

# Three squares theorem

Anton Danilkin, Loïc Chevalier

April 10, 2024

## Abstract

We formalize the Legendre's three squares theorem and its consequences, in particular the following results:

1. A natural number can be represented as the sum of three squares of natural numbers if and only if it is not of the form  $4^a(8k + 7)$ , where  $a$  and  $k$  are natural numbers.
2. If  $n$  is a natural number such that  $n \equiv 3 \pmod{8}$ , then  $n$  can be represented as the sum of three squares of odd natural numbers.

Consequences include the following:

1. An integer  $n$  can be written as  $n = x^2 + y^2 + z^2 + z$ , where  $x, y, z$  are integers, if and only if  $n \geq 0$ .
2. The Legendre's four squares theorem: any natural number can be represented as the sum of four squares of natural numbers.

We follow the book of Melvyn B. Nathanson 'Additive Number Theory: The Classical Bases' [1].

We plan to make use of the first consequence mentioned above in an upcoming AFP entry on Diophantine equations. More concretely, we intend to formalize universal pairs over the integers which requires expressing a natural number as a polynomial in integers while only using few variables.

## Contents

<b>1 Properties of residues, congruences, quadratic residues and the Legendre symbol</b>	<b>2</b>
1.1 Properties of residues and congruences . . . . .	2
1.2 Properties of quadratic residues . . . . .	4
1.3 Properties of the Legendre symbol . . . . .	4
<b>2 Vectors and matrices, determinants and their properties in dimensions 2 and 3</b>	<b>9</b>
<b>3 Properties of quadratic forms and their equivalences</b>	<b>22</b>

<b>4 Legendre's three squares theorem and its consequences</b>	<b>42</b>
4.1 Legendre's three squares theorem . . . . .	42
4.2 Consequences . . . . .	67

## 1 Properties of residues, congruences, quadratic residues and the Legendre symbol

```
theory Residues-Properties
imports HOL-Number-Theory.Quadratic-Reciprocity
begin
```

### 1.1 Properties of residues and congruences

```
lemma mod-diff-eq-nat:
  fixes a b m :: nat
  assumes a ≥ b
  shows (a - b) mod m = (m + (a mod m) - (b mod m)) mod m
proof cases
  assume m = 0
  thus ?thesis by auto
next
  assume 0: m ≠ 0
  have (a - b) mod m = nat (int (a - b) mod int m)
    unfolding nat-mod-as-int by blast
  also have ... = nat ((int a - int b) mod int m)
    using assms by (simp add: of-nat-diff)
  also have ... = nat ((int a mod int m - int b mod int m) mod int m)
    using mod-diff-eq by metis
  also have ... = nat ((int a mod int m + (int m - int b mod int m)) mod int m)
    by (metis add.left-commute add-uminus-conv-diff mod-add-self1)
  also have ... = nat ((int (nat (int a mod int m)) +
    (int m - int b mod int m)) mod int m)
    by (metis nat-int of-nat-mod)
  also have ... = nat ((int (nat (int a mod int m)) +
    int (m - nat (int b mod int m))) mod int m)
    by (metis 0 less-eq-nat.simps(1) mod-less-divisor
      nat-int nat-less-le of-nat-diff zmod-int)
  also have ... = nat (int (m + nat (int a mod int m) -
    nat (int b mod int m)) mod int m)
    by (metis 0 Nat.add-diff-assoc add.commute bot-nat-0.not-eq-extremum
      mod-less-divisor nat-less-le nat-mod-as-int of-nat-add)
  also have ... = (m + (a mod m) - (b mod m)) mod m
    unfolding nat-mod-as-int by blast
  finally show ?thesis .
qed
```

```
lemma prime-invertible-int:
  fixes a p :: int
```

```

assumes prime p
assumes ¬ p dvd a
shows ∃ b. [a * b = 1] (mod p)
using assms coprime-commute coprime-iff-invertible-int prime-imp-coprime by
blast

lemma power-cong:
fixes x y a m :: nat
assumes coprime a m
assumes [x = y] (mod totient m)
shows [a ^ x = a ^ y] (mod m)
proof -
obtain k :: int where 0: x + totient m * k = y
using assms by (metis cong-iff-lin cong-int-iff)
show ?thesis
proof cases
assume k ≥ 0
hence x + totient m * nat k = y
using 0 by (metis int-eq-iff nat-int-add of-nat-mult)
hence [a ^ x * (a ^ totient m) ^ nat k = a ^ y] (mod m)
unfolding cong-def by (metis power-add power-mult)
hence [a ^ x * (a ^ totient m mod m) ^ nat k = a ^ y] (mod m)
unfolding cong-def by (metis mod-mult-right-eq power-mod)
hence [a ^ x * (1 mod m) ^ nat k = a ^ y] (mod m)
using euler-theorem[OF assms(1)] unfolding cong-def by argo
hence [a ^ x * 1 ^ nat k = a ^ y] (mod m)
unfolding cong-def by (metis mod-mult-right-eq power-mod)
thus [a ^ x = a ^ y] (mod m) by auto
next
assume ¬ k ≥ 0
hence x = y + totient m * nat (- k)
using 0
by (smt (verit) int-nat-eq nat-int nat-minus-as-int of-nat-add of-nat-mult
right-diff-distrib')
hence [a ^ x = a ^ y * (a ^ totient m) ^ nat (- k)] (mod m)
unfolding cong-def by (metis power-add power-mult)
hence [a ^ x = a ^ y * (a ^ totient m mod m) ^ nat (- k)] (mod m)
unfolding cong-def by (metis mod-mult-right-eq power-mod)
hence [a ^ x = a ^ y * (1 mod m) ^ nat (- k)] (mod m)
using euler-theorem[OF assms(1)] unfolding cong-def by argo
hence [a ^ x = a ^ y * 1 ^ nat (- k)] (mod m)
unfolding cong-def by (metis mod-mult-right-eq power-mod)
thus [a ^ x = a ^ y] (mod m) by auto
qed
qed

lemma power-cong-alt:
fixes x a m :: nat
assumes coprime a m

```

**shows**  $a^{\wedge} x \text{ mod } m = a^{\wedge} (x \text{ mod totient } m) \text{ mod } m$   
**using** power-cong[*OF assms*] cong-def cong-mod-left **by** blast

## 1.2 Properties of quadratic residues

```

lemma QuadRes-cong:
  fixes a b p :: int
  assumes [a = b] (mod p)
  assumes QuadRes p a
  shows QuadRes p b
  using assms cong-trans unfolding QuadRes-def by blast

lemma QuadRes-mult:
  fixes a b p :: int
  assumes QuadRes p a
  assumes QuadRes p b
  shows QuadRes p (a * b)
  using assms
  unfolding QuadRes-def
  by (metis cong-mult mult.assoc mult.commute power2-eq-square)

lemma QuadRes-inv:
  fixes a b p :: int
  assumes prime p
  assumes [a * b = 1] (mod p)
  assumes QuadRes p a
  shows QuadRes p b
  proof -
    have 0:  $\neg p \text{ dvd } a$ 
    using assms
    by (metis cong-dvd-iff dvd-mult2 not-prime-unit)
    obtain x where 1:  $[x^2 = a] \text{ (mod } p)$  using assms unfolding QuadRes-def by blast
    have  $\neg p \text{ dvd } x$  using 0 1 assms cong-dvd-iff pos2 prime-dvd-power-iff by blast
    then obtain y where  $[x * y = 1] \text{ (mod } p)$ 
    using assms prime-invertible-int by blast
    hence 2:  $[(x * y)^2 = 1] \text{ (mod } p)$  using cong-pow by fastforce
    have  $[x^2 * b = 1] \text{ (mod } p)$  using 1 assms cong-scalar-right cong-trans by blast
    hence  $[y^2 * (x^2 * b) = y^2 * 1] \text{ (mod } p)$  using cong-scalar-left by blast
    hence  $[(x * y)^2 * b = y^2] \text{ (mod } p)$  by (simp add: algebra-simps)
    hence  $[b = y^2] \text{ (mod } p)$ 
    using 2
    by (metis cong-def cong-scalar-left mult.commute mult.right-neutral)
    hence  $[y^2 = b] \text{ (mod } p)$  by (rule cong-sym)
    thus ?thesis unfolding QuadRes-def by blast
  qed

```

## 1.3 Properties of the Legendre symbol

**lemma** Legendre-cong:

```

fixes a b p :: int
assumes [a = b] (mod p)
shows Legendre a p = Legendre b p
using assms QuadRes-cong[of a b p] QuadRes-cong[of b a p]
unfolding Legendre-def cong-def
by auto

lemma Legendre-one:
fixes p :: int
assumes p > 2
shows Legendre 1 p = 1
using assms
by (smt (verit) Legendre-def QuadRes-def cong-def
cong-less-imp-eq-int one-power2)

lemma Legendre-minus-one:
fixes p :: int
assumes prime p
assumes p > 2
shows Legendre (- 1) p = 1  $\longleftrightarrow$  [p = 1] (mod 4)
proof -
  have Legendre (- 1) p = 1  $\longleftrightarrow$  [Legendre (- 1) p = 1] (mod p)
  using assms
  by (metis Legendre-def QuadRes-def cong-0-iff cong-def not-prime-unit
one-power2)
  also have ...  $\longleftrightarrow$  [(- 1)  $\wedge$  ((nat p - 1) div 2) = 1] (mod p)
  using assms euler-criterion[of nat p - 1]
  by (smt (verit) cong-def nat-0-le nat-one-as-int of-nat-add one-add-one
prime-int-nat-transfer zless-nat-eq-int-zless)
  also have ...  $\longleftrightarrow$  ((- 1 :: int)  $\wedge$  ((nat p - 1) div 2)) = 1
  using assms
  by (simp add: cong-iff-dvd-diff minus-one-power-iff zdvd-not-zless)
  also have ...  $\longleftrightarrow$  even ((nat p - 1) div 2) by (simp add: minus-one-power-iff)
  also have ...  $\longleftrightarrow$  4 dvd (nat p - 1)
  using assms
  by (metis One-nat-def add-Suc-right div-dvd-div even-Suc even-diff-nat
even-numeral even-of-nat group-cancel.rule0 nat-0-le
numeral-Bit0-div-2 prime-int-nat-transfer prime-odd-int)
  also have ...  $\longleftrightarrow$  [p = 1] (mod 4)
  using assms
  unfolding cong-iff-dvd-diff int-dvd-int-iff[symmetric]
  by (simp add: int-ops)
  finally show ?thesis .
qed

lemma Legendre-minus-one-alt:
fixes p :: int
assumes prime p
assumes p > 2

```

```

shows Legendre (- 1) p = (if [p = 1] (mod 4) then 1 else - 1)
using assms Legendre-minus-one[OF assms]
unfolding Legendre-def cong-def
by (auto simp add: zmod-minus1)

lemma Legendre-two:
fixes p :: int
assumes prime p
assumes p > 2
shows Legendre 2 p = 1  $\longleftrightarrow$  [p = 1] (mod 8)  $\vee$  [p = 7] (mod 8)
proof -
let ?n = (p - 1) div 2 - (p - 1) div 4
have odd p using assms prime-odd-int by blast
hence 0: ( $\exists k. p = 8 * k + 1$ )  $\vee$  ( $\exists k. p = 8 * k + 3$ )  $\vee$ 
 $\quad$  ( $\exists k. p = 8 * k + 5$ )  $\vee$  ( $\exists k. p = 8 * k + 7$ )
by presburger
have 1: ( $j + 8 * k$ ) div 4 = j div 4 + 2 * k for k j :: int by linarith
have 2: GAUSS (nat p) 2
using assms unfolding GAUSS-def cong-def by auto
have Legendre 2 p = (- 1)  $\wedge$  card (GAUSS.E (nat p) 2)
using assms GAUSS.gauss-lemma[OF 2] by auto
also have ... = (- 1)  $\wedge$  card
(( $\lambda k. k \text{ mod } p$ ) ` (*) 2 ` {0 <..(p - 1) div 2}  $\cap$  {(p - 1) div 2 <..})
unfolding GAUSS.E-def[OF 2] GAUSS.C-def[OF 2]
GAUSS.B-def[OF 2] GAUSS.A-def[OF 2]
by (simp add: algebra-simps)
also have ... = (- 1)  $\wedge$  card
((*) 2 ` {0 <..(p - 1) div 2}  $\cap$  {(p - 1) div 2 <..})
by (rule arg-cong[of - -  $\lambda A. (- 1) \wedge \text{card } A$ ] [A  $\cap$  {(p - 1) div 2 <..}]; force)
also have ... = (- 1)  $\wedge$  card
{k  $\in$  (*) 2 ` {0 <..(p - 1) div 2}. k > (p - 1) div 2}
by (rule arg-cong[of - -  $\lambda A. (- 1) \wedge \text{card } A$ ]; blast)
also have ... = (- 1)  $\wedge$  card {k  $\in$  {0 <..(p - 1) div 2}. 2 * k > (p - 1) div 2}
apply (rule arg-cong[of - -  $\lambda n. (- 1) \wedge n$ ])
apply (rule card-bij-eq[where ?f =  $\lambda k. k \text{ div } 2$  and ?g = (*) 2])
subgoal unfolding inj-on-def by auto
subgoal by auto
subgoal by (simp add: inj-on-mult)
subgoal by auto
subgoal by (rule finite-Collect-conjI; auto)
subgoal by (rule finite-Collect-conjI; auto)
done
also have ... = (- 1)  $\wedge$  card {k  $\in$  {0 <..(p - 1) div 2}. k > (p - 1) div 4}
by (rule arg-cong[of - -  $\lambda f. (- 1) \wedge \text{card } f$ ] [k  $\in$  {0 <..(p - 1) div 2}. f k]);
auto
also have ... = (- 1)  $\wedge$  card {(p - 1) div 4 <..(p - 1) div 2}
by (rule arg-cong[of - -  $\lambda A. (- 1) \wedge \text{card } A$ ]; fastforce)
also have ... = (- 1)  $\wedge$  (nat ?n) by auto
finally have Legendre 2 p = 1  $\longleftrightarrow$  even ?n

```

```

unfolding minus-one-power-iff
  by (simp add: assms even-nat-iff prime-gt-0-int zdiv-mono2)
also have ...  $\longleftrightarrow [p = 1] \pmod{8} \vee [p = 7] \pmod{8}$  unfolding cong-def
  using 0 by (auto simp add: 1)
finally show ?thesis .
qed

lemma Legendre-two-alt:
  fixes p :: int
  assumes prime p
  assumes p > 2
  shows Legendre 2 p = (if [p = 1] (mod 8)  $\vee$  [p = 7] (mod 8) then 1 else -1)
  using assms Legendre-two[OF assms]
  unfolding Legendre-def cong-def
  by (auto simp add: zmod-minus1)

lemma Legendre-mult:
  fixes a b p :: int
  assumes prime p
  shows Legendre (a * b) p = Legendre a p * Legendre b p
proof cases
  assume 0: p = 2
  have 1: QuadRes p = ( $\lambda x$ . True) using 0
    by (metis QuadRes-def add-0 cong-iff-dvd-diff even-add odd-one power-one
          power-zero-numeral uminus-add-conv-diff)
  thus ?thesis using 0 unfolding 1 Legendre-def cong-0-iff by auto
next
  assume p ≠ 2
  hence 2: p > 2 using assms by (simp add: order-less-le prime-ge-2-int)
  have [Legendre a p = a  $\wedge$  ((nat p - 1) div 2)] (mod p)
    [Legendre b p = b  $\wedge$  ((nat p - 1) div 2)] (mod p)
    [Legendre (a * b) p = (a * b)  $\wedge$  ((nat p - 1) div 2)] (mod p)
  using 2 assms euler-criterion[of nat p a]
    euler-criterion[of nat p b]
    euler-criterion[of nat p a * b]
  by auto
  hence 3: [Legendre (a * b) p = Legendre a p * Legendre b p] (mod p)
    by (smt (verit, best) cong-mult cong-sym cong-trans power-mult-distrib)
  have 4: {Legendre a p, Legendre b p, Legendre (a * b) p} ⊆ {-1, 0, 1}
  unfolding Legendre-def by auto
  show ?thesis using 2 3 4 by (auto simp add: cong-iff-dvd-diff zdvd-not-zless)
qed

lemma Legendre-power:
  fixes a :: int
  fixes n :: nat
  fixes p :: int
  assumes prime p
  assumes p > 2

```

```

shows Legendre (a ^ n) p = (Legendre a p) ^ n
proof (induct n)
  case 0
  thus ?case using assms Legendre-one by auto
next
  case (Suc n)
  thus ?case using assms Legendre-mult by auto
qed

lemma Legendre-prod:
  fixes A :: 'a set
  fixes f :: 'a ⇒ int
  fixes p :: int
  assumes prime p
  assumes p > 2
  shows Legendre (prod f A) p = (Π x∈A. Legendre (f x) p)
proof (induct A rule: infinite-finite-induct)
  case (infinite A)
  thus ?case using assms Legendre-one by auto
next
  case empty
  thus ?case using assms Legendre-one by auto
next
  case (insert x F)
  thus ?case using assms Legendre-mult by auto
qed

lemma Legendre-equal:
  fixes p q :: int
  assumes prime p prime q
  assumes p > 2 q > 2
  assumes p ≠ q
  assumes [p = 1] (mod 4) ∨ [q = 1] (mod 4)
  shows Legendre p q = Legendre q p
proof -
  have 0: even (p - 1) even (q - 1) using assms prime-odd-int by auto
  have 1: ((p - 1) div 2) * ((q - 1) div 2) = (p - 1) * (q - 1) div 4
    using 0 by fastforce
  have 2: {Legendre p q, Legendre q p} ⊆ {-1, 0, 1}
    unfolding Legendre-def by auto
  have Legendre p q * Legendre q p =
    (- 1) ^ nat (((p - 1) div 2) * ((q - 1) div 2))
    using assms Quadratic-Reciprocity-int[of p q]
    by fastforce
  also have ... = (- 1) ^ nat ((p - 1) * (q - 1) div 4) unfolding 1 by rule
  also have ... = 1
    using 0 assms even-nat-iff
    unfolding minus-one-power-iff cong-iff-dvd-diff
    by auto

```

```

finally show ?thesis using 2 by auto
qed

lemma Legendre-opposite:
fixes p q :: int
assumes prime p prime q
assumes p > 2 q > 2
assumes p ≠ q
assumes [p = 3] (mod 4) ∧ [q = 3] (mod 4)
shows Legendre p q = - Legendre q p
proof -
have ⟨[p - 1 = 2] (mod 4)⟩ ⟨[q - 1 = 2] (mod 4)⟩
  using ⟨[p = 3] (mod 4) ∧ [q = 3] (mod 4)⟩
  by (simp-all add: cong-iff-dvd-diff)
from assms have ⟨even (p - 1)⟩ ⟨even (q - 1)⟩
  using assms by (auto dest: prime-odd-int)
then have *: (p - 1) * (q - 1) div 4 = ((p - 1) div 2) * ((q - 1) div 2)
  by fastforce
have odd ((p - 1) * (q - 1) div 4)
  using ⟨even (p - 1)⟩ ⟨even (q - 1)⟩ ⟨[p - 1 = 2] (mod 4)⟩ ⟨[q - 1 = 2] (mod 4)⟩
  by (auto elim!: oddE evenE dest: cong-dvd-iff simp add: *)
hence 2: odd (nat ((p - 1) * (q - 1) div 4))
  using assms even-nat-iff pos-imp-zdiv-nonneg-iff by fastforce
have 3: {Legendre p q, Legendre q p} ⊆ {-1, 0, 1}
  unfolding Legendre-def by auto
have Legendre p q * Legendre q p =
  (- 1) ^ nat (((p - 1) div 2) * ((q - 1) div 2))
  using assms Quadratic-Reciprocity-int[of p q]
  by fastforce
also have ... = (- 1) ^ nat ((p - 1) * (q - 1) div 4) unfolding * by rule
also have ... = - 1
  using 2 3
  unfolding minus-one-power-iff
  by auto
finally show ?thesis using 3 by auto
qed

end

```

## 2 Vectors and matrices, determinants and their properties in dimensions 2 and 3

```

theory Low-Dimensional-Linear-Algebra
imports HOL-Library.Adhoc-Overloading
begin

```

```

datatype vec2 =

```

```

datatype vec2 =
  vec2
  (vec21 : int)
  (vec22 : int)

datatype vec3 =
  vec3
  (vec31 : int)
  (vec32 : int)
  (vec33 : int)

datatype mat2 =
  mat2
  (mat211 : int) (mat212 : int)
  (mat221 : int) (mat222 : int)

datatype mat3 =
  mat3
  (mat311 : int) (mat312 : int) (mat313 : int)
  (mat321 : int) (mat322 : int) (mat323 : int)
  (mat331 : int) (mat332 : int) (mat333 : int)

instantiation vec2 :: ab-group-add
begin

definition zero-vec2 where
zero-vec2 =
  vec2
  0
  0

definition uminus-vec2 where
uminus-vec2 v =
  vec2
  (- vec21 v)
  (- vec22 v)

definition plus-vec2 where
plus-vec2 v1 v2 =
  vec2
  (vec21 v1 + vec21 v2)
  (vec22 v1 + vec22 v2)

definition minus-vec2 where
minus-vec2 v1 v2 =
  vec2
  (vec21 v1 - vec21 v2)
  (vec22 v1 - vec22 v2)

instance

```

```

apply intro-classes
unfolding zero-vec2-def uminus-vec2-def plus-vec2-def minus-vec2-def
apply simp-all
done

end

instantiation vec3 :: ab-group-add
begin

definition zero-vec3 where
zero-vec3 =
vec3
0
0
0

definition uminus-vec3 where
uminus-vec3 v =
vec3
(– vec31 v)
(– vec32 v)
(– vec33 v)

definition plus-vec3 where
plus-vec3 v1 v2 =
vec3
(vec31 v1 + vec31 v2)
(vec32 v1 + vec32 v2)
(vec33 v1 + vec33 v2)

definition minus-vec3 where
minus-vec3 v1 v2 =
vec3
(vec31 v1 – vec31 v2)
(vec32 v1 – vec32 v2)
(vec33 v1 – vec33 v2)

instance
apply intro-classes
unfolding zero-vec3-def uminus-vec3-def plus-vec3-def minus-vec3-def
apply simp-all
done

end

instantiation mat2 :: ring-1
begin

```

```

definition zero-mat2 where
zero-mat2 =
  mat2
  0 0
  0 0

definition one-mat2 where
one-mat2 =
  mat2
  1 0
  0 1

definition uminus-mat2 where
uminus-mat2 m =
  mat2
  (- mat211 m) (- mat212 m)
  (- mat221 m) (- mat222 m)

definition plus-mat2 where
plus-mat2 m1 m2 =
  mat2
  (mat211 m1 + mat211 m2) (mat212 m1 + mat212 m2)
  (mat221 m1 + mat221 m2) (mat222 m1 + mat222 m2)

definition minus-mat2 where
minus-mat2 m1 m2 =
  mat2
  (mat211 m1 - mat211 m2) (mat212 m1 - mat212 m2)
  (mat221 m1 - mat221 m2) (mat222 m1 - mat222 m2)

definition times-mat2 where
times-mat2 m1 m2 =
  mat2
  (mat211 m1 * mat211 m2 + mat212 m1 * mat221 m2) (mat211 m1 * mat212
m2 + mat212 m1 * mat222 m2)
  (mat221 m1 * mat211 m2 + mat222 m1 * mat221 m2) (mat221 m1 * mat212
m2 + mat222 m1 * mat222 m2)

instance
  apply intro-classes
  unfolding zero-mat2-def one-mat2-def uminus-mat2-def plus-mat2-def minus-mat2-def
  times-mat2-def
  apply (simp-all add: algebra-simps)
  done

end

instantiation mat3 :: ring-1
begin

```

```

definition zero-mat3 where
zero-mat3 =
mat3
0 0 0
0 0 0
0 0 0

definition one-mat3 where
one-mat3 =
mat3
1 0 0
0 1 0
0 0 1

definition uminus-mat3 where
uminus-mat3 m =
mat3
(- mat311 m) (- mat312 m) (- mat313 m)
(- mat321 m) (- mat322 m) (- mat323 m)
(- mat331 m) (- mat332 m) (- mat333 m)

definition plus-mat3 where
plus-mat3 m1 m2 =
mat3
(mat311 m1 + mat311 m2) (mat312 m1 + mat312 m2) (mat313 m1 + mat313 m2)
(mat321 m1 + mat321 m2) (mat322 m1 + mat322 m2) (mat323 m1 + mat323 m2)
(mat331 m1 + mat331 m2) (mat332 m1 + mat332 m2) (mat333 m1 + mat333 m2)

definition minus-mat3 where
minus-mat3 m1 m2 =
mat3
(mat311 m1 - mat311 m2) (mat312 m1 - mat312 m2) (mat313 m1 - mat313 m2)
(mat321 m1 - mat321 m2) (mat322 m1 - mat322 m2) (mat323 m1 - mat323 m2)
(mat331 m1 - mat331 m2) (mat332 m1 - mat332 m2) (mat333 m1 - mat333 m2)

definition times-mat3 where
times-mat3 m1 m2 =
mat3
(mat311 m1 * mat311 m2 + mat312 m1 * mat321 m2 + mat313 m1 * mat331 m2)
(mat311 m1 * mat312 m2 + mat312 m1 * mat322 m2 + mat313 m1 * mat332 m2)
(mat311 m1 * mat313 m2 + mat312 m1 * mat323 m2 + mat313 m1 * mat333 m2)

```

```


$$(mat3_{21} m1 * mat3_{11} m2 + mat3_{22} m1 * mat3_{21} m2 + mat3_{23} m1 * mat3_{31} m2) (mat3_{21} m1 * mat3_{12} m2 + mat3_{22} m1 * mat3_{22} m2 + mat3_{23} m1 * mat3_{32} m2) (mat3_{21} m1 * mat3_{13} m2 + mat3_{22} m1 * mat3_{23} m2 + mat3_{23} m1 * mat3_{33} m2)$$


$$(mat3_{31} m1 * mat3_{11} m2 + mat3_{32} m1 * mat3_{21} m2 + mat3_{33} m1 * mat3_{31} m2) (mat3_{31} m1 * mat3_{12} m2 + mat3_{32} m1 * mat3_{22} m2 + mat3_{33} m1 * mat3_{32} m2) (mat3_{31} m1 * mat3_{13} m2 + mat3_{32} m1 * mat3_{23} m2 + mat3_{33} m1 * mat3_{33} m2)$$


instance
  apply intro-classes
  unfolding zero-mat3-def one-mat3-def uminus-mat3-def plus-mat3-def minus-mat3-def times-mat3-def
  apply (simp-all add: algebra-simps)
  done

end

consts vec-dot :: 'a  $\Rightarrow$  'a  $\Rightarrow$  int (<- | -> 65)

definition vec2-dot :: vec2  $\Rightarrow$  vec2  $\Rightarrow$  int where
  vec2-dot v1 v2 = vec2_1 v1 * vec2_1 v2 + vec2_2 v1 * vec2_2 v2

adhoc-overloading vec-dot vec2-dot

definition vec3-dot :: vec3  $\Rightarrow$  vec3  $\Rightarrow$  int where
  vec3-dot v1 v2 = vec3_1 v1 * vec3_1 v2 + vec3_2 v1 * vec3_2 v2 + vec3_3 v1 * vec3_3 v2

adhoc-overloading vec-dot vec3-dot

lemma vec2-dot-zero-left [simp]:
  fixes v :: vec2
  shows <0 | v> = 0
  unfolding vec2-dot-def zero-vec2-def by auto

lemma vec2-dot-zero-right [simp]:
  fixes v :: vec2
  shows <v | 0> = 0
  unfolding vec2-dot-def zero-vec2-def by auto

lemma vec3-dot-zero-left [simp]:
  fixes v :: vec3
  shows <0 | v> = 0
  unfolding vec3-dot-def zero-vec3-def by auto

lemma vec3-dot-zero-right [simp]:
  fixes v :: vec3
  shows <v | 0> = 0

```

```

unfolding vec3-dot-def zero-vec3-def by auto

consts mat-app :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'b (infixr $ 65)

definition mat2-app :: mat2  $\Rightarrow$  vec2  $\Rightarrow$  vec2 where
mat2-app m v =
  vec2
  (mat211 m * vec21 v + mat212 m * vec22 v)
  (mat221 m * vec21 v + mat222 m * vec22 v)

adhoc-overloading mat-app mat2-app

definition mat3-app :: mat3  $\Rightarrow$  vec3  $\Rightarrow$  vec3 where
mat3-app m v =
  vec3
  (mat311 m * vec31 v + mat312 m * vec32 v + mat313 m * vec33 v)
  (mat321 m * vec31 v + mat322 m * vec32 v + mat323 m * vec33 v)
  (mat331 m * vec31 v + mat332 m * vec32 v + mat333 m * vec33 v)

adhoc-overloading mat-app mat3-app

lemma mat2-app-zero [simp]:
  fixes m :: mat2
  shows m $ 0 = 0
  unfolding mat2-app-def zero-vec2-def by auto

lemma mat3-app-zero [simp]:
  fixes m :: mat3
  shows m $ 0 = 0
  unfolding mat3-app-def zero-vec3-def by auto

lemma mat2-app-one [simp]:
  fixes v :: vec2
  shows 1 $ v = v
  unfolding mat2-app-def one-mat2-def by auto

lemma mat3-app-one [simp]:
  fixes v :: vec3
  shows 1 $ v = v
  unfolding mat3-app-def one-mat3-def by auto

lemma mat2-app-mul [simp]:
  fixes m1 m2 :: mat2
  fixes v :: vec2
  shows m1 * m2 $ v = m1 $ m2 $ v
  unfolding times-mat2-def mat2-app-def by (simp add: algebra-simps)

lemma mat3-app-mul [simp]:
  fixes m1 m2 :: mat3

```

```

fixes v :: vec3
shows m1 * m2 \$ v = m1 \$ m2 \$ v
unfolding times-mat3-def mat3-app-def by (simp add: algebra-simps)

consts mat-det :: 'a ⇒ int

definition mat2-det where
mat2-det m = mat211 m * mat222 m - mat212 m * mat221 m

adhoc-overloading mat-det mat2-det

definition mat3-det where
mat3-det m =
  mat311 m * mat322 m * mat333 m
+ mat312 m * mat323 m * mat331 m
+ mat313 m * mat321 m * mat332 m
- mat311 m * mat323 m * mat332 m
- mat312 m * mat321 m * mat333 m
- mat313 m * mat322 m * mat331 m

adhoc-overloading mat-det mat3-det

lemma mat2-mul-det [simp]:
fixes m1 m2 :: mat2
shows mat-det (m1 * m2) = mat-det m1 * mat-det m2
unfolding times-mat2-def mat2-det-def by (simp; algebra)

lemma mat3-mul-det [simp]:
fixes m1 m2 :: mat3
shows mat-det (m1 * m2) = mat-det m1 * mat-det m2
unfolding times-mat3-def mat3-det-def by (simp; algebra)

consts mat-sym :: 'a ⇒ bool

definition mat2-sym :: mat2 ⇒ bool where
mat2-sym m = (mat212 m = mat221 m)

adhoc-overloading mat-sym mat2-sym

definition mat3-sym :: mat3 ⇒ bool where
mat3-sym m = (mat312 m = mat321 m ∧ mat313 m = mat331 m ∧ mat323 m =
mat332 m)

adhoc-overloading mat-sym mat3-sym

consts mat-transpose :: 'a ⇒ 'a (‐T [91] 90)

definition mat2-transpose :: mat2 ⇒ mat2 where
mat2-transpose m =

```

```

mat2
(mat211 m) (mat221 m)
(mat212 m) (mat222 m)

adhoc-overloading mat-transpose mat2-transpose

definition mat3-transpose :: mat3  $\Rightarrow$  mat3 where
mat3-transpose m =
mat3
(mat311 m) (mat321 m) (mat331 m)
(mat312 m) (mat322 m) (mat332 m)
(mat313 m) (mat323 m) (mat333 m)

adhoc-overloading mat-transpose mat3-transpose

lemma mat2-transpose-involution [simp]:
fixes m :: mat2
shows (mT)T = m
unfolding mat2-transpose-def
by auto

lemma mat3-transpose-involution [simp]:
fixes m :: mat3
shows (mT)T = m
unfolding mat3-transpose-def
by auto

lemma mat2-sym-criterion:
fixes m :: mat2
shows mat-sym m  $\longleftrightarrow$  mT = m
unfolding mat2-sym-def mat2-transpose-def
by (cases m; auto)

lemma mat3-sym-criterion:
fixes m :: mat3
shows mat-sym m  $\longleftrightarrow$  mT = m
unfolding mat3-sym-def mat3-transpose-def
by (cases m; auto)

lemma mat2-transpose-one [simp]: (1 :: mat2)T = 1
unfolding mat2-transpose-def one-mat2-def by auto

lemma mat3-transpose-one [simp]: (1 :: mat3)T = 1
unfolding mat3-transpose-def one-mat3-def by auto

lemma mat2-transpose-mul [simp]:
fixes a b :: mat2
shows (a * b)T = bT * aT
unfolding mat2-transpose-def times-mat2-def by auto

```

```

lemma mat3-transpose-mul [simp]:
  fixes a b :: mat3
  shows (a * b)T = bT * aT
  unfolding mat3-transpose-def times-mat3-def by auto

lemma vec2-dot transpose-left:
  fixes m :: mat2
  fixes u v :: vec2
  shows <mT $ u | v> = <u | m $ v>
  unfolding vec2-dot-def mat2-app-def mat2-transpose-def
  by (simp add: algebra-simps)

lemma vec2-dot transpose-right:
  fixes m :: mat2
  fixes u v :: vec2
  shows <u | mT $ v> = <m $ u | v>
  unfolding vec2-dot-def mat2-app-def mat2-transpose-def
  by (simp add: algebra-simps)

lemma vec3-dot transpose-left:
  fixes m :: mat3
  fixes u v :: vec3
  shows <mT $ u | v> = <u | m $ v>
  unfolding vec3-dot-def mat3-app-def mat3-transpose-def
  by (simp add: algebra-simps)

lemma vec3-dot transpose-right:
  fixes m :: mat3
  fixes u v :: vec3
  shows <u | mT $ v> = <m $ u | v>
  unfolding vec3-dot-def mat3-app-def mat3-transpose-def
  by (simp add: algebra-simps)

lemma mat2-det-tranpose [simp]:
  fixes m :: mat2
  shows mat-det (mT) = mat-det m
  unfolding mat2-det-def mat2-transpose-def by auto

lemma mat3-det-tranpose [simp]:
  fixes m :: mat3
  shows mat-det (mT) = mat-det m
  unfolding mat3-det-def mat3-transpose-def by auto

consts mat-inverse :: 'a ⇒ 'a (⁻¹ [91] 90)

definition mat2-inverse :: mat2 ⇒ mat2 where
mat2-inverse m =
  mat2

```

$$(mat2_{22} m) (- mat2_{12} m) \\ (- mat2_{21} m) (mat2_{11} m)$$

**adhoc-overloading** *mat-inverse mat2-inverse*

```
definition mat3-inverse :: mat3  $\Rightarrow$  mat3 where
mat3-inverse m =
  mat3
    (mat3_{22} m * mat3_{33} m - mat3_{23} m * mat3_{32} m) (mat3_{13} m * mat3_{32} m -
    mat3_{12} m * mat3_{33} m) (mat3_{12} m * mat3_{23} m - mat3_{13} m * mat3_{22} m)
    (mat3_{23} m * mat3_{31} m - mat3_{21} m * mat3_{33} m) (mat3_{11} m * mat3_{33} m -
    mat3_{13} m * mat3_{31} m) (mat3_{13} m * mat3_{21} m - mat3_{11} m * mat3_{23} m)
    (mat3_{21} m * mat3_{32} m - mat3_{22} m * mat3_{31} m) (mat3_{12} m * mat3_{31} m -
    mat3_{11} m * mat3_{32} m) (mat3_{11} m * mat3_{22} m - mat3_{12} m * mat3_{21} m)
```

**adhoc-overloading** *mat-inverse mat3-inverse*

```
lemma mat2-inverse-cancel:
  fixes m :: mat2
  assumes mat-det m = 1
  shows m * m-1 = 1 m-1 * m = 1
  using assms unfolding mat2-det-def mat2-inverse-def times-mat2-def one-mat2-def
  by (auto simp add: algebra-simps)
```

```
lemma mat3-inverse-cancel:
  fixes m :: mat3
  assumes mat-det m = 1
  shows m * m-1 = 1 m-1 * m = 1
  using assms unfolding mat3-det-def mat3-inverse-def times-mat3-def one-mat3-def
  by (auto simp add: algebra-simps)
```

```
lemma mat2-inverse-cancel-left:
  fixes m a :: mat2
  assumes mat-det m = 1
  shows m * (m-1 * a) = a m-1 * (m * a) = a
  unfolding mult.assoc[symmetric]
  using assms mat2-inverse-cancel
  by auto
```

```
lemma mat3-inverse-cancel-left:
  fixes m a :: mat3
  assumes mat-det m = 1
  shows m * (m-1 * a) = a m-1 * (m * a) = a
  unfolding mult.assoc[symmetric]
  using assms mat3-inverse-cancel
  by auto
```

```

lemma mat2-inverse-cancel-right:
  fixes m a :: mat2
  assumes mat-det m = 1
  shows a * (m * m-1) = a a * (m-1 * m) = a
  using assms mat2-inverse-cancel
  by auto

lemma mat3-inverse-cancel-right:
  fixes m a :: mat3
  assumes mat-det m = 1
  shows a * (m * m-1) = a a * (m-1 * m) = a
  using assms mat3-inverse-cancel
  by auto

lemma mat2-inversable-cancel-left:
  fixes m a1 a2 :: mat2
  assumes mat-det m = 1
  assumes m * a1 = m * a2
  shows a1 = a2
  by (metis assms mat2-inverse-cancel-left(2))

lemma mat3-inversable-cancel-left:
  fixes m a1 a2 :: mat3
  assumes mat-det m = 1
  assumes m * a1 = m * a2
  shows a1 = a2
  by (metis assms mat3-inverse-cancel-left(2))

lemma mat2-inversable-cancel-right:
  fixes m a1 a2 :: mat2
  assumes mat-det m = 1
  assumes a1 * m = a2 * m
  shows a1 = a2
  by (metis assms mat2-inverse-cancel(1) mult.assoc mult.right-neutral)

lemma mat3-inversable-cancel-right:
  fixes m a1 a2 :: mat3
  assumes mat-det m = 1
  assumes a1 * m = a2 * m
  shows a1 = a2
  by (metis assms mat3-inverse-cancel(1) mult.assoc mult.right-neutral)

lemma mat2-inverse-det [simp]:
  fixes m :: mat2
  shows mat-det (m-1) = mat-det m
  unfolding mat2-inverse-def mat2-det-def
  by auto

lemma mat3-inverse-det [simp]:

```

```

fixes m :: mat3
shows mat-det (m-1) = (mat-det m)2
unfolding mat3-inverse-def mat3-det-def power2-eq-square
by (simp add: algebra-simps)

lemma mat2-inverse-transpose:
fixes m :: mat2
shows (mT)-1 = (m-1)T
unfolding mat2-inverse-def mat2-transpose-def
by auto

lemma mat3-inverse-transpose:
fixes m :: mat3
shows (mT)-1 = (m-1)T
unfolding mat3-inverse-def mat3-transpose-def
by auto

lemma mat2-special-preserves-zero:
fixes u :: mat2
fixes v :: vec2
assumes mat-det u = 1
shows u $ v = 0  $\longleftrightarrow$  v = 0
proof
  assume u $ v = 0
  hence u-1 $ u $ v = 0 by auto
  hence (u-1 * u) $ v = 0 by auto
  thus v = 0 using assms mat2-inverse-cancel by auto
next
  assume v = 0
  thus u $ v = 0 by auto
qed

lemma mat3-special-preserves-zero:
fixes u :: mat3
fixes v :: vec3
assumes mat-det u = 1
shows u $ v = 0  $\longleftrightarrow$  v = 0
proof
  assume u $ v = 0
  hence u-1 $ u $ v = 0 by auto
  hence (u-1 * u) $ v = 0 by auto
  thus v = 0 using assms mat3-inverse-cancel by auto
next
  assume v = 0
  thus u $ v = 0 by auto
qed

end

```

### 3 Properties of quadratic forms and their equivalences

```

theory Quadratic-Forms
imports Complex-Main Low-Dimensional-Linear-Algebra
begin

consts qf-app :: 'a ⇒ 'b ⇒ int (infixl §§ 65)

definition qf2-app :: mat2 ⇒ vec2 ⇒ int where
qf2-app m v = <v | m $ v>

adhoc-overloading qf-app qf2-app

definition qf3-app :: mat3 ⇒ vec3 ⇒ int where
qf3-app m v = <v | m $ v>

adhoc-overloading qf-app qf3-app

lemma qf2-app-zero [simp]:
fixes m :: mat2
shows m §§ 0 = 0
unfolding qf2-app-def by auto

lemma qf3-app-zero [simp]:
fixes m :: mat3
shows m §§ 0 = 0
unfolding qf3-app-def by auto

consts qf-positive-definite :: 'a ⇒ bool

definition qf2-positive-definite :: mat2 ⇒ bool where
qf2-positive-definite m = ( ∀ v. v ≠ 0 → m §§ v > 0 )

adhoc-overloading qf-positive-definite qf2-positive-definite

definition qf3-positive-definite :: mat3 ⇒ bool where
qf3-positive-definite m = ( ∀ v. v ≠ 0 → m §§ v > 0 )

adhoc-overloading qf-positive-definite qf3-positive-definite

lemma qf2-positive-definite-positive:
fixes m :: mat2
assumes qf-positive-definite m
shows ∀ v. m §§ v ≥ 0
using assms unfolding qf2-positive-definite-def
by (metis order-less-le order-refl qf2-app-zero)

lemma qf3-positive-definite-positive:

```

```

fixes m :: mat3
assumes qf-positive-definite m
shows  $\forall v. m \circledast v \geq 0$ 
using assms unfolding qf3-positive-definite-def
by (metis order-less-le order-refl qf3-app-zero)

consts qf-action :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl · 55)

definition qf2-action :: mat2  $\Rightarrow$  mat2  $\Rightarrow$  mat2 where
qf2-action a u =  $u^T * a * u$ 

adhoc-overloading qf-action qf2-action

definition qf3-action :: mat3  $\Rightarrow$  mat3  $\Rightarrow$  mat3 where
qf3-action a u =  $u^T * a * u$ 

adhoc-overloading qf-action qf3-action

lemma qf2-action-id:
fixes a :: mat2
shows a · 1 = a
unfolding qf2-action-def
by simp

lemma qf3-action-id:
fixes a :: mat3
shows a · 1 = a
unfolding qf3-action-def
by simp

lemma qf2-action-mul [simp]:
fixes a u v :: mat2
shows a · (u * v) = (a · u) · v
unfolding qf2-action-def
by (simp add: algebra-simps)

lemma qf3-action-mul [simp]:
fixes a u v :: mat3
shows a · (u * v) = (a · u) · v
unfolding qf3-action-def
by (simp add: algebra-simps)

consts qf-equiv :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infix ~ 65)

definition qf2-equiv :: mat2  $\Rightarrow$  mat2  $\Rightarrow$  bool where
qf2-equiv a b = ( $\exists u. \text{mat-det } u = 1 \wedge a \cdot u = b$ )

adhoc-overloading qf-equiv qf2-equiv

```

```

definition qf3-equiv :: mat3 ⇒ mat3 ⇒ bool where
qf3-equiv a b = (exists u. mat-det u = 1 ∧ a · u = b)

adhoc-overloading qf-equiv qf3-equiv

lemma qf2-equiv-sym-impl:
  fixes a b :: mat2
  shows a ~ b ⟹ b ~ a
  unfolding qf2-equiv-def qf2-action-def
  proof -
    assume ∃ u. mat-det u = 1 ∧ uT * a * u = b
    then obtain u where mat-det u = 1 ∧ uT * a * u = b by blast
    hence mat-det (u-1) = 1 ∧ ((u-1)T) * b * (u-1) = a
    unfolding mat2-inverse-transpose[symmetric]
    using mat2-inverse-cancel-left[of uT] mat2-inverse-cancel-right
    by (auto simp add: algebra-simps)
    thus ∃ u. mat-det u = 1 ∧ uT * b * u = a by blast
  qed

lemma qf3-equiv-sym-impl:
  fixes a b :: mat3
  shows a ~ b ⟹ b ~ a
  unfolding qf3-equiv-def qf3-action-def
  proof -
    assume ∃ u. mat-det u = 1 ∧ uT * a * u = b
    then obtain u where mat-det u = 1 ∧ uT * a * u = b by blast
    hence mat-det (u-1) = 1 ∧ ((u-1)T) * b * (u-1) = a
    unfolding mat3-inverse-transpose[symmetric]
    using mat3-inverse-cancel-left[of uT] mat3-inverse-cancel-right
    by (auto simp add: algebra-simps)
    thus ∃ u. mat-det u = 1 ∧ uT * b * u = a by blast
  qed

lemma qf2-equiv-sym:
  fixes a b :: mat2
  shows a ~ b ⟷ b ~ a
  using qf2-equiv-sym-impl by blast

lemma qf3-equiv-sym:
  fixes a b :: mat3
  shows a ~ b ⟷ b ~ a
  using qf3-equiv-sym-impl by blast

lemma qf2-equiv-trans:
  fixes a b c :: mat2
  assumes a ~ b
  assumes b ~ c
  shows a ~ c
  using assms by (metis mult-1 mat2-mul-det qf2-action-mul qf2-equiv-def)

```

```

lemma qf3-equiv-trans:
  fixes a b c :: mat3
  assumes a ~ b
  assumes b ~ c
  shows a ~ c
  using assms by (metis mult-1 mat3-mul-det qf3-action-mul qf3-equiv-def)

lemma qf2-action-app [simp]:
  fixes a u :: mat2
  fixes v :: vec2
  shows (a · u) $$ v = a $$ (u $ v)
  unfolding qf2-action-def qf2-app-def
  using vec2-dot transpose-right by auto

lemma qf3-action-app [simp]:
  fixes a u :: mat3
  fixes v :: vec3
  shows (a · u) $$ v = a $$ (u $ v)
  unfolding qf3-action-def qf3-app-def
  using vec3-dot transpose-right by auto

lemma qf2-equiv-preserves-positive-definite:
  fixes a b :: mat2
  assumes a ~ b
  shows qf-positive-definite a  $\longleftrightarrow$  qf-positive-definite b
  unfolding qf2-positive-definite-def
  by (metis assms mat2-special-preserves-zero qf2-action-app
      qf2-equiv-def qf2-equiv-sym)

lemma qf3-equiv-preserves-positive-definite:
  fixes a b :: mat3
  assumes a ~ b
  shows qf-positive-definite a  $\longleftrightarrow$  qf-positive-definite b
  unfolding qf3-positive-definite-def
  by (metis assms mat3-special-preserves-zero qf3-action-app
      qf3-equiv-def qf3-equiv-sym)

lemma qf2-equiv-preserves-sym:
  fixes a b :: mat2
  assumes a ~ b
  shows mat2-sym a  $\longleftrightarrow$  mat2-sym b
proof -
  obtain u where mat-det u = 1  $u^T * a * u = b$ 
  using assms unfolding qf2-action-def qf2-equiv-def by auto
  thus ?thesis
    unfolding mat2-sym-criterion
    using mat2-inversible-cancel-left[of  $u^T a^T * u a * u$ ]
          mat2-inversible-cancel-right[of  $u a^T a$ ]

```

```

    by (auto simp add: algebra-simps)
qed

lemma qf3-equiv-preserves-sym:
  fixes a b :: mat3
  assumes a ~ b
  shows mat3-sym a  $\longleftrightarrow$  mat3-sym b
proof -
  obtain u where mat-det u = 1  $u^T * a * u = b$ 
  using assms unfolding qf3-action-def qf3-equiv-def by auto
  thus ?thesis
    unfolding mat3-sym-criterion
    using mat3-inversible-cancel-left[of  $u^T a^T * u$  a * u]
      mat3-inversible-cancel-right[of u a^T a]
    by (auto simp add: algebra-simps)
qed

lemma qf2-equiv-preserves-det:
  fixes a b :: mat2
  assumes a ~ b
  shows mat-det a = mat-det b
  using assms unfolding qf2-action-def qf2-equiv-def by auto

lemma qf3-equiv-preserves-det:
  fixes a b :: mat3
  assumes a ~ b
  shows mat-det a = mat-det b
  using assms unfolding qf3-action-def qf3-equiv-def by auto

lemma qf2-equiv-preserves-range-subset:
  fixes a b :: mat2
  assumes a ~ b
  shows range (( $\lambda$ ) b)  $\subseteq$  range (( $\lambda$ ) a)
proof -
  obtain u where 0: mat-det u = 1 a · u = b
  using assms unfolding qf2-equiv-def by auto
  show ?thesis unfolding 0[symmetric] image-def by auto
qed

lemma qf3-equiv-preserves-range-subset:
  fixes a b :: mat3
  assumes a ~ b
  shows range (( $\lambda$ ) b)  $\subseteq$  range (( $\lambda$ ) a)
proof -
  obtain u where 0: mat-det u = 1 a · u = b
  using assms unfolding qf3-equiv-def by auto
  show ?thesis unfolding 0[symmetric] image-def by auto
qed

```

```

lemma qf2-equiv-preserves-range:
  fixes a b :: mat2
  assumes a ~ b
  shows range (((\$\$) a) = range (((\$\$) b)
  using assms qf2-equiv-sym qf2-equiv-preserves-range-subset by blast

```

```

lemma qf3-equiv-preserves-range:
  fixes a b :: mat3
  assumes a ~ b
  shows range (((\$\$) a) = range (((\$\$) b)
  using assms qf3-equiv-sym qf3-equiv-preserves-range-subset by blast

```

Lemma 1.1 from [1].

```

lemma qf2-positive-definite-criterion:
  fixes a
  assumes mat-sym a
  shows qf-positive-definite a <--> mat2_11 a > 0 ∧ mat-det a > 0
proof
  assume 0: qf-positive-definite a
  have vec2 1 0 ≠ 0 → a \$\$ vec2 1 0 > 0 using 0
    unfolding qf2-positive-definite-def by blast
  hence 1: mat2_11 a > 0
    unfolding zero-vec2-def vec2-dot-def times-mat2-def mat2-app-def qf2-app-def
    by auto
  let ?v = vec2 (- mat2_12 a) (mat2_11 a)
  have ?v ≠ 0 → a \$\$ ?v > 0 using 0
    unfolding qf2-positive-definite-def by blast
  hence mat2_11 a * mat-det a > 0 using assms 1
    unfolding zero-vec2-def vec2-dot-def times-mat2-def mat2-app-def
      mat2-det-def mat2-sym-def qf2-app-def
    by (simp add: mult.commute)
  hence 2: mat-det a > 0 using 1 zero-less-mult-pos by blast
  show mat2_11 a > 0 ∧ mat-det a > 0 using 1 2 by blast
next
  assume 3: mat2_11 a > 0 ∧ mat-det a > 0
  show qf-positive-definite a unfolding qf2-positive-definite-def
  proof (rule allI; rule impI)
    fix v :: vec2
    assume v ≠ 0
    hence 4: vec2_1 v ≠ 0 ∨ vec2_2 v ≠ 0 unfolding zero-vec2-def
      by (metis vec2.collapse)
    let ?n = (mat2_11 a * vec2_1 v + mat2_12 a * vec2_2 v)^2 + (mat-det a) * (vec2_2 v)^2
    have 5: mat2_11 a * (a \$\$ v) = ?n
      using assms
      unfolding vec2-dot-def times-mat2-def mat2-app-def mat2-det-def
        mat2-sym-def qf2-app-def power2-eq-square
      by (simp add: algebra-simps)
    have ?n > 0 using 3 4

```

```

by (metis add.commute add-0 add-pos-nonneg mult-eq-0-iff
      mult-pos-pos power-zero-numeral zero-le-power2 zero-less-power2)
thus a $$ v > 0 using 3 5 zero-less-mult-pos by metis
qed
qed

lemma congruence-class-close:
fixes k m :: int
assumes m > 0
shows ∃ t. 2 * |k + m * t| ≤ m (is ∃ t. ?P t)
proof -
let ?s = k div m
have 0: k - m * ?s ≥ 0 ∧ k - m * ?s < m using assms
by (metis pos-mod-sign pos-mod-bound add.commute add-diff-cancel-right'
      div-mod-decomp-int mult.commute)
show ?thesis proof cases
assume 2 * (k - m * ?s) ≤ m
hence ?P (- ?s) using 0 by auto
thus ?thesis by blast
next
assume ¬ (2 * (k - m * ?s) ≤ m)
hence 2 * (k - m * ?s) > m by auto
hence ?P (- (?s + 1)) using 0 by (simp add: algebra-simps)
thus ?thesis by blast
qed
qed

```

Lemma 1.2 from [1].

```

lemma lemma-1-2:
fixes b :: mat2
assumes mat-sym b
assumes qf-positive-definite b
shows ∃ a. a ~ b ∧
      2 * |mat2_12 a| ≤ mat2_11 a ∧
      mat2_11 a ≤ (2 / sqrt 3) * sqrt (mat-det a) (is ∃ a. ?P a)
proof -
define a11 where a11 ≡ LEAST y. y > 0 ∧ (∃ x. int y = b $$ x)
have 0: ∃ y. y > 0 ∧ (∃ x. int y = b $$ x)
apply (rule exI[of - nat (b $$ (vec2 1 1))])
using assms(2) unfolding qf2-positive-definite-def zero-vec2-def
apply (metis nat-0-le order-less-le vec2.inject zero-less-nat-eq zero-neq-one)
done
obtain r where 1: a11 > 0 int a11 = b $$ r
  using a11-def LeastI-ex[OF 0] by auto
let ?h = gcd (vec21 r) (vec22 r)
have r ≠ 0 using assms(2) 1 by fastforce
hence 2: ?h > 0 by (simp; metis vec2.collapse zero-vec2-def)
let ?r' = vec2 (vec21 r div ?h) (vec22 r div ?h)
have ?r' ≠ 0 using 2 unfolding zero-vec2-def

```

```

by (simp add: algebra-simps dvd-div-eq-0-iff)
hence nat (b $$ ?r') > 0 ∧ (∃ x. int (nat (b $$ ?r')) = b $$ x)
  using assms(2) unfolding qf2-positive-definite-def by auto
hence *: a11 ≤ nat (b $$ ?r') unfolding a11-def by (rule Least-le)
hence a11 ≤ b $$ ?r' using 1 by auto
also have **: ... = (b $$ r) div ?h2 proof -
  have (b $$ ?r') * ?h2 = b $$ r ?h2 dvd b $$ r
    unfolding qf2-app-def vec2-dot-def mat2-app-def power2-eq-square
    using 1
    by (auto simp add: algebra-simps mult-dvd-mono)
    thus b $$ ?r' = (b $$ r) div ?h2 using 2 by auto
qed
also have ... = a11 div ?h2 using 1 by auto
finally have a11 ≤ a11 div ?h2.
also have ... ≤ a11 using 1 2
  by (metis * ** assms(2) div-le-dividend int-eq-iff nat-div-distrib order-refl
      qf2-positive-definite-positive verit-la-disequality)
finally have ?h = 1 using 1 2
  by (smt (verit) int-div-less-self of-nat-0-less-iff power2-eq-square
      zero-less-power zmult-eq-1-iff)
then obtain s1 s2 where 3: 1 = (vec21 r) * s2 - (vec22 r) * s1
  by (metis bezout-int diff-minus-eq-add mult.commute mult-minus-right)
define a'12 where
  a'12 ≡
    (mat211 b) * (vec21 r) * s1
    + (mat212 b) * ((vec21 r) * s2 + (vec22 r) * s1)
    + (mat222 b) * (vec22 r) * s2

obtain t where 4: 2 * |a'12 + a11 * (t :: int)| ≤ a11
  using 1 congruence-class-close by fastforce
define a12 where a12 ≡ a'12 + a11 * t
define a22 where a22 ≡ b $$ (vec2 (s1 + (vec21 r) * t) (s2 + (vec22 r) * t))
let ?u =
  mat2
    (vec21 r) (s1 + (vec21 r) * t)
    (vec22 r) (s2 + (vec22 r) * t)

let ?a =
  mat2
    a11 a12
    a12 a22

have 5: mat-det ?u = 1 unfolding mat2-det-def
  using 3 by (simp add: algebra-simps)
have 6: ?a = b · ?u using assms(1) 1
  unfolding qf2-action-def mat2 transpose-def qf2-app-def vec2-dot-def
    mat2-app-def times-mat2-def mat2-sym-def
    a12-def a12-def a22-def a'12-def
  by (simp add: algebra-simps)

```

```

have  $b \sim ?a$  unfolding qf2-equiv-def using 5 6 by auto
hence 7:  $?a \sim b$  using qf2-equiv-sym by blast
have 8:  $2 * |\text{mat2}_{12} ?a| \leq \text{mat2}_{11} ?a$  using 4 unfolding a12-def by auto
have  $a_{11} \leq \text{int}(\text{nat } a_{22})$ 
  unfolding a11-def a22-def
  apply (rule rev-iffD1[OF - nat-int-comparison(3)])
  apply (rule wellorder-Least-lemma(2))
  using assms(2) 5 unfolding qf2-positive-definite-def zero-vec2-def mat2-det-def
  apply (metis add.right-neutral diff-add-cancel mat2.sel
        mult-zero-left mult-zero-right nat-0-le order-less-le vec2.inject
        zero-less-nat-eq zero-neq-one)
done
hence  $a_{11} \leq a_{22}$  using 1 by linarith
hence  $4 * a_{11}^2 \leq 4 * a_{11} * a_{22}$  unfolding power2-eq-square using 1 by auto
also have ... =  $4 * (\text{mat-det } ?a + a_{12}^2)$ 
  unfolding mat2-det-def power2-eq-square by auto
also have ... =  $4 * \text{mat-det } ?a + (2 * |a_{12}|)^2$ 
  unfolding power2-eq-square by auto
also have ...  $\leq 4 * \text{mat-det } ?a + (\text{int } a_{11})^2$ 
  using 4 power2-le-iff-abs-le unfolding a12-def by (smt (verit))
finally have  $3 * a_{11}^2 \leq 4 * (\text{mat-det } ?a)$  by auto
hence  $a_{11}^2 \leq (4 / 3) * \text{mat-det } ?a$  by linarith
hence  $\text{sqrt}(a_{11}^2) \leq \text{sqrt}((4 / 3) * \text{mat-det } ?a)$ 
  using real-sqrt-le-mono by blast
hence  $a_{11} \leq \text{sqrt}((4 / 3) * \text{mat-det } ?a)$  by auto
hence  $a_{11} \leq (\text{sqrt } 4) / (\text{sqrt } 3) * \text{sqrt}(\text{mat-det } ?a)$ 
  unfolding real-sqrt-mult real-sqrt-divide by blast
hence 9:  $\text{mat2}_{11} ?a \leq (2 / \text{sqrt } 3) * \text{sqrt}(\text{mat-det } ?a)$  by auto
have ?P ?a using 7 8 9 by blast
thus ?thesis by blast
qed

```

Theorem 1.2 from [1].

```

theorem qf2-det-one-equiv-canonical:
  fixes f :: mat2
  assumes mat-sym f
  assumes qf-positive-definite f
  assumes mat-det f = 1
  shows f ~ 1
proof -
  obtain a where
    0:  $f \sim a$ 
     $2 * |\text{mat2}_{12} a| \leq \text{mat2}_{11} a$ 
     $\text{mat2}_{11} a \leq (2 / \text{sqrt } 3) * \text{sqrt}(\text{mat-det } a)$ 
    using assms lemma-1-2[of f] qf2-equiv-sym by auto
  have 1: mat-sym a
    using assms 0 qf2-equiv-preserves-sym by auto
  have 2: qf-positive-definite a
    using assms 0 qf2-equiv-preserves-positive-definite by auto

```

```

have 3: mat-det a = 1 using assms 0 qf2-equiv-preserves-det by auto
have 4: mat2_11 a ≥ 1
  apply (rule allE[OF 2[unfolded qf2-positive-definite-def], of vec2 1 0])
  unfolding zero-vec2-def vec2-dot-def mat2-app-def qf2-app-def
    qf2-positive-definite-def
  apply auto
  done
have 5: mat2_11 a < 2 using 0 unfolding 3
  by (smt (verit, best) divide-le-eq int-less-real-le mult-cancel-left1
      of-int-1 real-sqrt-four real-sqrt-le-iff
      real-sqrt-mult-self real-sqrt-one)
have 6: mat2_11 a = 1 using 4 5 by auto
have 7: mat2_12 a = 0 mat2_21 a = 0 using 0 1 6 unfolding mat2-sym-def by
auto
have 8: mat2_22 a = 1 using 3 6 7 unfolding mat2-det-def by auto
have a = 1 using 6 7 8 unfolding one-mat2-def using mat2.collapse by metis
thus ?thesis using 0 by metis
qed

```

Lemma 1.3 from [1].

```

lemma lemma-1-3:
  fixes a :: mat3
  assumes mat-sym a
  defines a' ≡
    mat2
    (mat3_11 a * mat3_22 a - (mat3_12 a)^2) (mat3_11 a * mat3_23 a - mat3_12 a *
    mat3_13 a)
    (mat3_11 a * mat3_23 a - mat3_12 a * mat3_13 a) (mat3_11 a * mat3_33 a -
    (mat3_13 a)^2)

  defines d' ≡
    mat-det (
      mat2
      (mat3_11 a) (mat3_12 a)
      (mat3_12 a) (mat3_22 a)
    )

shows
  mat-det a' = mat3_11 a * mat-det a (is ?P)
  ⋀x. mat3_11 a * (a $$ x) =
    (mat3_11 a * vec3_1 x + mat3_12 a * vec3_2 x + mat3_13 a * vec3_3 x)^2 +
    (a' $$ (vec2 (vec3_2 x) (vec3_3 x))) (is ⋀x. ?Q x)
    qf-positive-definite a ==> qf-positive-definite a'
    qf-positive-definite a ↔ mat3_11 a > 0 ∧ d' > 0 ∧ mat-det a > 0
proof -
  show 0: ?P using assms
  unfolding a'-def mat2-det-def mat3-det-def mat3-sym-def power2-eq-square
  by (simp add: algebra-simps)
  show 1: ⋀x. ?Q x using assms

```

```

unfolding a'-def vec2-dot-def vec3-dot-def mat2-app-def mat3-app-def
    mat3-sym-def qf2-app-def qf3-app-def power2-eq-square
by (simp add: algebra-simps)
have 2: qf-positive-definite a  $\implies$  mat311 a > 0
proof -
  assume 3: qf-positive-definite a
  show mat311 a > 0
    using allE[OF iffD1[OF qf3-positive-definite-def 3], of vec3 1 0 0]
    unfolding zero-vec3-def vec3-dot-def mat3-app-def qf3-app-def
    by auto
  qed
  show 4: qf-positive-definite a  $\implies$  qf-positive-definite a'
  unfolding qf2-positive-definite-def
  proof
    fix v :: vec2
    assume 5: qf-positive-definite a
    hence 6: mat311 a > 0 using 2 by blast
    have a' $$ v \leq 0  $\implies$  v = 0 proof -
      assume 7: a' $$ v \leq 0
      obtain x2 x3 where 8: v = vec2 x2 x3 by (rule vec2.exhaust)
      define x1 where x1  $\equiv$  - (mat312 a * x2 + mat313 a * x3)
      have mat311 a * (a $$ (vec3 x1 (mat311 a * x2) (mat311 a * x3)) =
        (mat311 a * x1 + mat312 a * mat311 a * x2 + mat313 a * mat311 a * x3)2
+
        (a' $$ (vec2 (mat311 a * x2) (mat311 a * x3)))
      unfolding 1 by (simp add: algebra-simps)
      also have ... = a' $$ (vec2 (mat311 a * x2) (mat311 a * x3))
      unfolding x1-def by (simp add: algebra-simps)
      also have ... = (mat311 a)2 * (a' $$ v)
      unfolding 8 vec2-dot-def mat2-app-def qf2-app-def power2-eq-square
      by (simp add: algebra-simps)
      also have ...  $\leq$  0
      using 7 unfolding power2-eq-square by (simp add: mult-nonneg-nonpos)
      finally have mat311 a * (a $$ (vec3 x1 (mat311 a * x2) (mat311 a * x3)))  $\leq$  0
.
      hence a $$ (vec3 x1 (mat311 a * x2) (mat311 a * x3))  $\leq$  0
      using 6 by (simp add: mult-le-0-iff)
      hence vec3 x1 (mat311 a * x2) (mat311 a * x3) = 0
      using 5 unfolding qf3-positive-definite-def by fastforce
      hence x2 = 0 x3 = 0 using 6 unfolding zero-vec3-def by fastforce+
      thus v = 0 unfolding 8 zero-vec2-def by blast
      qed
      thus v  $\neq$  0  $\longrightarrow$  0 < a' $$ v by fastforce
    qed
    have 9: qf-positive-definite a  $\implies$  d' > 0  $\wedge$  mat-det a > 0
    proof -
      assume 10: qf-positive-definite a
      have 11: mat311 a > 0 using 2 10 by blast

```

```

have qf-positive-definite a' using 4 10 by blast
hence 12:  $\text{mat2}_{11} a' > 0 \wedge \text{mat-det } a' > 0$ 
  using qf2-positive-definite-criterion
  unfolding a'-def mat2-sym-def by fastforce
have 13:  $d' > 0$ 
  using 12 unfolding a'-def d'-def mat2-det-def power2-eq-square by fastforce
have 14:  $\text{mat-det } a > 0$ 
  using 11 12 unfolding 0 by (simp add: zero-less-mult-iff)
  show  $d' > 0 \wedge (\text{mat-det } a) > 0$  using 13 14 by blast
qed
have 15:  $\text{mat3}_{11} a > 0 \wedge d' > 0 \wedge \text{mat-det } a > 0 \implies \text{qf-positive-definite } a$ 
proof -
  assume 16:  $\text{mat3}_{11} a > 0 \wedge d' > 0 \wedge \text{mat-det } a > 0$ 
  have 17:  $\text{mat2}_{11} a' > 0$ 
    using 16 unfolding a'-def d'-def mat2-det-def power2-eq-square
    by (simp add: algebra-simps)
  have 18:  $\text{mat-det } a' > 0$  using 16 unfolding 0 by fastforce
  have 19: qf-positive-definite a'
    using qf2-positive-definite-criterion 17 18
    unfolding a'-def mat2-sym-def by fastforce
  show qf-positive-definite a
  unfolding qf3-positive-definite-def
  proof
    fix  $x :: \text{vec3}$ 
    have  $a \leq 0 \implies x = 0$  proof -
      assume  $a \leq 0$ 
      hence 20:  $\text{mat3}_{11} a * (a \leq 0) \leq 0$ 
        using 16 mult-le-0-iff order-less-le by auto
      have  $a' \leq (a * (a \leq 0)) \leq 0$ 
        using 20 unfolding 1 power2-eq-square by (smt (verit) zero-le-square)
      hence 21:  $a' \leq 0$ 
        using 19 qf2-positive-definite-positive
        using nle-le by blast
      have 22:  $\text{vec3}_2 x = 0 \wedge \text{vec3}_3 x = 0$ 
        using 19 21 unfolding zero-vec2-def qf2-positive-definite-def
        by (smt (verit) vec2.inject)+
      have  $\text{mat3}_{11} a * \text{vec3}_1 x + \text{mat3}_{12} a * \text{vec3}_2 x + \text{mat3}_{13} a * \text{vec3}_3 x = 0$ 
        using 20 21 unfolding 1 by fastforce
      hence  $\text{vec3}_1 x = 0$  using 16 22 by fastforce
      thus  $x = 0$  using 22 unfolding zero-vec3-def by (metis vec3.collapse)
    qed
    thus  $x \neq 0 \longrightarrow 0 < a \leq 0$  by fastforce
  qed
show qf-positive-definite a  $\longleftrightarrow$   $\text{mat3}_{11} a > 0 \wedge d' > 0 \wedge \text{mat-det } a > 0$ 
  using 2 9 15 by blast
qed

```

Lemma 1.4 from [1].

```

lemma lemma-1-4:
  fixes b :: mat3
  fixes v' :: mat2
  fixes r s :: int
  assumes mat-sym b
  assumes qf-positive-definite b
  assumes mat-det v' = 1
  defines b' ≡
    mat2
    (mat311 b * mat322 b - (mat312 b)2) (mat311 b * mat323 b - mat312 b * mat313 b)
    (mat311 b * mat323 b - mat312 b * mat313 b) (mat311 b * mat333 b - (mat313 b)2)
  defines a' ≡ b' · v'
  defines v ≡
    mat3
    1 r s
    0 (mat211 v') (mat212 v')
    0 (mat221 v') (mat222 v')

  defines a ≡ b · v
  shows
     $\bigwedge y. \text{mat3}_{11} b * (b \text{ ## } y) =$ 
     $(\text{mat3}_{11} b * \text{vec3}_1 y + \text{mat3}_{12} b * \text{vec3}_2 y + \text{mat3}_{13} b * \text{vec3}_3 y)^2 +$ 
     $(b' \text{ ## } (\text{vec2}(\text{vec3}_2 y) (\text{vec3}_3 y))) \text{ (is } \bigwedge y. ?P y)$ 
    mat311 a = mat311 b
     $\bigwedge x. \text{mat3}_{11} a * (a \text{ ## } x) =$ 
     $(\text{mat3}_{11} a * \text{vec3}_1 x + \text{mat3}_{12} a * \text{vec3}_2 x + \text{mat3}_{13} a * \text{vec3}_3 x)^2 +$ 
     $(a' \text{ ## } (\text{vec2}(\text{vec3}_2 x) (\text{vec3}_3 x))) \text{ (is } \bigwedge x. ?Q x)$ 
  proof -
    show  $\bigwedge y. ?P y$  using assms
    unfolding b'-def vec2-dot-def vec3-dot-def mat2-app-def mat3-app-def
      mat3-sym-def qf2-app-def qf3-app-def power2-eq-square
    by (simp add: algebra-simps)
    show mat311 a = mat311 b
    unfolding a-def v-def times-mat3-def mat3 transpose-def qf3-action-def
      by force
    show  $\bigwedge y. ?Q y$  using assms
    by (simp add: algebra-simps power2-eq-square
      a-def v-def a'-def b'-def vec2-dot-def vec3-dot-def
      times-mat2-def times-mat3-def mat2-app-def mat3-app-def
      mat2 transpose-def mat3 transpose-def mat3-sym-def
      qf2-app-def qf3-app-def qf2-action-def qf3-action-def)
  qed

```

Lemma 1.5 from [1].

```

lemma lemma-1-5:
  fixes u11 u21 u31

```

```

assumes Gcd {u11, u21, u31} = 1
shows ∃ u. mat311 u = u11 ∧ mat321 u = u21 ∧ mat331 u = u31 ∧ mat-det u
= 1
proof -
let ?a = gcd u11 u21
show ?thesis proof cases
assume ?a = 0
hence 0: u11 = 0 u21 = 0 u31 = 1 ∨ u31 = -1 using assms by auto
let ?u =
mat3
0 0 (- 1)
0 u31 0
u31 0 0
show ?thesis
apply (rule exI[of - ?u])
unfolding mat3-det-def using 0 apply auto
done
next
assume 1: ?a ≠ 0
obtain u12 u22 where 2: u11 * u22 - u21 * u12 = ?a using bezout-int
by (metis diff-minus-eq-add mult.commute mult-minus-right)
have gcd ?a u31 = 1 using assms by (simp add: gcd.assoc)
then obtain u33 b where 3: ?a * u33 - b * u31 = 1 using bezout-int
by (metis diff-minus-eq-add mult.commute mult-minus-right)
let ?u =
mat3
u11 u12 ((u11 div ?a) * b)
u21 u22 ((u21 div ?a) * b)
u31 0 u33
have mat-det ?u =
u11 * u22 * u33
+ u12 * ((u21 div ?a) * b) * u31
- ((u11 div ?a) * b) * u22 * u31
- u12 * u21 * u33
unfolding mat3-det-def by auto
also have ... =
u11 * u22 * u33
+ u12 * (u21 div ?a) * (b * u31)
- u22 * (u11 div ?a) * (b * u31)
- u12 * u21 * u33
by auto
also have ... =
u11 * u22 * u33
+ u12 * (u21 div ?a) * (?a * u33 - 1)
- u22 * (u11 div ?a) * (?a * u33 - 1)
- u12 * u21 * u33
using 3 by (simp add: algebra-simps)
also have ... =
u11 * u22 * u33

```

```

+ u12 * ((u21 div ?a) * ?a) * u33
- u12 * (u21 div ?a)
- u22 * ((u11 div ?a) * ?a) * u33
+ u22 * (u11 div ?a)
- u12 * u21 * u33
by (simp add: algebra-simps)
also have ... =
- u12 * (u21 div ?a)
+ u22 * (u11 div ?a)
by (simp add: algebra-simps)
also have ... =
- u12 * (u21 div ?a)
+ u22 * u11 div ?a
by (metis dvd-div-mult gcd-dvd1 mult.commute)
also have ... =
- u12 * (u21 div ?a)
+ (?a + u21 * u12) div ?a
using 2 by (simp add: algebra-simps)
also have ... =
- u12 * (u21 div ?a)
+ 1 + (u21 * u12) div ?a
using 1 by auto
also have ... = 1 by (simp add: algebra-simps div-mult1-eq)
finally have 4: mat-det ?u = 1 .
show ?thesis
apply (rule exI[of - ?u])
using 4 apply auto
done
qed
qed

```

Lemma 1.6 from [1].

```

lemma lemma-1-6:
fixes c :: mat3
assumes mat-sym c
assumes qf-positive-definite c
shows  $\exists a. a \sim c \wedge$ 

$$2 * (\max |mat3_{12} a| |mat3_{13} a|) \leq mat3_{11} a \wedge$$


$$mat3_{11} a \leq (4 / 3) * \text{root } 3 (mat-det a)$$

proof -
define a11 where a11 ≡ LEAST y. y > 0 ∧ (∃x. int y = c §§ x)
have 0: ∃y. y > 0 ∧ (∃x. int y = c §§ x)
apply (rule exI[of - nat (c §§ (vec3 1 1 1))])
using assms(2) unfolding qf3-positive-definite-def zero-vec3-def
apply (metis nat-0-le order-less-le vec3.inject zero-less-nat-eq zero-neq-one)
done
obtain t where 1: a11 > 0 int a11 = c §§ t
using a11-def LeastI-ex[OF 0] by auto
let ?h = Gcd {vec31 t, vec32 t, vec33 t}

```

```

have  $t \neq 0$  using assms(2) 1 by fastforce
hence  $?h > 0$  by (simp; metis vec3.collapse zero-vec3-def)
let  $?t' = \text{vec3}(\text{vec3}_1 t \text{ div } ?h)(\text{vec3}_2 t \text{ div } ?h)(\text{vec3}_3 t \text{ div } ?h)$ 
have  $?t' \neq 0$  using 2 unfolding zero-vec3-def
  by (auto simp add: algebra-simps dvd-div-eq-0-iff)
hence  $\text{nat}(c \$\$ ?t') > 0 \wedge (\exists x. \text{int}(\text{nat}(c \$\$ ?t')) = c \$\$ x)$ 
  using assms(2) unfolding qf3-positive-definite-def by auto
hence  $a_{11} \leq \text{nat}(c \$\$ ?t')$  unfolding  $a_{11}$ -def by (rule Least-le)
hence  $a_{11} \leq c \$\$ ?t'$  using 1 by auto
also have ... =  $(c \$\$ t) \text{ div } ?h^2$  proof -
  have  $?h \text{ dvd } \text{vec3}_1 t ?h \text{ dvd } \text{vec3}_2 t ?h \text{ dvd } \text{vec3}_3 t$ 
    by (meson Gcd-dvd insertCI)+
  then have  $(c \$\$ ?t') * ?h^2 = c \$\$ t ?h^2 \text{ dvd } c \$\$ t$ 
    unfolding qf3-app-def vec3-dot-def mat3-app-def power2-eq-square
    using 1
    by (auto simp add: algebra-simps mult-dvd-mono)
  thus  $c \$\$ ?t' = (c \$\$ t) \text{ div } ?h^2$  using 2 by auto
qed
also have ... =  $a_{11} \text{ div } ?h^2$  using 1 by auto
finally have  $a_{11} \leq a_{11} \text{ div } ?h^2$  .
also have ...  $\leq a_{11}$  using 1 2
  by (metis div-le-dividend nat-div-distrib nat-int nat-le-iff of-nat-0-le-iff)
finally have  $?h = 1$  using 1 2
  by (smt (verit) int-div-less-self of-nat-0-less-iff power2-eq-square
      zero-less-power zmult-eq-1-iff)
then obtain  $u$  where 3:  $\text{mat3}_{11} u = \text{vec3}_1 t \text{ mat3}_{21} u = \text{vec3}_2 t$ 
 $\text{mat3}_{31} u = \text{vec3}_3 t \text{ mat-det } u = 1$ 
  using lemma-1-5 by blast
define  $b$  where  $b \equiv c \cdot u$ 
have 4: mat-sym  $b$ 
  using 3 assms(1) qf3-equiv-preserves-sym
  unfolding b-def qf3-equiv-def
  by auto
have 5: qf-positive-definite  $b$ 
  using 3 assms(2) qf3-equiv-preserves-positive-definite
  unfolding b-def qf3-equiv-def
  by auto
have 6:  $a_{11} = (\text{LEAST } y. y > 0 \wedge (\exists x. \text{int } y = b \$\$ x))$ 
  unfolding  $a_{11}$ -def apply (rule arg-cong[of - - Least])
  using 3 qf3-equiv-preserves-range[of  $c b$ ]
  unfolding b-def image-def qf3-equiv-def
  apply fast
done
have 7:  $a_{11} = \text{mat3}_{11} b$ 
  using 1 3
  by (simp add: algebra-simps
    b-def times-mat3-def vec3-dot-def mat3-app-def
    mat3 transpose-def qf3-app-def qf3-action-def)
define  $b'$  where  $b' \equiv$ 

```

```

mat2
  (mat311 b * mat322 b - (mat312 b)2) (mat311 b * mat323 b - mat312 b *
mat313 b)
  (mat311 b * mat323 b - mat312 b * mat313 b) (mat311 b * mat333 b -
(mat313 b)2)

have 8: mat-sym b' unfolding b'-def mat2-sym-def by auto
have 9: mat-det b' = mat311 b * mat-det b
   $\bigwedge x. \text{mat3}_{11} b * (b \$\$ x) =$ 
    (mat311 b * vec31 x + mat312 b * vec32 x + mat313 b * vec33 x)2 +
    (b' \$\$ (vec2 (vec32 x) (vec33 x)))
  qf-positive-definite b'
using 4 5 b'-def lemma-1-3 by blast+
obtain a' v' where 10: a' = b' · v'
  mat-det v' = 1
  mat211 a' ≤ (2 / sqrt 3) * sqrt (mat311 b * mat-det b)
using 8 9 qf2-equiv-sym qf2-equiv-preserves-det lemma-1-2[of b']
unfold qf2-equiv-def by metis
obtain r s where 11: 2 * |(mat312 b) * (mat211 v') +
  (mat313 b) * (mat221 v') + a11 * (r :: int)| ≤ a11
  2 * |(mat312 b) * (mat212 v') +
  (mat313 b) * (mat222 v') + a11 * (s :: int)| ≤ a11
using 1 congruence-class-close by fastforce
define a12 where a12 ≡ (mat312 b) * (mat211 v') +
  (mat313 b) * (mat221 v') + a11 * r
define a13 where a13 ≡ (mat312 b) * (mat212 v') +
  (mat313 b) * (mat222 v') + a11 * s
define v where v ≡
  mat3
  1 r s
  0 (mat211 v') (mat212 v')
  0 (mat221 v') (mat222 v')

have 12: mat-det v = 1
  using 10 unfold v-def mat2-det-def mat3-det-def by (simp add: algebra-simps)
define a where a ≡ b · v
have 13: mat-det a = mat-det b
  using 12 qf3-equiv-preserves-det
  unfold a-def qf3-equiv-def
  by metis
have 14: a11 = (LEAST y. y > 0 ∧ (∃ x. int y = a \$\$ x))
  unfold 6 apply (rule arg-cong[of - - Least])
  using 12 qf3-equiv-preserves-range[of b a]
  unfold a-def image-def qf3-equiv-def
  apply fast
  done
have 15: mat311 a = mat311 b
   $\bigwedge x. \text{mat3}_{11} a * (a \$\$ x) =$ 

```

```


$$(mat3_{11} a * vec3_1 x + mat3_{12} a * vec3_2 x + mat3_{13} a * vec3_3 x)^2 +$$


$$(a' \$\$ (vec2 (vec3_2 x) (vec3_3 x)))$$

using 4 5 10 lemma-1-4 unfolding b'-def v-def a-def by blast+
have 16: mat2_{11} a' = mat3_{11} a * mat3_{22} a - (mat3_{12} a)^2
using 15(2)[of vec3 0 1 0]
unfolding vec3-dot-def vec2-dot-def mat3-app-def mat2-app-def
qf3-app-def qf2-app-def
by simp
define a_{22} where a_{22} ≡ a \$\$ (vec3 0 1 0)
have 17: a_{11} = mat3_{11} a unfolding 7 15 by auto
have 18: a_{12} = mat3_{12} a a_{13} = mat3_{13} a a_{22} = mat3_{22} a a_{22} = mat3_{22} a
using 13(1) 17
unfolding a-def v-def a_{12}-def a_{13}-def a_{22}-def
by (auto simp add: algebra-simps
times-mat3-def vec3-dot-def mat3-app-def
mat3 transpose-def qf3-app-def qf3-action-def)
have 19: a ~ c
unfolding qf3-equiv-sym[of a c]
unfolding a-def b-def qf3-equiv-def qf3-action-mul[symmetric]
using 3 12 by (metis mat3-mul-det mult-1)
have 20: 2 * (max |mat3_{12} a| |mat3_{13} a|) ≤ mat3_{11} a
using 11 unfolding 17[symmetric] 18 a_{12}-def[symmetric] a_{13}-def[symmetric]
by auto
have 21: (2 / sqrt 3) ^ 6 = 64 / 27 unfolding power-def by auto
have a_{11} ≤ int (nat a_{22})
unfolding a_{11}-def a_{22}-def
apply (rule rev-iffD1[OF - nat-int-comparison(3)])
apply (rule wellorder-Least-lemma(2))
using assms(2) 5 12
apply (metis a-def b-def nat-0-le nat-int of-nat-0 qf3-action-app
qf3-equiv-def qf3-equiv-preserves-positive-definite
qf3-positive-definite-def qf3-positive-definite-positive
vec3.sel(2) zero-neq-one zero-vec3-def zless-nat-conj)
done
hence a_{11} ≤ a_{22} using 1 by linarith
hence (mat3_{11} a)^2 ≤ a_{11} * a_{22} unfolding power2-eq-square using 1 17 by auto
also have ... = mat2_{11} a' + a_{12}^2 using 16 17 18 by auto
also have ... ≤ (2 / sqrt 3) * sqrt (mat3_{11} b * mat-det b) + a_{12}^2
using 10 by auto
also have ... ≤ (2 / sqrt 3) * sqrt (mat3_{11} a * mat-det a) + (mat3_{11} a)^2 / 4
using 11 13 15 17 18 a_{12}-def
power2-le-iff-abs-le[of real-of-int (mat3_{11} a) (mat3_{12} a) * 2]
by auto
finally have (3 / 4) * (mat3_{11} a)^2 ≤ (2 / sqrt 3) * sqrt (mat3_{11} a * mat-det a)
by (simp add: algebra-simps)
hence (mat3_{11} a)^2 ≤ (2 / sqrt 3) ^ 3 * sqrt (mat3_{11} a * mat-det a)
unfolding power2-eq-square power3-eq-cube by (simp add: algebra-simps)
hence ((mat3_{11} a)^2)^2 ≤ ((2 / sqrt 3) ^ 3 * sqrt (mat3_{11} a * mat-det a))^2

```

```

using 1(1) unfolding 17[symmetric] power2-eq-square
by (metis of-int-power power2-eq-square power-mono zero-le-square)
hence (mat311 a) 4 ≤ (2 / sqrt 3) 6 * (sqrt (mat311 a * mat-det a))2
by (simp add: power-mult-distrib)
hence (mat311 a) 4 ≤ (2 / sqrt 3) 6 * mat311 a * mat-det a
using 4 5 13 17 lemma-1-3 by auto
hence (mat311 a) 3 ≤ (2 / sqrt 3) 6 * mat-det a
using 1(1) unfolding 17[symmetric]
unfolding power-def apply (simp add: algebra-simps)
apply (metis mult-left-le-imp-le of-nat-0-less-iff times-divide-eq-right)
done
hence (mat311 a) 3 ≤ (64 / 27) * mat-det a using 21 by auto
hence root 3 ((mat311 a) 3) ≤ root 3 ((64 / 27) * mat-det a) by auto
hence root 3 ((mat311 a) 3) ≤ root 3 (64 / 27) * root 3 (mat-det a)
unfolding real-root-mult by auto
hence mat311 a ≤ root 3 (64 / 27) * root 3 (mat-det a)
using odd-real-root-power-cancel by auto
hence 22: mat311 a ≤ (4 / 3) * root 3 (mat-det a)
using real-root-divide by force
show ?thesis using 19 20 22 by blast
qed

```

Theorem 1.3 from [1].

```

theorem qf3-det-one-equiv-canonical:
fixes f :: mat3
assumes mat-sym f
assumes qf-positive-definite f
assumes mat-det f = 1
shows f ∼ 1
proof –
obtain a where
  0: f ∼ a ∧
    2 * (max |mat312 a| |mat313 a|) ≤ mat311 a ∧
    mat311 a ≤ (4 / 3) * root 3 (mat-det a)
using assms lemma-1-6[of f] qf3-equiv-sym by auto
have 1: mat-sym a
using assms 0 qf3-equiv-preserves-sym by auto
have 2: qf-positive-definite a
using assms 0 qf3-equiv-preserves-positive-definite by auto
have 3: mat-det a = 1 using assms 0 qf3-equiv-preserves-det by auto
have 4: mat312 a = 0 mat313 a = 0 using 0 3 by auto
have mat311 a ≥ 1
apply (rule allE[OF 2[unfolded qf3-positive-definite-def], of vec3 1 0 0])
unfolding zero-vec3-def vec3-dot-def mat3-app-def qf3-app-def
  qf3-positive-definite-def
apply auto
done
hence 6: mat311 a = 1 using 0 3 by auto
define a' where a' ≡

```

```

mat2
  (mat322 a) (mat323 a)
  (mat332 a) (mat333 a)

have 7: mat-det a' = 1
  using 3 4 6 unfolding a'-def mat2-det-def mat3-det-def by auto
have 8: mat-sym a' using 1 unfolding a'-def mat2-sym-def mat3-sym-def by
auto
have 9: qf-positive-definite a'
  using 2 unfolding qf2-positive-definite-def qf3-positive-definite-def
proof -
  assume 10:  $\forall v. v \neq 0 \longrightarrow a \quad v > 0$ 
  show  $\forall v. v \neq 0 \longrightarrow a' \quad v > 0$ 
  proof (rule; rule)
    fix v :: vec2
    assume v ≠ 0
    hence vec3 0 (vec21 v) (vec22 v) ≠ 0
      unfolding zero-vec2-def zero-vec3-def by (metis vec2.collapse vec3.inject)
    hence a $$ (vec3 0 (vec21 v) (vec22 v)) > 0 using 10 by auto
    thus a' $$ v > 0
      unfolding a'-def vec2-dot-def vec3-dot-def
        mat2-app-def mat3-app-def qf2-app-def qf3-app-def
        qf2-positive-definite-def qf3-positive-definite-def
      by auto
    qed
  qed
obtain u' where 11: mat-det u' = 1 a' · u' = 1
  using 7 8 9 qf2-det-one-equiv-canonical[of a']
  unfolding qf2-equiv-def
  by auto
define u where u ≡
  mat3
    1 0 0
    0 (mat211 u') (mat212 u')
    0 (mat221 u') (mat222 u')

have 12: mat-det u = 1
  using 11 unfolding u-def mat2-det-def mat3-det-def by auto
have 13: a · u = 1
  using 1 4 6 11
  by (simp add: algebra-simps
    a'-def u-def one-mat2-def one-mat3-def
    times-mat2-def times-mat3-def
    mat2 transpose-def mat3 transpose-def
    qf2-action-def qf3-action-def mat3-sym-def)
have a ∼ 1 using 12 13 unfolding qf3-equiv-def by blast
  thus ?thesis using 0 qf3-equiv-trans by blast
qed

```

end

## 4 Legendre's three squares theorem and its consequences

```
theory Three-Squares
imports Dirichlet-L.Dirichlet-Theorem Residues-Properties Quadratic-Forms
begin
```

### 4.1 Legendre's three squares theorem

```
definition quadratic-residue-alt :: int ⇒ int ⇒ bool where
quadratic-residue-alt m a = (exists x y. x2 − a = y * m)
```

```
lemma quadratic-residue-alt-equiv: quadratic-residue-alt = QuadRes
  unfolding quadratic-residue-alt-def QuadRes-def
  by (metis cong-iff-dvd-diff cong-modulus-mult dvdE dvd-refl mult.commute)
```

```
lemma sq-nat-abs: (nat |v|)2 = nat (v2)
  by (simp add: nat-power-eq[symmetric])
```

Lemma 1.7 from [1].

```
lemma three-squares-using-quadratic-residue:
fixes n d' :: nat
assumes n ≥ 2
assumes d' > 0
assumes QuadRes (d' * n − 1) (− d')
shows ∃ x1 x2 x3. n = x12 + x22 + x32
proof −
  define a where a ≡ d' * n − 1
  from assms(3) obtain x y where x2 + int d' = y * int a
    unfolding a-def quadratic-residue-alt-equiv[symmetric]
      quadratic-residue-alt-def
    by auto
  hence Hxy: x2 + d' = y * a by auto
  have y ≥ 1
    using assms Hxy
    unfolding a-def
    by (smt (verit) bot-nat-0.not-eq-extremum mult-le-0-iff of-nat-0-le-iff
        of-nat-le-0-iff power2-eq-square zero-le-square)
  moreover from Hxy have Hxy2: d' = y * a − x2 by (simp add: algebra-simps)
  define M where M ≡ mat3 y x 1 x a 0 1 0 n
  moreover have Msym: mat-sym M
    unfolding mat3-sym-def M-def mat3.sel
    by simp
  moreover have Mdet-eq-1: mat-det M = 1
  proof −
    have mat-det M = (y * a − x2) * n − a
```

```

unfolding mat3-det-def M-def mat3.sel power2-eq-square
by (simp add: algebra-simps)
also have ... = int d' * n - a unfolding Hxy2 by simp
also have ... = 1 unfolding a-def using assms int-ops by force
finally show ?thesis .
qed
moreover have mat-det (mat2 y x x a) > 0
using Hxy2 assms
unfolding mat2-det-def power2-eq-square
by simp
ultimately have qf-positive-definite M
by (auto simp add: lemma-1-3(4))
hence M ~ 1
using Msym and Mdet-eq-1
by (simp add: qf3-det-one-equiv-canonical)
moreover have M $$ vec3 0 0 1 = n
unfolding M-def qf3-app-def mat3-app-def vec3-dot-def mat3.sel vec3.sel
by (simp add: algebra-simps)
hence n ∈ range (( $$ ) M) by (metis rangeI)
ultimately have n ∈ range (( $$ ) (1 :: mat3))
using qf3-equiv-preserves-range by simp
then obtain u :: vec3 where 1 $$ u = n
by auto
hence <u | u> = n
unfolding qf3-app-def mat3-app-def one-mat3-def
by simp
hence ∃ x1 x2 x3. int n = x12 + x22 + x32
unfolding vec3-dot-def power2-eq-square by metis
hence ∃ x1 x2 x3. n = (nat |x1|)2 + (nat |x2|)2 + (nat |x3|)2
unfolding sq-nat-abs
apply (simp add: nat-add-distrib[symmetric])
apply (metis nat-int)
done
thus ∃ x1 x2 x3. n = x12 + x22 + x32 by blast
qed

```

```

lemma prime-linear-combination:
fixes a m :: nat
assumes m > 1
assumes coprime a m
obtains j :: nat where prime (a + m * j) ∧ j ≠ 0
proof –
assume 0: ∀j. prime (a + m * j) ∧ j ≠ 0  $\implies$  thesis

have 1: infinite {p. prime p ∧ [p = a] (mod m)}
using assms
by (rule Dirichlet-Theorem.residues-nat.Dirichlet[unfolded residues-nat-def])

have 2: finite {j. prime (nat (int a + int m * j)) ∧ j ≤ 0}

```

```

apply (rule finite-subset[of - {-(int a) div (int m)..0}])
subgoal
  apply (rule subsetI)
  subgoal for j
    proof -
      assume 1:  $j \in \{j. \text{prime}(\text{nat}(int a + int m * j)) \wedge j \leq 0\}$ 
      have  $\text{int } a + \text{int } m * j \geq 0$  using 1 prime-ge-0-int by force
      hence  $\text{int } m * j \geq -\text{int } a$  by auto
      hence  $j \geq -(int a) \text{ div } int m$ 
        using assms 1
        by (smt (verit) unique-euclidean-semiring-class.cong-def
            coprime-crossproduct-nat coprime-iff-invertible-int
            coprime-int-iff int-distrib(3) int-ops(2) int-ops(7)
            mem-Collect-eq mult-cancel-right1 zdiv-mono1
            nonzero-mult-div-cancel-left of-nat-0-eq-iff
            of-nat-le-0-iff prime-ge-2-int prime-nat-iff-prime)
      thus  $j \in \{-(int a) \text{ div } (int m)..0\}$  using 1 by auto
    qed
    done
  subgoal by blast
  done

  have  $\{p. \text{prime } p \wedge [p = a] \pmod{m}\} =$ 
     $\{p. \text{prime } p \wedge (\exists j. \text{int } p = \text{int } a + \text{int } m * j)\}$ 
  unfolding cong-sym-eq[of - a]
  unfolding cong-int-iff[symmetric] cong-iff-lin
  ..
  also have ... =  $\{p. \text{prime } p \wedge (\exists j. p = \text{nat}(int a + int m * j))\}$ 
  by (metis (opaque-lifting) nat-int nat-eq-iff
      prime-ge-0-int prime-nat-iff-prime)
  also have ... =  $(\lambda j. \text{nat}(int a + int m * j))`$ 
     $\{j. \text{prime}(\text{nat}(int a + int m * j))\}$ 
  by blast
  finally have infinite  $((\lambda j. \text{nat}(int a + int m * j))`$ 
     $\{j. \text{prime}(\text{nat}(int a + int m * j))\})$ 
  using 1 by metis
  hence infinite  $\{j. \text{prime}(\text{nat}(int a + int m * j))\}$ 
  using finite-imageI by blast
  hence infinite  $(\{j. \text{prime}(\text{nat}(int a + int m * j))\} -$ 
     $\{j. \text{prime}(\text{nat}(int a + int m * j)) \wedge j \leq 0\})$ 
  using 2 Diff-infinite-finite by blast
  hence infinite  $\{j. \text{prime}(\text{nat}(int a + int m * j)) \wedge j > 0\}$ 
  by (rule back-subst[of infinite]; auto)
  hence infinite  $(int` \{j. \text{prime}(\text{nat}(int a + int m * j)) \wedge j \neq (0 :: \text{nat})\})$ 
  apply (rule back-subst[of infinite])
  unfolding image-def using zero-less-imp-eq-int apply auto
  done
  hence infinite  $\{j. \text{prime}(\text{nat}(int a + int m * j)) \wedge (j :: \text{nat}) \neq 0\}$ 
  using finite-imageI by blast

```

```

hence infinite {j. prime (a + m * j) ∧ j ≠ 0}
  apply (rule back-subst[of infinite])
  apply (auto simp add: int-ops nat-plus-as-int)
  done
thus thesis using 0 not-finite-existsD by blast
qed

```

Lemma 1.8 from [1].

```

lemma three-squares-using-mod-four:
  fixes n :: nat
  assumes n mod 4 = 2
  shows ∃ x1 x2 x3. n = x12 + x22 + x32
proof -
  have n > 1 using assms by auto
  have coprime (n - 1) (4 * n)
    by (smt (verit, del-insts) Suc-pred assms bot-nat-0.not-eq-extremum
        coprime-commute coprime-diff-one-right-nat
        coprime-mod-right-iff coprime-mult-left-iff
        diff-Suc-1 mod-Suc mod-mult-self1-is-0 mult-0-right
        numeral-2-eq-2 zero-neq-numeral)
  then obtain j where H-j:
    prime ((n - 1) + (4 * n) * j) ∧ j ≠ 0
    using prime-linear-combination[of 4 * n n - 1] ⟨n > 1⟩ by auto
  have j > 0 using H-j by blast

  define d' where d' ≡ 4 * j + 1
  define p where p ≡ d' * n - 1
  have prime p
    unfolding p-def d'-def
    using conjunct1[OF H-j] apply (rule back-subst[of prime])
    using ⟨n > 1⟩ apply (simp add: algebra-simps nat-minus-as-int nat-plus-as-int)
    done
  have p mod 4 = 1
    unfolding p-def
    apply (subst mod-diff-eq-nat)
    subgoal unfolding d'-def using ⟨n > 1⟩ ⟨j > 0⟩ by simp
    subgoal
      apply (subst mod-mult-eq[symmetric])
      unfolding assms d'-def apply simp
      done
    done
  have d' * n mod 4 = 2
    using assms p-def d'-def ⟨p mod 4 = 1⟩
    by (metis mod-mult-cong mod-mult-self4 nat-mult-1)
  hence d' mod 4 = 1 using assms by (simp add: d'-def)

  have QuadRes p (- d')
  proof -
    have d'-expansion: d' = (∏ q∈prime-factors d'. q ^ multiplicity q d')

```

```

using prime-factorization-nat unfolding d'-def by auto

have odd d' unfolding d'-def by simp
hence d'-prime-factors-odd: q ∈ prime-factors d' ⇒ odd q for q
by fastforce

have d'-prime-factors-gt-2: q ∈ prime-factors d' ⇒ q > 2 for q
using d'-prime-factors-odd
by (metis even-numeral in-prime-factors-imp-prime
order-less-le prime-ge-2-nat)

have [p = - 1] (mod d')
unfolding p-def cong-iff-dvd-diff apply simp
using ⟨n > 1⟩ apply (smt (verit) Suc-as-int Suc-pred add-gr-0 d'-def
dvd-nat-abs-iff dvd-triv-left
less-numeral-extra(1) mult-pos-pos
of-nat-less-0-iff order-le-less-trans
zero-less-one-class.zero-le-one)
done
hence d'-prime-factors-2-p-mod:
q ∈ prime-factors d' ⇒ [p = - 1] (mod q) for q
by (rule cong-dvd-modulus; auto)

have d' mod 4 = (Π q∈prime-factors d'. q ^ multiplicity q d') mod 4
using d'-expansion by argo
also have ... = (Π q∈prime-factors d'. (q mod 4) ^ multiplicity q d') mod 4
apply (subst mod-prod-eq[symmetric])
apply (subst power-mod[symmetric])
apply (subst mod-prod-eq)
apply blast
done
also have ... = (Π q∈{q ∈ prime-factors d'. [q = 3] (mod 4)}.
(q mod 4) ^ multiplicity q d') mod 4
apply (rule arg-cong[of _ - λx. x mod 4])
apply (rule prod.mono-neutral-right)
subgoal by blast
subgoal by blast
subgoal
unfold unique-euclidean-semiring-class.cong-def
apply (rule ballI)
using d'-prime-factors-odd apply simp
apply (metis One-nat-def dvd-0-right even-even-mod-4-iff
even-numeral mod-exhaust-less-4)
done
done
also have ... = (Π q∈{q ∈ prime-factors d'. [q = 3] (mod 4)}.
((int q) mod 4) ^ multiplicity q d') mod 4
by (simp add: int-ops)
also have ... = (Π q∈{q ∈ prime-factors d'. [q = 3] (mod 4)}).

```

```

 $((- 1) \text{ mod } 4) \wedge \text{multiplicity } q \text{ } d' \text{ mod } 4$ 
apply (rule arg-cong[of - -  $\lambda x. x \text{ mod } 4$ ])
apply (rule prod.cong[OF refl])
unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int apply
simp
apply (metis nat-int of-nat-mod of-nat-numeral)
done
also have ... =  $(\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 4)}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d' \text{ mod } 4$ 
apply (subst mod-prod-eq[symmetric])
apply (subst power-mod)
apply (subst mod-prod-eq)
apply blast
done
finally have  $[\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 4)}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d' = 1 :: \text{int} \text{ (mod 4)}$ 
using < $d' \text{ mod } 4 = 1$ >
by (simp add: unique-euclidean-semiring-class.cong-def)
hence prod-prime-factors-minus-one:
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 4)}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d' = (1 :: \text{int})$ 
unfolding power-sum[symmetric]
unfolding minus-one-power-iff unique-euclidean-semiring-class.cong-def
by presburger

have  $p > 2$  using <prime p> < $p \text{ mod } 4 = 1$ > nat-less-le prime-ge-2-nat by force

have  $d'$ -prime-factors-Legendre:
 $q \in \text{prime-factors } d' \implies$ 
Legendre  $p$   $q = \text{Legendre } q$   $p$  for  $q$ 
proof -
assume  $q \in \text{prime-factors } d'$ 
have prime  $q$  using < $q \in \text{prime-factors } d'$ > by blast
have  $q > 2$  using  $d'$ -prime-factors-gt-2 < $q \in \text{prime-factors } d'$ > by blast
show Legendre  $p$   $q = \text{Legendre } q$   $p$ 
using <prime p> < $p > 2$ > < $p \text{ mod } 4 = 1$ >
<prime  $q$ > < $q > 2$ > Legendre-equal[of  $p$   $q$ ]
unfolding unique-euclidean-semiring-class.cong-def
using zmod-int[of  $p$  4]
by auto
qed

have Legendre  $(- d')$   $p = \text{Legendre } (- 1)$   $p * \text{Legendre } d' p$ 
using <prime p> Legendre-mult[of  $p - 1$   $d'$ ] by auto
also have ... = Legendre  $d'$   $p$ 
using <prime p> < $p > 2$ > < $p \text{ mod } 4 = 1$ > Legendre-minus-one[of  $p$ ]
unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int
by auto
also have ... =  $(\prod q \in \text{prime-factors } d'. \text{Legendre } (q \wedge \text{multiplicity } q \text{ } d') \text{ } p)$ 

```

```

apply (subst d'-expansion)
using <prime p> <p > 2> Legendre-prod[of p] apply auto
done
also have ... = ( $\prod_{q \in \text{prime-factors } d'} (\text{Legendre } q \ p) \wedge \text{multiplicity } q \ d'$ )
  using <prime p> <p > 2> Legendre-power by auto
also have ... = ( $\prod_{q \in \text{prime-factors } d'} (\text{Legendre } p \ q) \wedge \text{multiplicity } q \ d'$ )
  using d'-prime-factors-Legendre by auto
also have ... = ( $\prod_{q \in \text{prime-factors } d'} (\text{Legendre } (-1) \ q) \wedge \text{multiplicity } q \ d'$ )
  apply (rule prod.cong[OF refl])
  using d'-prime-factors-2-p-mod apply (metis Legendre-cong)
done
also have ... = ( $\prod_{q \in \text{prime-factors } d'} (\text{if } [q = 1] \ (\text{mod } 4) \text{ then } 1 \text{ else } -1) \wedge \text{multiplicity } q \ d'$ )
  apply (rule prod.cong[OF refl])
subgoal for q
  using Legendre-minus-one-alt[of q] d'-prime-factors-gt-2[of q]
  by (smt (verit) cong-int-iff in-prime-factors-iff int-eq-iff-numeral
      less-imp-of-nat-less numeral-Bit0 numeral-One of-nat-1
      prime-nat-int-transfer)
done
also have ... = ( $\prod_{q \in \{q \in \text{prime-factors } d' \mid [q = 3] \ (\text{mod } 4)\}} (\text{if } [q = 1] \ (\text{mod } 4) \text{ then } 1 \text{ else } -1) \wedge \text{multiplicity } q \ d'$ )
  apply (rule prod.mono-neutral-right)
subgoal by blast
subgoal by blast
subgoal
  unfolding unique-euclidean-semiring-class.cong-def
  apply (rule ballI)
  using d'-prime-factors-odd apply simp
  apply (metis One-nat-def dvd-0-right even-even-mod-4-iff
        even-numeral mod-exhaust-less-4)
done
done
also have ... = ( $\prod_{q \in \{q \in \text{prime-factors } d' \mid [q = 3] \ (\text{mod } 4)\}} (-1) \wedge \text{multiplicity } q \ d'$ )
  by (rule prod.cong[OF refl];
      simp add: unique-euclidean-semiring-class.cong-def)
also have ... = 1 using prod-prime-factors-minus-one .
finally show QuadRes p (- d')
  unfolding Legendre-def
  by (metis one-neq-neg-one one-neq-zero)
qed
thus  $\exists x_1 x_2 x_3. n = x_1^2 + x_2^2 + x_3^2$ 
  using <n > 1> three-squares-using-quadratic-residue[of n d']
  unfolding d'-def p-def
  by auto
qed

```

```

lemma three-mod-eight-power-iff:
  fixes n :: nat
  shows (3 :: int) ^ n mod 8 = (if even n then 1 else 3)
proof (induction n)
  case 0
  thus ?case by auto
next
  case (Suc n)
  thus ?case
    apply (cases even n)
    subgoal
      using mod-mult-left-eq[of 3 8 3 ^ n] apply simp
      apply presburger
      done
    subgoal
      using mod-mult-left-eq[of 3 8 3 * 3 ^ n] apply simp
      apply presburger
      done
    done
  qed

```

Lemma 1.9 from [1].

```

lemma three-squares-using-mod-eight:
  fixes n :: nat
  assumes n mod 8 ∈ {1, 3, 5}
  shows ∃ x1 x2 x3. n = x1^2 + x2^2 + x3^2
proof cases
  assume n = 1
  hence n = 1^2 + 0^2 + 0^2 unfolding power2-eq-square by auto
  thus ∃ x1 x2 x3. n = x1^2 + x2^2 + x3^2 by blast
next
  assume n ≠ 1
  hence n > 1 using assms by auto

  have H-n:
    (n mod 8 = 1 ⇒ P) ⇒
    (n mod 8 = 3 ⇒ P) ⇒
    (n mod 8 = 5 ⇒ P) ⇒ P for P
    using assms by auto

define c :: nat where c ≡ if n mod 8 = 3 then 1 else 3
have c * n ≥ 1 unfolding c-def using ⟨n > 1⟩ by auto

obtain k where H-k: 2 * k = c * n - 1
  using H-n
  unfolding c-def
  by (smt (verit, ccfv-threshold) dvd-mod even-mult-iff even-numeral
    odd-numeral odd-one odd-two-times-div-two-nat)
have k-mod-4: k mod 4 = (if n mod 8 = 5 then 3 else 1) (is k mod 4 = ?v)

```

```

proof -
  have  $c * n \bmod 8 = (\text{if } n \bmod 8 = 5 \text{ then } 7 \text{ else } 3)$ 
  using  $H\text{-}n$ 
  proof cases
    assume  $n \bmod 8 = 1$ 
    have  $3 * n \bmod 8 = 3$ 
    using  $\langle n \bmod 8 = 1 \rangle \text{ mod-mult-right-eq}[of 3 n 8]$ 
    by auto
    thus ?thesis unfolding  $c\text{-def}$  using  $\langle n \bmod 8 = 1 \rangle$  by auto
  next
    assume  $n \bmod 8 = 3$ 
    thus ?thesis unfolding  $c\text{-def}$  by auto
  next
    assume  $n \bmod 8 = 5$ 
    have  $3 * n \bmod 8 = 7$ 
    using  $\langle n \bmod 8 = 5 \rangle \text{ mod-mult-right-eq}[of 3 n 8]$ 
    by auto
    thus ?thesis unfolding  $c\text{-def}$  using  $\langle n \bmod 8 = 5 \rangle$  by auto
  qed
  hence  $2 * k \bmod 8 = (\text{if } n \bmod 8 = 5 \text{ then } 6 \text{ else } 2)$ 
  unfolding  $H\text{-}k$  using  $\langle c * n \geq 1 \rangle \text{ mod-diff-eq-nat}$  by simp
  hence  $2 * (k \bmod 4) = 2 * ?v$  by (simp add: mult-mod-right)
  thus ?thesis by simp
  qed

have  $\text{coprime } k (4 * n)$ 
  using  $k\text{-mod-4 } H\text{-}k \langle c * n \geq 1 \rangle$ 
  by (metis One-nat-def coprime-Suc-left-nat coprime-commute
        coprime-diff-one-right-nat coprime-mod-left-iff
        coprime-mult-right-iff mult-2-right numeral-2-eq-2 numeral-3-eq-3
        numeral-Bit0 order-less-le-trans zero-less-one zero-neq-numeral)
then obtain  $j$  where  $H\text{-}j$ :
  prime  $(k + (4 * n) * j) \wedge j \neq 0$ 
  using  $\text{prime-linear-combination}[of k n - 1] \langle n > 1 \rangle$ 
  by (metis One-nat-def Suc-leI bot-nat-0.not-eq-extremum mult-is-0
        nat-1-eq-mult-iff nat-less-le prime-linear-combination
        zero-neq-numeral)
have  $j > 0$  using  $H\text{-}j$  by blast

define  $p$  where  $p \equiv k + (4 * n) * j$ 
have  $\text{prime } p$ 
  unfolding  $p\text{-def}$ 
  using  $\text{conjunct1}[OF H\text{-}j]$  apply (rule back-subst[of prime])
  apply (simp add: int-ops nat-plus-as-int)
  done
have  $[p = k] (\bmod 4 * n)$ 
  unfolding  $p\text{-def unique-euclidean-semiring-class.cong-def}$  by auto

have  $p > 2$ 

```

```

using <prime p> <[p = k] (mod 4 * n)> <coprime k (4 * n)>
by (metis cong-dvd-iff cong-dvd-modulus-nat coprime-common-divisor-nat
      dvd-mult2 even-numeral le-neq-implies-less odd-one prime-ge-2-nat)

define d' where d' ≡ 8 * j + c
have d' > 1 unfolding d'-def using <j > 0> by simp
have H-2-p: 2 * p = d' * n - 1
unfolding p-def d'-def
using <c * n ≥ 1> H-k
by (smt (verit, del-insts) Nat.add-diff-assoc add.commute
      add-mult-distrib mult.commute mult-2 numeral-Bit0)

have QuadRes p (– d')
proof –
  have d'-expansion: d' = ( $\prod_{q \in \text{prime-factors } d'} q^{\text{multiplicity } q}$ ) using <j > 0> prime-factorization-nat unfolding d'-def by auto

  have odd d' unfolding c-def d'-def by simp
  hence d'-prime-factors-odd: q ∈ prime-factors d' ⇒ odd q for q
    by fastforce

  have d'-prime-factors-gt-2: q ∈ prime-factors d' ⇒ q > 2 for q
    using d'-prime-factors-odd
    by (metis even-numeral in-prime-factors-imp-prime
         order-less-le prime-ge-2-nat)

  have [2 * p = – 1] (mod d')
    using <n > 1> <d' > 1>
    unfolding H-2-p cong-iff-dvd-diff
    by (simp add: int-ops less-1-mult order-less-imp-not-less)
  hence d'-prime-factors-2-p-mod:
    q ∈ prime-factors d' ⇒ [2 * p = – 1] (mod q) for q
    by (rule cong-dvd-modulus; auto)

  have q ∈ prime-factors d' ⇒ coprime (2 * int p) q for q
    using d'-prime-factors-2-p-mod
    by (metis cong-imp-coprime cong-sym coprime-1-left
         coprime-minus-left-iff mult-2 of-nat-add)
  hence d'-prime-factors-coprime:
    q ∈ prime-factors d' ⇒ coprime (int p) q for q
    using d'-expansion by auto

  have Legendre-using-quadratic-reciprocity:
    Legendre (– d') p =
      ( $\prod_{q \in \text{prime-factors } d'} (\text{Legendre } p \ q)^{\text{multiplicity } q}$ ) using <n mod 8 ∈ {1, 3}>
    proof cases
      assume n mod 8 ∈ {1, 3}

      have k mod 4 = 1 using <n mod 8 ∈ {1, 3}> k-mod-4 by auto

```

```

hence  $p \bmod 4 = 1$ 
  using  $\langle p = k \rangle \bmod 4 * n \rangle$ 
    by (metis unique-euclidean-semiring-class.cong-def cong-modulus-mult-nat)
hence  $[int p = 1] \bmod 4$ 
  by (metis cong-mod-left cong-refl int-ops(2) int-ops(3) of-nat-mod)

have  $d'$ -prime-factors-Legendre:
   $q \in \text{prime-factors } d' \implies$ 
    Legendre  $p q = \text{Legendre } q p$  for  $q$ 
proof -
  assume  $q \in \text{prime-factors } d'$ 
  have prime  $q$  using  $\langle q \in \text{prime-factors } d' \rangle$  by blast
  have  $q > 2$  using  $d'$ -prime-factors-gt-2  $\langle q \in \text{prime-factors } d' \rangle$  by blast
  show Legendre  $p q = \text{Legendre } q p$ 
    using  $\langle \text{prime } p \rangle \langle p > 2 \rangle \langle [int p = 1] \bmod 4 \rangle$ 
       $\langle \text{prime } q \rangle \langle q > 2 \rangle \text{Legendre-equal}[of p q]$ 
    by auto
qed

have Legendre  $(- d') p = \text{Legendre } (- 1) p * \text{Legendre } d' p$ 
  using  $\langle \text{prime } p \rangle \text{Legendre-mult}[of p - 1 d']$  by auto
also have ... = Legendre  $d' p$ 
  using  $\langle \text{prime } p \rangle \langle p > 2 \rangle \langle [int p = 1] \bmod 4 \rangle$  Legendre-minus-one
  by auto
also have ... =  $(\prod_{q \in \text{prime-factors } d'} \text{Legendre } (q \wedge \text{multiplicity } q d')) p$ 
  apply (subst  $d'$ -expansion)
  using  $\langle \text{prime } p \rangle \langle p > 2 \rangle \text{Legendre-prod}[of p]$  apply auto
  done
also have ... =  $(\prod_{q \in \text{prime-factors } d'} (\text{Legendre } q p) \wedge \text{multiplicity } q d')$ 
  using  $\langle \text{prime } p \rangle \langle p > 2 \rangle$  Legendre-power by auto
also have ... =  $(\prod_{q \in \text{prime-factors } d'} (\text{Legendre } p q) \wedge \text{multiplicity } q d')$ 
  using  $d'$ -prime-factors-Legendre by auto
finally show ?thesis .

next
  assume  $n \bmod 8 \notin \{1, 3\}$ 
  hence  $n \bmod 8 = 5$  using assms by auto

have  $[p = 3] \bmod 4$ 
  using  $\langle n \bmod 8 = 5 \rangle \text{k-mod-4} \langle [p = k] \bmod 4 * n \rangle$ 
    by (metis cong-mod-right cong-modulus-mult-nat)
have  $[d' = 3] \bmod 8$ 
  using  $\langle n \bmod 8 = 5 \rangle$ 
  unfolding  $d'$ -def c-def cong-iff-dvd-diff
    by (simp add: unique-euclidean-semiring-class.cong-def)
have  $[d' = -1] \bmod 4$ 
  using  $\langle [d' = 3] \bmod 8 \rangle \text{cong-modulus-mult}[of d' 3 4 2]$ 
  unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int
  by auto

```

```

have  $d'$ -prime-factors-cases:
   $q \in \text{prime-factors } d' \implies$ 
    multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$  for  $q$ 
proof -
  assume  $q \in \text{prime-factors } d'$ 
  consider  $[q = 0] \pmod{4}$ 
    |  $[q = 1] \pmod{4}$ 
    |  $[q = 2] \pmod{4}$ 
    |  $[q = 3] \pmod{4}$ 
  unfolding unique-euclidean-semiring-class.cong-def by (simp; linarith)
  thus multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$ 
  proof cases
    assume  $[q = 0] \pmod{4}$ 
    hence False
    using  $d'$ -prime-factors-odd  $\langle q \in \text{prime-factors } d' \rangle$ 
    by (meson cong-0-iff dvd-trans even-numeral)
    thus multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$ 
      by blast
  next
    assume  $[q = 1] \pmod{4}$ 
    thus multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$ 
      by blast
  next
    assume  $[q = 2] \pmod{4}$ 
    hence  $q = 2$ 
    using  $d'$ -prime-factors-odd  $\langle q \in \text{prime-factors } d' \rangle$ 
    by (metis unique-euclidean-semiring-class.cong-def
        dvd-mod-iff even-numeral)
    thus multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$ 
      by (simp add: ⟨odd  $d'$ ⟩ not-dvd-imp-multiplicity-0)
  next
    assume  $[q = 3] \pmod{4}$ 
    thus multiplicity  $q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4}$ 
      by blast
qed
qed

have  $d' = (\prod_{q \in \{\text{prime-factors } d'\}} \text{True}) \cdot q \wedge \text{multiplicity } q d'$ 
  using  $d'$ -expansion by auto
also have ... =  $(\prod_{q \in \{\text{prime-factors } d'\}} \text{multiplicity } q d' = 0 \vee [q = 1] \pmod{4} \vee [q = 3] \pmod{4})$ .
  q  $\wedge$  multiplicity  $q d'$ 
using  $d'$ -prime-factors-cases by meson
also have ... =  $(\prod_{q \in \{\text{prime-factors } d'\}} \text{multiplicity } q d' = 0) \cup$ 
   $\{q \in \{\text{prime-factors } d'\} \mid [q = 1] \pmod{4} \vee [q = 3] \pmod{4}\} \cdot q \wedge \text{multiplicity } q d'$ 
by (rule prod.cong; blast)
also have ... =  $(\prod_{q \in \{\text{prime-factors } d'\}} [q = 1] \pmod{4} \vee [q = 3] \pmod{4}) \cdot q \wedge \text{multiplicity } q d'$ 

```

```

by (rule prod.mono-neutral-left[symmetric]; auto)
also have ... = ( $\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}} \cup$ 
 $\{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}.$ 
 $q \wedge^{\text{multiplicity}} q d')$ 
by (rule prod.cong; blast)
also have ... = ( $\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d') *$ 
 $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d')$ 
by (rule prod.union-disjoint;
auto simp add: unique-euclidean-semiring-class.cong-def)
finally have  $d'$ -expansion-mod-4:
 $d' = (\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d') *$ 
 $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d').$ 

have  $\text{int } d' \pmod{4} = \text{int } ((\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d') *$ 
 $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $q \wedge^{\text{multiplicity}} q d') \pmod{4})$ 
using  $d'$ -expansion-mod-4
by presburger
also have ... = ( $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}}.$ 
 $((q \pmod{4}) \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} *$ 
 $((\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $((q \pmod{4}) \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} \pmod{4}$ 
unfolding mod-mult-eq mod-prod-eq power-mod ..
also have ... =  $\text{int } (((\prod_{q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}}.$ 
 $(1 \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} *$ 
 $((\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $(3 \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} \pmod{4})$ 
unfolding unique-euclidean-semiring-class.cong-def by auto
also have ... = ( $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $((\text{int } 3) \pmod{4}) \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} \pmod{4}$ 
by (simp add: int-ops)
also have ... = ( $(\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $(((-1) \pmod{4}) \wedge^{\text{multiplicity}} q d') \pmod{4}) \pmod{4} \pmod{4}$ 
by auto
also have ... = ( $\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $((-1) \wedge^{\text{multiplicity}} q d') \pmod{4})$ 
unfolding power-mod mod-prod-eq mod-mod-trivial ..
finally have [ $d' = \prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $((-1) \wedge^{\text{multiplicity}} q d') \pmod{4})$ 
unfolding unique-euclidean-semiring-class.cong-def .
hence [ $\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}}.$ 
 $((-1) \wedge^{\text{multiplicity}} q d') = -1 :: \text{int} \pmod{4}$ )
using ⟨ $d' = -1 \pmod{4}$ ⟩
unfolding unique-euclidean-semiring-class.cong-def

```

```

by argo
hence prod-d'-prime-factors-q-3-mod-4-minus-one:
  ( $\prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}} q$ ) =
    ((-1) ^ multiplicity q d') = (-1 :: int)
  unfolding power-sum[symmetric]
  unfolding minus-one-power-if unique-euclidean-semiring-class.cong-def
  by auto

have d'-prime-factors-q-1-mod-4-Legendre:
  q ∈ prime-factors d' ==>
  [q = 1] (mod 4) ==>
  Legendre p q = Legendre q p for q
proof -
  assume q ∈ prime-factors d'
  assume [q = 1] (mod 4)
  have prime q using <math>\langle q \in \text{prime-factors } d' \rangle</math> by blast
  have q > 2 using d'-prime-factors-gt-2 <math>\langle q \in \text{prime-factors } d' \rangle</math> by blast
  show Legendre p q = Legendre q p
    using <math>\langle \text{prime } p \rangle \langle p > 2 \rangle \langle [p = 3] \pmod{4} \rangle \langle [q = 1] \pmod{4} \rangle</math>
      <math>\langle \text{prime } q \rangle \langle q > 2 \rangle \text{Legendre-equal}[of p q]</math>
    unfolding unique-euclidean-semiring-class.cong-def
    using zmod-int[of q 4]
    by auto
qed

have d'-prime-factors-q-3-mod-4-Legendre:
  q ∈ prime-factors d' ==>
  [q = 3] (mod 4) ==>
  Legendre p q = -Legendre q p for q
proof -
  assume q ∈ prime-factors d'
  assume [q = 3] (mod 4)
  have prime q using <math>\langle q \in \text{prime-factors } d' \rangle</math> by blast
  have q > 2 using d'-prime-factors-gt-2 <math>\langle q \in \text{prime-factors } d' \rangle</math> by blast
  have p ≠ q
    using d'-prime-factors-coprime[of q] <math>\langle q \in \text{prime-factors } d' \rangle \langle \text{prime } p \rangle</math>
    by auto
  show Legendre p q = -Legendre q p
    using <math>\langle \text{prime } p \rangle \langle p > 2 \rangle \langle [p = 3] \pmod{4} \rangle \langle [q = 3] \pmod{4} \rangle</math>
      <math>\langle \text{prime } q \rangle \langle q > 2 \rangle \langle p \neq q \rangle \text{Legendre-opposite}[of p q]</math>
    unfolding unique-euclidean-semiring-class.cong-def
    using zmod-int[of p 4] zmod-int[of q 4]
    by fastforce
qed

have d'-prime-factors-q-0-2-mod-4:
  q ∈ prime-factors d' ==>
  ([q = 0] (mod 4) ∨ [q = 2] (mod 4)) ==>
  Legendre p q = 1 for q

```

```

unfolding unique-euclidean-semiring-class.cong-def
using d'-prime-factors-odd mod-mod-cancel[of 2 4 q]
by fastforce

have Legendre (- d') p = Legendre (- 1) p * Legendre d' p
  using <prime p> Legendre-mult[of p - 1 d'] by auto
also have ... = - Legendre d' p
  using <prime p> <p > 2> <[p = 3] (mod 4)> Legendre-minus-one[of p]
  unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int
  by (auto simp add: cong-0-iff Legendre-def)
also have ... = - ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre q p)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (Legendre q p)  $\wedge$  multiplicity q d'
apply (subst d'-expansion-mod-4)
using <prime p> <p > 2> Legendre-mult[of p]
  Legendre-prod[of p  $\lambda q. q \wedge$  multiplicity q d'] Legendre-power[of p]
apply simp
done
also have ... = - ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (- Legendre p q)  $\wedge$  multiplicity q d'
using d'-prime-factors-q-1-mod-4-Legendre
  d'-prime-factors-q-3-mod-4-Legendre
by auto
also have ... = - ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (Legendre p q * (- 1))  $\wedge$  multiplicity q d'
by auto
also have ... = - ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (- 1)  $\wedge$  multiplicity q d'
unfolding power-mult-distrib prod.distrib by auto
also have ... = ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 3] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
unfolding prod-d'-prime-factors-q-3-mod-4-minus-one by auto
also have ... = ( $\prod_{q \in \text{prime-factors } d'} [q = 0] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 1] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'
  ( $\prod_{q \in \text{prime-factors } d'} [q = 2] \pmod{4}$ )
  (Legendre p q)  $\wedge$  multiplicity q d'

```

```


$$\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}} (\text{Legendre } p q) \wedge \text{multiplicity } q d' \right)$$

using  $d'$ -prime-factors-q-0-2-mod-4 by auto
also have ... =  $\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 0] \pmod{4}\}} \cup \right.$ 

$$\left. \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\} \right)$$


$$(\text{Legendre } p q) \wedge \text{multiplicity } q d' *$$


$$\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 2] \pmod{4}\}} \cup \right.$$


$$\left. \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\} \right)$$


$$(\text{Legendre } p q) \wedge \text{multiplicity } q d'$$

using prod.union-disjoint[of { $q \in \text{prime-factors } d'. [q = 0] \pmod{4}$ }]

$$\{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\}$$


$$\lambda q. (\text{Legendre } p (\text{int } q)) \wedge$$


$$\text{multiplicity } q d'$$

prod.union-disjoint[of { $q \in \text{prime-factors } d'. [q = 2] \pmod{4}$ }]

$$\{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\}$$


$$\lambda q. (\text{Legendre } p (\text{int } q)) \wedge$$


$$\text{multiplicity } q d'$$

by (fastforce simp add: unique-euclidean-semiring-class.cong-def)
also have ... =  $\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 0] \pmod{4}\}} \cup \right.$ 

$$\left. \{q \in \text{prime-factors } d'. [q = 1] \pmod{4}\} \right) \cup$$


$$\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 2] \pmod{4}\}} \cup \right.$$


$$\left. \{q \in \text{prime-factors } d'. [q = 3] \pmod{4}\} \right)$$


$$(\text{Legendre } p q) \wedge \text{multiplicity } q d'$$

by (rule prod.union-disjoint[symmetric];
auto simp add: unique-euclidean-semiring-class.cong-def)
also have ... =  $\left( \prod_{q \in \{q \in \text{prime-factors } d'. [q = 0] \pmod{4}\}} \vee \right.$ 

$$\left. [q = 1] \pmod{4} \vee [q = 2] \pmod{4} \vee [q = 3] \pmod{4} \right)$$


$$(\text{Legendre } p q) \wedge \text{multiplicity } q d'$$

by (rule prod.cong; auto)
also have ... =  $\left( \prod_{q \in \text{prime-factors } d'.} (\text{Legendre } p q) \wedge \text{multiplicity } q d' \right)$ 
by (rule prod.cong;
auto simp add: unique-euclidean-semiring-class.cong-def)
finally show ?thesis .
qed

```

```

have  $q \in \text{prime-factors } d' \implies \text{Legendre } 4 q = 1$  for  $q$ 
proof -
assume  $q \in \text{prime-factors } d'$ 
have  $q \text{ dvd } 4 \implies q \leq 4$  by (simp add: dvd-imp-le)
hence  $q \text{ dvd } 4 \implies q \in \{0, 1, 2, 3, 4\}$  by auto
hence  $q \text{ dvd } 4 \implies q \in \{1, 2, 4\}$  by auto
hence  $\neg q \text{ dvd } 4$  using ‹ $q \in \text{prime-factors } d'$ ›  $d'$ -prime-factors-odd[of  $q$ ]
by (metis empty-iff even-numeral in-prime-factors-imp-prime
    insert-iff not-prime-1)
hence  $\neg \text{int } q \text{ dvd } 4$  by presburger
thus  $\text{Legendre } 4 q = 1$ 
unfolding Legendre-def QuadRes-def cong-0-iff power2-eq-square
by (metis cong-refl mult-2 numeral-Bit0)

```

```

qed
hence Legendre ( $- d'$ )  $p = (\prod q \in \text{prime-factors } d')$ .
  ( $\text{Legendre } (2 * 2) q * \text{Legendre } p q \wedge \text{multiplicity } q d'$ )
  using Legendre-using-quadratic-reciprocity by auto
also have ... =  $(\prod q \in \text{prime-factors } d')$ .
  ( $\text{Legendre } 2 q * \text{Legendre } (2 * p) q \wedge \text{multiplicity } q d'$ )
apply (rule prod.cong[OF refl])
using  $d'$ -prime-factors-gt-2 Legendre-mult in-prime-factors-imp-prime
by (metis int-ops(7) of-nat-numeral prime-nat-int-transfer mult.assoc)
also have ... =  $(\prod q \in \text{prime-factors } d')$ .
  ( $\text{Legendre } 2 q * \text{Legendre } (- 1) q \wedge \text{multiplicity } q d'$ )
apply (rule prod.cong[OF refl])
using  $d'$ -prime-factors-2-p-mod Legendre-cong
unfolding unique-euclidean-semiring-class.cong-def
apply metis
done
also have ... =  $(\prod q \in \text{prime-factors } d')$ .
  ((if [ $q = 1$ ] (mod 8)  $\vee$  [ $q = 7$ ] (mod 8) then 1 else -1) *
   (if [ $q = 1$ ] (mod 4) then 1 else -1))  $\wedge$  multiplicity  $q d'$ )
apply (rule prod.cong[OF refl])
subgoal for  $q$ 
apply (rule arg-cong2[of  $\lambda a. b. (a * b)$   $\wedge$  multiplicity  $q d'$ ])
subgoal
  using Legendre-two-alt[of  $q$ ]  $d'$ -prime-factors-gt-2[of  $q$ ]
  unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int
  by force
subgoal
  using Legendre-minus-one-alt[of  $q$ ]  $d'$ -prime-factors-gt-2[of  $q$ ]
  unfolding unique-euclidean-semiring-class.cong-def nat-mod-as-int
  by force
done
done
also have ... =  $(\prod q \in \text{prime-factors } d')$ .
  ((if [ $q = 5$ ] (mod 8)  $\vee$  [ $q = 7$ ] (mod 8) then -1 else 1))  $\wedge$ 
  multiplicity  $q d'$ )
apply (rule prod.cong)
subgoal by blast
subgoal for  $q$ 
  apply (rule arg-cong[of  $\lambda a. a$   $\wedge$  multiplicity  $q d'$ ])
  unfolding unique-euclidean-semiring-class.cong-def apply (simp; pres-
burger)
done
done
also have ... =  $(\prod q \in \text{prime-factors } d')$ .
  (if [ $q = 5$ ] (mod 8)  $\vee$  [ $q = 7$ ] (mod 8)
   then (-1)  $\wedge$  multiplicity  $q d'$  else 1))
by (rule prod.cong; auto)
also have ... =  $(\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)} \vee [q = 7] \text{ (mod 8)}\})$ .

```

```

 $(- 1) \wedge \text{multiplicity } q d'$ 
using prod.inter-filter[symmetric] by fast
also have ... =  $(- 1) \wedge (\sum_{q \in \{q \in \text{prime-factors } d'\}} [q = 5] \pmod{8} \vee [q = 7] \pmod{8})$ .
 $\text{multiplicity } q d'$ 
by (simp add: power-sum)
finally have Legendre-using-sum:
Legendre  $(- d') p =$ 
 $(- 1) \wedge (\sum_{q \in \{q \in \text{prime-factors } d'\}} [q = 5] \pmod{8} \vee [q = 7] \pmod{8})$ .
 $\text{multiplicity } q d' .$ 

have  $[\sum_{q \in \{q \in \text{prime-factors } d'\}} [q = 5] \pmod{8} \vee [q = 7] \pmod{8}]$ .
 $\text{multiplicity } q d' = 0 \pmod{2}$ 
proof -
have  $d' = (\prod_{q \in \{q \in \text{prime-factors } d'\}} [q = 1] \pmod{8} \vee [q = 3] \pmod{8} \vee$ 
 $[q = 5] \pmod{8} \vee [q = 7] \pmod{8})$ .  $q \wedge \text{multiplicity } q d'$ 
apply (subst d'-expansion)
apply (rule prod.cong)
subgoal
apply (rule Set.set-eqI)
subgoal for q
apply (rule iffI)
subgoal
using d'-prime-factors-odd[of q]
unfolding unique-euclidean-semiring-class.cong-def
apply simp
apply presburger
done
subgoal by blast
done
done
subgoal by blast
done
also have ... =  $(\prod_{q \in \{\{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\}} [q = 1] \pmod{8} \vee$ 
 $[q = 3] \pmod{8} \vee [q = 5] \pmod{8} \vee [q = 7] \pmod{8})$ .
 $q \wedge \text{multiplicity } q d'$ 
by (rule prod.cong; auto)
also have ... =  $(\prod_{q \in \{\{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\}} [q = 1] \pmod{8} \vee$ 
 $[q = 3] \pmod{8})$ .
 $q \wedge \text{multiplicity } q d' *$ 
 $(\prod_{q \in \{\{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\} \cup \{q \in \text{prime-factors } d'\}} [q = 5] \pmod{8} \vee$ 
 $[q = 7] \pmod{8})$ .
 $q \wedge \text{multiplicity } q d'$ 
by (rule prod.union-disjoint;
force simp add: unique-euclidean-semiring-class.cong-def)
also have ... =  $(\prod_{q \in \{q \in \text{prime-factors } d'\}} [q = 1] \pmod{8})$ .

```

```


$$q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\} \\ q \wedge \text{multiplicity } q d')$$

using prod.union-disjoint[of { $q \in \text{prime-factors } d'. [q = 1] \pmod{8}$ }  

{ $q \in \text{prime-factors } d'. [q = 3] \pmod{8}$ }  

 $\lambda q. q \wedge \text{multiplicity } q d'$ ]  

prod.union-disjoint[of { $q \in \text{prime-factors } d'. [q = 5] \pmod{8}$ }  

{ $q \in \text{prime-factors } d'. [q = 7] \pmod{8}$ }  

 $\lambda q. q \wedge \text{multiplicity } q d'$ ]

by (force simp add: unique-euclidean-semiring-class.cong-def)
finally have int (d' mod 8) = ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' * \\ (\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\} \\ q \wedge \text{multiplicity } q d') \bmod 8$ )
by auto
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\} \\ q \wedge \text{multiplicity } q d' \bmod 8$ )
by (metis (no-types, lifting) mod-mult-eq mod-mod-trivial)
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{8}\} \\ (q \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \\ (q \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \\ (q \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\} \\ (q \wedge \text{multiplicity } q d' \bmod 8) \bmod 8$ )
unfolding mod-prod-eq ..
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \pmod{8}\} \\ ((q \bmod 8) \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \\ ((q \bmod 8) \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \\ ((q \bmod 8) \wedge \text{multiplicity } q d' \bmod 8) \bmod 8 * \\ ((\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\} \\ ((q \bmod 8) \wedge \text{multiplicity } q d' \bmod 8) \bmod 8$ )

```

```

unfolding power-mod ..
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \text{ (mod 8)}\}$ .
    (((int q) mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
        (((int q) mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)}\}$ .
        (((int q) mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \text{ (mod 8)}\}$ .
        (((int q) mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (((int q) mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8

by (simp add: int-ops)
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \text{ (mod 8)}\}$ .
    ((1 mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
        ((3 mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)}\}$ .
        ((-3 mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \text{ (mod 8)}\}$ .
        ((-1 mod 8)  $\wedge$  multiplicity q d') mod 8) mod 8) mod 8

unfolding cong-int-iff[symmetric] unique-euclidean-semiring-class.cong-def
by auto
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \text{ (mod 8)}\}$ .
    (1  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
        (3  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)}\}$ .
        ((-3)  $\wedge$  multiplicity q d') mod 8) mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \text{ (mod 8)}\}$ .
        ((-1)  $\wedge$  multiplicity q d') mod 8) mod 8) mod 8

unfolding power-mod ..
also have ... = (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \text{ (mod 8)}\}$ .
    1  $\wedge$  multiplicity q d') mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
        3  $\wedge$  multiplicity q d') mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)}\}$ .
        (-3)  $\wedge$  multiplicity q d') mod 8) *
    (( $\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \text{ (mod 8)}\}$ .
        (-1)  $\wedge$  multiplicity q d') mod 8) mod 8

unfolding mod-prod-eq ..
also have ... = ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 1] \text{ (mod 8)}\}$ .
    1  $\wedge$  multiplicity q d') *
    ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
        3  $\wedge$  multiplicity q d') *
    ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \text{ (mod 8)}\}$ .
        (-3)  $\wedge$  multiplicity q d') *
    ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \text{ (mod 8)}\}$ .
        (-1)  $\wedge$  multiplicity q d') mod 8

by (metis (no-types, lifting) mod-mult-eq mod-mod-trivial)
also have ... = ( $\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \text{ (mod 8)}\}$ .
    3  $\wedge$  multiplicity q d') *

```

```


$$(\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\}.$$


$$(- 3) \wedge \text{multiplicity } q \text{ } d' *$$


$$(\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\}.$$


$$(- 1) \wedge \text{multiplicity } q \text{ } d' \bmod 8$$


by auto
also have ... =  $(\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d' *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d' * (- 1) \wedge \text{multiplicity } q \text{ } d' *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d' \bmod 8$ 

unfolding power-mult-distrib[symmetric] by auto
also have ... =  $((\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d' *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d') *$ 
 $((\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d') *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d') \bmod 8$ 

unfolding prod.distrib by (simp add: algebra-simps)
also have ... =  $((\prod q \in \{q \in \text{prime-factors } d'. [q = 3] \pmod{8}\} \cup$ 
 $\{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d' *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'. [q = 5] \pmod{8}\} \cup$ 
 $\{q \in \text{prime-factors } d'. [q = 7] \pmod{8}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d') \bmod 8$ 

apply (subst
prod.union-disjoint[of {q∈prime-factors d'. [q = 5] (mod 8)}
{q∈prime-factors d'. [q = 7] (mod 8)}
λq. (- 1) ^ multiplicity q d']
 $)$ 
apply ((force simp add: unique-euclidean-semiring-class.cong-def)+)[3]
apply (subst
prod.union-disjoint[of {q∈prime-factors d'. [q = 3] (mod 8)}
{q∈prime-factors d'. [q = 5] (mod 8)}
λq. 3 ^ multiplicity q d']
 $)$ 
apply ((force simp add: unique-euclidean-semiring-class.cong-def)+)[3]
apply blast
done
also have ... =  $((\prod q \in \{q \in \text{prime-factors } d'.$ 
 $[q = 3] \pmod{8} \vee [q = 5] \pmod{8}\}.$ 
 $3 \wedge \text{multiplicity } q \text{ } d' *$ 
 $(\prod q \in \{q \in \text{prime-factors } d'.$ 
 $[q = 5] \pmod{8} \vee [q = 7] \pmod{8}\}.$ 
 $(- 1) \wedge \text{multiplicity } q \text{ } d') \bmod 8$ 
by (rule arg-cong2[of - - - λA B. ((prod q∈A. - q) * (prod q∈B. - q)) mod 8];
auto)

```

```

also have ... = ((( $\prod q \in \{q \in \text{prime-factors } d'\}$ .
                   $[q = 3] \pmod{8} \vee [q = 5] \pmod{8}\}.$ 
                   $3 \wedge \text{multiplicity } q \text{ } d' \pmod{8}) *$ 
                  ( $\left(\prod q \in \{q \in \text{prime-factors } d'\}$ .
                       $[q = 5] \pmod{8} \vee [q = 7] \pmod{8}\}.$ 
                       $(-1) \wedge \text{multiplicity } q \text{ } d' \pmod{8}) \pmod{8}$ 
unfolding mod-mult-eq ..
also have ... = (( $3 \wedge (\sum q \in \{q \in \text{prime-factors } d'\}$ .
                   $[q = 3] \pmod{8} \vee [q = 5] \pmod{8}\}.$ 
                   $\text{multiplicity } q \text{ } d' \pmod{8}) *$ 
                  ( $(-1) \wedge (\sum q \in \{q \in \text{prime-factors } d'\}$ .
                       $[q = 5] \pmod{8} \vee [q = 7] \pmod{8}\}.$ 
                       $\text{multiplicity } q \text{ } d' \pmod{8}) \pmod{8}$ 
unfolding power-sum ..
also have ... =
int (case (( $\sum q \in \{q \in \text{prime-factors } d'\}$ .
               $[q = 3] \pmod{8} \vee [q = 5] \pmod{8}\}.$ 
               $\text{multiplicity } q \text{ } d' \pmod{2},$ 
              ( $\sum q \in \{q \in \text{prime-factors } d'\}$ .
                   $[q = 5] \pmod{8} \vee [q = 7] \pmod{8}\}.$ 
                   $\text{multiplicity } q \text{ } d' \pmod{2}) \text{ of}$ 
                  ( $(0, 0) \Rightarrow 1$ 
                  | ( $0, \text{Suc } 0) \Rightarrow 7$ 
                  | ( $\text{Suc } 0, 0) \Rightarrow 3$ 
                  | ( $\text{Suc } 0, \text{Suc } 0) \Rightarrow 5$ ) (is - = int ?d'-mod-8)
unfolding three-mod-eight-power-iff minus-one-power-iff
by (simp; simp add: odd-iff-mod-2-eq-one)
finally have d'-mod-8:  $d' \pmod{8} = ?d' \text{-mod-8}$  by linarith

have  $[d' = 1] \pmod{8} \vee [d' = 3] \pmod{8}$ 
unfolding d'-def c-def unique-euclidean-semiring-class.cong-def
using assms
by auto
hence ?d'-mod-8 = 1  $\vee ?d' \text{-mod-8} = 3$ 
unfolding unique-euclidean-semiring-class.cong-def d'-mod-8 by auto
thus ?thesis
unfolding unique-euclidean-semiring-class.cong-def
by (smt (verit) Collect-cong One-nat-def cong-exp-iff-simps(11)
      even-mod-2-iff even-numeral nat.case(2) numeral-eq-iff
      numerals(1) old.nat.simps(4) parity-cases prod.simps(2)
      semiring-norm(84))
qed
hence Legendre (- d') p = 1
using Legendre-using-sum
unfolding minus-one-power-iff cong-0-iff
by argo
thus QuadRes p (- d')
unfolding Legendre-def
by (metis one-neq-neg-one one-neq-zero)

```

qed

```
from ⟨QuadRes p (– d')⟩ obtain x₀ y where x₀² – (– d') = y * (int p)
  unfolding quadratic-residue-alt-equiv[symmetric]
    quadratic-residue-alt-def
  by auto
hence (int p) dvd (x₀² – – d') by simp

define x :: int where x ≡ if odd x₀ then x₀ else (x₀ + p)

have even (4 * int n * j) by simp
moreover have odd k using ⟨coprime k (4 * n)⟩ by auto
ultimately have odd (int p) unfolding p-def by simp

have odd x unfolding x-def using ⟨odd (int p)⟩ by auto

have QuadRes (2 * p) (– d')
  unfolding quadratic-residue-alt-equiv[symmetric]
    quadratic-residue-alt-def
proof –
  have 2 dvd (x² – – d') unfolding d'-def c-def using ⟨odd x⟩ by auto
  moreover from ⟨(int p) dvd (x₀² – – d')⟩
  have (int p) dvd (x² – – d')
    unfolding x-def power2-eq-square
    apply (simp add: algebra-simps)
    unfolding add.assoc[symmetric, of d' x₀ * x₀]
    apply auto
    done
  moreover have coprime 2 (int p) using ⟨odd (int p)⟩ by auto
  ultimately have (int (2 * p)) dvd (x² – – d') by (simp add: divides-mult)
  hence (x² – – d') mod (int (2 * p)) = 0 by simp
  hence ∃ y. x² – – d' = int (2 * p) * y by auto
  hence ∃ y. x² – – d' = y * int (2 * p) by (simp add: algebra-simps)
  thus ∃ x y. x² – – d' = y * int (2 * p) by (rule exI[where ?x=x])
qed

have n ≥ 2 using ⟨1 < n⟩ by auto
moreover have 0 < nat d' unfolding d'-def using ⟨j > 0⟩ by simp
moreover have QuadRes (int (nat d' * n – 1)) (– d')
  using ⟨d' > 1⟩ H-2-p ⟨QuadRes (2 * p) (– d')⟩ by (simp add: int-ops)
ultimately show ∃ x₁ x₂ x₃. n = x₁² + x₂² + x₃²
  using three-squares-using-quadratic-residue[of n nat d']
  by auto
qed

lemma power-two-mod-eight:
  fixes n :: nat
  shows n² mod 8 ∈ {0, 1, 4}
proof –
```

```

have 0:  $n^2 \bmod 8 = (n \bmod 8)^2 \bmod 8$ 
  unfolding power2-eq-square by (simp add: mod-mult-eq)
have  $n \bmod 8 \in \{0, 1, 2, 3, 4, 5, 6, 7\}$  by auto
hence  $(n \bmod 8)^2 \bmod 8 \in \{0, 1, 4\}$ 
  unfolding power2-eq-square by auto
thus  $n^2 \bmod 8 \in \{0, 1, 4\}$  unfolding 0 .
qed

```

```

lemma power-two-mod-four:
  fixes n :: nat
  shows  $n^2 \bmod 4 \in \{0, 1\}$ 
  using power-two-mod-eight[of n] mod-mod-cancel[of 4 8  $n^2$ ] by auto

```

Theorem 1.4 from [1].

```

theorem three-squares-iff:
  fixes n :: nat
  shows  $(\exists x_1 x_2 x_3. n = x_1^2 + x_2^2 + x_3^2) \longleftrightarrow (\nexists a k. n = 4 \wedge a * (8 * k + 7))$ 
proof
  assume  $\exists x_1 x_2 x_3. n = x_1^2 + x_2^2 + x_3^2$ 
  then obtain x1 x2 x3 where 0:  $n = x_1^2 + x_2^2 + x_3^2$  by blast
  show  $\nexists a k. n = 4 \wedge a * (8 * k + 7)$ 
proof
  assume  $\exists a k. n = 4 \wedge a * (8 * k + 7)$ 
  then obtain a k where 1:  $n = 4 \wedge a * (8 * k + 7)$  by blast
  from 0 1 show False
proof (induction a arbitrary: n x1 x2 x3 rule: nat.induct)
  fix n x1 x2 x3 :: nat
  assume 2:  $n = x_1^2 + x_2^2 + x_3^2$ 
  assume 3:  $n = 4 \wedge 0 * (8 * k + 7)$ 
  hence 3:  $n \bmod 8 = 7$  unfolding 1 by auto
  have  $(x_1^2 \bmod 8 + x_2^2 \bmod 8 + x_3^2 \bmod 8) \bmod 8 = 7$ 
    unfolding 2 3[symmetric]
    by (meson mod-add-cong mod-mod-trivial)
  thus False
    using power-two-mod-eight[of x1]
      power-two-mod-eight[of x2]
        power-two-mod-eight[of x3]
    by auto
next
  fix a' n x1 x2 x3 :: nat
  assume 4:  $\bigwedge n' x_1' x_2' x_3' :: nat.$ 
     $n' = x_1'^2 + x_2'^2 + x_3'^2 \implies n' = 4 \wedge a' * (8 * k + 7) \implies False$ 
  assume 5:  $n = x_1^2 + x_2^2 + x_3^2$ 
  assume 6:  $n = 4 \wedge Suc a' * (8 * k + 7)$ 
  hence 6:  $4 * ?m = x_1^2 + x_2^2 + x_3^2$  unfolding 5 by auto
  have  $(x_1^2 + x_2^2 + x_3^2) \bmod 4 = 0$  using 6 by presburger
  hence  $((x_1^2 \bmod 4) + (x_2^2 \bmod 4) + (x_3^2 \bmod 4)) \bmod 4 = 0$ 
    by presburger

```

```

hence  $x_1^2 \bmod 4 = 0 \wedge x_2^2 \bmod 4 = 0 \wedge x_3^2 \bmod 4 = 0$ 
  using power-two-mod-four[of  $x_1$ ]
    power-two-mod-four[of  $x_2$ ]
      power-two-mod-four[of  $x_3$ ]
    by (auto; presburger)
hence even  $x_1 \wedge$  even  $x_2 \wedge$  even  $x_3$ 
  by (metis dvd-0-right even-even-mod-4-iff even-power)
hence  $4 * ?m = 4 * ((x_1 \bmod 2)^2 + (x_2 \bmod 2)^2 + (x_3 \bmod 2)^2)$ 
  unfolding 6 by fastforce
hence  $?m = (x_1 \bmod 2)^2 + (x_2 \bmod 2)^2 + (x_3 \bmod 2)^2$  by auto
thus False using 4 by blast
qed
qed
next
assume 7:  $\nexists a k. n = 4 \wedge a * (8 * k + 7)$ 
show  $\exists x_1 x_2 x_3. n = x_1^2 + x_2^2 + x_3^2$ 
proof cases
  assume  $n = 0$ 
  thus  $\exists x_1 x_2 x_3. n = x_1^2 + x_2^2 + x_3^2$  by auto
next
assume 8:  $n \neq 0$ 
have  $n > 0 \implies \exists a m. n = 4 \wedge a * m \wedge \neg 4 \bmod m$ 
proof (induction n rule: less-induct)
  fix  $n :: nat$ 
  assume 9:  $\bigwedge n'. n' < n \implies n' > 0 \implies \exists a' m'. n' = 4 \wedge a' * m' \wedge \neg 4 \bmod m'$ 
  assume 10:  $n > 0$ 
  show  $\exists a m. n = 4 \wedge a * m \wedge \neg 4 \bmod m$ 
  proof cases
    assume 11:  $4 \bmod n$ 
    have  $n \bmod 4 < n$   $n \bmod 4 > 0$  using 10 11 by auto
    then obtain  $a' m'$  where 12:  $n \bmod 4 = 4 \wedge a' * m' \wedge \neg 4 \bmod m'$ 
      using 9 by blast
    have  $n = 4 \wedge (Suc a') * m' \wedge \neg 4 \bmod m'$ 
      using 11 12 by auto
    thus  $\exists a m. n = 4 \wedge a * m \wedge \neg 4 \bmod m$  by blast
  next
  assume  $\neg 4 \bmod n$ 
  hence  $n = 4 \wedge 0 * n \wedge \neg 4 \bmod n$  by auto
  thus  $\exists a m. n = 4 \wedge a * m \wedge \neg 4 \bmod m$  by blast
qed
qed
then obtain  $a m$  where 13:  $n = 4 \wedge a * m \wedge \neg 4 \bmod m$  using 8 by auto
have 14:  $m \bmod 8 \neq 7$ 
proof
  assume  $m \bmod 8 = 7$ 
  then obtain  $k$  where  $m = 8 * k + 7$  by (metis div-mod-decomp mult.commute)
  hence  $n = 4 \wedge a * (8 * k + 7)$  unfolding 13 by blast
  thus False using 7 by blast

```

```

qed
have m mod 4 = 2 ∨ m mod 8 ∈ {1, 3, 5, 7}
  using 13 by (simp; presburger)
hence m mod 4 = 2 ∨ m mod 8 ∈ {1, 3, 5}
  using 14 by blast
hence ∃ x1 x2 x3. m = x12 + x22 + x32
  using three-squares-using-mod-four three-squares-using-mod-eight
  by blast
hence ∃ x1 x2 x3. n = (2 ^ a * x1)2 + (2 ^ a * x2)2 + (2 ^ a * x3)2
  unfolding 13 power2-eq-square
  unfolding mult.assoc[of 2 ^ a]
  unfolding mult.commute[of 2 ^ a]
  unfolding mult.assoc
  unfolding power-add[symmetric]
  unfolding mult-2[symmetric]
  unfolding power-mult
  unfolding mult.assoc[symmetric]
  unfolding add-mult-distrib[symmetric]
  unfolding mult.commute[of 4 ^ a]
  by simp
thus ∃ x1 x2 x3. n = x12 + x22 + x32 by blast
qed
qed

```

Theorem 1.5 from [1].

```

theorem odd-three-squares-using-mod-eight:
fixes n :: nat
assumes n mod 8 = 3
shows ∃ x1 x2 x3. odd x1 ∧ odd x2 ∧ odd x3 ∧ n = x12 + x22 + x32
proof -
obtain x1 x2 x3 where 0: n = x12 + x22 + x32
  using assms three-squares-using-mod-eight by blast
have (x12 mod 8 + x22 mod 8 + x32 mod 8) mod 8 = 3
  unfolding 0 assms[symmetric]
  by (meson mod-add-cong mod-mod-trivial)
hence x12 mod 8 = 1 ∧ x22 mod 8 = 1 ∧ x32 mod 8 = 1
  using power-two-mod-eight[of x1]
    power-two-mod-eight[of x2]
    power-two-mod-eight[of x3]
  by auto
hence odd x1 ∧ odd x2 ∧ odd x3
  by (metis dvd-mod even-numeral even-power odd-one pos2)
hence odd x1 ∧ odd x2 ∧ odd x3 ∧ n = x12 + x22 + x32 using 0 by blast
thus ∃ x1 x2 x3. odd x1 ∧ odd x2 ∧ odd x3 ∧ n = x12 + x22 + x32 by blast
qed

```

## 4.2 Consequences

lemma four-decomposition:

```

fixes n :: nat
shows  $\exists x y z. n = x^2 + y^2 + z^2 + z$ 
proof -
have  $(4 * n + 1) \text{ mod } 8 \in \{1, 3, 5\}$  by (simp; presburger)
then obtain x y z where 0:  $4 * n + 1 = x^2 + y^2 + z^2$ 
  using three-squares-using-mod-eight by blast
hence 1:  $1 = (x^2 + y^2 + z^2) \text{ mod } 4$ 
  by (metis Suc-0-mod-numeral(2) Suc-eq-plus1 mod-add-left-eq
       mod-mult-self1-is-0)
show ?thesis
proof cases
  assume even x
  then obtain x' where H-x:  $x = 2 * x'$  by blast
  show ?thesis
proof cases
  assume even y
  then obtain y' where H-y:  $y = 2 * y'$  by blast
  show ?thesis
proof cases
  assume even z
  then obtain z' where H-z:  $z = 2 * z'$  by blast
  show ?thesis using 1 unfolding H-x H-y H-z by auto
next
  assume odd z
  then obtain z' where H-z:  $z = 2 * z' + 1$  using oddE by blast
  have n =  $x'^2 + y'^2 + z'^2 + z'$ 
    using 0 unfolding H-x H-y H-z power2-eq-square by auto
  thus ?thesis by blast
qed
next
  assume odd y
  then obtain y' where H-y:  $y = 2 * y' + 1$  using oddE by blast
  show ?thesis
proof cases
  assume even z
  then obtain z' where H-z:  $z = 2 * z'$  by blast
  have n =  $x'^2 + z'^2 + y'^2 + y'$ 
    using 0 unfolding H-x H-y H-z power2-eq-square by auto
  thus ?thesis by blast
next
  assume odd z
  then obtain z' where H-z:  $z = 2 * z' + 1$  using oddE by blast
  show ?thesis
    using 1
    unfolding H-x H-y H-z power2-eq-square
    by (metis dvd-mod even-add even-mult-iff even-numeral odd-one)
qed
qed
next

```

```

assume odd x
then obtain x' where H-x: x = 2 * x' + 1 using oddE by blast
show ?thesis
proof cases
  assume even y
  then obtain y' where H-y: y = 2 * y' by blast
  show ?thesis
  proof cases
    assume even z
    then obtain z' where H-z: z = 2 * z' by blast
    have n = y'^2 + z'^2 + x'^2 + x'
      using 0 unfolding H-x H-y H-z power2-eq-square by auto
    thus ?thesis by blast
  next
  assume odd z
  then obtain z' where H-z: z = 2 * z' + 1 using oddE by blast
  show ?thesis
    using 1
    unfolding H-x H-y H-z power2-eq-square
    by (metis dvd-mod even-add even-mult-iff even-numeral odd-one)
qed
next
assume odd y
then obtain y' where H-y: y = 2 * y' + 1 using oddE by blast
show ?thesis
proof cases
  assume even z
  then obtain z' where H-z: z = 2 * z' by blast
  show ?thesis
    using 1
    unfolding H-x H-y H-z power2-eq-square
    by (metis dvd-mod even-add even-mult-iff even-numeral odd-one)
next
assume odd z
then obtain z' where H-z: z = 2 * z' + 1 using oddE by blast
show ?thesis
  using 1
  unfolding H-x H-y H-z power2-eq-square
  by (simp add: mod-add-eq[symmetric])
qed
qed
qed
qed

```

```

theorem four-decomposition-int:
  fixes n :: int
  shows ( $\exists x y z. n = x^2 + y^2 + z^2 + z$ )  $\longleftrightarrow n \geq 0$ 
proof
  assume  $\exists x y z. n = x^2 + y^2 + z^2 + z$ 

```

```

then obtain x y z where 0:  $n = x^2 + y^2 + z^2 + z$  by blast
show  $n \geq 0$ 
  unfolding 0 power2-eq-square
  by (smt (verit) div-pos-neg-trivial mult-le-0-iff
       nonzero-mult-div-cancel-right sum-squares-ge-zero)
next
  assume  $n \geq 0$ 
  thus  $\exists x y z. n = x^2 + y^2 + z^2 + z$ 
    using four-decomposition[of nat n]
    by (smt (verit) int-eq-iff int-plus of-nat-power)
qed

theorem four-squares:
  fixes n :: nat
  shows  $\exists x_1 x_2 x_3 x_4. n = x_1^2 + x_2^2 + x_3^2 + x_4^2$ 
proof cases
  assume  $\exists a k. n = 4 \wedge a * (8 * k + 7)$ 
  then obtain a k where  $n = 4 \wedge a * (8 * k + 7)$  by blast
  hence 0:  $n = 4 \wedge a * (8 * k + 6) + (2 \wedge a)^2$ 
    by (metis add-mult-distrib left-add-mult-distrib mult.commute mult-numeral-1
         numeral-Bit0 numeral-plus-numeral power2-eq-square power-mult-distrib
         semiring-norm(5))
  have  $\nexists a' k'. 4 \wedge a * (8 * k + 6) = 4 \wedge a' * (8 * k' + 7)$ 
  proof
    assume  $\exists a' k'. 4 \wedge a * (8 * k + 6) = 4 \wedge a' * (8 * k' + 7)$ 
    then obtain a' k' where 1:  $4 \wedge a * (8 * k + 6) = 4 \wedge a' * (8 * k' + 7)$ 
      by blast
    show False
  proof (cases rule: linorder-cases[of a a'])
    assume  $a < a'$ 
    hence 2:  $a' = a + (a' - a) a' - a > 0$  by auto
    have 3:  $4 \wedge a * (8 * k + 6) = 4 \wedge a * 4 \wedge (a' - a) * (8 * k' + 7)$ 
      using 1 2 by (metis power-add)
    have 2 =  $(8 * k + 6) \bmod 4$  by presburger
    also have ... =  $(4 \wedge (a' - a) * (8 * k' + 7)) \bmod 4$  using 3 by auto
    also have ... = 0 using 2 by auto
    finally show False by auto
  next
    assume a = a'
    hence  $8 * k + 6 = 8 * k' + 7$  using 1 by auto
    thus False by presburger
  next
    assume a > a'
    hence 4:  $a = a' + (a - a') a - a' > 0$  by auto
    have 5:  $4 \wedge a' * 4 \wedge (a - a') * (8 * k + 6) = 4 \wedge a' * (8 * k' + 7)$ 
      using 1 4 by (metis power-add)
    have 0 =  $(4 \wedge (a - a') * (8 * k + 6)) \bmod 4$  using 4 by auto
    also have ... =  $(8 * k' + 7) \bmod 4$  using 5 by auto
    also have ... = 3 by presburger

```

```

    finally show False by auto
qed
qed
then obtain x1 x2 x3 where 4 ^ a * (8 * k + 6) = x12 + x22 + x32
  using three-squares-iff by blast
thus ∃ x1 x2 x3 x4. n = x12 + x22 + x32 + x42 unfolding θ by auto
next
assume ∉ a k. n = 4 ^ a * (8 * k + 7)
hence ∃ x1 x2 x3. n = x12 + x22 + x32 using three-squares-iff by blast
thus ∃ x1 x2 x3 x4. n = x12 + x22 + x32 + x42 by (metis zero-power2 add-0)
qed
end

```

## References

- [1] M. B. Nathanson. *Additive Number Theory: The Classical Bases*, volume 164 of *Graduate Texts in Mathematics*. Springer, New York, 1996.