Sums of two and four squares

Roelof Oosterhuis
University of Groningen

June 16, 2019

Abstract

This document gives the formal proofs of the following results about the sums of two and four squares:

1. Any prime number $p \equiv 1 \bmod 4$ can be written as the sum of two squares.
2. (Lagrange) Any natural number can be written as the sum of four squares.

The proofs are largely based on chapters II and III of the book by Weil [Wei83].

The results have been formalised before in the proof assistant HOL Light [Har]. A more complete study of the sum of two squares, including the first result, has been formalised in Coq [The04]. The results can also be found as numbers 20 and 19 on the list of 'top 100 mathematical theorems' [Wie].

This research is part of an M.Sc. thesis under supervision of Jaap Top and Wim H. Hesselink (RU Groningen). For more information see [Oos07].

Contents

1 Lagrange's four-square theorem 13

theory *TwoSquares*
imports
HOL−Number-Theory.Number-Theory
begin

context

fixes *sum2sq-nat* :: *nat* ⇒ *nat* ⇒ *nat*
defines *sum2sq-nat a b* ≡ *a^2+b^2*

fixes *is-sum2sq-nat* :: *nat* ⇒ *bool*
defines *is-sum2sq-nat n* ≡ (∃ *a b*. *n* = *sum2sq-nat a b*)

begin

private lemma *best-division-abs*: (*n*::*int*) > *0* ⟹ ∃ *k*. *2* ∗ |*a* − *k*∗*n*| ≤ *n*
proof −
assume *a*: *n* > *0*
define *k* **where** *k* = *a div n*
hence *h*: *a* − *k* ∗ *n* = *a mod n* **by** (*simp add*: *mod-div-mult-eq algebra-simps*)
thus *?thesis*
proof (*cases 2* ∗ (*a mod n*) ≤ *n*)
case *True*
hence *2* ∗ |*a* − *k*∗*n*| ≤ *n* **using** *h pos-mod-sign a* **by** *auto*
thus *?thesis* **by** *blast*
next
case *False*
hence *2* ∗ (*n* − *a mod n*) ≤ *n* **by** *auto*
have *a* − (*k*+*1*)∗*n* = *a mod n* − *n* **using** *h* **by** (*simp add*: *algebra-simps*)
hence *2* ∗ |*a* − (*k*+*1*)∗*n*| ≤ *n* **using** *h pos-mod-bound*[*of n a*] *a False* **by** *fastforce*
thus *?thesis* **by** *blast*
qed
qed

private definition
sum2sq-int :: *int* × *int* ⇒ *int* **where**
sum2sq-int = (λ(*a*,*b*). *a^2+b^2*)

private definition
is-sum2sq-int :: *int* ⇒ *bool* **where**
is-sum2sq-int n ⟷ (∃ *a b*. *n* = *sum2sq-int*(*a*,*b*))

private lemma *sum2sq-int-nat-eq*: *sum2sq-nat a b* = *sum2sq-int* (*a*, *b*)
unfolding *sum2sq-nat-def sum2sq-int-def* **by** *simp*

private lemma *is-sum2sq-int-nat-eq*: *is-sum2sq-nat n* = *is-sum2sq-int* (*int n*)

unfolding *is-sum2sq-nat-def is-sum2sq-int-def*
proof
assume ∃ *a b*. *n* = *sum2sq-nat a b*
thus ∃ *a b*. *int n* = *sum2sq-int* (*a*, *b*) **using** *sum2sq-int-nat-eq* **by** *force*
next
assume ∃ *a b*. *int n* = *sum2sq-int* (*a*, *b*)
then obtain *a b* **where** *int n* = *sum2sq-int* (*a*, *b*) **by** *blast*
hence *int n* = *sum2sq-int* (*int* (*nat* |*a*|), *int* (*nat* |*b*|)) **unfolding** *sum2sq-int-def* **by** *simp*
hence *int n* = *int* (*sum2sq-nat* (*nat* |*a*|) (*nat* |*b*|)) **using** *sum2sq-int-nat-eq* **by** *presburger*
thus ∃ *a b*. *n* = *sum2sq-nat a b* **by** *auto*
qed

private lemma *product-two-squares-aux*: *sum2sq-int*(*a*, *b*) ∗ *sum2sq-int*(*c*, *d*) = *sum2sq-int*(*a*∗*c* − *b*∗*d*, *a*∗*d* + *b*∗*c*)
unfolding *power2-eq-square sum2sq-int-def* **by** (*simp add*: *algebra-simps*)

private lemma *product-two-squares-int*: *is-sum2sq-int m* ⟹ *is-sum2sq-int n* ⟹ *is-sum2sq-int* (*m*∗*n*)
by (*unfold is-sum2sq-int-def*, *auto simp only*: *product-two-squares-aux*, *blast*)

private lemma *product-two-squares-nat*: *is-sum2sq-nat m* ⟹ *is-sum2sq-nat n* ⟹ *is-sum2sq-nat* (*m*∗*n*)
using *product-two-squares-int is-sum2sq-int-nat-eq* **by** *simp*

private lemma *sots1-aux*:
assumes *prime* (*4*∗*k*+*3*)
assumes *odd* (*multiplicity* (*4*∗*k*+*3*) *n*)
shows ¬ *is-sum2sq-nat n*
proof
assume *is-sum2sq-nat n*
then obtain *a b* **where** *h1*: *n* = *a*^*2* + *b*^*2* **unfolding** *is-sum2sq-nat-def sum2sq-nat-def* **by** *blast*
have *ab-nz*: *a* ≠ *0* ∨ *b* ≠ *0* **by** (*rule ccontr*) (*insert assms*, *auto simp*: *h1*)
let *?p* = *4*∗*k*+*3*
let *?g* = *gcd a b*
have *h2*: *?g* ≠ *0* **using** *assms*(*2*) *h1 odd-pos* **by** *fastforce*
then obtain *a′ b′* **where** *h3*: *a* = *a′* ∗ *?g b* = *b′* ∗ *?g coprime a′ b′*
using *gcd-coprime-exists* **by** *blast*
have *?g*^*2 dvd n* **using** *dvd-add h1* **by** *auto*
then obtain *m* **where** *h4*: *m* ∗ *?g*^*2* = *n* **using** *dvd-div-mult-self* **by** *blast*
also have ... = (*a′* ∗ *?g*)^*2* + (*b′* ∗ *?g*)^*2* **unfolding** *h1* **using** *h3* **by** *presburger*
also have ... = *?g*^*2* ∗ *a′*^*2* + *?g*^*2* ∗ *b′*^*2* **unfolding** *power2-eq-square* **by** *simp*
finally have *?g*^*2* ∗ *m* = *?g*^*2* ∗ (*a′*^*2* + *b′*^*2*) **by** (*simp add*: *distrib-left mult.commute*)
hence *h5*: *m* = *a′*^*2* + *b′*^*2* **using** *h2* **by** *auto*
let *?mul* = *multiplicity ?p ?g*
have *multiplicity ?p* (*?g*^*2*) = *?mul* + *?mul*
unfolding *power2-eq-square* **using** *h2 assms*
by (*subst prime-elem-multiplicity-mult-distrib*) *simp-all*
hence *even* (*multiplicity ?p* (*?g*^*2*)) **by** *simp*
moreover have *m* ≠ *0* **using** *assms*(*2*) *h4 odd-pos* **by** *fastforce*
ultimately have *odd* (*multiplicity ?p m*)

using *assms ab-nz* **by** (*simp-all add*: *h4* [*symmetric*] *prime-elem-multiplicity-mult-distrib*)
hence *?p dvd m* **using** *not-dvd-imp-multiplicity-0* **by** *force*
hence *h6*: *?p dvd a'^2+b'^2* **using** *h5* **by** *auto*
{
assume *?p dvd a'^2*
moreover hence *?p dvd b'^2* **using** *h6 dvd-add-right-iff* **by** *blast*
ultimately have *?p dvd a' ?p dvd b'* **using** *assms*(*1*) *prime-dvd-power-nat* **by** *blast+*
hence *False*
using *assms*(*1*) *h3*(*3*) *coprime-common-divisor-nat*[*of a' b' ?p*] *not-prime-1* **by** *linarith*
}
hence ¬ (*?p dvd a'^2*) **..**
hence *h7*: ¬ (*?p dvd a'*) **using** *assms*(*1*)
by (*simp add*: *power2-eq-square prime-dvd-mult-iff*)
hence *coprime ?p a'*
using *assms*(*1*) **by** (*simp add*: *prime-imp-coprime*)
thm *prime-imp-coprime-nat*
moreover have *a' ≠ 0* **using** *h7 dvd-0-right*[*of ?p*] **by** *meson*
ultimately obtain *ainv aux* **where** *a' * ainv = ?p * aux + 1*
using *bezout-nat*[*of a' ?p*]
by (*auto simp*: *ac-simps*)
hence [*a' * ainv = 1*] (*mod ?p*) **using** *cong-to-1'-nat* **by** *auto*
from *cong-mult* [*OF this this*] **have** *h11*: [*1 = ainv^2 * a'^2*] (*mod ?p*)
unfolding *power2-eq-square* **by** (*simp add*: *algebra-simps cong-sym*)
let *?bdiva = ainv * b'*
have [*ainv^2 * (a'^2 + b'^2) = 0*] (*mod ?p*)
using *h6 cong-dvd-modulus-nat cong-mult-self-right* **by** *blast*
from *cong-add* [*OF h11 this*] **have** [*1 + ainv^2 * b'^2 = 0*] (*mod ?p*)
unfolding *add-mult-distrib2* **using** *cong-add-lcancel-nat*[*of ainv^2 * a'^2*]
by *fastforce*
hence *h8*: [*?bdiva^2 + 1 = 0*] (*mod ?p*) **by** (*simp add*: *power-mult-distrib*)
{
assume *?p dvd ?bdiva*
hence *?p dvd* (*?bdiva^2*) **by** (*simp add*: *assms*(*1*) *prime-dvd-power-nat-iff*)
hence [*?bdiva^2 = 0*] (*mod ?p*) **using** *cong-altdef-nat* **by** *auto*
hence [*?bdiva^2 +1 = 1*] (*mod ?p*) **using** *cong-add-rcancel-0-nat* **by** *blast*
from *this h8* **have** [*0 = 1*] (*mod ?p*) **using** *cong-sym cong-trans* **by** *blast*
hence *?p dvd 1* **using** *cong-0-1-nat* **by** *auto*
hence *False* **using** *assms*(*1*) **by** *simp*
}
hence ¬ (*?p dvd ?bdiva*) **..**
hence *h9*: [*?bdiva^(?p−1) = 1*] (*mod ?p*)
using *assms*(*1*) *fermat-theorem* [*of ?p ?bdiva*] **by** *simp*
have *h10*: *?p≥3* **by** *simp*
have *h11*: [*?bdiva^(4*k+2) = 1*] (*mod ?p*) **using** *h9* **by** *auto*
have [(*?bdiva^2 + 1*)*^2 = 0*] (*mod ?p*) **using** *h8 cong-pow* [*of ?bdiva^2 + 1 0 ?p 2*] **by** *auto*
moreover have *?bdiva ^ 4 = (?bdiva ^ 2) ^ 2* **by** *auto*
hence (*?bdiva^2 + 1*)*^2 = ?bdiva^4 + ?bdiva^2 + ?bdiva^2 + 1*
by (*auto simp*: *algebra-simps power2-eq-square*)
ultimately have [*?bdiva^4 + ?bdiva^2 + ?bdiva^2 + 1 = 0*] (*mod ?p*) **by** *simp*
moreover have [*?bdiva^4 + ?bdiva^2 + (?bdiva^2 + 1) = ?bdiva^4 + ?bdiva^2 + 0*]

(*mod ?p*)
using *h8 cong-add-lcancel-nat* **by** *blast*
ultimately have [*?bdiva^4 + ?bdiva^2 = 0*] (*mod ?p*) **by** (*simp add*: *cong-def*)
hence [*?bdiva^4 + ?bdiva^2 + 1 = 0 + 1*] (*mod ?p*) **using** *cong-add-rcancel-nat* **by** *blast*
moreover have [*?bdiva^4 + (?bdiva^2 + 1) = ?bdiva^4 + 0*] (*mod ?p*)
using *h8 cong-add-lcancel-nat* **by** *blast*
ultimately have [*?bdiva^4 = 1*] (*mod ?p*) **by** (*simp add*: *cong-def*)
hence [(*?bdiva^4*) *^k = 1^k*] (*mod ?p*) **using** *cong-pow* **by** *blast*
hence *h12*: [*?bdiva^(4*k) = 1*] (*mod ?p*) **by** (*simp add*: *power-mult*)
hence *h13*: [*?bdiva^(4*k)*(?bdiva^2 + 1) = 1*(?bdiva^2 + 1)*] (*mod ?p*)
using *cong-scalar-right* **by** *blast*
have *?bdiva^(4*k)*(?bdiva^2 + 1) = ?bdiva^(4*k+2)+?bdiva^(4*k)*
unfolding *add-mult-distrib2 power-add* **by** *simp*
hence [*?bdiva^(4*k+2)+?bdiva^(4*k) = ?bdiva^2 + 1*] (*mod ?p*)
using *h13* **unfolding** *nat-mult-1* **by** *presburger*
moreover have [*?bdiva^(4*k+2) + ?bdiva^(4*k) = 1 + 1*] (*mod ?p*)
using *h11 h12 cong-add* **by** *blast*
ultimately have [*?bdiva^2 + 1 = 2*] (*mod ?p*)
by (*auto simp add*: *cong-def*)
hence [*0 = 2*] (*mod ?p*) **using** *h8* **by** (*simp add*: *cong-def*)
then have *?p dvd 2* **by** (*auto dest*: *cong-dvd-iff*)
then show *False*
by (*auto dest*: *dvd-imp-le*)
qed

private lemma *sots1*: **assumes** *is-sum2sq-nat n*
shows $\bigwedge$ *k*. *prime* (*4*k+3*) $\longrightarrow$ *even* (*multiplicity* (*4*k+3*) *n*)
using *sots1-aux assms* **by** *blast*

private lemma *aux-lemma*: **assumes** [(*a*::*nat*) = *b*] (*mod c*) *b* < *c*
shows $\exists$ *k*. *a* = *c*k* + *b*
proof −
have *a mod c* = *b* **using** *assms* **by** (*simp add*: *cong-def mod-if*)
hence *b* ≤ *a* **using** *assms* **by** *auto*
thus *?thesis* **using** *cong-le-nat assms*(*1*) **by** *auto*
qed

private lemma *Legendre-1mod4*: *prime* (*4*k+1*::*nat*) ⟹ (*Legendre* (−*1*) (*4*k+1*)) = *1*
proof −
let *?p* = *4*k+1*
let *?L* = *Legendre* (−*1*) *?p*
assume *p*: *prime ?p*
from *p* **have** *k* ≠ *0* **by** (*intro notI*) *simp-all*
hence *p2*: *?p* > *2* **by** *simp*
with *p* **have** [*?L* = (−*1*) *^*((*?p* − *1*) *div 2*)] (*mod ?p*)
by (*rule euler-criterion*)
hence [*?L* = (−*1*) *^*(*2* ∗ *nat k*)] (*mod ?p*) **by** *auto*
hence [*?L* = *1*] (*mod ?p*) **unfolding** *power-mult* **by** *simp*
hence *?p dvd 1*−*?L*
using *cong-iff-dvd-diff dvd-minus-iff*[*of ?p ?L*−*1*] **by** *auto*

```
  moreover have ?L=1 ∨ ?L=0 ∨ ?L=−1 by (simp add: Legendre-def)
  ultimately have ?L = 1 ∨ ?p dvd 1 ∨ ?p dvd (2::int) by auto
  moreover
  { assume ?p dvd 1 ∨ ?p dvd (2::int)
    with p2 have False by (auto simp add: zdvd-not-zless) }
  ultimately show ?thesis by auto
qed

private lemma qf1-prime-exists: prime (4*k+1) ⟹ is-sum2sq-nat (4*k+1)
proof −
  let ?p = 4*k+1
  assume p: prime ?p
  hence Legendre (−1) ?p = 1 by (rule Legendre-1mod4)
  moreover
  { assume ¬ QuadRes ?p (−1)
    hence Legendre (−1) ?p ≠ 1 by (unfold Legendre-def, auto) }
  ultimately have QuadRes ?p (−1) by auto
  then obtain s1 where s1: [s1^2 = −1] (mod ?p) by (auto simp add: QuadRes-def)
  hence s1': [s1^2 + 1 = 0] (mod ?p) by (simp add: cong-iff-dvd-diff)
  define s2 where s2 = nat |s1|
  hence int (s2^2 + 1) = s1^2 + 1 by auto
  with s1' have [int (s2^2 + 1) = 0] (mod ?p) by presburger
  hence s2: [s2^2 + 1 = 0] (mod ?p)
    using cong-int-iff by fastforce
  from p have p0: ?p > 0 by simp
  then obtain s where s0p: 0 ≤ s ∧ s < ?p ∧ [s2 = s] (mod ?p)
    using cong-less-unique-nat[of ?p] by fastforce
  then have [s^2 = s2^2] (mod ?p)
    by (simp add: cong-sym cong-pow)
  with s2 have s: [s^2 + 1 = 0] (mod ?p)
    using cong-trans cong-add-rcancel-nat by blast
  hence ?p dvd s^2 + 1 using cong-altdef-nat by auto
  then obtain t where t: s^2 + 1 = ?p*t by (auto simp add: dvd-def)
  hence ?p*t = sum2sq-nat s 1 by (simp add: sum2sq-nat-def)
  hence qf1pt: is-sum2sq-nat (?p*t) by (auto simp add: is-sum2sq-nat-def)
  have t-l-p: t < ?p
  proof (rule ccontr)
    assume ¬ t < ?p
    hence t > ?p − 1 by simp
    with p0 have ?p*(?p − 1) < ?p*t by (simp only: mult-less-mono2)
    also with t have ... = s^2 + 1 by simp
    also have ... ≤ ?p*(?p − 1) − ?p + 2
    proof −
      from s0p have s ≤ ?p − 1 by (auto simp add: less-le)
      with s0p have s^2 ≤ (?p − 1)^2 by (simp only: power-mono)
        also have ... = ?p*(?p − 1) − 1*(?p − 1) by (simp only: power2-eq-square
diff-mult-distrib)
      finally show ?thesis by auto
    qed
    finally have ?p < 2 by simp
    with p show False by (unfold prime-def, auto)
  qed
```

```
  have tpos: t ≥ 1
  proof (rule ccontr)
    assume ¬ t ≥ 1
    hence t < 1 by auto
    moreover
    { assume t = 0 with t have s^2 + 1 = 0 by simp }
     moreover
    { assume t < 0
      with p0 have ?p*t < ?p*0 by (simp only: zmult-zless-mono2)
      with t have s^2 + 1 < 0 by auto }
    moreover have s^2 ≥ 0 by (simp only: zero-le-power2)
    ultimately show False by (auto simp add: less-le)
  qed
  moreover
  { assume t1: t > 0
    then obtain tn where tn: tn = t - 1 by auto
    have is-sum2sq-nat (?p*(1+ 0)) (is ?Q 0)
      — So, Q n = there exist x, y such that x^2 + y^2 = (p * (1 + int(n)))
    proof (rule ccontr)
      assume nQ1: ¬ ?Q 0
      have (1 + tn) < ?p ⟹ ¬ ?Q tn
      proof (induct tn rule: infinite-descent0)
        case 0
        from nQ1 show 1+ 0 < ?p ⟹ ¬ ?Q 0 by simp
      next
        case (smaller n)
        hence n0: n > 0 and IH: 1+ n < ?p ∧ ?Q n by auto
        then obtain x y where x^2 + y^2 = int (?p*(1+ n))
          using is-sum2sq-int-nat-eq by (unfold is-sum2sq-int-def sum2sq-int-def, auto)
       hence xy: x^2 + y^2 = (int ?p)*(int (1+ n)) unfolding of-nat-mult by presburger
        let ?n1 = int (1 + n)
        from n0 have n1pos: ?n1 > 0 by simp
        then obtain r v where rv: v = x-r*?n1 ∧ 2*|v| ≤ ?n1
          by (frule-tac n=?n1 in best-division-abs, auto)
        from n1pos obtain s w where sw: w = y-s*?n1 ∧ 2*|w| ≤ ?n1
          by (frule-tac n=?n1 in best-division-abs, auto)
        let ?C = v^2 + w^2
        have ?n1 dvd ?C
        proof
          from rv sw have ?C = (x-r*?n1)^2 + (y-s*?n1)^2 by simp
           also have ... = x^2 + y^2 - 2*x*(r*?n1) - 2*y*(s*?n1) + (r*?n1)^2 +
(s*?n1)^2
            unfolding power2-eq-square by (simp add: algebra-simps)
          also with xy have ... = ?n1*?p - ?n1*(2*x*r) - ?n1*(2*y*s) + ?n1^2*r^2
+ ?n1^2*s^2
            by (simp only: ac-simps power-mult-distrib)
          finally show ?C = ?n1*(?p - 2*x*r - 2*y*s + ?n1*(r^2 + s^2))
            by (simp only: power-mult-distrib distrib-left ac-simps
              left-diff-distrib right-diff-distrib power2-eq-square)
        qed
        then obtain m1 where m1: ?C = ?n1*m1 by (auto simp add: dvd-def)
        have mn: m1 < ?n1
```

proof (*rule ccontr*)
assume ¬ *m1* < *?n1* **hence** *?n1*−*m1* ≤ *0* **by** *simp*
hence *4*∗*?n1* − *4*∗*m1* ≤ *0* **by** *simp*
with *n1pos* **have** *2*∗*?n1* − *4*∗*m1* < *0* **by** *simp*
with *n1pos* **have** *?n1*∗(*2*∗*?n1* − *4*∗*m1*) < *?n1*∗*0* **by** (*simp only*: *zmult-zless-mono2*)
hence *contr*: *?n1*∗(*2*∗*?n1*− *4*∗*m1*) < *0* **by** *simp*
have *hlp*: *2*∗|*v*| ≥ *0* ∧ *2*∗|*w*| ≥ *0* **by** *simp*
from *m1* **have** *4*∗*?n1*∗*m1* = *4*∗*v*ˆ*2* + *4*∗*w*ˆ*2* **by** *arith*
also have ... = (*2*∗|*v*|)ˆ*2* + (*2*∗|*w*|)ˆ*2*
by (*auto simp add*: *power-mult-distrib*)
also from *rv hlp* **have** ... ≤ *?n1*ˆ*2* + (*2*∗|*w*|)ˆ*2*
using *power-mono* [*of* *2*∗|*b*| *1* + *int n 2* **for** *b*] **by** *auto*
also from *sw hlp* **have** ... ≤ *?n1*ˆ*2* + *?n1*ˆ*2*
using *power-mono* [*of* *2*∗|*b*| *1* + *int n 2* **for** *b*] **by** *auto*
finally have *?n1*∗*m1*∗*4* ≤ *?n1*∗*?n1*∗*2* **by** (*simp add*: *power2-eq-square ac-simps*)
hence *?n1*∗(*2*∗*?n1*− *4*∗*m1*) ≥ *0* **by** (*simp only*: *right-diff-distrib ac-simps*)
with *contr* **show** *False* **by** *auto*
qed
have *?p*∗*m1* = (*r*∗*v* + *s*∗*w* + *m1*)ˆ*2* + (*r*∗*w* − *s*∗*v*)ˆ*2*
proof −
from *m1 xy* **have** (*?p*∗*?n1*)∗*?C* = (*x*ˆ*2*+*y*ˆ*2*)∗(*v*ˆ*2*+*w*ˆ*2*) **by** *simp*
also have ... = (*x*∗*v* + *y*∗*w*)ˆ*2* + (*x*∗*w* − *y*∗*v*)ˆ*2*
by (*simp add*: *eval-nat-numeral field-simps*)
also with *rv sw* **have** ... = ((*r*∗*?n1*+*v*)∗*v* + (*s*∗*?n1*+*w*)∗*w*)ˆ*2* + ((*r*∗*?n1*+*v*)∗*w* − (*s*∗*?n1*+*w*)∗*v*)ˆ*2*
by *simp*
also have ... = (*?n1*∗(*r*∗*v*) + *?n1*∗(*s*∗*w*) + (*v*ˆ*2*+*w*ˆ*2*))ˆ*2* + (*?n1*∗(*r*∗*w*) − *?n1*∗(*s*∗*v*))ˆ*2*
by (*simp add*: *eval-nat-numeral field-simps*)
also from *m1* **have** ... = (*?n1*∗(*r*∗*v*) + *?n1*∗(*s*∗*w*) + *?n1*∗*m1*)ˆ*2* + (*?n1*∗(*r*∗*w*) − *?n1*∗(*s*∗*v*))ˆ*2*
by *simp*
finally have (*?p*∗*?n1*)∗*?C* = *?n1*ˆ*2*∗(*r*∗*v* + *s*∗*w* + *m1*)ˆ*2* + *?n1*ˆ*2*∗(*r*∗*w* − *s*∗*v*)ˆ*2*
by (*simp add*: *eval-nat-numeral field-simps*)
with *m1* **have** *?n1*ˆ*2*∗(*?p*∗*m1*) = *?n1*ˆ*2*∗((*r*∗*v* + *s*∗*w* + *m1*)ˆ*2* + (*r*∗*w* − *s*∗*v*)ˆ*2*)
by (*simp only*: *ac-simps power2-eq-square*, *simp add*: *distrib-left*)
hence *?n1*ˆ*2*∗(*?p*∗*m1* − (*r*∗*v*+*s*∗*w*+*m1*)ˆ*2* − (*r*∗*w*−*s*∗*v*)ˆ*2*) = *0*
by (*auto simp add*: *distrib-left right-diff-distrib*)
moreover from *n1pos* **have** *?n1*ˆ*2* ≠ *0* **by** (*simp add*: *power2-eq-square*)
ultimately show *?thesis* **by** *simp*
qed
hence *qf1pm1*: *is-sum2sq-int* ((*int ?p*)∗*m1*)
by (*unfold is-sum2sq-int-def sum2sq-int-def*, *auto*)
have *m1pos*: *m1* > *0*
proof −
{ assume *v*ˆ*2* + *w*ˆ*2* = *0*
hence *v* = *0* ∧ *w* = *0* **using** *sum-power2-eq-zero-iff* **by** *blast*
with *rv sw* **have** *?n1 dvd x* ∧ *?n1 dvd y* **by** (*unfold dvd-def*, *auto*)
hence *?n1*ˆ*2 dvd x*ˆ*2* ∧ *?n1*ˆ*2 dvd y*ˆ*2* **by** *simp*
hence *?n1*ˆ*2 dvd x*ˆ*2* + *y*ˆ*2* **by** (*simp only*: *dvd-add*)

```
        with xy have ?n1*?n1 dvd ?n1*?p by (simp only: power2-eq-square ac-simps)
         moreover from n1pos have ?n1 ≠ 0 by simp
         ultimately have ?n1 dvd ?p by (rule zdvd-mult-cancel)
         with n1pos have ?n1 ≥ 0 ∧ ?n1 dvd ?p by simp
         with p have ?n1 = 1 ∨ ?n1 = ?p unfolding prime-nat-iff by presburger
         with IH have ?Q 0 by auto
         with nQ1 have False by simp }
       moreover
       { assume v^2 + 1*w^2 ≠ 0
         moreover have v^2 + w^2 ≥ 0 by simp
         ultimately have vwpos: v^2 + w^2 > 0 by arith
         with m1 have m1 ≠ 0 by auto
         moreover have m1 ≥ 0
         proof (rule ccontr)
           assume ¬ m1 ≥ 0
           hence m1 < 0 by simp
           with n1pos have ?n1*m1 < ?n1*0 by (simp only: zmult-zless-mono2)
           with m1 vwpos show False by simp
         qed
         ultimately have ?thesis by auto }
       ultimately show ?thesis by auto
     qed
     hence 1 + int((nat m1) - 1) = m1 by arith
     with qf1pm1 have Qm1: ?Q ((nat m1) - 1)
       using is-sum2sq-int-nat-eq by (simp add: algebra-simps)
     then obtain mm where tmp: mm = (nat m1) - 1 ∧ ?Q mm by auto
     moreover have mm<n using tmp mn m1pos by arith
     moreover with IH have 1 + int mm < ?p by auto
     ultimately show ?case by auto
   qed
   hence ¬ is-sum2sq-nat (?p*t) using tn tpos t-l-p by auto
   with qf1pt show False by simp
  qed
  hence ?thesis by auto }
 ultimately show ?thesis by (auto simp add: less-le)
qed

private lemma fermat-two-squares: assumes prime p (¬ [p = 3] (mod 4))
 shows is-sum2sq-nat p
proof (cases p=2)
case True
 have (2::nat)=1^2+1^2 using power2-eq-square by simp
 thus ?thesis unfolding is-sum2sq-nat-def sum2sq-nat-def using True by fast
next
case False
 hence p > 2 using assms(1) unfolding prime-nat-iff by auto
 hence h1: odd p using assms(1) prime-odd-nat by simp
 hence h2: ¬ [p = 0] (mod 4) unfolding cong-def by fastforce
 have h3: ¬ [p = 2] (mod 4) using h1 cong-dvd-iff [of p 2 2] cong-dvd-modulus-nat by
auto
 obtain x where h4: [p = x] (mod 4) ∧ x<4 by (meson cong-less-unique-nat zero-less-numeral)
 from h1 h2 h3 h4 assms have x≠0 ∧ x≠2 ∧ x≠3 ∧ x<4 by meson
```

```
  hence x=1 by linarith
  from this h4 have [p = 1] (mod 4) by simp
  then obtain k where p = 4*k+1 using aux-lemma by fastforce
  thus ?thesis using qf1-prime-exists assms by blast
qed

private lemma sots2: assumes ⋀ k. prime (4*k+3) ⟶ even (multiplicity (4*k+3)
n)
  shows is-sum2sq-nat n using assms
proof (induction n rule: nat-less-induct)
case (1 n)
  thus ?case
  proof (cases n>1)
  case f: False
    thus ?thesis
    proof (cases n=1)
    case True
      have (1::nat)=0^2+1^2 by (simp add: power2-eq-square)
      thus ?thesis using True unfolding is-sum2sq-nat-def sum2sq-nat-def by blast
    next
    case False
      hence n=0 using f by simp
      moreover have (0::nat)=0^2+0^2 by (simp add: power2-eq-square)
      ultimately show ?thesis unfolding is-sum2sq-nat-def sum2sq-nat-def by blast
    qed
  next
  case True
  then obtain p m where h1: prime p ∧ n = p * m using prime-divisor-exists[of n]
    by (auto elim: dvdE)
  with True have m-nz: m ≠ 0 by (intro notI) auto
  from h1 have h2: m<n using n-less-m-mult-n[of m p] prime-gt-Suc-0-nat[of p] True
by linarith
    {
      assume a1: [p = 3] (mod 4)
      then obtain kp where p = 4*kp+3 using aux-lemma by fastforce
      hence even (multiplicity p n) using 1.prems h1 by auto
      moreover have multiplicity p n ≠ 0 using h1 True m-nz
        by (subst multiplicity-eq-zero-iff) (auto simp: prime-gt-0-nat)
      ultimately have h3: multiplicity p n ≥ 2 by presburger
      have p dvd m
      proof (rule ccontr)
        assume a2: ¬ p dvd m
        hence multiplicity p m = 0 by (rule not-dvd-imp-multiplicity-0)
        moreover from h1 have multiplicity p p = 1 by (intro multiplicity-prime) auto
        moreover have m > 0 using h1 True by (cases m = 0) simp-all
        ultimately have multiplicity p n = 1 using h1
          using prime-elem-multiplicity-mult-distrib [of p p m] m-nz prime-gt-0-nat
          by auto
        thus False using h3 by simp
      qed
      then obtain m' where h4: m = p * m' using dvdE by blast
      with h1 have h5: n = p^2 * m' by (simp add: power2-eq-square)
```

```
    have h6: m'<n
      using dual-order.strict-trans h1 h2 h4 nat-mult-less-cancel1 prime-gt-0-nat[of p]
by blast
    have ⋀ kq. prime (4*kq + 3) ⟹ even (multiplicity (4*kq + 3) m')
    proof −
      fix kq::nat
      let ?q = 4*kq + 3
      assume a2: prime ?q
      {
        assume p: p=?q
        hence h7: multiplicity ?q (p^2) = 2 using h1
          by (auto intro!: multiplicity-prime-power)
        have even (multiplicity ?q n) using 1(2)[of kq] a2 by blast
        also note h5
        also from p h1 h4 m-nz
          have multiplicity (4 * kq + 3) (p^2 * m') =
                Suc (Suc (multiplicity (4 * kq + 3) m'))
          by (subst prime-elem-multiplicity-mult-distrib) auto
        finally have even (multiplicity ?q m') by simp
      }
      moreover {
        assume p≠?q
        from a2 h4 m-nz have multiplicity ?q n =
                             multiplicity (4 * kq + 3) (p^2) + multiplicity (4 * kq + 3) m'
          unfolding h5 by (subst prime-elem-multiplicity-mult-distrib) simp-all
        also from ‹p ≠ ?q› a2 h1 have multiplicity ?q (p^2) = 0
          by (intro multiplicity-distinct-prime-power) simp-all
        finally have multiplicity ?q n = multiplicity ?q m' by simp
        moreover have even (multiplicity ?q n) using 1(2)[of kq] a2 by blast
        ultimately have even (multiplicity ?q m') by simp
      }
      ultimately show even (multiplicity ?q m') by blast
    qed
    hence is-sum2sq-nat m' by (simp add: 1 h6)
    moreover have p^2 = p^2 + 0^2 by simp
    hence is-sum2sq-nat (p^2) unfolding is-sum2sq-nat-def sum2sq-nat-def by blast
    ultimately have ?thesis using product-two-squares-nat h5 by blast
  } moreover
  {
    assume a1: ¬ [p = 3] (mod 4)
    have ⋀ kq. prime (4*kq+3) ⟹ even (multiplicity (4*kq+3) m)
    proof −
      fix kq
      let ?q = 4*(kq::nat) + 3
      assume a2: prime ?q
      { assume p = ?q
        then have False using a1 cong-add-rcancel-0-nat [of 4 * kq 3 4]
          by (auto simp add: cong-def)
      }
      hence p≠?q ..

      have n = p * m using h1 by simp
```

```
      also from h1 a2 m-nz have multiplicity ?q ... =
                                multiplicity (4 * kq + 3) p + multiplicity (4 * kq + 3) m
        by (subst prime-elem-multiplicity-mult-distrib) (simp-all add: prime-gt-0-nat)
      also from ‹p ≠ ?q› a2 h1 have multiplicity ?q p = 0
        by (intro prime-multiplicity-other) simp-all
      finally have multiplicity ?q n = multiplicity ?q m by simp
      moreover have even (multiplicity ?q n) using 1(2)[of kq] a2 by blast
      ultimately show even (multiplicity ?q m) by simp
    qed
    hence is-sum2sq-nat m by (simp add: 1 h2)
    moreover have is-sum2sq-nat p using fermat-two-squares a1 h1 by blast
    ultimately have ?thesis using product-two-squares-nat h1 by blast
  } ultimately
  show ?thesis by blast
 qed
qed

theorem sum-of-two-squares:
   is-sum2sq-nat n ⟷ (∀ k. prime (4*k+3) ⟶ even (multiplicity (4*k+3) n))
 using sots1[of n] sots2[of n] by blast

private lemma k-mod-eq: (∀p::nat. prime p ∧ [p = 3] (mod 4) ⟶ P p) = (∀k. prime
(4*k+3) ⟶ P (4*k+3))
proof
 assume a1: ∀p. prime p ∧ [p = 3] (mod 4) ⟶ P p
 {
  fix k :: nat
  assume prime (4 * k + 3)
  moreover hence [4*k+3 = 3] (mod 4)
   by (simp add: cong-add-rcancel-0-nat cong-mult-self-left)
  ultimately have P (4 * k + 3) using a1 by blast
 }
 thus ∀k. prime (4 * k + 3) ⟶ P (4 * k + 3) by blast
next
 assume a1: ∀k. prime (4 * k + 3) ⟶ P (4 * k + 3)
 {
  fix p :: nat
  assume prime p [p = 3] (mod 4)
  moreover with aux-lemma obtain k where p = 4*k+3 by fastforce
  ultimately have P p using a1 by blast
 }
 thus ∀p. prime p ∧ [p = 3] (mod 4) ⟶ P p by blast
qed

theorem sum-of-two-squares':
   is-sum2sq-nat n ⟷ (∀ p. prime p ∧ [p = 3] (mod 4) ⟶ even (multiplicity p n))
 using sum-of-two-squares k-mod-eq by presburger

theorem sum-of-two-squares-prime: assumes prime p
 shows is-sum2sq-nat p = [p≠3] (mod 4)
proof (cases [p=3] (mod 4))
case True
```

```
  have odd (multiplicity p p) using assms by simp
  hence ¬ (is-sum2sq-nat p) using assms True sum-of-two-squares' by blast
  with True show ?thesis by simp
qed (simp add: fermat-two-squares assms)

end

end
```

1 Lagrange's four-square theorem

```
theory FourSquares
  imports HOL−Number-Theory.Number-Theory
begin

context

  fixes sum4sq-nat :: nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat
  defines sum4sq-nat a b c d ≡ a^2+b^2+c^2+d^2

  fixes is-sum4sq-nat :: nat ⇒ bool
  defines is-sum4sq-nat n ≡ (∃ a b c d. n = sum4sq-nat a b c d)

begin


private lemma best-division-abs: (n::int) > 0 ⟹ ∃ k. 2 * |a − k*n| ≤ n
proof −
  assume a: n > 0
  define k where k = a div n
  have h: a − k * n = a mod n by (simp add: div-mult-mod-eq algebra-simps k-def)
  thus ?thesis
  proof (cases 2 * (a mod n) ≤ n)
    case True
    hence 2 * |a − k*n| ≤ n using h pos-mod-sign a by auto
    thus ?thesis by blast
  next
    case False
    hence 2 * (n − a mod n) ≤ n by auto
    have a − (k+1)*n = a mod n − n using h by (simp add: algebra-simps)
    hence 2 * |a − (k+1)*n| ≤ n using h pos-mod-bound[of n a] a False by fastforce
    thus ?thesis by blast
  qed
qed
```

Shows that all nonnegative integers can be written as the sum of four squares. The proof consists of the following steps:

- For every prime $p = 2n + 1$ the two sets of residue classes

$$\{x^2 \bmod p \mid 0 \leq x \leq n\} \text{ and } \{-1 - y^2 \bmod p \mid 0 \leq y \leq n\}$$

both contain $n+1$ different elements and therefore they must have at least one element in common.

- Hence there exist x, y such that $x^2 + y^2 + 1^2 + 0^2$ is a multiple of p.
- The next step is to show, by an infinite descent, that p itself can be written as the sum of four squares.
- Finally, using the multiplicity of this form, the same holds for all positive numbers.

private definition
sum4sq-int :: *int* × *int* × *int* × *int* ⇒ *int* **where**
sum4sq-int = (λ(*a*,*b*,*c*,*d*). *a*^2+*b*^2+*c*^2+*d*^2)

private definition
is-sum4sq-int :: *int* ⇒ *bool* **where**
is-sum4sq-int n ⟷ (∃ *a b c d*. *n* = *sum4sq-int*(*a*,*b*,*c*,*d*))

private lemma *mult-sum4sq-int*: *sum4sq-int*(*a*,*b*,*c*,*d*) ∗ *sum4sq-int*(*p*,*q*,*r*,*s*) =
sum4sq-int(*a*∗*p*+*b*∗*q*+*c*∗*r*+*d*∗*s*, *a*∗*q*−*b*∗*p*−*c*∗*s*+*d*∗*r*,
a∗*r*+*b*∗*s*−*c*∗*p*−*d*∗*q*, *a*∗*s*−*b*∗*r*+*c*∗*q*−*d*∗*p*)
by (*unfold sum4sq-int-def*, *simp add*: *eval-nat-numeral field-simps*)

private lemma *sum4sq-int-nat-eq*: *sum4sq-nat a b c d* = *sum4sq-int* (*a*, *b*, *c*, *d*)
unfolding *sum4sq-nat-def sum4sq-int-def* **by** *simp*

private lemma *is-sum4sq-int-nat-eq*: *is-sum4sq-nat n* = *is-sum4sq-int* (*int n*)
unfolding *is-sum4sq-nat-def is-sum4sq-int-def*
proof
assume ∃ *a b c d*. *n* = *sum4sq-nat a b c d*
thus ∃ *a b c d*. *int n* = *sum4sq-int* (*a*, *b*, *c*, *d*) **using** *sum4sq-int-nat-eq* **by** *force*
next
assume ∃ *a b c d*. *int n* = *sum4sq-int* (*a*, *b*, *c*, *d*)
then obtain *a b c d* **where** *int n* = *sum4sq-int* (*a*, *b*, *c*, *d*) **by** *blast*
hence *int n* = *sum4sq-int* (*int* (*nat* |*a*|), *int* (*nat* |*b*|), *int* (*nat* |*c*|), *int* (*nat* |*d*|))
unfolding *sum4sq-int-def* **by** *simp*
hence *int n* = *int* (*sum4sq-nat* (*nat* |*a*|) (*nat* |*b*|) (*nat* |*c*|) (*nat* |*d*|))
using *sum4sq-int-nat-eq* **by** *presburger*
thus ∃ *a b c d*. *n* = *sum4sq-nat a b c d* **by** *auto*
qed

private lemma *is-mult-sum4sq-int*: *is-sum4sq-int x* ⟹ *is-sum4sq-int y* ⟹ *is-sum4sq-int* (*x*∗*y*)
by (*unfold is-sum4sq-int-def*, *auto simp only*: *mult-sum4sq-int*, *blast*)

private lemma *is-mult-sum4sq-nat*: *is-sum4sq-nat x* ⟹ *is-sum4sq-nat y* ⟹ *is-sum4sq-nat* (*x*∗*y*)
using *is-mult-sum4sq-int is-sum4sq-int-nat-eq* **by** *simp*

private lemma *mult-oddprime-is-sum4sq*: ⟦ *prime* (*nat p*); *odd p* ⟧ ⟹
∃ *t*. 0<*t* ∧ *t*<*p* ∧ *is-sum4sq-int* (*p*∗*t*)

```
proof −
  assume p1: prime (nat p)
  then have p0: p > 1 and prime p
    by (simp-all add: prime-int-nat-transfer prime-nat-iff)
  assume p2: odd p
  then obtain n where n: p = 2*n+1 using oddE by blast
  with p1 have n0: n > 0 by (auto simp add: prime-nat-iff)
  let ?C = {0 ..< p}
  let ?D = {0 .. n}
  let ?f = %x. x^2 mod p
  let ?g = %x. (−1−x^2) mod p
  let ?A = ?f ‘ ?D
  let ?B = ?g ‘ ?D
  have finC: finite ?C by simp
  have finD: finite ?D by simp
  from p0 have AsubC: ?A ⊆ ?C and BsubC: ?B ⊆ ?C
    by (auto simp add: pos-mod-conj)
  with finC have finA: finite ?A and finB: finite ?B
    by (auto simp add: finite-subset)
  from AsubC BsubC have AunBsubC: ?A ∪ ?B ⊆ ?C by (rule Un-least)
  from p0 have cardC: card ?C = nat p using card-atLeastZeroLessThan-int by blast
  from n0 have cardD: card ?D = 1+ nat n by simp
  have cardA: card ?A = card ?D
  proof −
    have inj-on ?f ?D
    proof (unfold inj-on-def, auto)
      fix x fix y
      assume x0: 0 ≤ x and xn: x ≤ n and y0: 0 ≤ y and yn: y ≤ n
        and xyp: x^2 mod p = y^2 mod p
      with p0 have [x^2 = y^2] (mod p) using cong-def by blast
      hence p dvd x^2−y^2 using cong-iff-dvd-diff by blast
      hence p dvd (x+y)*(x−y) by (simp add: power2-eq-square algebra-simps)
      hence p dvd x+y ∨ p dvd x−y using ‹prime p› p0
        by (auto dest: prime-dvd-multD)
      moreover
      { assume p dvd x+y
        moreover from xn yn n have x+y < p by auto
        ultimately have ¬ x+y > 0 by (auto simp add: zdvd-not-zless)
        with x0 y0 have x = y by auto } — both are zero
      moreover
      { assume ass: p dvd x−y
        have x = y
        proof (rule ccontr, case-tac x−y ≥ 0)
          assume x−y ≥ 0 and x ≠ y hence x−y > 0 by auto
          with ass have ¬ x−y < p by (auto simp add: zdvd-not-zless)
          with xn y0 n p0 show False by auto
        next
          assume ¬ 0 ≤ x−y hence y−x > 0 by auto
          moreover from x0 yn n p0 have y−x < p by auto
          ultimately have ¬ p dvd y−x by (auto simp add: zdvd-not-zless)
          moreover from ass have p dvd −(x−y) by (simp only: dvd-minus-iff)
          ultimately show False by auto
```

```
    qed }
   ultimately show x=y by auto
  qed
  with finD show ?thesis by (simp only: inj-on-iff-eq-card)
 qed
 have cardB: card ?B = card ?D
 proof −
  have inj-on ?g ?D
  proof (unfold inj-on-def, auto)
   fix x fix y
   assume x0: 0 ≤ x and xn: x ≤ n and y0: 0 ≤ y and yn:  y ≤ n
    and xyp: (−1−x^2) mod p = (−1−y^2) mod p
   with p0 have [−1−y^2 = −1−x^2] (mod p) by (simp only: cong-def)
   hence p dvd (−1−y^2) − (−1−x^2) by (simp only: cong-iff-dvd-diff)
   moreover have −1−y^2 − (−1−x^2) = x^2 − y^2 by arith
   ultimately have p dvd x^2−y^2 by simp
   hence p dvd (x+y)∗(x−y) by (simp add: power2-eq-square algebra-simps)
   with p1 have p dvd x+y ∨ p dvd x−y using ‹prime p› p0
    by (auto dest: prime-dvd-multD)
   moreover
   { assume p dvd x+y
    moreover from xn yn n have x+y < p by auto
    ultimately have ¬ x+y > 0 by (auto simp add: zdvd-not-zless)
    with x0 y0 have x = y by auto } — both are zero
   moreover
   { assume ass: p dvd x−y
    have x = y
    proof (rule ccontr, case-tac x−y ≥ 0)
     assume x−y ≥ 0 and x ≠ y hence x−y > 0 by auto
     with ass have ¬ x−y < p by (auto simp add: zdvd-not-zless)
     with xn y0 n p0 show False by auto
    next
     assume ¬ 0 ≤ x−y hence y−x > 0 by auto
     moreover from x0 yn n p0 have y−x < p by auto
     ultimately have ¬ p dvd y−x by (auto simp add: zdvd-not-zless)
     moreover from ass have p dvd −(x−y) by (simp only: dvd-minus-iff)
     ultimately show False by auto
    qed }
   ultimately show x=y by auto
  qed
  with finD show ?thesis by (simp only: inj-on-iff-eq-card)
 qed
 have ?A ∩ ?B ≠ {}
 proof (rule ccontr, auto)
  assume ABdisj: ?A ∩ ?B = {}
  from cardA cardB cardD have 2 + 2∗(nat n) = card ?A + card ?B by auto
  also with finA finB ABdisj have ... = card (?A ∪ ?B)
   by (simp only: card-Un-disjoint)
  also with finC AunBsubC have ... ≤ card ?C by (simp only: card-mono)
  also with cardC have ... = nat p by simp
  finally have 2 + 2∗(nat n) ≤ nat p by simp
  with n show False by arith
```

```
  qed
  then obtain z where z ∈ ?A ∧ z ∈ ?B by auto
  then obtain x y where xy: x ∈ ?D ∧ y ∈ ?D ∧ z = x^2 mod p ∧ z = (−1−y^2)
mod p by blast
  with p0 have [x^2=−1−y^2](mod p) by (simp add: cong-def)
  hence p dvd x^2−(−1−y^2) by (simp only: cong-iff-dvd-diff)
  moreover have x^2−(−1−y^2)=x^2+y^2+1 by arith
  ultimately have p dvd sum4sq-int(x,y,1,0) by (auto simp add: sum4sq-int-def)
  then obtain t where t: p * t = sum4sq-int(x,y,1,0) by (auto simp only: dvd-def
eq-refl)
  hence is-sum4sq-int (p*t) by (unfold is-sum4sq-int-def, auto)
  moreover have t > 0 ∧ t < p
  proof
    have x^2 ≥ 0 ∧ y^2 ≥ 0 by simp
    hence x^2+y^2+1 > 0 by arith
    with t have p*t > 0 by (unfold sum4sq-int-def, auto)
    moreover
    { assume t < 0 with p0 have p*t < p*0 by (simp only: zmult-zless-mono2)
      hence p*t < 0 by simp }
    moreover
    { assume t = 0 hence p*t = 0 by simp }
    ultimately have ¬ t < 0 ∧ t ≠ 0 by auto
    thus t > 0 by simp
    from xy have x^2 ≤ n^2 ∧ y^2 ≤ n^2 by (auto simp add: power-mono)
    hence x^2+y^2+1 ≤ 2*n^2 + 1 by auto
    with t have contr: p*t ≤ 2*n^2+1 by (simp add: sum4sq-int-def)
    moreover
    { assume t > n+1
      with p0 have p*(n+1) < p*t by (simp only: zmult-zless-mono2)
      with n have p*t > (2*n+1)*n + (2*n+1)*1 by (simp only: distrib-left)
      hence p*t > 2*n*n + n + 2*n + 1 by (simp only: distrib-right mult-1-left)
      with n0 have p*t > 2*n^2 + 1 by (simp add: power2-eq-square) }
    ultimately have ¬ t > n+1 by auto
    with n0 n show t < p by auto
  qed
  ultimately show ?thesis by blast
qed

private lemma zprime-is-sum4sq: prime (nat p) ⟹ is-sum4sq-int p
proof (cases)
  assume p2: p=2
  hence p = sum4sq-int(1,1,0,0) by (auto simp add: sum4sq-int-def)
  thus ?thesis by (auto simp add: is-sum4sq-int-def)
next
  assume ¬ p =2 and prp: prime (nat p)
  hence ¬ nat p = 2 by simp
  with prp have 2 < nat p using prime-nat-iff by force
  moreover with prp have odd (nat p) using prime-odd-nat[of nat p] by blast
  ultimately have odd p by (simp add: even-nat-iff)
 with prp have ∃ t. 0<t ∧ t<p ∧ is-sum4sq-int (p*t) by (rule mult-oddprime-is-sum4sq)
  then obtain a b c d t where pt-sol: 0<t ∧ t<p ∧ p*t = sum4sq-int(a,b,c,d)
    by (unfold is-sum4sq-int-def, blast)
```

```
  hence Qt: 0<t ∧ t<p ∧ (∃ a1 a2 a3 a4. p*t = sum4sq-int(a1,a2,a3,a4))
    (is ?Q t) by blast
  have ?Q 1
  proof (rule ccontr)
    assume nQ1: ¬ ?Q 1
    have ¬ ?Q t
    proof (induct t rule: infinite-descent0-measure[where V=λx. (nat x)- 1], clarify)
      fix x a b c d
      assume nat x - 1 = 0 and x > 0 and s: p*x=sum4sq-int(a,b,c,d) and x < p
      moreover hence x = 1 by arith
      ultimately have ?Q 1 by auto
      with nQ1 show False by auto
    next
      fix x
      assume 0 < nat x - 1 and ¬ ¬ ?Q x
     then obtain a1 a2 a3 a4 where ass: 1<x ∧ x<p ∧ p*x = sum4sq-int(a1,a2,a3,a4)
        by auto
      have ∃ y. nat y - 1 < nat x - 1 ∧ ?Q y
      proof (cases)
        assume evx: even x
        hence even (x*p) by simp
        with ass have ev1234: even (a1^2+a2^2 + a3^2+a4^2)
          by (auto simp add: sum4sq-int-def ac-simps)
           have ∃ b1 b2 b3 b4. p*x=sum4sq-int(b1,b2,b3,b4) ∧ even (b1+b2) ∧ even
(b3+b4)
        proof (cases)
          assume ev12: even (a1^2+a2^2)
          with ev1234 ass show ?thesis by auto
        next
          assume ¬ even (a1^2+a2^2)
          hence odd12: odd (a1^2+a2^2) by simp
          with ev1234 have odd34: odd (a3^2+a4^2) by auto
          show ?thesis
          proof (cases)
            assume ev1: even (a1^2)
            with odd12 have odd2: odd (a2^2) by simp
            show ?thesis
            proof (cases)
              assume even (a3^2)
               moreover from ass have p*x = sum4sq-int(a1,a3,a2,a4) by (auto simp
add: sum4sq-int-def)
              ultimately show ?thesis using odd2 odd34 ev1 by auto
            next
              assume ¬ even (a3^2)
               moreover from ass have p*x = sum4sq-int(a1,a4,a2,a3) by (auto simp
add: sum4sq-int-def)
              ultimately show ?thesis using odd34 odd2 ev1 by auto
            qed
          next
            assume odd1: ¬ even (a1^2)
            with odd12 have ev2: even (a2^2) by simp
            show ?thesis
```

proof (*cases*)
assume *even* (*a3^2*)
moreover from *ass* **have** *sum4sq-int*(*a1*,*a4*,*a2*,*a3*)=*p*∗*x* **by** (*auto simp add*: *sum4sq-int-def*)
ultimately show *?thesis* **using** *odd34 odd1 ev2* **by** *force*
next
assume ¬ *even* (*a3^2*)
moreover from *ass* **have** *sum4sq-int*(*a1*,*a3*,*a2*,*a4*)=*p*∗*x* **by** (*auto simp add*: *sum4sq-int-def*)
ultimately show *?thesis* **using** *odd34 odd1 ev2* **by** *force*
qed
qed
qed
then obtain *b1 b2 b3 b4*
where *b*: *p*∗*x* = *sum4sq-int*(*b1*,*b2*,*b3*,*b4*) ∧ *even* (*b1*+*b2*) ∧ *even* (*b3*+*b4*) **by** *auto*
then obtain *c1 c3* **where** *c13*: *b1*+*b2* = *2*∗*c1* ∧ *b3*+*b4* = *2*∗*c3*
using *evenE*[*of b1*+*b2*] *evenE*[*of b3*+*b4*] **by** *meson*
from *b* **have** *even* (*b1*−*b2*) ∧ *even* (*b3*−*b4*) **by** *simp*
then obtain *c2 c4* **where** *c24*: *b1*−*b2* = *2*∗*c2* ∧ *b3*−*b4* = *2*∗*c4*
using *evenE*[*of b1*−*b2*] *evenE*[*of b3*−*b4*] **by** *meson*
from *evx* **obtain** *y* **where** *y*: *x* = *2*∗*y* **using** *evenE* **by** *blast*
hence *4*∗(*p*∗*y*) = *2*∗(*p*∗*x*) **by** (*simp add*: *ac-simps*)
also from *b* **have** ... = *2*∗*b1^2* + *2*∗*b2^2* + *2*∗*b3^2* + *2*∗*b4^2*
by (*auto simp*: *sum4sq-int-def*)
also have ... = (*b1* + *b2*)^*2* + (*b1* − *b2*)^*2* + (*b3* + *b4*)^*2* + (*b3* − *b4*)^*2*
by (*auto simp add*: *power2-eq-square algebra-simps*)
also with *c13 c24* **have** ... = *4*∗(*c1^2* + *c2^2* + *c3^2* + *c4^2*)
by (*auto simp add*: *power-mult-distrib*)
finally have *p* ∗ *y* = *sum4sq-int*(*c1*,*c2*,*c3*,*c4*) **by** (*auto simp add*: *sum4sq-int-def*)
moreover from *y ass* **have** *0* < *y* ∧ *y* < *p* ∧ (*nat y*) − *1* < (*nat x*) − *1* **by** *arith*
ultimately show *?thesis* **by** *blast*
next
assume *xodd*: ¬ *even x*
with *ass* **have** ∃ *c1 c2 c3 c4*. *2*∗|*a1*−*c1*∗*x*|≤*x* ∧ *2*∗|*a2*−*c2*∗*x*|≤*x* ∧ *2*∗|*a3*−*c3*∗*x*|≤*x* ∧ *2*∗|*a4*−*c4*∗*x*|≤*x*
by (*simp add*: *best-division-abs*)
then obtain *b1 c1 b2 c2 b3 c3 b4 c4* **where**
bc-def: *b1* = *a1*−*c1*∗*x* ∧ *b2* = *a2*−*c2*∗*x* ∧ *b3* = *a3*−*c3*∗*x* ∧ *b4* = *a4*−*c4*∗*x*
and *2*∗|*b1*|≤*x* ∧ *2*∗|*b2*|≤*x* ∧ *2*∗|*b3*|≤*x* ∧ *2*∗|*b4*|≤*x*
by *blast*
moreover have *2*∗|*b1*|≠*x* ∧ *2*∗|*b2*|≠*x* ∧ *2*∗|*b3*|≠*x* ∧ *2*∗|*b4*|≠*x* **using** *xodd* **by** *fastforce*
ultimately have *bc-abs*: *2*∗|*b1*|<*x* ∧ *2*∗|*b2*|<*x* ∧ *2*∗|*b3*|<*x* ∧ *2*∗|*b4*|<*x* **by** *auto*
let *?B* = *b1^2* + *b2^2* + *b3^2* + *b4^2*
let *?C* = *c1^2* + *c2^2* + *c3^2* + *c4^2*
have *x dvd ?B*
proof
from *bc-def ass* **have**
?B = *p*∗*x* − *2*∗(*a1*∗*c1*+*a2*∗*c2*+*a3*∗*c3*+*a4*∗*c4*)∗*x* + *?C*∗*x^2*
unfolding *sum4sq-int-def* **by** (*auto simp add*: *power2-eq-square algebra-simps*)
thus *?B* = *x*∗(*p* − *2*∗(*a1*∗*c1*+*a2*∗*c2*+*a3*∗*c3*+*a4*∗*c4*) + *?C*∗*x*)

```
    by (auto simp add: ac-simps power2-eq-square
      distrib-left right-diff-distrib)
qed
then obtain y where y: ?B = x * y by (auto simp add: dvd-def)
let ?A1 = a1*b1 + a2*b2 + a3*b3 + a4*b4
let ?A2 = a1*b2 - a2*b1 - a3*b4 + a4*b3
let ?A3 = a1*b3 + a2*b4 - a3*b1 - a4*b2
let ?A4 = a1*b4 - a2*b3 + a3*b2 - a4*b1
let ?A = sum4sq-int(?A1,?A2,?A3,?A4)
have x dvd ?A1 ∧ x dvd ?A2 ∧ x dvd ?A3 ∧ x dvd ?A4
proof (safe)
  from bc-def have
    ?A1 = (b1+c1*x)*b1 + (b2+c2*x)*b2 + (b3+c3*x)*b3 + (b4+c4*x)*b4
    by simp
  also with y have ... = x*(y + c1*b1 + c2*b2 + c3*b3 + c4*b4)
    by (auto simp add: distrib-left power2-eq-square ac-simps)
  finally show x dvd ?A1 by auto
  from bc-def have
    ?A2 = (b1+c1*x)*b2 - (b2+c2*x)*b1 - (b3+c3*x)*b4 + (b4+c4*x)*b3
    by simp
  also have ... = x*(c1*b2 - c2*b1 - c3*b4 + c4*b3)
    by (auto simp add: distrib-left right-diff-distrib ac-simps)
  finally show x dvd ?A2 by auto
  from bc-def have
    ?A3 = (b1+c1*x)*b3 + (b2+c2*x)*b4 - (b3+c3*x)*b1 - (b4+c4*x)*b2
    by simp
  also have ... = x*(c1*b3 + c2*b4 - c3*b1 - c4*b2)
    by (auto simp add: distrib-left right-diff-distrib ac-simps)
  finally show x dvd ?A3 by auto
  from bc-def have
    ?A4 = (b1+c1*x)*b4 - (b2+c2*x)*b3 + (b3+c3*x)*b2 - (b4+c4*x)*b1
    by simp
  also have ... = x*(c1*b4 - c2*b3 + c3*b2 - c4*b1)
    by (auto simp add: distrib-left right-diff-distrib ac-simps)
  finally show x dvd ?A4 by auto
qed
then obtain d1 d2 d3 d4 where d:
  ?A1=x*d1 ∧ ?A2=x*d2 ∧ ?A3=x*d3 ∧ ?A4=x*d4
  by (auto simp add: dvd-def)
let ?D = sum4sq-int(d1,d2,d3,d4)
from d have x^2*?D = ?A
  by (auto simp only: sum4sq-int-def power-mult-distrib distrib-left)
also have ... = sum4sq-int(a1,a2,a3,a4)*sum4sq-int(b1,b2,b3,b4)
  by (simp only: mult-sum4sq-int)
also with y ass have ... = (p*x)*(x*y) by (auto simp add: sum4sq-int-def)
also have ... = x^2*(p*y) by (simp only: power2-eq-square ac-simps)
finally have x^2*(?D - p*y) = 0 by (auto simp add: right-diff-distrib)
with ass have p*y = ?D by auto
moreover have y-l-x: y < x
proof -
  have 4*b1^2 = (2*|b1|)^2 ∧ 4*b2^2 = (2*|b2|)^2 ∧
    4*b3^2 = (2*|b3|)^2 ∧ 4*b4^2 = (2*|b4|)^2 by simp
```

```
      with bc-abs have 4*b1^2<x^2 ∧ 4*b2^2<x^2 ∧ 4*b3^2<x^2 ∧ 4*b4^2<x^2
        using power-strict-mono [of 2*|b| x 2 for b]
        by auto
      hence ?B < x^2 by auto
      with y have x*(x-y) > 0
        by (auto simp add: power2-eq-square right-diff-distrib)
      moreover from ass have x > 0 by simp
      ultimately show ?thesis using zero-less-mult-pos by fastforce
    qed
    moreover have y > 0
    proof -
      have b2pos: b1^2 ≥ 0 ∧ b2^2 ≥ 0 ∧ b3^2 ≥ 0 ∧ b4^2 ≥ 0 by simp
      hence ?B = 0 ∨ ?B > 0 by arith
      moreover
      { assume ?B = 0
        moreover from b2pos have
          ?B-b1^2 ≥ 0 ∧ ?B-b2^2 ≥ 0 ∧ ?B-b3^2 ≥ 0 ∧ ?B-b4^2 ≥ 0 by arith
        ultimately have b1^2 ≤ 0 ∧ b2^2 ≤ 0 ∧ b3^2 ≤ 0 ∧ b4^2 ≤ 0 by auto
        with b2pos have  b1^2 = 0 ∧ b2^2 = 0 ∧ b3^2 = 0 ∧ b4^2 = 0 by arith
        hence b1 = 0 ∧ b2 = 0 ∧ b3 = 0 ∧ b4 = 0 by auto
        with bc-def have x dvd a1 ∧ x dvd a2 ∧ x dvd a3 ∧ x dvd a4
          by auto
        hence x^2 dvd a1^2 ∧ x^2 dvd a2^2 ∧ x^2 dvd a3^2 ∧ x^2 dvd a4^2 by simp
        hence x^2 dvd a1^2+a2^2+a3^2+a4^2 by (simp only: dvd-add)
        with ass have x^2 dvd p*x by (auto simp only: sum4sq-int-def)
        hence x*x dvd x*p by (simp only: power2-eq-square ac-simps)
        with ass have nat x dvd nat p
          by (simp add: nat-dvd-iff)
        moreover from ass prp have x ≥ 0 ∧ x ≠ 1 ∧ x ≠ p ∧ prime (nat p) by
simp
        ultimately have False unfolding prime-nat-iff by auto }
      moreover
      { assume ?B > 0
        with y have x*y > 0 by simp
        moreover from ass have x > 0 by simp
        ultimately have ?thesis using zero-less-mult-pos by blast }
      ultimately show ?thesis by auto
    qed
    moreover with y-l-x have (nat y) - 1 < (nat x) - 1 by arith
    moreover from y-l-x ass have y < p by auto
    ultimately show ?thesis by blast
  qed
  thus ∃ y. nat y - 1 < nat x - 1 ∧ ¬ ¬ ?Q y by blast
 qed
 with Qt show False by simp
qed
thus is-sum4sq-int p by (auto simp add: is-sum4sq-int-def)
qed

private lemma prime-is-sum4sq: prime p ⟹ is-sum4sq-nat p
 using zprime-is-sum4sq is-sum4sq-int-nat-eq by simp
```

```
theorem sum-of-four-squares: is-sum4sq-nat n
proof (induction n rule: nat-less-induct)
case (1 n)
  show ?case
  proof (cases n>1)
  case False
    hence n = 0 ∨ n = 1 by auto
      moreover have 0 = sum4sq-nat 0 0 0 0 1 = sum4sq-nat 1 0 0 0 unfolding
sum4sq-nat-def by auto
    ultimately show ?thesis unfolding is-sum4sq-nat-def by blast
  next
  case True
    then obtain p m where dec: prime p ∧ n = p * m using prime-factor-nat[of n]
      by (auto elim: dvdE)
    moreover hence m<n using n-less-m-mult-n[of m p] prime-gt-Suc-0-nat[of p] True
by linarith
   ultimately have is-sum4sq-nat m is-sum4sq-nat p using 1 prime-is-sum4sq by blast+
    thus ?thesis using dec is-mult-sum4sq-nat by blast
  qed
qed

end

end
```

References

[Har] John Harrison. The HOL Light theorem prover. http://www.cl.cam.ac.uk/~jrh13/hol-light/.

[Oos07] Roelof Oosterhuis. Mechanised theorem proving: Exponents 3 and 4 of Fermat's Last Theorem in Isabelle. Master's thesis, University of Groningen, 2007. http://www.roelofoosterhuis.nl/MScthesis.pdf.

[The04] Laurent Thery. Numbers equal to the sum of two square numbers. http://coq.inria.fr/contribs/SumOfTwoSquare.html, 2004.

[Wei83] André Weil. *Number Theory: An Approach Through History; From Hammurapi to Legendre*. Birkhäuser, 1983.

[Wie] Freek Wiedijk. Formalizing 100 theorems. http://www.cs.ru.nl/~freek/100/.