 c1 = angle a2 b2 c2
  shows   congruent-triangle a1 b1 c1 a2 b2 c2
proof (rule congruent-triangleI-sss)
  show dist a1 c1 = dist a2 c2
  proof (rule power2-eq-imp-eq)
    from cosine-law-triangle[of a1 c1 b1] cosine-law-triangle[of a2 c2 b2] assms
      show (dist a1 c1)^2 = (dist a2 c2)^2 by (simp add: dist-commute)
  qed simp-all
qed fact+

lemmas congruent-triangle-sas = congruent-triangleD[OF congruent-triangleI-sas]

lemma congruent-triangleI-aas:
  assumes angle a1 b1 c1 = angle a2 b2 c2
  assumes angle b1 c1 a1 = angle b2 c2 a2
  assumes dist a1 b1 = dist a2 b2
  assumes ¬collinear {a1,b1,c1}
  shows   congruent-triangle a1 b1 c1 a2 b2 c2
proof (rule congruent-triangleI-sas)
  from ‹¬collinear {a1,b1,c1}› have neq: a1 ≠ b1 by auto
  with assms(3) have neq': a2 ≠ b2 by auto
  have A: angle c1 a1 b1 = angle c2 a2 b2 using neq neq' assms
```

using *angle-sum-triangle*[*of a1 b1 c1*] *angle-sum-triangle*[*of a2 b2 c2*]
by *simp*
from *assms* **have** *B*: *angle b1 a1 c1* ∈ {*0*<..<*pi*}
by (*intro not-collinear-angle*) (*simp-all add*: *insert-commute*)
from *sine-law-triangle*[*of c1 a1 b1*] *sine-law-triangle*[*of c2 a2 b2*] *assms A B*
show *dist b1 c1* = *dist b2 c2*
by (*auto simp*: *angle-commute dist-commute sin-angle-zero-iff*)
qed *fact*+

lemmas *congruent-triangle-aas* = *congruent-triangleD*[*OF congruent-triangleI-aas*]

lemma *congruent-triangleI-asa*:
assumes *angle a1 b1 c1* = *angle a2 b2 c2*
assumes *dist a1 b1* = *dist a2 b2*
assumes *angle b1 a1 c1* = *angle b2 a2 c2*
assumes ¬*collinear* {*a1*, *b1*, *c1*}
shows *congruent-triangle a1 b1 c1 a2 b2 c2*
proof (*rule congruent-triangleI-aas*)
from *assms* **have** *neq*: *a1* ≠ *b1 a2* ≠ *b2* **by** *auto*
show *angle b1 c1 a1* = *angle b2 c2 a2*
by (*rule similar-triangle-aa*) (*insert assms neq*, *simp-all add*: *angle-commute*)
qed *fact*+

lemmas *congruent-triangle-asa* = *congruent-triangleD*[*OF congruent-triangleI-asa*]

2.5 Isosceles Triangle Theorem

We now prove the Isosceles Triangle Theorem: in a triangle where two sides have the same length, the two angles that are adjacent to only one of the two sides must be equal.

lemma *isosceles-triangle*:
assumes *dist a c* = *dist b c*
shows *angle b a c* = *angle a b c*
by (*rule congruent-triangle-sss*) (*insert assms*, *simp-all add*: *dist-commute*)

For the non-degenerate case (i.e. the three points are not collinear), We also prove the converse.

lemma *isosceles-triangle-converse*:
assumes *angle a b c* = *angle b a c* ¬*collinear* {*a*,*b*,*c*}
shows *dist a c* = *dist b c*
by (*rule congruent-triangle-asa*[*OF assms*(*1*) - - *assms*(*2*)])
(*simp-all add*: *dist-commute angle-commute assms*)

2.6 Contributions by Lukas Bulwahn

lemma *Pythagoras*:
fixes *A B C* :: ′*a* :: *real-inner*
assumes *orthogonal* (*A* − *C*) (*B* − *C*)

```
  shows (dist B C) ^ 2 + (dist C A) ^ 2 = (dist A B) ^ 2
proof -
  from assms have cos (angle A C B) = 0
    by (metis orthogonal-iff-angle cos-pi-half)
  from this show ?thesis
    by (simp add: cosine-law-triangle[of A B C] dist-commute)
qed

lemma isosceles-triangle-orthogonal-on-midpoint:
  fixes A B C :: 'a :: euclidean-space
  assumes dist C A = dist C B
  shows orthogonal (C − midpoint A B) (A − midpoint A B)
proof (cases A = B)
  assume A ≠ B
  let ?M = midpoint A B
  from ‹A ≠ B› have angle A ?M C = pi − angle B ?M C
    by (intro angle-inverse between-midpoint)
      (auto simp: between-midpoint eq-commute[of - midpoint A B for A B])
  moreover have angle A ?M C = angle C ?M B
  proof -
    have congruence: congruent-triangle C A ?M C B ?M
    proof (rule congruent-triangleI-sss)
      show dist C A = dist C B using assms .
      show dist A ?M = dist B ?M by (simp add: dist-midpoint)
      show dist C (midpoint A B) = dist C (midpoint A B) ..
    qed
    from this show ?thesis by (simp add: congruent-triangle.angles(6))
  qed
  ultimately have angle A ?M C = pi / 2 by (simp add: angle-commute)
  from this show ?thesis
    by (simp add: orthogonal-iff-angle orthogonal-commute)
next
  assume A = B
  from this show ?thesis
    by (simp add: orthogonal-clauses(1))
qed

end
```