

Gaussian Integers

Manuel Eberl

February 6, 2026

Abstract

The Gaussian integers are the subring $\mathbb{Z}[i]$ of the complex numbers, i. e. the ring of all complex numbers with integral real and imaginary part. This article provides a definition of this ring as well as proofs of various basic properties, such as that they form a Euclidean ring and a full classification of their primes. An executable (albeit not very efficient) factorisation algorithm is also provided.

Lastly, this Gaussian integer formalisation is used in two short applications:

1. The characterisation of all positive integers that can be written as sums of two squares
2. Euclid's formula for primitive Pythagorean triples

While elementary proofs for both of these are already available in the AFP, the theory of Gaussian integers provides more concise proofs and a more high-level view.

Contents

1	Gaussian Integers	3
1.1	Auxiliary material	3
1.2	Definition	9
1.3	Pretty-printing	14
1.4	Norm	15
1.5	Division and normalisation	17
1.6	Prime elements	25
1.6.1	The factorisation of 2	26
1.6.2	Inert primes	27
1.6.3	Non-inert primes	28
1.6.4	Full classification of Gaussian primes	34
1.6.5	Multiplicities of primes	37
1.7	Coprimality of an element and its conjugate	40
1.8	Square decompositions of prime numbers congruent 1 mod 4	43
1.9	Executable factorisation of Gaussian integers	48
1.10	Sums of two squares	51
1.11	Primitive Pythagorean triples	54

1 Gaussian Integers

theory *Gaussian-Integers*

imports

HOL-Computational-Algebra.Computational-Algebra

HOL-Number-Theory.Number-Theory

begin

1.1 Auxiliary material

lemma *coprime-iff-prime-factors-disjoint*:

fixes $x\ y :: 'a :: \text{factorial-semiring}$

assumes $x \neq 0\ y \neq 0$

shows $\text{coprime } x\ y \longleftrightarrow \text{prime-factors } x \cap \text{prime-factors } y = \{\}$

proof

assume *coprime* $x\ y$

have *False* **if** $p \in \text{prime-factors } x\ p \in \text{prime-factors } y$ **for** p

proof –

from *that assms* **have** $p \text{ dvd } x\ p \text{ dvd } y$

by (*auto simp: prime-factors-dvd*)

with $\langle \text{coprime } x\ y \rangle$ **have** $p \text{ dvd } 1$

using *coprime-common-divisor* **by** *auto*

with *that assms* **show** *False* **by** (*auto simp: prime-factors-dvd*)

qed

thus $\text{prime-factors } x \cap \text{prime-factors } y = \{\}$ **by** *auto*

next

assume *disjoint*: $\text{prime-factors } x \cap \text{prime-factors } y = \{\}$

show *coprime* $x\ y$

proof (*rule coprimeI*)

fix d **assume** $d: d \text{ dvd } x\ d \text{ dvd } y$

show *is-unit* d

proof (*rule ccontr*)

assume $\neg \text{is-unit } d$

moreover from *this* **and** d *assms* **have** $d \neq 0$ **by** *auto*

ultimately obtain p **where** $p: \text{prime } p\ p \text{ dvd } d$

using *prime-divisor-exists* **by** *auto*

with d **and** *assms* **have** $p \in \text{prime-factors } x \cap \text{prime-factors } y$

by (*auto simp: prime-factors-dvd*)

with *disjoint* **show** *False* **by** *auto*

qed

qed

qed

lemma *product-dvd-irreducibleD*:

fixes $a\ b\ x :: 'a :: \text{algebraic-semidom}$

assumes *irreducible* x

assumes $a * b \text{ dvd } x$

shows $a \text{ dvd } 1 \vee b \text{ dvd } 1$

proof –

from *assms* **obtain** c **where** $x = a * b * c$

by *auto*
 hence $x = a * (b * c)$
 by (*simp add: mult-ac*)
 from *irreducibleD[OF assms(1) this]* show $a \text{ dvd } 1 \vee b \text{ dvd } 1$
 by (*auto simp: is-unit-mult-iff*)
qed

lemma *prime-elem-mult-dvdI*:
 assumes *prime-elem* $p \text{ dvd } c \ b \text{ dvd } c \ \neg p \text{ dvd } b$
 shows $p * b \text{ dvd } c$
proof –
 from *assms(3)* obtain *a* where $c = a * b$
 using *mult.commute* by *blast*
 with *assms(2)* have $p \text{ dvd } a * b$
 by *simp*
 with *assms* have $p \text{ dvd } a$
 by (*subst (asm) prime-elem-dvd-mult-iff*) *auto*
 with *c* show *?thesis* by (*auto intro: mult-dvd-mono*)
qed

lemma *prime-elem-power-mult-dvdI*:
 fixes $p :: 'a :: \text{algebraic-semidom}$
 assumes *prime-elem* $p \text{ dvd } c \ b \text{ dvd } c \ \neg p \text{ dvd } b$
 shows $p \wedge^n * b \text{ dvd } c$
proof (*cases n = 0*)
 case *False*
 from *assms(3)* obtain *a* where $c = a * b$
 using *mult.commute* by *blast*
 with *assms(2)* have $p \wedge^n \text{ dvd } b * a$
 by (*simp add: mult-ac*)
 hence $p \wedge^n \text{ dvd } a$
 by (*rule prime-power-dvd-multD[OF assms(1)]*) (*use assms False in auto*)
 with *c* show *?thesis* by (*auto intro: mult-dvd-mono*)
qed (*use assms in auto*)

lemma *prime-mod-4-cases*:
 fixes $p :: \text{nat}$
 assumes *prime* p
 shows $p = 2 \vee [p = 1] \pmod{4} \vee [p = 3] \pmod{4}$
proof (*cases p = 2*)
 case *False*
 with *prime-gt-1-nat[of p]* *assms* have $p > 2$ by *auto*
 have $\neg 4 \text{ dvd } p$
 using *assms product-dvd-irreducibleD[of p 2 2]*
 by (*auto simp: prime-elem-iff-irreducible simp flip: prime-elem-nat-iff*)
 hence $p \text{ mod } 4 \neq 0$
 by (*auto simp: mod-eq-0-iff-dvd*)
 moreover have $p \text{ mod } 4 \neq 2$
proof

```

assume  $p \bmod 4 = 2$ 
hence  $p \bmod 4 \bmod 2 = 0$ 
  by (simp add: cong-def)
thus False using  $\langle \text{prime } p \rangle \langle p > 2 \rangle$  prime-odd-nat[of p]
  by (auto simp: mod-mod-cancel)
qed
moreover have  $p \bmod 4 \in \{0,1,2,3\}$ 
  by auto
ultimately show ?thesis by (auto simp: cong-def)
qed auto

```

```

lemma of-nat-prod-mset:  $\text{of-nat } (\text{prod-mset } A) = \text{prod-mset } (\text{image-mset of-nat } A)$ 
  by (induction A) auto

```

```

lemma multiplicity-0-left [simp]:  $\text{multiplicity } 0 \ x = 0$ 
  by (cases x = 0) (auto simp: not-dvd-imp-multiplicity-0)

```

```

lemma is-unit-power [intro]:  $\text{is-unit } x \implies \text{is-unit } (x \wedge n)$ 
  by (subst is-unit-power-iff) auto

```

```

lemma (in factorial-semiring) pow-divides-pow-iff:

```

```

  assumes  $n > 0$ 
  shows  $a \wedge n \text{ dvd } b \wedge n \iff a \text{ dvd } b$ 
proof (cases b = 0)
  case False
  show ?thesis
  proof
    assume dvd:  $a \wedge n \text{ dvd } b \wedge n$ 
    with  $\langle b \neq 0 \rangle$  have  $a \neq 0$ 
      using  $\langle n > 0 \rangle$  by (auto simp: power-0-left)
    show  $a \text{ dvd } b$ 
  proof (rule multiplicity-le-imp-dvd)
    fix  $p :: 'a$  assume  $p$ : prime p
    from dvd  $\langle b \neq 0 \rangle$  have  $\text{multiplicity } p \ (a \wedge n) \leq \text{multiplicity } p \ (b \wedge n)$ 
      by (intro dvd-imp-multiplicity-le) auto
    thus  $\text{multiplicity } p \ a \leq \text{multiplicity } p \ b$ 
    using  $p \langle a \neq 0 \rangle \langle b \neq 0 \rangle \langle n > 0 \rangle$  by (simp add: prime-elem-multiplicity-power-distrib)
  qed fact+
  qed (auto intro: dvd-power-same)
qed (use assms in <auto simp: power-0-left>)

```

```

lemma multiplicity-power-power:

```

```

  fixes  $p :: 'a :: \{\text{factorial-semiring, algebraic-semidom}\}$ 
  assumes  $n > 0$ 
  shows  $\text{multiplicity } (p \wedge n) \ (x \wedge n) = \text{multiplicity } p \ x$ 
proof (cases x = 0  $\vee$   $p = 0$   $\vee$  is-unit p)
  case True
  thus ?thesis using  $\langle n > 0 \rangle$ 
    by (auto simp: power-0-left is-unit-power-iff multiplicity-unit-left)

```

```

next
  case False
  show ?thesis
  proof (intro antisym multiplicity-geI)
    have  $(p \wedge \text{multiplicity } p \ x) \wedge n \ \text{dvd } x \wedge n$ 
      by (intro dvd-power-same) (simp add: multiplicity-dvd)
    thus  $(p \wedge n) \wedge \text{multiplicity } p \ x \ \text{dvd } x \wedge n$ 
      by (simp add: mult-ac flip: power-mult)
  next
    have  $(p \wedge n) \wedge \text{multiplicity } (p \wedge n) \ (x \wedge n) \ \text{dvd } x \wedge n$ 
      by (simp add: multiplicity-dvd)
    hence  $(p \wedge \text{multiplicity } (p \wedge n) \ (x \wedge n)) \wedge n \ \text{dvd } x \wedge n$ 
      by (simp add: mult-ac flip: power-mult)
    thus  $p \wedge \text{multiplicity } (p \wedge n) \ (x \wedge n) \ \text{dvd } x$ 
      by (subst (asm) pow-divides-pow-iff) (use assms in auto)
  qed (use False <n > 0 in <auto simp: is-unit-power-iff>)
qed

```

```

lemma even-square-cong-4-int:
   $\langle [x^2 = 0] \pmod{4} \rangle$  if  $\langle \text{even } x \rangle$  for  $x :: \text{int}$ 
proof -
  from that obtain  $y$  where  $\langle x = 2 * y \rangle ..$ 
  then show ?thesis by (simp add: cong-def)
qed

```

```

lemma even-square-cong-4-nat:
   $\langle [x^2 = 0] \pmod{4} \rangle$  if  $\langle \text{even } x \rangle$  for  $x :: \text{nat}$ 
  using that even-square-cong-4-int [of  $\langle \text{int } x \rangle$ ] by (simp flip: cong-int-iff)

```

```

lemma odd-square-cong-4-int:
   $\langle [x^2 = 1] \pmod{4} \rangle$  if  $\langle \text{odd } x \rangle$  for  $x :: \text{int}$ 
proof -
  from that obtain  $y$  where  $\langle x = 2 * y + 1 \rangle ..$ 
  then have  $\langle x^2 = 4 * (y^2 + y) + 1 \rangle$ 
    by (simp add: power2-eq-square algebra-simps)
  also have  $\langle \dots \pmod{4} = ((4 * (y^2 + y)) \pmod{4} + 1 \pmod{4}) \pmod{4} \rangle$ 
    by (simp only: mod-simps)
  also have  $\langle \dots = 1 \pmod{4} \rangle$ 
    by simp
  finally show ?thesis
    by (simp only: cong-def)
qed

```

```

lemma odd-square-cong-4-nat:
   $\langle [x^2 = 1] \pmod{4} \rangle$  if  $\langle \text{odd } x \rangle$  for  $x :: \text{nat}$ 
  using that odd-square-cong-4-int [of  $\langle \text{int } x \rangle$ ] by (simp flip: cong-int-iff)

```

Gaussian integers will require a notion of an element being a power up to a unit, so we introduce this here. This should go in the library eventually.

definition *is-nth-power-upto-unit* **where**

is-nth-power-upto-unit $n\ x \longleftrightarrow (\exists u. \text{is-unit } u \wedge \text{is-nth-power } n\ (u * x))$

lemma *is-nth-power-upto-unit-base*: *is-nth-power* $n\ x \implies \text{is-nth-power-upto-unit } n\ x$

by (*auto simp: is-nth-power-upto-unit-def intro: exI[of - 1]*)

lemma *is-nth-power-upto-unitI*:

assumes *normalize* $(x \wedge n) = \text{normalize } y$

shows *is-nth-power-upto-unit* $n\ y$

proof –

from *associatedE1[OF assms]* **obtain** u **where** *is-unit* $u\ u * y = x \wedge n$

by *metis*

thus *?thesis*

by (*auto simp: is-nth-power-upto-unit-def intro!: exI[of - u]*)

qed

lemma *is-nth-power-upto-unit-conv-multiplicity*:

fixes $x :: 'a :: \text{factorial-semiring}$

assumes $n > 0$

shows *is-nth-power-upto-unit* $n\ x \longleftrightarrow (\forall p. \text{prime } p \longrightarrow n\ \text{dvd } \text{multiplicity } p\ x)$

proof (*cases* $x = 0$)

case *False*

show *?thesis*

proof *safe*

fix $p :: 'a$ **assume** p : *prime* p

assume *is-nth-power-upto-unit* $n\ x$

then obtain $u\ y$ **where** *is-unit* $u\ u * x = y \wedge n$

by (*auto simp: is-nth-power-upto-unit-def elim!: is-nth-powerE*)

from $p\ uy$ *assms* *False* **have** [*simp*]: $y \neq 0$ **by** (*auto simp: power-0-left*)

have *multiplicity* $p\ (u * x) = \text{multiplicity } p\ (y \wedge n)$

by (*subst uy(2) [symmetric] simp*)

also have *multiplicity* $p\ (u * x) = \text{multiplicity } p\ x$

by (*simp add: multiplicity-times-unit-right uy(1)*)

finally show $n\ \text{dvd } \text{multiplicity } p\ x$

using *False* **and** p **and** uy **and** *assms*

by (*auto simp: prime-elem-multiplicity-power-distrib*)

next

assume $*$: $\forall p. \text{prime } p \longrightarrow n\ \text{dvd } \text{multiplicity } p\ x$

have *multiplicity* $p\ ((\prod_{p \in \text{prime-factors } x} p \wedge (\text{multiplicity } p\ x\ \text{div } n)) \wedge n) =$
multiplicity $p\ x$ **if** *prime* p **for** p

proof –

from *that* **and** $*$ **have** $n\ \text{dvd } \text{multiplicity } p\ x$ **by** *blast*

have *multiplicity* $p\ x = 0$ **if** $p \notin \text{prime-factors } x$

using *that* **and** $\langle \text{prime } p \rangle$ **by** (*simp add: prime-factors-multiplicity*)

with *that* **and** $*$ **and** *assms* **show** *?thesis* **unfolding** *prod-power-distrib*

power-mult [symmetric]

by (*subst multiplicity-prod-prime-powers*) (*auto simp: in-prime-factors-imp-prime elim: dvdE*)

qed
with *assms False*
have *normalize* $((\prod_{p \in \text{prime-factors } x} p \wedge (\text{multiplicity } p \ x \ \text{div } n)) \wedge n) =$
normalize x
by (*intro multiplicity-eq-imp-eq*) (*auto simp: multiplicity-prod-prime-powers*)
thus *is-nth-power-upto-unit n x*
by (*auto intro: is-nth-power-upto-unitI*)
qed
qed (*use assms in <auto simp: is-nth-power-upto-unit-def>*)

lemma *is-nth-power-upto-unit-0-left* [*simp, intro*]: *is-nth-power-upto-unit 0 x* \longleftrightarrow
is-unit x

proof
assume *is-unit x*
thus *is-nth-power-upto-unit 0 x*
unfolding *is-nth-power-upto-unit-def* **by** (*intro exI[of - 1 div x]*) *auto*
next
assume *is-nth-power-upto-unit 0 x*
then obtain *u* **where** *is-unit u u * x = 1*
by (*auto simp: is-nth-power-upto-unit-def*)
thus *is-unit x*
by (*metis dvd-triv-right*)
qed

lemma *is-nth-power-upto-unit-unit* [*simp, intro*]:
assumes *is-unit x*
shows *is-nth-power-upto-unit n x*
using *assms* **by** (*auto simp: is-nth-power-upto-unit-def intro!: exI[of - 1 div x]*)

lemma *is-nth-power-upto-unit-1-left* [*simp, intro*]: *is-nth-power-upto-unit 1 x*
by (*auto simp: is-nth-power-upto-unit-def intro: exI[of - 1]*)

lemma *is-nth-power-upto-unit-mult-coprimeD1*:
fixes *x y :: 'a :: factorial-semiring*
assumes *coprime x y is-nth-power-upto-unit n (x * y)*
shows *is-nth-power-upto-unit n x*

proof –
consider $n = 0 \mid x = 0 \ n > 0 \mid x \neq 0 \ y = 0 \ n > 0 \mid n > 0 \ x \neq 0 \ y \neq 0$
by *force*
thus *?thesis*
proof *cases*
assume [*simp*]: $n = 0$
from *assms* **have** *is-unit (x * y)*
by *auto*
hence *is-unit x*
using *is-unit-mult-iff* **by** *blast*
thus *?thesis* **using** *assms* **by** *auto*
next
assume $x = 0 \ n > 0$

```

    thus ?thesis by (auto simp: is-nth-power-upto-unit-def)
next
  assume *:  $x \neq 0 \ y = 0 \ n > 0$ 
  with assms show ?thesis by auto
next
  assume *:  $n > 0$  and [simp]:  $x \neq 0 \ y \neq 0$ 
  show ?thesis
  proof (subst is-nth-power-upto-unit-conv-multiplicity[OF <n > 0>]; safe)
    fix p :: 'a assume p: prime p
    show n dvd multiplicity p x
    proof (cases p dvd x)
      case False
      thus ?thesis
      by (simp add: not-dvd-imp-multiplicity-0)
    next
      case True
      have n dvd multiplicity p (x * y)
      using assms(2) <n > 0> p by (auto simp: is-nth-power-upto-unit-conv-multiplicity)
      also have ... = multiplicity p x + multiplicity p y
      using p by (subst prime-elem-multiplicity-mult-distrib) auto
      also have  $\neg p \text{ dvd } y$ 
      using <coprime x y> <p dvd x> p not-prime-unit coprime-common-divisor
    by blast
    hence multiplicity p y = 0
    by (rule not-dvd-imp-multiplicity-0)
    finally show ?thesis by simp
  qed
  qed
  qed
  qed

```

```

lemma is-nth-power-upto-unit-mult-coprimeD2:
  fixes x y :: 'a :: factorial-semiring
  assumes coprime x y is-nth-power-upto-unit n (x * y)
  shows is-nth-power-upto-unit n y
  using assms is-nth-power-upto-unit-mult-coprimeD1[of y x]
  by (simp-all add: mult-ac coprime-commute)

```

1.2 Definition

Gaussian integers are the ring $\mathbb{Z}[i]$ which is formed either by formally adjoining an element i with $i^2 = -1$ to \mathbb{Z} or by taking all the complex numbers with integer real and imaginary part.

We define them simply by giving an appropriate ring structure to \mathbb{Z}^2 , with the first component representing the real part and the second component the imaginary part:

```

codatatype gauss-int = Gauss-Int (ReZ: int) (ImZ: int)

```

The following is the imaginary unit i in the Gaussian integers, which we will denote as iz :

primcorec *gauss-i* **where**

$\text{ReZ } \text{gauss-i} = 0$
 $| \text{ImZ } \text{gauss-i} = 1$

lemma *gauss-int-eq-iff*: $x = y \longleftrightarrow \text{ReZ } x = \text{ReZ } y \wedge \text{ImZ } x = \text{ImZ } y$
by (*cases x*; *cases y*) *auto*

Next, we define the canonical injective homomorphism from the Gaussian integers into the complex numbers:

primcorec *gauss2complex* **where**

$\text{Re } (\text{gauss2complex } z) = \text{of-int } (\text{ReZ } z)$
 $| \text{Im } (\text{gauss2complex } z) = \text{of-int } (\text{ImZ } z)$

declare $[[\text{coercion } \text{gauss2complex}]]$

lemma *gauss2complex-eq-iff* [*simp*]: $\text{gauss2complex } z = \text{gauss2complex } u \longleftrightarrow z = u$
by (*simp add: complex-eq-iff gauss-int-eq-iff*)

Gaussian integers also have conjugates, just like complex numbers:

primcorec *gauss-cnj* **where**

$\text{ReZ } (\text{gauss-cnj } z) = \text{ReZ } z$
 $| \text{ImZ } (\text{gauss-cnj } z) = -\text{ImZ } z$

In the remainder of this section, we prove that Gaussian integers are a commutative ring of characteristic 0 and several other trivial algebraic properties.

instantiation *gauss-int* :: *comm-ring-1*
begin

primcorec *zero-gauss-int* **where**

$\text{ReZ } \text{zero-gauss-int} = 0$
 $| \text{ImZ } \text{zero-gauss-int} = 0$

primcorec *one-gauss-int* **where**

$\text{ReZ } \text{one-gauss-int} = 1$
 $| \text{ImZ } \text{one-gauss-int} = 0$

primcorec *uminus-gauss-int* **where**

$\text{ReZ } (\text{uminus-gauss-int } x) = -\text{ReZ } x$
 $| \text{ImZ } (\text{uminus-gauss-int } x) = -\text{ImZ } x$

primcorec *plus-gauss-int* **where**

$\text{ReZ } (\text{plus-gauss-int } x \ y) = \text{ReZ } x + \text{ReZ } y$

| $\text{Im}Z$ (*plus-gauss-int* $x y$) = $\text{Im}Z x + \text{Im}Z y$

primcorec *minus-gauss-int* **where**

$\text{Re}Z$ (*minus-gauss-int* $x y$) = $\text{Re}Z x - \text{Re}Z y$
| $\text{Im}Z$ (*minus-gauss-int* $x y$) = $\text{Im}Z x - \text{Im}Z y$

primcorec *times-gauss-int* **where**

$\text{Re}Z$ (*times-gauss-int* $x y$) = $\text{Re}Z x * \text{Re}Z y - \text{Im}Z x * \text{Im}Z y$
| $\text{Im}Z$ (*times-gauss-int* $x y$) = $\text{Re}Z x * \text{Im}Z y + \text{Im}Z x * \text{Re}Z y$

instance

by *intro-classes* (*auto simp: gauss-int-eq-iff algebra-simps*)

end

lemma *gauss-i-times-i* [*simp*]: $i_Z * i_Z = (-1 :: \text{gauss-int})$
and *gauss-cnj-i* [*simp*]: *gauss-cnj* $i_Z = -i_Z$
by (*simp-all add: gauss-int-eq-iff*)

lemma *gauss-cnj-eq-0-iff* [*simp*]: *gauss-cnj* $z = 0 \longleftrightarrow z = 0$
by (*auto simp: gauss-int-eq-iff*)

lemma *gauss-cnj-eq-self*: $\text{Im} z = 0 \implies \text{gauss-cnj} z = z$
and *gauss-cnj-eq-minus-self*: $\text{Re} z = 0 \implies \text{gauss-cnj} z = -z$
by (*auto simp: gauss-int-eq-iff*)

lemma *ReZ-of-nat* [*simp*]: $\text{Re}Z$ (*of-nat* n) = *of-nat* n
and *ImZ-of-nat* [*simp*]: $\text{Im}Z$ (*of-nat* n) = 0
by (*induction n; simp*)+

lemma *ReZ-of-int* [*simp*]: $\text{Re}Z$ (*of-int* n) = n
and *ImZ-of-int* [*simp*]: $\text{Im}Z$ (*of-int* n) = 0
by (*induction n; simp*)+

lemma *ReZ-numeral* [*simp*]: $\text{Re}Z$ (*numeral* n) = *numeral* n
and *ImZ-numeral* [*simp*]: $\text{Im}Z$ (*numeral* n) = 0
by (*subst of-nat-numeral [symmetric], subst ReZ-of-nat ImZ-of-nat, simp*)+

lemma *gauss2complex-0* [*simp*]: *gauss2complex* 0 = 0
and *gauss2complex-1* [*simp*]: *gauss2complex* 1 = 1
and *gauss2complex-i* [*simp*]: *gauss2complex* $i_Z = i$
and *gauss2complex-add* [*simp*]: *gauss2complex* ($x + y$) = *gauss2complex* $x +$
gauss2complex y
and *gauss2complex-diff* [*simp*]: *gauss2complex* ($x - y$) = *gauss2complex* $x -$
gauss2complex y
and *gauss2complex-mult* [*simp*]: *gauss2complex* ($x * y$) = *gauss2complex* $x *$
gauss2complex y
and *gauss2complex-uminus* [*simp*]: *gauss2complex* ($-x$) = $-$ *gauss2complex* x
and *gauss2complex-cnj* [*simp*]: *gauss2complex* (*gauss-cnj* x) = *cnj* (*gauss2complex* x)

x)

by (simp-all add: complex-eq-iff)

lemma *gauss2complex-of-nat* [simp]: $\text{gauss2complex (of-nat } n) = \text{of-nat } n$
by (simp add: complex-eq-iff)

lemma *gauss2complex-eq-0-iff* [simp]: $\text{gauss2complex } x = 0 \longleftrightarrow x = 0$
and *gauss2complex-eq-1-iff* [simp]: $\text{gauss2complex } x = 1 \longleftrightarrow x = 1$
and *zero-eq-gauss2complex-iff* [simp]: $0 = \text{gauss2complex } x \longleftrightarrow x = 0$
and *one-eq-gauss2complex-iff* [simp]: $1 = \text{gauss2complex } x \longleftrightarrow x = 1$
by (simp-all add: complex-eq-iff gauss-int-eq-iff)

lemma *gauss-i-times-gauss-i-times* [simp]: $i_{\mathbb{Z}} * (i_{\mathbb{Z}} * x) = (-x :: \text{gauss-int})$
by (subst mult.assoc [symmetric], subst gauss-i-times-i) auto

lemma *gauss-i-neq-0* [simp]: $i_{\mathbb{Z}} \neq 0$ $0 \neq i_{\mathbb{Z}}$
and *gauss-i-neq-1* [simp]: $i_{\mathbb{Z}} \neq 1$ $1 \neq i_{\mathbb{Z}}$
and *gauss-i-neq-of-nat* [simp]: $i_{\mathbb{Z}} \neq \text{of-nat } n$ $\text{of-nat } n \neq i_{\mathbb{Z}}$
and *gauss-i-neq-of-int* [simp]: $i_{\mathbb{Z}} \neq \text{of-int } n$ $\text{of-int } n \neq i_{\mathbb{Z}}$
and *gauss-i-neq-numeral* [simp]: $i_{\mathbb{Z}} \neq \text{numeral } m$ $\text{numeral } m \neq i_{\mathbb{Z}}$
by (auto simp: gauss-int-eq-iff)

lemma *gauss-cnj-0* [simp]: $\text{gauss-cnj } 0 = 0$
and *gauss-cnj-1* [simp]: $\text{gauss-cnj } 1 = 1$
and *gauss-cnj-cnj* [simp]: $\text{gauss-cnj (gauss-cnj } z) = z$
and *gauss-cnj-uminus* [simp]: $\text{gauss-cnj } (-a) = -\text{gauss-cnj } a$
and *gauss-cnj-add* [simp]: $\text{gauss-cnj } (a + b) = \text{gauss-cnj } a + \text{gauss-cnj } b$
and *gauss-cnj-diff* [simp]: $\text{gauss-cnj } (a - b) = \text{gauss-cnj } a - \text{gauss-cnj } b$
and *gauss-cnj-mult* [simp]: $\text{gauss-cnj } (a * b) = \text{gauss-cnj } a * \text{gauss-cnj } b$
and *gauss-cnj-of-nat* [simp]: $\text{gauss-cnj (of-nat } n1) = \text{of-nat } n1$
and *gauss-cnj-of-int* [simp]: $\text{gauss-cnj (of-int } n2) = \text{of-int } n2$
and *gauss-cnj-numeral* [simp]: $\text{gauss-cnj (numeral } n3) = \text{numeral } n3$
by (simp-all add: gauss-int-eq-iff)

lemma *gauss-cnj-power* [simp]: $\text{gauss-cnj } (a \wedge n) = \text{gauss-cnj } a \wedge n$
by (induction n) auto

lemma *gauss-cnj-sum* [simp]: $\text{gauss-cnj (sum } f A) = (\sum x \in A. \text{gauss-cnj } (f x))$
by (induction A rule: infinite-finite-induct) auto

lemma *gauss-cnj-prod* [simp]: $\text{gauss-cnj (prod } f A) = (\prod x \in A. \text{gauss-cnj } (f x))$
by (induction A rule: infinite-finite-induct) auto

lemma *of-nat-dvd-of-nat*:
assumes *a dvd b*
shows *of-nat a dvd (of-nat b :: 'a :: comm-semiring-1)*
using *assms* by auto

lemma *of-int-dvd-imp-dvd-gauss-cnj*:

```

fixes z :: gauss-int
assumes of-int n dvd z
shows of-int n dvd gauss-cnj z
proof –
  from assms obtain u where z = of-int n * u by blast
  hence gauss-cnj z = of-int n * gauss-cnj u
    by simp
  thus ?thesis by auto
qed

```

```

lemma of-nat-dvd-imp-dvd-gauss-cnj:
  fixes z :: gauss-int
  assumes of-nat n dvd z
  shows of-nat n dvd gauss-cnj z
  using of-int-dvd-imp-dvd-gauss-cnj[of int n] assms by simp

```

```

lemma of-int-dvd-of-int-gauss-int-iff:
  (of-int m :: gauss-int) dvd of-int n  $\longleftrightarrow$  m dvd n
proof
  assume of-int m dvd (of-int n :: gauss-int)
  then obtain a :: gauss-int where of-int n = of-int m * a
    by blast
  thus m dvd n
    by (auto simp: gauss-int-eq-iff)
qed auto

```

```

lemma of-nat-dvd-of-nat-gauss-int-iff:
  (of-nat m :: gauss-int) dvd of-nat n  $\longleftrightarrow$  m dvd n
  using of-int-dvd-of-int-gauss-int-iff[of int m int n] by simp

```

```

lemma gauss-cnj-dvd:
  assumes a dvd b
  shows gauss-cnj a dvd gauss-cnj b
proof –
  from assms obtain c where b = a * c
    by blast
  hence gauss-cnj b = gauss-cnj a * gauss-cnj c
    by simp
  thus ?thesis by auto
qed

```

```

lemma gauss-cnj-dvd-iff: gauss-cnj a dvd gauss-cnj b  $\longleftrightarrow$  a dvd b
  using gauss-cnj-dvd[of a b] gauss-cnj-dvd[of gauss-cnj a gauss-cnj b] by auto

```

```

lemma gauss-cnj-dvd-left-iff: gauss-cnj a dvd b  $\longleftrightarrow$  a dvd gauss-cnj b
  by (subst gauss-cnj-dvd-iff [symmetric]) auto

```

```

lemma gauss-cnj-dvd-right-iff: a dvd gauss-cnj b  $\longleftrightarrow$  gauss-cnj a dvd b
  by (rule gauss-cnj-dvd-left-iff [symmetric])

```

```

instance gauss-int :: idom
proof
  fix z u :: gauss-int
  assume z ≠ 0 u ≠ 0
  hence gauss2complex z * gauss2complex u ≠ 0
    by simp
  also have gauss2complex z * gauss2complex u = gauss2complex (z * u)
    by simp
  finally show z * u ≠ 0
    unfolding gauss2complex-eq-0-iff .
qed

```

```

instance gauss-int :: ring-char-0
  by intro-classes (auto intro!: injI simp: gauss-int-eq-iff)

```

1.3 Pretty-printing

The following lemma collection provides better pretty-printing of Gaussian integers so that e.g. evaluation with the ‘value’ command produces nicer results.

lemma *gauss-int-code-post* [*code-post*]:

```

Gauss-Int 0 0 = 0
Gauss-Int 0 1 = iZ
Gauss-Int 0 (-1) = -iZ
Gauss-Int 1 0 = 1
Gauss-Int 1 1 = 1 + iZ
Gauss-Int 1 (-1) = 1 - iZ
Gauss-Int (-1) 0 = -1
Gauss-Int (-1) 1 = -1 + iZ
Gauss-Int (-1) (-1) = -1 - iZ
Gauss-Int (numeral b) 0 = numeral b
Gauss-Int (-numeral b) 0 = -numeral b
Gauss-Int (numeral b) 1 = numeral b + iZ
Gauss-Int (-numeral b) 1 = -numeral b + iZ
Gauss-Int (numeral b) (-1) = numeral b - iZ
Gauss-Int (-numeral b) (-1) = -numeral b - iZ
Gauss-Int 0 (numeral b) = numeral b * iZ
Gauss-Int 0 (-numeral b) = -numeral b * iZ
Gauss-Int 1 (numeral b) = 1 + numeral b * iZ
Gauss-Int 1 (-numeral b) = 1 - numeral b * iZ
Gauss-Int (-1) (numeral b) = -1 + numeral b * iZ
Gauss-Int (-1) (-numeral b) = -1 - numeral b * iZ
Gauss-Int (numeral a) (numeral b) = numeral a + numeral b * iZ
Gauss-Int (numeral a) (-numeral b) = numeral a - numeral b * iZ
Gauss-Int (-numeral a) (numeral b) = -numeral a + numeral b * iZ
Gauss-Int (-numeral a) (-numeral b) = -numeral a - numeral b * iZ
by (simp-all add: gauss-int-eq-iff)

```

```

value iZ ^ 3
value 2 * (3 + iZ)
value (2 + iZ) * (2 - iZ)

```

1.4 Norm

The square of the complex norm (or complex modulus) on the Gaussian integers gives us a norm that always returns a natural number. We will later show that this is also a Euclidean norm (in the sense of a Euclidean ring).

definition *gauss-int-norm* :: *gauss-int* ⇒ *nat* **where**
gauss-int-norm *z* = *nat* (*ReZ* *z* ^ 2 + *ImZ* *z* ^ 2)

lemma *gauss-int-norm-0* [*simp*]: *gauss-int-norm* 0 = 0
and *gauss-int-norm-1* [*simp*]: *gauss-int-norm* 1 = 1
and *gauss-int-norm-i* [*simp*]: *gauss-int-norm* i_Z = 1
and *gauss-int-norm-cnj* [*simp*]: *gauss-int-norm* (*gauss-cnj* *z*) = *gauss-int-norm* *z*
and *gauss-int-norm-of-nat* [*simp*]: *gauss-int-norm* (*of-nat* *n*) = *n* ^ 2
and *gauss-int-norm-of-int* [*simp*]: *gauss-int-norm* (*of-int* *m*) = *nat* (*m* ^ 2)
and *gauss-int-norm-of-numeral* [*simp*]: *gauss-int-norm* (*numeral* *n*[′]) = *numeral* (*Num.sqr* *n*[′])
by (*simp-all* *add*: *gauss-int-norm-def* *nat-power-eq*)

lemma *gauss-int-norm-uminus* [*simp*]: *gauss-int-norm* (−*z*) = *gauss-int-norm* *z*
by (*simp* *add*: *gauss-int-norm-def*)

lemma *gauss-int-norm-eq-0-iff* [*simp*]: *gauss-int-norm* *z* = 0 ↔ *z* = 0

proof

```

assume gauss-int-norm z = 0
hence ReZ z ^ 2 + ImZ z ^ 2 ≤ 0
by (simp add: gauss-int-norm-def)
moreover have ReZ z ^ 2 + ImZ z ^ 2 ≥ 0
by simp
ultimately have ReZ z ^ 2 + ImZ z ^ 2 = 0
by linarith
thus z = 0
by (auto simp: gauss-int-eq-iff)

```

qed *auto*

lemma *gauss-int-norm-pos-iff* [*simp*]: *gauss-int-norm* *z* > 0 ↔ *z* ≠ 0
using *gauss-int-norm-eq-0-iff*[*of* *z*] **by** (*auto* *intro*: *Nat.gr0I*)

lemma *real-gauss-int-norm*: *real* (*gauss-int-norm* *z*) = *norm* (*gauss2complex* *z*) ^ 2
by (*auto* *simp*: *cmod-def* *gauss-int-norm-def*)

lemma *gauss-int-norm-mult*: *gauss-int-norm* (*z* * *u*) = *gauss-int-norm* *z* * *gauss-int-norm* *u*

u

proof –

have $\text{real}(\text{gauss-int-norm}(z * u)) = \text{real}(\text{gauss-int-norm } z * \text{gauss-int-norm } u)$
unfolding *of-nat-mult* **by** (*simp add: real-gauss-int-norm norm-power norm-mult power-mult-distrib*)
thus *?thesis* **by** (*subst (asm) of-nat-eq-iff*)
qed

lemma *self-mult-gauss-cnj*: $z * \text{gauss-cnj } z = \text{of-nat}(\text{gauss-int-norm } z)$
by (*simp add: gauss-int-norm-def gauss-int-eq-iff algebra-simps power2-eq-square*)

lemma *gauss-cnj-mult-self*: $\text{gauss-cnj } z * z = \text{of-nat}(\text{gauss-int-norm } z)$
by (*subst mult.commute, rule self-mult-gauss-cnj*)

lemma *self-plus-gauss-cnj*: $z + \text{gauss-cnj } z = \text{of-int}(2 * \text{Re } z)$
and *self-minus-gauss-cnj*: $z - \text{gauss-cnj } z = \text{of-int}(2 * \text{Im } z) * i_{\mathbb{Z}}$
by (*auto simp: gauss-int-eq-iff*)

lemma *gauss-int-norm-dvd-mono*:
assumes $a \text{ dvd } b$
shows $\text{gauss-int-norm } a \text{ dvd } \text{gauss-int-norm } b$
proof –
from *assms* **obtain** c **where** $b = a * c$ **by** *blast*
hence $\text{gauss-int-norm } b = \text{gauss-int-norm}(a * c)$
by *metis*
thus *?thesis* **by** (*simp add: gauss-int-norm-mult*)
qed

lemma *gauss-int-norm-power*: $\text{gauss-int-norm}(x \wedge n) = \text{gauss-int-norm } x \wedge n$
by (*metis gauss-cnj-mult-self gauss-cnj-power of-nat-eq-of-nat-power-cancel-iff power-mult-distrib*)

A Gaussian integer is a unit iff its norm is 1, and this is the case precisely for the four elements ± 1 and $\pm i$:

lemma *is-unit-gauss-int-iff*: $x \text{ dvd } 1 \iff x \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}} :: \text{gauss-int}\}$
and *is-unit-gauss-int-iff'*: $x \text{ dvd } 1 \iff \text{gauss-int-norm } x = 1$

proof –

have $x \text{ dvd } 1$ **if** $x \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}}\}$

proof –

from *that* **have** $*$: $x * \text{gauss-cnj } x = 1$

by (*auto simp: gauss-int-norm-def*)

show $x \text{ dvd } 1$ **by** (*subst * [symmetric] simp*)

qed

moreover **have** $\text{gauss-int-norm } x = 1$ **if** $x \text{ dvd } 1$

using *gauss-int-norm-dvd-mono[OF that]* **by** *simp*

moreover **have** $x \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}}\}$ **if** $\text{gauss-int-norm } x = 1$

proof –

from *that* **have** $*$: $(\text{Re } x)^2 + (\text{Im } x)^2 = 1$

by (*auto simp: gauss-int-norm-def nat-eq-iff*)

hence $\text{Re}Z\ x \wedge 2 \leq 1$ **and** $\text{Im}Z\ x \wedge 2 \leq 1$
using *zero-le-power2[of ImZ x] zero-le-power2[of ReZ x]* **by** *linarith+*
hence $|\text{Re}Z\ x| \leq 1$ **and** $|\text{Im}Z\ x| \leq 1$
by *(auto simp: abs-square-le-1)*
hence $\text{Re}Z\ x \in \{-1, 0, 1\}$ **and** $\text{Im}Z\ x \in \{-1, 0, 1\}$
by *auto*
thus $x \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}} :: \text{gauss-int}\}$
using * **by** *(auto simp: gauss-int-eq-iff)*
qed
ultimately show $x \text{ dvd } 1 \longleftrightarrow x \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}} :: \text{gauss-int}\}$
and $x \text{ dvd } 1 \longleftrightarrow \text{gauss-int-norm } x = 1$
by *blast+*
qed

lemma *is-unit-gauss-i [simp, intro]: (gauss-i :: gauss-int) dvd 1*
by *(simp add: is-unit-gauss-int-iff)*

lemma *gauss-int-norm-eq-Suc-0-iff: gauss-int-norm x = Suc 0 \longleftrightarrow x dvd 1*
by *(simp add: is-unit-gauss-int-iff')*

lemma *is-unit-gauss-cn timer [intro]: z dvd 1 \implies gauss-cn timer z dvd 1*
by *(simp add: is-unit-gauss-int-iff')*

lemma *is-unit-gauss-cn timer-iff [simp]: gauss-cn timer z dvd 1 \longleftrightarrow z dvd 1*
by *(simp add: is-unit-gauss-int-iff')*

1.5 Division and normalisation

We define a rounding operation that takes a complex number and returns a Gaussian integer by rounding the real and imaginary parts separately:

primcorec *round-complex :: complex \Rightarrow gauss-int where*
 $\text{Re}Z\ (\text{round-complex } z) = \text{round } (\text{Re } z)$
 $|\ \text{Im}Z\ (\text{round-complex } z) = \text{round } (\text{Im } z)$

The distance between a rounded complex number and the original one is no more than $\frac{1}{2}\sqrt{2}$:

lemma *norm-round-complex-le: norm (z - gauss2complex (round-complex z)) $\wedge 2 \leq 1 / 2$*

proof -

have $(\text{Re } z - \text{Re}Z\ (\text{round-complex } z)) \wedge 2 \leq (1 / 2) \wedge 2$
using *of-int-round-abs-le[of Re z]*
by *(subst abs-le-square-iff [symmetric]) (auto simp: abs-minus-commute)*
moreover have $(\text{Im } z - \text{Im}Z\ (\text{round-complex } z)) \wedge 2 \leq (1 / 2) \wedge 2$
using *of-int-round-abs-le[of Im z]*
by *(subst abs-le-square-iff [symmetric]) (auto simp: abs-minus-commute)*
ultimately have $(\text{Re } z - \text{Re}Z\ (\text{round-complex } z)) \wedge 2 + (\text{Im } z - \text{Im}Z\ (\text{round-complex } z)) \wedge 2 \leq$
 $(1 / 2) \wedge 2 + (1 / 2) \wedge 2$

```

    by (rule add-mono)
  thus norm (z - gauss2complex (round-complex z)) ^ 2 ≤ 1 / 2
    by (simp add: cmod-def power2-eq-square)
qed

```

lemma *dist-round-complex-le*: $\text{dist } z \text{ (gauss2complex (round-complex } z)) \leq \text{sqrt } 2 / 2$

```

proof -
  have dist z (gauss2complex (round-complex z)) ^ 2 =
    norm (z - gauss2complex (round-complex z)) ^ 2
    by (simp add: dist-norm)
  also have ... ≤ 1 / 2
    by (rule norm-round-complex-le)
  also have ... = (sqrt 2 / 2) ^ 2
    by (simp add: power2-eq-square)
  finally show ?thesis
    by (rule power2-le-imp-le) auto
qed

```

We can now define division on Gaussian integers simply by performing the division in the complex numbers and rounding the result. This also gives us a remainder operation defined accordingly for which the norm of the remainder is always smaller than the norm of the divisor.

We can also define a normalisation operation that returns a canonical representative for each association class. Since the four units of the Gaussian integers are ± 1 and $\pm i$, each association class (other than 0) has four representatives, one in each quadrant. We simply define the one in the upper-right quadrant (i.e. the one with non-negative imaginary part and positive real part) as the canonical one.

Thus, the Gaussian integers form a Euclidean ring. This gives us many things, most importantly the existence of GCDs and LCMs and unique factorisation.

```

instantiation gauss-int :: algebraic-semidom
begin

```

```

definition divide-gauss-int :: gauss-int ⇒ gauss-int ⇒ gauss-int where
  divide-gauss-int a b = round-complex (gauss2complex a / gauss2complex b)

```

```

instance proof
  fix a :: gauss-int
  show a div 0 = 0
    by (auto simp: gauss-int-eq-iff divide-gauss-int-def)
next
  fix a b :: gauss-int assume b ≠ 0
  thus a * b div b = a
    by (auto simp: gauss-int-eq-iff divide-gauss-int-def)
qed

```

end

instantiation *gauss-int* :: *semidom-divide-unit-factor*
begin

definition *unit-factor-gauss-int* :: *gauss-int* \Rightarrow *gauss-int* **where**

unit-factor-gauss-int $z =$
 (*if* $z = 0$ *then* 0 *else*
 if $\text{Im}Z\ z \geq 0 \wedge \text{Re}Z\ z > 0$ *then* 1
 else if $\text{Re}Z\ z \leq 0 \wedge \text{Im}Z\ z > 0$ *then* i_Z
 else if $\text{Im}Z\ z \leq 0 \wedge \text{Re}Z\ z < 0$ *then* -1
 else $-i_Z$)

instance proof

show *unit-factor* $(0 :: \text{gauss-int}) = 0$
 by (*simp add: unit-factor-gauss-int-def*)

next

fix $z :: \text{gauss-int}$
assume *is-unit* z
thus *unit-factor* $z = z$
 by (*subst (asm) is-unit-gauss-int-iff*) (*auto simp: unit-factor-gauss-int-def*)

next

fix $z :: \text{gauss-int}$
assume $z: z \neq 0$
thus *is-unit* (*unit-factor* z)
 by (*subst is-unit-gauss-int-iff*) (*auto simp: unit-factor-gauss-int-def*)

next

fix $z\ u :: \text{gauss-int}$
assume *is-unit* z
hence $z \in \{1, -1, i_Z, -i_Z\}$
 by (*subst (asm) is-unit-gauss-int-iff*)
thus *unit-factor* $(z * u) = z * \text{unit-factor}\ u$
 by (*safe; auto simp: unit-factor-gauss-int-def gauss-int-eq-iff[of u 0]*)

qed

end

instantiation *gauss-int* :: *normalization-semidom*
begin

definition *normalize-gauss-int* :: *gauss-int* \Rightarrow *gauss-int* **where**

normalize-gauss-int $z =$
 (*if* $z = 0$ *then* 0 *else*
 if $\text{Im}Z\ z \geq 0 \wedge \text{Re}Z\ z > 0$ *then* z
 else if $\text{Re}Z\ z \leq 0 \wedge \text{Im}Z\ z > 0$ *then* $-i_Z * z$
 else if $\text{Im}Z\ z \leq 0 \wedge \text{Re}Z\ z < 0$ *then* $-z$
 else $i_Z * z$)

```

instance proof
  show normalize (0 :: gauss-int) = 0
    by (simp add: normalize-gauss-int-def)
next
  fix z :: gauss-int
  show unit-factor z * normalize z = z
    by (auto simp: normalize-gauss-int-def unit-factor-gauss-int-def algebra-simps)
qed

end

lemma normalize-gauss-int-of-nat [simp]: normalize (of-nat n :: gauss-int) = of-nat
n
  and normalize-gauss-int-of-int [simp]: normalize (of-int m :: gauss-int) = of-int
  |m|
  and normalize-gauss-int-of-numeral [simp]: normalize (numeral n' :: gauss-int)
  = numeral n'
  by (auto simp: normalize-gauss-int-def)

lemma normalize-gauss-i [simp]: normalize iZ = 1
  by (simp add: normalize-gauss-int-def)

lemma gauss-int-norm-normalize [simp]: gauss-int-norm (normalize x) = gauss-int-norm
x
  by (simp add: normalize-gauss-int-def gauss-int-norm-mult)

lemma normalized-gauss-int:
  assumes normalize z = z
  shows ReZ z ≥ 0 ImZ z ≥ 0
  using assms
  by (cases ReZ z 0 :: int rule: linorder-cases;
    cases ImZ z 0 :: int rule: linorder-cases;
    simp add: normalize-gauss-int-def gauss-int-eq-iff)+

lemma normalized-gauss-int':
  assumes normalize z = z z ≠ 0
  shows ReZ z > 0 ImZ z ≥ 0
  using assms
  by (cases ReZ z 0 :: int rule: linorder-cases;
    cases ImZ z 0 :: int rule: linorder-cases;
    simp add: normalize-gauss-int-def gauss-int-eq-iff)+

lemma normalized-gauss-int-iff:
  normalize z = z ↔ z = 0 ∨ ReZ z > 0 ∧ ImZ z ≥ 0
  by (cases ReZ z 0 :: int rule: linorder-cases;
    cases ImZ z 0 :: int rule: linorder-cases;
    simp add: normalize-gauss-int-def gauss-int-eq-iff)+

instantiation gauss-int :: idom-modulo

```

begin

definition *modulo-gauss-int* :: *gauss-int* \Rightarrow *gauss-int* \Rightarrow *gauss-int* **where**
modulo-gauss-int *a b* = *a* - *a div b* * *b*

instance proof

fix *a b* :: *gauss-int*
show *a div b* * *b* + *a mod b* = *a*
by (*simp add: modulo-gauss-int-def*)
qed

end

lemma *gauss-int-norm-mod-less-aux*:

assumes [*simp*]: *b* \neq 0
shows 2 * *gauss-int-norm* (*a mod b*) \leq *gauss-int-norm* *b*
proof -
define *a'* *b'* **where** *a'* = *gauss2complex* *a* **and** *b'* = *gauss2complex* *b*
have [*simp*]: *b'* \neq 0 **by** (*simp add: b'-def*)
have *gauss-int-norm* (*a mod b*) =
 norm (*gauss2complex* (*a* - *round-complex* (*a'* / *b'*) * *b*)) \wedge 2
unfolding *modulo-gauss-int-def*
by (*subst real-gauss-int-norm [symmetric]*) (*auto simp add: divide-gauss-int-def*
a'-def b'-def)
also have *gauss2complex* (*a* - *round-complex* (*a'* / *b'*) * *b*) =
 a' - *gauss2complex* (*round-complex* (*a'* / *b'*)) * *b'*
by (*simp add: a'-def b'-def*)
also have ... = (*a'* / *b'* - *gauss2complex* (*round-complex* (*a'* / *b'*))) * *b'*
by (*simp add: field-simps*)
also have *norm* ... \wedge 2 = *norm* (*a'* / *b'* - *gauss2complex* (*round-complex* (*a'* /
b'))) \wedge 2 * *norm* *b'* \wedge 2
by (*simp add: norm-mult power-mult-distrib*)
also have ... \leq 1 / 2 * *norm* *b'* \wedge 2
by (*intro mult-right-mono norm-round-complex-le*) *auto*
also have *norm* *b'* \wedge 2 = *gauss-int-norm* *b*
by (*simp add: b'-def real-gauss-int-norm*)
finally show ?*thesis* **by** *linarith*
qed

lemma *gauss-int-norm-mod-less*:

assumes [*simp*]: *b* \neq 0
shows *gauss-int-norm* (*a mod b*) < *gauss-int-norm* *b*
proof -
have *gauss-int-norm* *b* > 0 **by** *simp*
thus *gauss-int-norm* (*a mod b*) < *gauss-int-norm* *b*
using *gauss-int-norm-mod-less-aux*[*OF* *assms*, of *a*] **by** *presburger*
qed

lemma *gauss-int-norm-dvd-imp-le*:

```

    assumes  $b \neq 0$ 
    shows  $\text{gauss-int-norm } a \leq \text{gauss-int-norm } (a * b)$ 
  proof (cases  $a = 0$ )
    case False
      thus ?thesis using assms by (intro dvd-imp-le gauss-int-norm-dvd-mono) auto
    qed auto

  instantiation gauss-int :: euclidean-ring
  begin

  definition euclidean-size-gauss-int :: gauss-int  $\Rightarrow$  nat where
    [simp]: euclidean-size-gauss-int = gauss-int-norm

  instance proof
    show euclidean-size ( $0 :: \text{gauss-int}$ ) = 0
      by simp
    next
    fix  $a b :: \text{gauss-int}$  assume [simp]:  $b \neq 0$ 
    show euclidean-size ( $a \bmod b$ ) < euclidean-size  $b$ 
      using gauss-int-norm-mod-less[of  $b a$ ] by simp
    show euclidean-size  $a \leq \text{euclidean-size } (a * b)$ 
      by (simp add: gauss-int-norm-dvd-imp-le)
    qed

  end

  instance gauss-int :: normalization-euclidean-semiring ..

  instantiation gauss-int :: euclidean-ring-gcd
  begin

  definition gcd-gauss-int :: gauss-int  $\Rightarrow$  gauss-int  $\Rightarrow$  gauss-int where
    gcd-gauss-int  $\equiv$  normalization-euclidean-semiring-class.gcd
  definition lcm-gauss-int :: gauss-int  $\Rightarrow$  gauss-int  $\Rightarrow$  gauss-int where
    lcm-gauss-int  $\equiv$  normalization-euclidean-semiring-class.lcm
  definition Gcd-gauss-int :: gauss-int set  $\Rightarrow$  gauss-int where
    Gcd-gauss-int  $\equiv$  normalization-euclidean-semiring-class.Gcd
  definition Lcm-gauss-int :: gauss-int set  $\Rightarrow$  gauss-int where
    Lcm-gauss-int  $\equiv$  normalization-euclidean-semiring-class.Lcm

  instance
    by intro-classes
      (simp-all add: gcd-gauss-int-def lcm-gauss-int-def Gcd-gauss-int-def Lcm-gauss-int-def)

  end

  lemma gcd-gauss-cnj:  $\text{gcd } (\text{gauss-cnj } x) (\text{gauss-cnj } y) = \text{normalize } (\text{gauss-cnj } (\text{gcd } x y))$ 
  proof (rule sym, rule gcdI)

```

show $\bigwedge d. \llbracket d \text{ dvd } \text{gauss-cnj } x; d \text{ dvd } \text{gauss-cnj } y \rrbracket \implies d \text{ dvd } \text{normalize } (\text{gauss-cnj } (\text{gcd } x \ y))$

by (*auto simp: gauss-cnj-dvd-right-iff*)

qed (*auto simp: gauss-cnj-dvd-left-iff*)

lemma *gcd-gauss-cnj-left*: $\text{gcd } (\text{gauss-cnj } x) \ y = \text{normalize } (\text{gauss-cnj } (\text{gcd } x \ (\text{gauss-cnj } y)))$

by (*metis gauss-cnj-cnj gcd-gauss-cnj*)

lemma *gcd-gauss-cnj-right*: $\text{gcd } x \ (\text{gauss-cnj } y) = \text{normalize } (\text{gauss-cnj } (\text{gcd } (\text{gauss-cnj } x) \ y))$

by (*subst gcd-gauss-cnj [symmetric]*) *auto*

lemma *multiplicity-gauss-cnj*: $\text{multiplicity } (\text{gauss-cnj } a) \ (\text{gauss-cnj } b) = \text{multiplicity } a \ b$

unfolding *multiplicity-def gauss-cnj-power [symmetric] gauss-cnj-dvd-iff ..*

lemma *multiplicity-gauss-int-of-nat*:

$\text{multiplicity } (\text{of-nat } a) \ (\text{of-nat } b :: \text{gauss-int}) = \text{multiplicity } a \ b$

unfolding *multiplicity-def of-nat-power [symmetric] of-nat-dvd-of-nat-gauss-int-iff ..*

..

lemma *gauss-int-dvd-same-norm-imp-associated*:

assumes $z1 \text{ dvd } z2 \ \text{gauss-int-norm } z1 = \text{gauss-int-norm } z2$

shows $\text{normalize } z1 = \text{normalize } z2$

proof (*cases z1 = 0*)

case [*simp*]: *False*

from *assms(1)* **obtain** u **where** $z2 = z1 * u$ **by** *blast*

from *assms* **have** $\text{gauss-int-norm } u = 1$

by (*auto simp: gauss-int-norm-mult u*)

hence *is-unit u*

by (*simp add: is-unit-gauss-int-iff'*)

with u **show** *?thesis* **by** *simp*

qed (*use assms in auto*)

lemma *gcd-of-int-gauss-int*: $\text{gcd } (\text{of-int } a :: \text{gauss-int}) \ (\text{of-int } b) = \text{of-int } (\text{gcd } a \ b)$

proof (*induction nat |b| arbitrary: a b rule: less-induct*)

case (*less b a*)

show *?case*

proof (*cases b = 0*)

case *False*

have $\text{of-int } (\text{gcd } a \ b) = \text{of-int } (\text{gcd } b \ (a \ \text{mod } b)) :: \text{gauss-int}$

by (*subst gcd-red-int*) *auto*

also have $\dots = \text{gcd } (\text{of-int } b) \ (\text{of-int } (a \ \text{mod } b))$

using *False* **by** (*intro less [symmetric]*) (*auto intro!: abs-mod-less*)

also have $a \ \text{mod } b = (a - a \ \text{div } b * b)$

by (*simp add: minus-div-mult-eq-mod*)

also have $\text{of-int } \dots = \text{of-int } (-(a \ \text{div } b)) * \text{of-int } b + (\text{of-int } a :: \text{gauss-int})$

by (*simp add: algebra-simps*)

also have $\text{gcd} (\text{of-int } b) \dots = \text{gcd} (\text{of-int } b) (\text{of-int } a)$
by (rule gcd-add-mult)
finally show ?thesis **by** (simp add: gcd.commute)
qed auto
qed

lemma coprime-of-int-gauss-int: coprime (of-int a :: gauss-int) (of-int b) = coprime a b
unfolding coprime-iff-gcd-eq-1 gcd-of-int-gauss-int **by** auto

lemma gcd-of-nat-gauss-int: gcd (of-nat a :: gauss-int) (of-nat b) = of-nat (gcd a b)
using gcd-of-int-gauss-int[of int a int b] **by** simp

lemma coprime-of-nat-gauss-int: coprime (of-nat a :: gauss-int) (of-nat b) = coprime a b
unfolding coprime-iff-gcd-eq-1 gcd-of-nat-gauss-int **by** auto

lemma gauss-cnj-dvd-self-iff: gauss-cnj z dvd z \longleftrightarrow $\text{Re}Z z = 0 \vee \text{Im}Z z = 0 \vee |\text{Re}Z z| = |\text{Im}Z z|$
proof

assume gauss-cnj z dvd z
hence normalize (gauss-cnj z) = normalize z
by (rule gauss-int-dvd-same-norm-imp-associated) auto
then obtain u :: gauss-int **where** is-unit u **and** u: gauss-cnj z = u * z
using associatedE1 **by** blast
hence $u \in \{1, -1, i\mathbf{z}, -i\mathbf{z}\}$
by (simp add: is-unit-gauss-int-iff)
thus $\text{Re}Z z = 0 \vee \text{Im}Z z = 0 \vee |\text{Re}Z z| = |\text{Im}Z z|$
proof (elim insertE emptyE)
assume [simp]: $u = i\mathbf{z}$
have $\text{Re}Z z = \text{Re}Z (\text{gauss-cnj } z)$
by simp
also have $\text{gauss-cnj } z = i\mathbf{z} * z$
using u **by** simp
also have $\text{Re}Z \dots = -\text{Im}Z z$
by simp
finally show $\text{Re}Z z = 0 \vee \text{Im}Z z = 0 \vee |\text{Re}Z z| = |\text{Im}Z z|$
by auto

next
assume [simp]: $u = -i\mathbf{z}$
have $\text{Re}Z z = \text{Re}Z (\text{gauss-cnj } z)$
by simp
also have $\text{gauss-cnj } z = -i\mathbf{z} * z$
using u **by** simp
also have $\text{Re}Z \dots = \text{Im}Z z$
by simp
finally show $\text{Re}Z z = 0 \vee \text{Im}Z z = 0 \vee |\text{Re}Z z| = |\text{Im}Z z|$
by auto

```

next
  assume [simp]: u = 1
  have ImZ z = -ImZ (gauss-cnj z)
    by simp
  also have gauss-cnj z = z
    using u by simp
  finally show ReZ z = 0 ∨ ImZ z = 0 ∨ |ReZ z| = |ImZ z|
    by auto
next
  assume [simp]: u = -1
  have ReZ z = ReZ (gauss-cnj z)
    by simp
  also have gauss-cnj z = -z
    using u by simp
  also have ReZ ... = -ReZ z
    by simp
  finally show ReZ z = 0 ∨ ImZ z = 0 ∨ |ReZ z| = |ImZ z|
    by auto
qed
next
  assume ReZ z = 0 ∨ ImZ z = 0 ∨ |ReZ z| = |ImZ z|
  thus gauss-cnj z dvd z
  proof safe
    assume |ReZ z| = |ImZ z|
    then obtain u :: int where is-unit u and u: ImZ z = u * ReZ z
      using associatedE2[of ReZ z ImZ z] by auto
    from ⟨is-unit u⟩ have u ∈ {1, -1}
      by auto
    hence z = gauss-cnj z * (of-int u * iZ)
      using u by (auto simp: gauss-int-eq-iff)
    thus ?thesis
      by (metis dvd-triv-left)
  qed (auto simp: gauss-cnj-eq-self gauss-cnj-eq-minus-self)
qed

```

lemma *self-dvd-gauss-cnj-iff*: $z \text{ dvd gauss-cnj } z \iff \text{ReZ } z = 0 \vee \text{ImZ } z = 0 \vee |\text{ReZ } z| = |\text{ImZ } z|$
 using *gauss-cnj-dvd-self-iff*[of *z*] by (subst (asm) *gauss-cnj-dvd-left-iff*) auto

1.6 Prime elements

Next, we analyse what the prime elements of the Gaussian integers are. First, note that according to the conventions of Isabelle's computational algebra library, a prime element is called a prime iff it is also normalised, i.e. in our case it lies in the upper right quadrant.

As a first fact, we can show that a Gaussian integer whose norm is \mathbb{Z} -prime must be $\mathbb{Z}[i]$ -prime:

lemma *prime-gauss-int-norm-imp-prime-elem*:

```

assumes prime (gauss-int-norm q)
shows prime-elem q
proof –
  have irreducible q
  proof (rule irreducibleI)
    fix a b assume q = a * b
    hence gauss-int-norm q = gauss-int-norm a * gauss-int-norm b
    by (simp-all add: gauss-int-norm-mult)
    thus is-unit a ∨ is-unit b
    using assms by (auto dest!: prime-product simp: gauss-int-norm-eq-Suc-0-iff)
  qed (use assms in ⟨auto simp: is-unit-gauss-int-iff ⟩)
  thus prime-elem q
  using irreducible-imp-prime-elem-gcd by blast
qed

```

Also, a conjugate is a prime element iff the original element is a prime element:

```

lemma prime-elem-gauss-cnj [intro]: prime-elem z ⇒ prime-elem (gauss-cnj z)
  by (auto simp: prime-elem-def gauss-cnj-dvd-left-iff)

```

```

lemma prime-elem-gauss-cnj-iff [simp]: prime-elem (gauss-cnj z) ⇔ prime-elem z
  using prime-elem-gauss-cnj[of z] prime-elem-gauss-cnj[of gauss-cnj z] by auto

```

1.6.1 The factorisation of 2

2 factors as $-i(1+i)^2$ in the Gaussian integers, where $-i$ is a unit and $1+i$ is prime.

```

lemma gauss-int-2-eq: 2 = -iZ * (1 + iZ) ^ 2
  by (simp add: gauss-int-eq-iff power2-eq-square)

```

```

lemma prime-elem-one-plus-i-gauss-int: prime-elem (1 + iZ)
  by (rule prime-gauss-int-norm-imp-prime-elem) (auto simp: gauss-int-norm-def)

```

```

lemma prime-one-plus-i-gauss-int: prime (1 + iZ)
  by (simp add: prime-def prime-elem-one-plus-i-gauss-int
    gauss-int-eq-iff normalize-gauss-int-def)

```

```

lemma prime-factorization-2-gauss-int:
  prime-factorization (2 :: gauss-int) = {#1 + iZ, 1 + iZ#}
proof –
  have prime-factorization (2 :: gauss-int) =
    (prime-factorization (prod-mset {#1 + gauss-i, 1 + gauss-i#}))
  by (subst prime-factorization-unique) (auto simp: gauss-int-eq-iff normalize-gauss-int-def)
  also have prime-factorization (prod-mset {#1 + gauss-i, 1 + gauss-i#}) =
    {#1 + gauss-i, 1 + gauss-i#}
  using prime-one-plus-i-gauss-int by (subst prime-factorization-prod-mset-primes)
  auto

```

finally show *?thesis* .
qed

1.6.2 Inert primes

Any \mathbb{Z} -prime congruent 3 modulo 4 is also a Gaussian prime. These primes are called *inert*, because they do not decompose when moving from \mathbb{Z} to $\mathbb{Z}[i]$.

lemma *gauss-int-norm-not-3-mod-4*: [gauss-int-norm $z \neq 3$] (mod 4)

proof –

have A : $\text{Re}Z\ z \bmod 4 \in \{0..3\}$ $\text{Im}Z\ z \bmod 4 \in \{0..3\}$ by *auto*

have B : $\{0..3\} = \{0, 1, 2, 3 :: \text{int}\}$ by *auto*

have $[\text{Re}Z\ z^2 + \text{Im}Z\ z^2 = (\text{Re}Z\ z \bmod 4)^2 + (\text{Im}Z\ z \bmod 4)^2]$ (mod 4)

by (*intro cong-add cong-pow*) (*auto simp: cong-def*)

moreover have $((\text{Re}Z\ z \bmod 4)^2 + (\text{Im}Z\ z \bmod 4)^2) \bmod 4 \neq 3 \bmod 4$

using A *unfolding B* by *auto*

ultimately have $[\text{Re}Z\ z^2 + \text{Im}Z\ z^2 \neq 3]$ (mod 4)

unfolding cong-def by *metis*

hence $[\text{int} (\text{nat} (\text{Re}Z\ z^2 + \text{Im}Z\ z^2)) \neq \text{int} 3]$ (mod (int 4))

by *simp*

thus *?thesis* *unfolding gauss-int-norm-def*

by (*subst (asm) cong-int-iff*)

qed

lemma *prime-elem-gauss-int-of-nat*:

fixes $n :: \text{nat}$

assumes *prime*: *prime* n and $[n = 3]$ (mod 4)

shows *prime-elem* (*of-nat* $n :: \text{gauss-int}$)

proof (*intro irreducible-imp-prime-elem irreducibleI*)

from *assms* show *of-nat* $n \neq (0 :: \text{gauss-int})$

by (*auto simp: gauss-int-eq-iff*)

next

show $\neg \text{is-unit}$ (*of-nat* $n :: \text{gauss-int}$)

using *assms* by (*subst is-unit-gauss-int-iff*) (*auto simp: gauss-int-eq-iff*)

next

fix $a\ b :: \text{gauss-int}$

assume $*$: *of-nat* $n = a * b$

hence *gauss-int-norm* ($a * b$) = *gauss-int-norm* (*of-nat* n)

by *metis*

hence $*$: *gauss-int-norm* $a * \text{gauss-int-norm } b = n^2$

by (*simp add: gauss-int-norm-mult power2-eq-square flip: nat-mult-distrib*)

from *prime-power-mult-nat*[*OF prime this*] obtain $i\ j :: \text{nat}$

where ij : *gauss-int-norm* $a = n^i$ *gauss-int-norm* $b = n^j$ by *blast*

have $i + j = 2$

proof –

have $n^{i+j} = n^2$

using $ij * \mathbf{by}$ (*simp add: power-add*)
from *prime-power-inj*[*OF prime this*] **show** *?thesis by simp*
qed
hence $i = 0 \wedge j = 2 \vee i = 1 \wedge j = 1 \vee i = 2 \wedge j = 0$
by *auto*
thus *is-unit a* \vee *is-unit b*
proof (*elim disjE*)
assume $i = 1 \wedge j = 1$
with ij **have** *gauss-int-norm a = n*
by *auto*
hence [*gauss-int-norm a = n*] (*mod 4*)
by *simp*
also have [$n = 3$] (*mod 4*) **by** *fact*
finally have [*gauss-int-norm a = 3*] (*mod 4*) .
moreover have [*gauss-int-norm a \neq 3*] (*mod 4*)
by (*rule gauss-int-norm-not-3-mod-4*)
ultimately show *?thesis by contradiction*
qed (*use ij in <auto simp: is-unit-gauss-int-iff>*)
qed

theorem *prime-gauss-int-of-nat*:
fixes $n :: \mathit{nat}$
assumes *prime: prime n and* [$n = 3$] (*mod 4*)
shows *prime (of-nat n :: gauss-int)*
using *prime-elem-gauss-int-of-nat*[*OF assms*]
unfolding *prime-def by simp*

1.6.3 Non-inert primes

Any \mathbb{Z} -prime congruent 1 modulo 4 factors into two conjugate Gaussian primes.

lemma *minimal-QuadRes-neg1*:
assumes *QuadRes n (-1) n > 1 odd n*
obtains $x :: \mathit{nat}$ **where** $x \leq (n - 1) \mathit{div} 2$ **and** [$x^2 + 1 = 0$] (*mod n*)
proof –
from $\langle \mathit{QuadRes} \ n \ (-1) \rangle$ **obtain** x **where** [$x^2 = (-1)$] (*mod (int n)*)
by (*auto simp: QuadRes-def*)
hence [$x^2 + 1 = -1 + 1$] (*mod (int n)*)
by (*intro cong-add*) *auto*
also have $x^2 + 1 = \mathit{int} \ (\mathit{nat} \ |x| \ ^2 + 1)$
by *simp*
finally have [$\mathit{int} \ (\mathit{nat} \ |x| \ ^2 + 1) = \mathit{int} \ 0$] (*mod (int n)*)
by *simp*
hence [$\mathit{nat} \ |x| \ ^2 + 1 = 0$] (*mod n*)
by (*subst (asm) cong-int-iff*)

define x' **where**
 $x' = (\mathit{if} \ \mathit{nat} \ |x| \ \mathit{mod} \ n \leq (n - 1) \ \mathit{div} \ 2 \ \mathit{then} \ \mathit{nat} \ |x| \ \mathit{mod} \ n \ \mathit{else} \ n - (\mathit{nat} \ |x| \ \mathit{mod} \ n))$

```

have  $x'$ -quadres:  $[x' \wedge 2 + 1 = 0] \pmod n$ 
proof (cases nat  $|x| \pmod n \leq (n - 1) \text{ div } 2$ )
  case True
    hence  $[x' \wedge 2 + 1 = (\text{nat } |x| \pmod n) \wedge 2 + 1] \pmod n$ 
      by (simp add:  $x'$ -def)
    also have  $[(\text{nat } |x| \pmod n) \wedge 2 + 1 = \text{nat } |x| \wedge 2 + 1] \pmod n$ 
      by (intro cong-add cong-pow) (auto simp: cong-def)
    also have  $[\text{nat } |x| \wedge 2 + 1 = 0] \pmod n$  by fact
    finally show ?thesis .
  next
    case False
      hence  $[\text{int } (x' \wedge 2 + 1) = (\text{int } n - \text{int } (\text{nat } |x| \pmod n)) \wedge 2 + 1] \pmod{\text{int } n}$ 
        using  $\langle n > 1 \rangle$  by (simp add:  $x'$ -def of-nat-diff add-ac)
      also have  $[(\text{int } n - \text{int } (\text{nat } |x| \pmod n)) \wedge 2 + 1 =$ 
         $(0 - \text{int } (\text{nat } |x| \pmod n)) \wedge 2 + 1] \pmod{\text{int } n}$ 
        by (intro cong-add cong-pow) (auto simp: cong-def)
      also have  $[(0 - \text{int } (\text{nat } |x| \pmod n)) \wedge 2 + 1 = \text{int } ((\text{nat } |x| \pmod n) \wedge 2 +$ 
         $1)] \pmod{\text{int } n}$ 
        by (simp add: add-ac)
      finally have  $[x' \wedge 2 + 1 = (\text{nat } |x| \pmod n)^2 + 1] \pmod n$ 
        by (subst (asm) cong-int-iff)
      also have  $[(\text{nat } |x| \pmod n)^2 + 1 = \text{nat } |x| \wedge 2 + 1] \pmod n$ 
        by (intro cong-add cong-pow) (auto simp: cong-def)
      also have  $[\text{nat } |x| \wedge 2 + 1 = 0] \pmod n$  by fact
      finally show ?thesis .
    qed
  moreover have  $x'$ -le:  $x' \leq (n - 1) \text{ div } 2$ 
    using  $\langle \text{odd } n \rangle$  by (auto elim!: oddE simp:  $x'$ -def)
  ultimately show ?thesis by (intro that[of  $x'$ ])
qed

```

Let p be some prime number that is congruent 1 modulo 4.

```

locale noninert-gauss-int-prime =
  fixes  $p :: \text{nat}$ 
  assumes prime-p: prime  $p$  and cong-1-p:  $[p = 1] \pmod 4$ 
begin

```

```

lemma p-gt-2:  $p > 2$  and odd-p: odd  $p$ 
proof -
  from prime-p and cong-1-p have  $p > 1$   $p \neq 2$ 
    by (auto simp: prime-gt-Suc-0-nat cong-def)
  thus  $p > 2$  by auto
  with prime-p show odd  $p$ 
    using primes-dvd-imp-eq two-is-prime-nat by blast
qed

```

-1 is a quadratic residue modulo p , so there exists some x such that $x^2 + 1$ is divisible by p . Moreover, we can choose x such that it is positive and no greater than $\frac{1}{2}(p - 1)$:

lemma *minimal-QuadRes-neg1*:
obtains x **where** $x > 0$ $x \leq (p - 1) \operatorname{div} 2$ $[x^2 + 1 = 0] \pmod{p}$
proof –
have $[\operatorname{Legendre} (-1) (\operatorname{int} p) = (-1)^{\operatorname{int} ((p - 1) \operatorname{div} 2)}] \pmod{(\operatorname{int} p)}$
using *prime-p p-gt-2* **by** (*intro euler-criterion*) *auto*
also have $[p - 1 = 1 - 1] \pmod{4}$
using *p-gt-2* **by** (*intro cong-diff-nat cong-refl*) (*use cong-1-p in auto*)
hence $2 * 2 \operatorname{dvd} p - 1$
by (*simp add: cong-0-iff*)
hence $\operatorname{even} ((p - 1) \operatorname{div} 2)$
using *dvd-mult-imp-div* **by** *blast*
hence $(-1)^{\operatorname{int} ((p - 1) \operatorname{div} 2)} = (1 :: \operatorname{int})$
by *simp*
finally have $\operatorname{Legendre} (-1) (\operatorname{int} p) \pmod{p} = 1$
using *p-gt-2* **by** (*auto simp: cong-def*)
hence $\operatorname{Legendre} (-1) (\operatorname{int} p) = 1$
using *p-gt-2* **by** (*auto simp: Legendre-def cong-def zmod-minus1 split: if-splits*)
hence $\operatorname{QuadRes} p (-1)$
by (*simp add: Legendre-def split: if-splits*)
from *minimal-QuadRes-neg1* [*OF this*] *p-gt-2 odd-p*
obtain x **where** $x: x \leq (p - 1) \operatorname{div} 2$ $[x^2 + 1 = 0] \pmod{p}$ **by** *auto*
have $x > 0$
using *x p-gt-2* **by** (*auto intro!: Nat.gr0I simp: cong-def*)
from x **and** *this* **show** *?thesis* **by** (*intro that[of x] auto*)
qed

We can show from this that p is not prime as a Gaussian integer.

lemma *not-prime*: $\neg \operatorname{prime}\text{-elem} (\operatorname{of}\text{-nat} p :: \operatorname{gauss}\text{-int})$
proof
assume *prime*: $\operatorname{prime}\text{-elem} (\operatorname{of}\text{-nat} p :: \operatorname{gauss}\text{-int})$
obtain x **where** $x: x > 0$ $x \leq (p - 1) \operatorname{div} 2$ $[x^2 + 1 = 0] \pmod{p}$
using *minimal-QuadRes-neg1* .

have $\operatorname{of}\text{-nat} p \operatorname{dvd} (\operatorname{of}\text{-nat} (x^2 + 1) :: \operatorname{gauss}\text{-int})$
using x **by** (*intro of-nat-dvd-of-nat*) (*auto simp: cong-0-iff*)
also have $\operatorname{eq}: \operatorname{of}\text{-nat} (x^2 + 1) = ((\operatorname{of}\text{-nat} x + \mathbf{i}_{\mathbb{Z}}) * (\operatorname{of}\text{-nat} x - \mathbf{i}_{\mathbb{Z}}) :: \operatorname{gauss}\text{-int})$
using $\langle x > 0 \rangle$ **by** (*simp add: algebra-simps gauss-int-eq-iff power2-eq-square of-nat-diff*)
finally have $\operatorname{of}\text{-nat} p \operatorname{dvd} ((\operatorname{of}\text{-nat} x + \mathbf{i}_{\mathbb{Z}}) * (\operatorname{of}\text{-nat} x - \mathbf{i}_{\mathbb{Z}}) :: \operatorname{gauss}\text{-int})$.

from *prime* **and** *this*
have $\operatorname{of}\text{-nat} p \operatorname{dvd} (\operatorname{of}\text{-nat} x + \mathbf{i}_{\mathbb{Z}} :: \operatorname{gauss}\text{-int}) \vee \operatorname{of}\text{-nat} p \operatorname{dvd} (\operatorname{of}\text{-nat} x - \mathbf{i}_{\mathbb{Z}} :: \operatorname{gauss}\text{-int})$
by (*rule prime-elim-dvd-multD*)
hence $\operatorname{dvd}: \operatorname{of}\text{-nat} p \operatorname{dvd} (\operatorname{of}\text{-nat} x + \mathbf{i}_{\mathbb{Z}} :: \operatorname{gauss}\text{-int}) \operatorname{of}\text{-nat} p \operatorname{dvd} (\operatorname{of}\text{-nat} x - \mathbf{i}_{\mathbb{Z}} :: \operatorname{gauss}\text{-int})$
by (*auto dest: of-nat-dvd-imp-dvd-gauss-cnj*)

have $\operatorname{of}\text{-nat} (p^2) = (\operatorname{of}\text{-nat} p * \operatorname{of}\text{-nat} p :: \operatorname{gauss}\text{-int})$

```

    by (simp add: power2-eq-square)
  also from dvd have ... dvd ((of-nat x + iZ) * (of-nat x - iZ}))
    by (intro mult-dvd-mono)
  also have ... = of-nat (x ^ 2 + 1)
    by (rule eq [symmetric])
  finally have p ^ 2 dvd (x ^ 2 + 1)
    by (subst (asm) of-nat-dvd-of-nat-gauss-int-iff)
  hence p ^ 2 ≤ x ^ 2 + 1
    by (intro dvd-imp-le) auto
  moreover have p ^ 2 > x ^ 2 + 1
  proof -
    have x ^ 2 + 1 ≤ ((p - 1) div 2) ^ 2 + 1
      using x by (intro add-mono power-mono) auto
    also have ... ≤ (p - 1) ^ 2 + 1
      by auto
    also have (p - 1) * (p - 1) < (p - 1) * (p + 1)
      using p-gt-2 by (intro mult-strict-left-mono) auto
    hence (p - 1) ^ 2 + 1 < p ^ 2
      by (simp add: algebra-simps power2-eq-square)
    finally show ?thesis .
  qed
  ultimately show False by linarith
qed

```

Any prime factor of p in the Gaussian integers must have norm p .

lemma *norm-prime-divisor*:

```

  fixes q :: gauss-int
  assumes q: prime-elem q q dvd of-nat p
  shows gauss-int-norm q = p
  proof -
    from assms obtain r where r: of-nat p = q * r
      by auto
    have p ^ 2 = gauss-int-norm (of-nat p)
      by simp
    also have ... = gauss-int-norm q * gauss-int-norm r
      by (auto simp: r gauss-int-norm-mult)
    finally have *: gauss-int-norm q * gauss-int-norm r = p ^ 2
      by simp
    hence ∃ i j. gauss-int-norm q = p ^ i ∧ gauss-int-norm r = p ^ j
      using prime-p by (intro prime-power-mult-nat)
    then obtain i j where ij: gauss-int-norm q = p ^ i gauss-int-norm r = p ^ j
      by blast
    have ij-eq-2: i + j = 2
    proof -
      from * have p ^ (i + j) = p ^ 2
        by (simp add: power-add ij)
      thus ?thesis
        using p-gt-2 by (subst (asm) power-inject-exp) auto
    qed
  qed

```

hence $i = 0 \wedge j = 2 \vee i = 1 \wedge j = 1 \vee i = 2 \wedge j = 0$ **by auto**
 hence $i = 1$
proof (*elim disjE*)
 assume $i = 2 \wedge j = 0$
 hence *is-unit r*
 using *ij* **by** (*simp add: gauss-int-norm-eq-Suc-0-iff*)
 hence *prime-elem (of-nat p :: gauss-int)* **using** \langle *prime-elem q* \rangle
 by (*simp add: prime-elem-mult-unit-left r mult.commute[of - r]*)
 with not-prime **show** $i = 1$ **by contradiction**
qed (*use q ij in <auto simp: gauss-int-norm-eq-Suc-0-iff>*)
thus *?thesis* **using** *ij* **by simp**
qed

We now show two lemmas that characterise the two prime factors of p in the Gaussian integers: they are two conjugates $x \pm iy$ for positive integers x and y such that $x^2 + y^2 = p$.

lemma *prime-divisor-exists*:

obtains q **where** *prime q prime-elem (gauss-cnj q) ReZ q > 0 ImZ q > 0*
 *of-nat p = q * gauss-cnj q gauss-int-norm q = p*

proof –

have $\exists q :: \text{gauss-int. } q \text{ dvd of-nat } p \wedge \text{prime } q$
 by (*rule prime-divisor-exists*) (*use prime-p in <auto simp: is-unit-gauss-int-iff'>*)
then obtain $q :: \text{gauss-int}$ **where** $q : \text{prime } q \wedge q \text{ dvd of-nat } p$
 by blast
from \langle *prime q* \rangle **have** [*simp*]: $q \neq 0$ **by auto**
have *normalize q = q*
 using q **by simp**
hence *q-signs: ReZ q > 0 ImZ q ≥ 0*
 by (*subst (asm) normalized-gauss-int-iff; simp*) $+$

from q **have** *gauss-int-norm q = p*
 using *norm-prime-divisor[of q]* **by simp**
moreover from this **have** *gauss-int-norm (gauss-cnj q) = p*
 by simp
hence *prime-elem (gauss-cnj q)*
 using *prime-p* **by** (*intro prime-gauss-int-norm-imp-prime-elem*) *auto*
moreover have *of-nat p = q * gauss-cnj q*
 using \langle *gauss-int-norm q = p* \rangle **by** (*simp add: self-mult-gauss-cnj*)
moreover have *ImZ q ≠ 0*

proof

assume [*simp*]: $\text{ImZ } q = 0$
define m **where** $m = \text{nat } (\text{ReZ } q)$
have [*simp*]: $q = \text{of-nat } m$
 using *q-signs* **by** (*auto simp: gauss-int-eq-iff m-def*)
with q **have** $m \text{ dvd } p$
 by (*simp add: of-nat-dvd-of-nat-gauss-int-iff*)
with *prime-p* **have** $m = 1 \vee m = p$
 using *prime-nat-iff* **by blast**
with q **show** *False* **using** *not-prime* **by auto**

qed
with q -signs have $\text{Im}Z\ q > 0$ by simp
ultimately show ?thesis using q q -signs by (intro that[*of* q])
qed

theorem prime-factorization:

obtains $q1\ q2$
where prime $q1$ prime $q2$ prime-factorization (of-nat p) = {# $q1$, $q2$ #}
gauss-int-norm $q1$ = p gauss-int-norm $q2$ = p $q2$ = $i_{\mathbb{Z}} * \text{gauss-cnj}\ q1$
 $\text{Re}Z\ q1 > 0$ $\text{Im}Z\ q1 > 0$ $\text{Re}Z\ q2 > 0$ $\text{Im}Z\ q2 > 0$
proof –
obtain q where q : prime q prime-elem (gauss-cnj q) $\text{Re}Z\ q > 0$ $\text{Im}Z\ q > 0$
of-nat $p = q * \text{gauss-cnj}\ q$ gauss-int-norm $q = p$
using prime-divisor-exists by metis
from \langle prime q \rangle have [simp]: $q \neq 0$ by auto
define q' where $q' = \text{normalize}\ (\text{gauss-cnj}\ q)$
have prime-factorization (of-nat p) = prime-factorization (prod-mset {# q , q' #})
by (subst prime-factorization-unique) (auto simp: $q\ q'$ -def)
also have ... = {# q , q' #}
using q by (subst prime-factorization-prod-mset-primes) (auto simp: q' -def)
finally have prime-factorization (of-nat p) = {# q , q' #} .
moreover have $q' = i_{\mathbb{Z}} * \text{gauss-cnj}\ q$
using q by (auto simp: normalize-gauss-int-def q' -def)
moreover have prime q'
using q by (auto simp: q' -def)
ultimately show ?thesis using q
by (intro that[*of* $q\ q'$]) (auto simp: q' -def gauss-int-norm-mult)
qed

end

In particular, a consequence of this is that any prime congruent 1 modulo 4 can be written as a sum of squares of positive integers.

lemma prime-cong-1-mod-4-gauss-int-norm-exists:

fixes $p :: \text{nat}$
assumes prime p [$p = 1$] (mod 4)
shows $\exists z. \text{gauss-int-norm}\ z = p \wedge \text{Re}Z\ z > 0 \wedge \text{Im}Z\ z > 0$
proof –
from *assms* **interpret** noninert-gauss-int-prime p
by unfold-locales
from prime-divisor-exists **obtain** q
where q : prime q of-nat $p = q * \text{gauss-cnj}\ q$
 $\text{Re}Z\ q > 0$ $\text{Im}Z\ q > 0$ gauss-int-norm $q = p$ by metis
have $p = \text{gauss-int-norm}\ q$
using q by simp
thus ?thesis using q by blast
qed

1.6.4 Full classification of Gaussian primes

Any prime in the ring of Gaussian integers is of the form

- $1 + i\mathbb{Z}$
- p where $p \in \mathbb{N}$ is prime in \mathbb{N} and congruent 1 modulo 4
- $x + iy$ where x, y are positive integers and $x^2 + y^2$ is a prime congruent 3 modulo 4

or an associated element of one of these.

theorem *gauss-int-prime-classification*:

fixes $x :: \text{gauss-int}$

assumes *prime x*

obtains

(*one-plus-i*) $x = 1 + i\mathbb{Z}$

| (*cong-3-mod-4*) p **where** $x = \text{of-nat } p$ *prime p* [$p = 3$] (*mod 4*)

| (*cong-1-mod-4*) *prime (gauss-int-norm x)* [*gauss-int-norm x = 1*] (*mod 4*)

$\text{ReZ } x > 0$ $\text{ImZ } x > 0$ $\text{ReZ } x \neq \text{ImZ } x$

proof –

define N **where** $N = \text{gauss-int-norm } x$

have $x \text{ dvd } x * \text{gauss-cnj } x$

by *simp*

also have $\dots = \text{of-nat } (\text{gauss-int-norm } x)$

by (*simp add: self-mult-gauss-cnj*)

finally have $x \in \text{prime-factors } (\text{of-nat } N)$

using *assms* **by** (*auto simp: in-prime-factors-iff N-def*)

also have $N = \text{prod-mset } (\text{prime-factorization } N)$

using *assms* **unfolding** $N\text{-def}$ **by** (*subst prod-mset-prime-factorization-nat*)

auto

also have ($\text{of-nat } \dots :: \text{gauss-int}$) =

$\text{prod-mset } (\text{image-mset } \text{of-nat } (\text{prime-factorization } N))$

by (*subst of-nat-prod-mset*) *auto*

also have $\text{prime-factors } \dots = (\bigcup_{p \in \text{prime-factors } N} \text{prime-factors } (\text{of-nat } p))$

by (*subst prime-factorization-prod-mset*) *auto*

finally obtain p **where** $p: p \in \text{prime-factors } N$ $x \in \text{prime-factors } (\text{of-nat } p)$

by *auto*

have *prime p*

using p **by** *auto*

hence $\neg(2 * 2) \text{ dvd } p$

using *product-dvd-irreducibleD*[$\text{of } p$ 2 2]

by (*auto simp flip: prime-elem-iff-irreducible*)

hence [$p \neq 0$] (*mod 4*)

using p **by** (*auto simp: cong-0-iff in-prime-factors-iff*)

hence $p \text{ mod } 4 \in \{1, 2, 3\}$ **by** (*auto simp: cong-def*)

thus *?thesis*

proof (*elim singletonE insertE*)

```

assume  $p \bmod 4 = 2$ 
hence  $p \bmod 4 \bmod 2 = 0$ 
  by simp
hence  $p \bmod 2 = 0$ 
  by (simp add: mod-mod-cancel)
with  $\langle \text{prime } p \rangle$  have [simp]:  $p = 2$ 
  using prime-prime-factor two-is-prime-nat by blast
have prime-factors (of-nat  $p$ ) =  $\{1 + i_{\mathbb{Z}} :: \text{gauss-int}\}$ 
  by (simp add: prime-factorization-2-gauss-int)
with  $p$  show ?thesis using that(1) by auto
next
assume  $\ast: p \bmod 4 = 3$ 
hence prime-factors (of-nat  $p$ ) =  $\{\text{of-nat } p :: \text{gauss-int}\}$ 
  using prime-gauss-int-of-nat[of p]  $\langle \text{prime } p \rangle$ 
  by (subst prime-factorization-prime) (auto simp: cong-def)
with  $p$  show ?thesis using that(2)[of p]  $\ast$ 
  by (auto simp: cong-def)
next
assume  $\ast: p \bmod 4 = 1$ 
then interpret noninert-gauss-int-prime  $p$ 
  by unfold-locales (use  $\langle \text{prime } p \rangle$  in auto simp: cong-def)
obtain  $q1\ q2 :: \text{gauss-int}$  where  $q12$ :
  prime  $q1$  prime  $q2$  prime-factorization (of-nat  $p$ ) =  $\{\#q1, q2\#$ 
  gauss-int-norm  $q1 = p$  gauss-int-norm  $q2 = p$   $q2 = i_{\mathbb{Z}} \ast \text{gauss-cnj } q1$ 
  ReZ  $q1 > 0$  ImZ  $q1 > 0$  ReZ  $q1 > 0$  ImZ  $q2 > 0$ 
  using prime-factorization by metis
from  $p\ q12$  have  $x = q1 \vee x = q2$  by auto
with  $q12$  have  $\ast\ast: \text{gauss-int-norm } x = p$  ReZ  $x > 0$  ImZ  $x > 0$ 
  by auto
have ReZ  $x \neq \text{ImZ } x$ 
proof
  assume ReZ  $x = \text{ImZ } x$ 
  hence even (gauss-int-norm  $x$ )
    by (auto simp: gauss-int-norm-def nat-mult-distrib)
  hence even  $p$  using  $\langle \text{gauss-int-norm } x = p \rangle$ 
    by simp
  with  $\langle p \bmod 4 = 1 \rangle$  show False
    by presburger
qed
thus ?thesis using that(3)  $\langle \text{prime } p \rangle \ast \ast$ 
  by (simp add: cong-def)
qed
qed

lemma prime-gauss-int-norm-squareD:
fixes  $z :: \text{gauss-int}$ 
assumes prime  $z$  gauss-int-norm  $z = p \wedge 2$ 
shows prime  $p \wedge z = \text{of-nat } p$ 
using assms(1)

```

```

proof (cases rule: gauss-int-prime-classification)
  case one-plus-i
  have prime (2 :: nat) by simp
  also from one-plus-i have 2 = p ^ 2
    using assms(2) by (auto simp: gauss-int-norm-def)
  finally show ?thesis by (simp add: prime-power-iff)
next
  case (cong-3-mod-4 p)
  thus ?thesis using assms by auto
next
  case cong-1-mod-4
  with assms show ?thesis
    by (auto simp: prime-power-iff)
qed

lemma gauss-int-norm-eq-prime-squareD:
  assumes prime p and [p = 3] (mod 4) and gauss-int-norm z = p ^ 2
  shows normalize z = of-nat p and prime-elem z
proof -
  have  $\exists q::\text{gauss-int. } q \text{ dvd } z \wedge \text{prime } q$ 
    by (rule prime-divisor-exists) (use assms in  $\langle \text{auto simp: is-unit-gauss-int-iff} \rangle$ )
  then obtain q :: gauss-int where q: q dvd z prime q by blast
  have gauss-int-norm q dvd gauss-int-norm z
    by (rule gauss-int-norm-dvd-mono) fact
  also have ... = p ^ 2 by fact
  finally obtain i where i:  $i \leq 2$  gauss-int-norm q = p ^ i
    by (subst (asm) divides-primelpow-nat) (use assms q in auto)
  from i assms q have  $i \neq 0$ 
    by (auto intro!: Nat.gr0I simp: gauss-int-norm-eq-Suc-0-iff)
  moreover from i assms q have  $i \neq 1$ 
    using gauss-int-norm-not-3-mod-4[of q] by auto
  ultimately have  $i = 2$  using i by auto
  with i have gauss-int-norm q = p ^ 2 by auto
  hence [simp]: q = of-nat p
    using prime-gauss-int-norm-squareD[of q p] q by auto
  have normalize (of-nat p) = normalize z
    using q assms
    by (intro gauss-int-dvd-same-norm-imp-associated) auto
  thus *: normalize z = of-nat p by simp

  have prime (normalize z)
    using prime-gauss-int-of-nat[of p] assms by (subst *) auto
  thus prime-elem z by simp
qed

```

The following can be used as a primality test for Gaussian integers. It effectively reduces checking the primality of a Gaussian integer to checking the primality of an integer.

A Gaussian integer is prime if either its norm is either \mathbb{Z} -prime or the square

of a \mathbb{Z} -prime that is congruent 3 modulo 4.

lemma *prime-elem-gauss-int-iff*:

fixes $z :: \text{gauss-int}$

defines $n \equiv \text{gauss-int-norm } z$

shows $\text{prime-elem } z \longleftrightarrow \text{prime } n \vee (\exists p. n = p^2 \wedge \text{prime } p \wedge [p = 3] \pmod{4})$

proof

assume $\text{prime } n \vee (\exists p. n = p^2 \wedge \text{prime } p \wedge [p = 3] \pmod{4})$

thus $\text{prime-elem } z$

by (*auto intro: gauss-int-norm-eq-prime-squareD(2)*
prime-gauss-int-norm-imp-prime-elem simp: n-def)

next

assume $\text{prime-elem } z$

hence $\text{prime } (\text{normalize } z)$ **by** *simp*

thus $\text{prime } n \vee (\exists p. n = p^2 \wedge \text{prime } p \wedge [p = 3] \pmod{4})$

proof (*cases rule: gauss-int-prime-classification*)

case *one-plus-i*

have $n = \text{gauss-int-norm } (\text{normalize } z)$

by (*simp add: n-def*)

also have $\text{normalize } z = 1 + iz$

by *fact*

also have $\text{gauss-int-norm } \dots = 2$

by (*simp add: gauss-int-norm-def*)

finally show *?thesis* **by** *simp*

next

case (*cong-3-mod-4 p*)

have $n = \text{gauss-int-norm } (\text{normalize } z)$

by (*simp add: n-def*)

also have $\text{normalize } z = \text{of-nat } p$

by *fact*

also have $\text{gauss-int-norm } \dots = p^2$

by *simp*

finally show *?thesis* **using** *cong-3-mod-4* **by** *simp*

next

case *cong-1-mod-4*

thus *?thesis* **by** (*simp add: n-def*)

qed

qed

1.6.5 Multiplicities of primes

In this section, we will show some results connecting the multiplicity of a Gaussian prime p in a Gaussian integer z to the \mathbb{Z} -multiplicity of the norm of p in the norm of z .

The multiplicity of the Gaussian prime $1 + iz$ in an integer c is simply twice the \mathbb{Z} -multiplicity of 2 in c :

lemma *multiplicity-prime-1-plus-i-aux*: $\text{multiplicity } (1 + iz) (\text{of-nat } c) = 2 * \text{mul-}$

```

multiplicity 2 c
proof (cases c = 0)
  case [simp]: False
  have 2 * multiplicity 2 c = multiplicity 2 (c ^ 2)
    by (simp add: prime-elem-multiplicity-power-distrib)
  also have multiplicity 2 (c ^ 2) = multiplicity (of-nat 2) (of-nat c ^ 2 ::
gauss-int)
    by (simp flip: multiplicity-gauss-int-of-nat)
  also have of-nat 2 = (-i $\mathbb{Z}$ ) * (1 + i $\mathbb{Z}$ ) ^ 2
    by (simp add: algebra-simps power2-eq-square)
  also have multiplicity ... (of-nat c ^ 2) = multiplicity ((1 + i $\mathbb{Z}$ ) ^ 2) (of-nat c
^ 2)
    by (subst multiplicity-times-unit-left) auto
  also have ... = multiplicity (1 + i $\mathbb{Z}$ ) (of-nat c)
    by (subst multiplicity-power-power) auto
  finally show ?thesis ..
qed auto

```

The multiplicity of an inert Gaussian prime $q \in \mathbb{Z}$ in a Gaussian integer z is precisely half the \mathbb{Z} -multiplicity of q in the norm of z .

```

lemma multiplicity-prime-cong-3-mod-4:
  assumes prime (of-nat q :: gauss-int)
  shows multiplicity q (gauss-int-norm z) = 2 * multiplicity (of-nat q) z
proof (cases z = 0)
  case [simp]: False
  have multiplicity q (gauss-int-norm z) =
    multiplicity (of-nat q) (of-nat (gauss-int-norm z) :: gauss-int)
    by (simp add: multiplicity-gauss-int-of-nat)
  also have ... = multiplicity (of-nat q) (z * gauss-cnj z)
    by (simp add: self-mult-gauss-cnj)
  also have ... = multiplicity (of-nat q) z + multiplicity (gauss-cnj (of-nat q))
(gauss-cnj z)
    using assms by (subst prime-elem-multiplicity-mult-distrib) auto
  also have multiplicity (gauss-cnj (of-nat q)) (gauss-cnj z) = multiplicity (of-nat
q) z
    by (subst multiplicity-gauss-cnj) auto
  also have ... + ... = 2 * ...
    by simp
  finally show ?thesis .
qed auto

```

For Gaussian primes p whose norm is congruent 1 modulo 4, the $\mathbb{Z}[i]$ -multiplicity of p in an integer c is just the \mathbb{Z} -multiplicity of their norm in c .

```

lemma multiplicity-prime-cong-1-mod-4-aux:
  fixes p :: gauss-int
  assumes prime-elem p ReZ p > 0 ImZ p > 0 ImZ p ≠ ReZ p
  shows multiplicity p (of-nat c) = multiplicity (gauss-int-norm p) c
proof (cases c = 0)

```

```

case [simp]: False
show ?thesis
proof (intro antisym multiplicity-geI)
  define k where k = multiplicity p (of-nat c)
  have p ^ k dvd of-nat c
    by (simp add: multiplicity-dvd k-def)
  moreover have gauss-cn timer p ^ k dvd of-nat c
    using multiplicity-dvd[of gauss-cn timer p of-nat c]
      multiplicity-gauss-cn timer[of p of-nat c] by (simp add: k-def)
  moreover have ¬p dvd gauss-cn timer p
    using assms by (subst self-dvd-gauss-cn timer-iff) auto
  hence ¬p dvd gauss-cn timer p ^ k
    using assms prime-elem-dvd-power by blast
  ultimately have p ^ k * gauss-cn timer p ^ k dvd of-nat c
    using assms by (intro prime-elem-power-mult-dvdI) auto
  also have p ^ k * gauss-cn timer p ^ k = of-nat (gauss-int-norm p ^ k)
    by (simp flip: self-mult-gauss-cn timer add: power-mult-distrib)
  finally show gauss-int-norm p ^ k dvd c
    by (subst (asm) of-nat-dvd-of-nat-gauss-int-iff)
next
define k where k = multiplicity (gauss-int-norm p) c
have p ^ k dvd (p * gauss-cn timer p) ^ k
  by (intro dvd-power-same) auto
also have ... = of-nat (gauss-int-norm p ^ k)
  by (simp add: self-mult-gauss-cn timer)
also have ... dvd of-nat c
  unfolding of-nat-dvd-of-nat-gauss-int-iff by (auto simp: k-def multiplicity-dvd)
finally show p ^ k dvd of-nat c .
qed (use assms in ⟨auto simp: gauss-int-norm-eq-Suc-0-iff⟩)
qed auto

```

The multiplicity of a Gaussian prime with norm congruent 1 modulo 4 in some Gaussian integer z and the multiplicity of its conjugate in z sum to the the \mathbb{Z} -multiplicity of their norm in the norm of z :

```

lemma multiplicity-prime-cong-1-mod-4:
  fixes p :: gauss-int
  assumes prime-elem p ReZ p > 0 ImZ p > 0 ImZ p ≠ ReZ p
  shows multiplicity (gauss-int-norm p) (gauss-int-norm z) =
    multiplicity p z + multiplicity (gauss-cn timer p) z
proof (cases z = 0)
case [simp]: False
have multiplicity (gauss-int-norm p) (gauss-int-norm z) =
  multiplicity p (of-nat (gauss-int-norm z))
  using assms by (subst multiplicity-prime-cong-1-mod-4-aux) auto
also have ... = multiplicity p (z * gauss-cn timer z)
  by (simp add: self-mult-gauss-cn timer)
also have ... = multiplicity p z + multiplicity p (gauss-cn timer z)
  using assms by (subst prime-elem-multiplicity-mult-distrib) auto
also have multiplicity p (gauss-cn timer z) = multiplicity (gauss-cn timer p) z

```

by (subst multiplicity-gauss-cnj [symmetric]) auto
 finally show ?thesis .
 qed auto

The multiplicity of the Gaussian prime $1 + i\mathbf{z}$ in a Gaussian integer z is precisely the \mathbf{Z} -multiplicity of 2 in the norm of z :

lemma multiplicity-prime-1-plus-i: multiplicity $(1 + i\mathbf{z}) z = \text{multiplicity } 2 (\text{gauss-int-norm } z)$

proof (cases $z = 0$)

case [simp]: False

note [simp] = prime-elem-one-plus-i-gauss-int

have $2 * \text{multiplicity } 2 (\text{gauss-int-norm } z) = \text{multiplicity } (1 + i\mathbf{z}) (\text{of-nat } (\text{gauss-int-norm } z))$

by (rule multiplicity-prime-1-plus-i-aux [symmetric])

also have $\dots = \text{multiplicity } (1 + i\mathbf{z}) (z * \text{gauss-cnj } z)$

by (simp add: self-mult-gauss-cnj)

also have $\dots = \text{multiplicity } (1 + i\mathbf{z}) z + \text{multiplicity } (\text{gauss-cnj } (1 - i\mathbf{z})) (\text{gauss-cnj } z)$

by (subst prime-elem-multiplicity-mult-distrib) auto

also have $\text{multiplicity } (\text{gauss-cnj } (1 - i\mathbf{z})) (\text{gauss-cnj } z) = \text{multiplicity } (1 - i\mathbf{z}) z$

z

by (subst multiplicity-gauss-cnj) auto

also have $1 - i\mathbf{z} = (-i\mathbf{z}) * (1 + i\mathbf{z})$

by (simp add: algebra-simps)

also have $\text{multiplicity } \dots z = \text{multiplicity } (1 + i\mathbf{z}) z$

by (subst multiplicity-times-unit-left) auto

also have $\dots + \dots = 2 * \dots$

by simp

finally show ?thesis by simp

qed auto

1.7 Coprimality of an element and its conjugate

Using the classification of the primes, we now show that if the real and imaginary parts of a Gaussian integer are coprime and its norm is odd, then it is coprime to its own conjugate.

lemma coprime-self-gauss-cnj:

assumes coprime $(\text{Re } z) (\text{Im } z)$ and odd $(\text{gauss-int-norm } z)$

shows coprime $z (\text{gauss-cnj } z)$

proof (rule coprimeI)

fix d assume $d \text{ dvd } z \text{ } d \text{ dvd } \text{gauss-cnj } z$

have *: False if $p \in \text{prime-factors } z \text{ } p \in \text{prime-factors } (\text{gauss-cnj } z)$ for p

proof –

from that have p : prime $p \text{ } p \text{ dvd } z \text{ } p \text{ dvd } \text{gauss-cnj } z$

by auto

define p' where $p' = \text{gauss-cnj } p$

define d where $d = \text{gauss-int-norm } p$

```

have of-nat-d-eq: of-nat d = p * p'
  by (simp add: p'-def self-mult-gauss-cn timer d-def)
have prime-elem p prime-elem p' p dvd z p' dvd z p dvd gauss-cn timer z p' dvd
gauss-cn timer z
  using that by (auto simp: in-prime-factors-iff p'-def gauss-cn timer dvd-left-iff)

have prime p
  using that by auto
then obtain q where q: prime q of-nat q dvd z
proof (cases rule: gauss-int-prime-classification)
  case one-plus-i
  hence 2 = gauss-int-norm p
    by (auto simp: gauss-int-norm-def)
  also have gauss-int-norm p dvd gauss-int-norm z
    using p by (intro gauss-int-norm-dvd-mono) auto
  finally have even (gauss-int-norm z) .
  with ⟨odd (gauss-int-norm z)⟩ show ?thesis
    by contradiction
next
  case (cong-3-mod-4 q)
  thus ?thesis using that[of q] p by simp
next
  case cong-1-mod-4
  hence ¬p dvd p'
    unfolding p'-def by (subst self-dvd-gauss-cn timer-iff) auto
  hence p * p' dvd z using p
    by (intro prime-elem-mult-dvdI) (auto simp: p'-def gauss-cn timer dvd-left-iff)
  also have p * p' = of-nat (gauss-int-norm p)
    by (simp add: p'-def self-mult-gauss-cn timer)
  finally show ?thesis using that[of gauss-int-norm p] cong-1-mod-4
    by simp
qed

have of-nat q dvd gcd (2 * of-int (ReZ z)) (2 * iZ * of-int (ImZ z))
proof (rule gcd-greatest)
  have of-nat q dvd (z + gauss-cn timer z)
    using q by (auto simp: gauss-cn timer dvd-right-iff)
  also have ... = 2 * of-int (ReZ z)
    by (simp add: self-plus-gauss-cn timer)
  finally show of-nat q dvd (2 * of-int (ReZ z) :: gauss-int) .
next
  have of-nat q dvd (z - gauss-cn timer z)
    using q by (auto simp: gauss-cn timer dvd-right-iff)
  also have ... = 2 * iZ * of-int (ImZ z)
    by (simp add: self-minus-gauss-cn timer)
  finally show of-nat q dvd (2 * iZ * of-int (ImZ z)) .
qed
also have ... = 2
proof -

```

```

have odd (ReZ z) ∨ odd (ImZ z)
  using assms by (auto simp: gauss-int-norm-def even-nat-iff)
thus ?thesis
proof
  assume odd (ReZ z)
  hence coprime (of-int (ReZ z)) (of-int 2 :: gauss-int)
    unfolding coprime-of-int-gauss-int coprime-right-2-iff-odd .
  thus ?thesis
    using assms
    by (subst gcd-mult-left-right-cancel)
      (auto simp: coprime-of-int-gauss-int coprime-commute is-unit-left-imp-coprime
        is-unit-right-imp-coprime gcd-proj1-if-dvd gcd-proj2-if-dvd)
next
  assume odd (ImZ z)
  hence coprime (of-int (ImZ z)) (of-int 2 :: gauss-int)
    unfolding coprime-of-int-gauss-int coprime-right-2-iff-odd .
  hence gcd (2 * of-int (ReZ z)) (2 * iZ * of-int (ImZ z)) = gcd (2 * of-int
(ReZ z)) (2 * iZ)
    using assms
    by (subst gcd-mult-right-right-cancel)
      (auto simp: coprime-of-int-gauss-int coprime-commute is-unit-left-imp-coprime
        is-unit-right-imp-coprime)
  also have ... = normalize (2 * gcd (of-int (ReZ z)) iZ)
    by (subst gcd-mult-left) auto
  also have gcd (of-int (ReZ z)) iZ = 1
    by (subst coprime-iff-gcd-eq-1 [symmetric], rule is-unit-right-imp-coprime)
auto
  finally show ?thesis by simp
qed
qed
finally have of-nat q dvd (of-nat 2 :: gauss-int)
  by simp
hence q dvd 2
  by (simp only: of-nat-dvd-of-nat-gauss-int-iff)
with ⟨prime q⟩ have q = 2
  using primes-dvd-imp-eq two-is-prime-nat by blast
with q have 2 dvd z
  by auto

have 2 dvd gauss-int-norm 2
  by simp
also have ... dvd gauss-int-norm z
  using ⟨2 dvd z⟩ by (intro gauss-int-norm-dvd-mono)
finally show False using ⟨odd (gauss-int-norm z)⟩ by contradiction
qed

fix d :: gauss-int
assume d: d dvd z d dvd gauss-cnj z
show is-unit d

```

```

proof (rule ccontr)
  assume  $\neg$ is-unit  $d$ 
  moreover from  $d$  assms have  $d \neq 0$ 
  by auto
  ultimately obtain  $p$  where  $p$ : prime  $p \mid d$ 
  using prime-divisorE by blast
  with  $d$  have  $p \in \text{prime-factors } z \mid p \in \text{prime-factors } (\text{gauss-cnj } z)$ 
  using assms by (auto simp: in-prime-factors-iff)
  with  $!$ [of  $p$ ] show False by blast
qed

```

1.8 Square decompositions of prime numbers congruent 1 mod 4

```

lemma prime-1-mod-4-sum-of-squares-unique-aux:
  fixes  $p \ x \ y :: \text{nat}$ 
  assumes prime  $p$  [ $p = 1$ ] (mod 4)  $x^2 + y^2 = p$ 
  shows  $x > 0 \wedge y > 0 \wedge x \neq y$ 
proof safe
  from assms show  $x > 0 \ y > 0$ 
  by (auto intro!: Nat.gr0I simp: prime-power-iff)
next
  assume  $x = y$ 
  with assms have  $p = 2 * x^2$ 
  by simp
  with  $\langle$ prime  $p$  $\rangle$  have  $p = 2$ 
  by (auto dest: prime-product)
  with  $\langle$ [ $p = 1$ ] (mod 4) $\rangle$  show False
  by (simp add: cong-def)
qed

```

Any prime number congruent 1 modulo 4 can be written *uniquely* as a sum of two squares $x^2 + y^2$ (up to commutativity of the addition). Additionally, we have shown above that x and y are both positive and $x \neq y$.

```

lemma prime-1-mod-4-sum-of-squares-unique:
  fixes  $p :: \text{nat}$ 
  assumes prime  $p$  [ $p = 1$ ] (mod 4)
  shows  $\exists!$ ( $x, y$ ).  $x \leq y \wedge x^2 + y^2 = p$ 
proof (rule ex-ex1I)
  obtain  $z$  where  $z$ : gauss-int-norm  $z = p$ 
  using prime-cong-1-mod-4-gauss-int-norm-exists[OF assms] by blast
  show  $\exists z$ . case  $z$  of ( $x, y$ )  $\Rightarrow x \leq y \wedge x^2 + y^2 = p$ 
  proof (cases  $|ReZ \ z| \leq |ImZ \ z|$ )
  case True
  with  $z$  show ?thesis by
    (intro exI[of - (nat |ReZ z|, nat |ImZ z|)])
    (auto simp: gauss-int-norm-def nat-add-distrib simp flip: nat-power-eq)
next

```

```

    case False
  with z show ?thesis by
    (intro exI[of - (nat |ImZ z|, nat |ReZ z|)])
    (auto simp: gauss-int-norm-def nat-add-distrib simp flip: nat-power-eq)
qed
next
fix z1 z2
assume z1: case z1 of (x, y) ⇒ x ≤ y ∧ x2 + y2 = p
assume z2: case z2 of (x, y) ⇒ x ≤ y ∧ x2 + y2 = p
define z1' :: gauss-int where z1' = of-nat (fst z1) + iZ * of-nat (snd z1)
define z2' :: gauss-int where z2' = of-nat (fst z2) + iZ * of-nat (snd z2)
from assms interpret noninert-gauss-int-prime p
  by unfold-locales auto
have norm-z1': gauss-int-norm z1' = p
  using z1 by (simp add: z1'-def gauss-int-norm-def case-prod-unfold nat-add-distrib
nat-power-eq)
have norm-z2': gauss-int-norm z2' = p
  using z2 by (simp add: z2'-def gauss-int-norm-def case-prod-unfold nat-add-distrib
nat-power-eq)

  have sgns: fst z1 > 0 snd z1 > 0 fst z2 > 0 snd z2 > 0 fst z1 ≠ snd z1 fst z2
  ≠ snd z2
  using prime-1-mod-4-sum-of-squares-unique-aux[OF assms, of fst z1 snd z1] z1
  prime-1-mod-4-sum-of-squares-unique-aux[OF assms, of fst z2 snd z2] z2
by auto
have [simp]: normalize z1' = z1' normalize z2' = z2'
  using sgns by (subst normalized-gauss-int-iff; simp add: z1'-def z2'-def)+
have prime z1' prime z2'
  using norm-z1' norm-z2' assms unfolding prime-def
  by (auto simp: prime-gauss-int-norm-imp-prime-elem)

have of-nat p = z1' * gauss-cnj z1'
  by (simp add: self-mult-gauss-cnj norm-z1')
hence z1' dvd of-nat p
  by simp
also have of-nat p = z2' * gauss-cnj z2'
  by (simp add: self-mult-gauss-cnj norm-z2')
finally have z1' dvd z2' ∨ z1' dvd gauss-cnj z2' using assms
  by (subst (asm) prime-elem-dvd-mult-iff)
  (simp add: norm-z1' prime-gauss-int-norm-imp-prime-elem)
thus z1 = z2
proof
  assume z1' dvd z2'
  with ⟨prime z1'⟩ ⟨prime z2'⟩ have z1' = z2'
    by (simp add: primes-dvd-imp-eq)
  thus ?thesis
    by (simp add: z1'-def z2'-def gauss-int-eq-iff prod-eq-iff)
next
  assume dvd: z1' dvd gauss-cnj z2'

```

```

have normalize (i $\mathbb{Z}$  * gauss-cnj z2') = i $\mathbb{Z}$  * gauss-cnj z2'
  using sgns by (subst normalized-gauss-int-iff) (auto simp: z2'-def)
moreover have prime-elem (i $\mathbb{Z}$  * gauss-cnj z2')
  by (rule prime-gauss-int-norm-imp-prime-elem)
  (simp add: gauss-int-norm-mult norm-z2' <prime p>)
ultimately have prime (i $\mathbb{Z}$  * gauss-cnj z2')
  by (simp add: prime-def)
moreover from dvd have z1' dvd i $\mathbb{Z}$  * gauss-cnj z2'
  by simp
ultimately have z1' = i $\mathbb{Z}$  * gauss-cnj z2'
  using <prime z1'> by (simp add: primes-dvd-imp-eq)
hence False using z1 z2 sgns
  by (auto simp: gauss-int-eq-iff z1'-def z2'-def)
thus ?thesis ..
qed
qed

```

lemma two-sum-of-squares-nat-iff: $(x :: \text{nat})^2 + y^2 = 2 \iff x = 1 \wedge y = 1$

proof

```

assume eq: x^2 + y^2 = 2
have square-neq-2: n^2 ≠ 2 for n :: nat
proof
  assume *: n^2 = 2
  have prime (2 :: nat)
  by simp
  thus False by (subst (asm) * [symmetric]) (auto simp: prime-power-iff)
qed

```

```

from eq have x^2 < 2^2 y^2 < 2^2
  by simp-all
hence x < 2 y < 2
  using power2-less-imp-less[of x 2] power2-less-imp-less[of y 2] by auto
moreover have x > 0 y > 0
  using eq square-neq-2[of x] square-neq-2[of y] by (auto intro!: Nat.gr0I)
ultimately show x = 1 ∧ y = 1
  by auto
qed auto

```

lemma prime-sum-of-squares-unique:

```

fixes p :: nat
assumes prime p p = 2 ∨ [p = 1] (mod 4)
shows ∃!(x,y). x ≤ y ∧ x^2 + y^2 = p
using assms(2)
proof
  assume [simp]: p = 2
  have **: (λ(x,y). x ≤ y ∧ x^2 + y^2 = p) = (λz. z = (1,1 :: nat))
    using two-sum-of-squares-nat-iff by (auto simp: fun-eq-iff)
  thus ?thesis

```

by (subst **) auto
qed (use prime-1-mod-4-sum-of-squares-unique[of p] assms in auto)

We now give a simple and inefficient algorithm to compute the canonical decomposition $x^2 + y^2$ with $x \leq y$.

definition prime-square-sum-nat-decomp :: nat \Rightarrow nat \times nat **where**

prime-square-sum-nat-decomp p =
(if prime p \wedge (p = 2 \vee [p = 1] (mod 4))
then THE (x,y). x \leq y \wedge x² + y² = p else (0, 0))

lemma prime-square-sum-nat-decomp-eqI:

assumes prime p x² + y² = p x \leq y
shows prime-square-sum-nat-decomp p = (x, y)

proof –

have [gauss-int-norm (of-nat x + i_Z * of-nat y) \neq 3] (mod 4)

by (rule gauss-int-norm-not-3-mod-4)

also have gauss-int-norm (of-nat x + i_Z * of-nat y) = p

using assms by (auto simp: gauss-int-norm-def nat-add-distrib nat-power-eq)

finally have [p \neq 3] (mod 4) .

with prime-mod-4-cases[of p] assms have *: p = 2 \vee [p = 1] (mod 4)

by auto

have prime-square-sum-nat-decomp p = (THE (x,y). x \leq y \wedge x² + y² = p)

using * \langle prime p \rangle by (simp add: prime-square-sum-nat-decomp-def)

also have ... = (x, y)

proof (rule the1-equality)

show $\exists!(x,y). x \leq y \wedge x^2 + y^2 = p$

using \langle prime p \rangle * by (rule prime-sum-of-squares-unique)

qed (use assms in auto)

finally show ?thesis .

qed

lemma prime-square-sum-nat-decomp-correct:

assumes prime p p = 2 \vee [p = 1] (mod 4)

defines z \equiv prime-square-sum-nat-decomp p

shows fst z² + snd z² = p fst z \leq snd z

proof –

define z' **where** z' = (THE (x,y). x \leq y \wedge x² + y² = p)

have z = z'

unfolding z-def z'-def using assms by (simp add: prime-square-sum-nat-decomp-def)

also have $\exists!(x,y). x \leq y \wedge x^2 + y^2 = p$

using assms by (intro prime-sum-of-squares-unique)

hence case z' of (x, y) \Rightarrow x \leq y \wedge x² + y² = p

unfolding z'-def by (rule theI')

finally show fst z² + snd z² = p fst z \leq snd z

by auto

qed

```

lemma sum-of-squares-nat-bound:
  fixes  $x\ y\ n :: \text{nat}$ 
  assumes  $x^2 + y^2 = n$ 
  shows  $x \leq n$ 
proof (cases  $x = 0$ )
  case False
  hence  $x * 1 \leq x^2$ 
    unfolding power2-eq-square by (intro mult-mono) auto
  also have  $\dots \leq x^2 + y^2$ 
    by simp
  also have  $\dots = n$ 
    by fact
  finally show ?thesis by simp
qed auto

lemma sum-of-squares-nat-bound':
  fixes  $x\ y\ n :: \text{nat}$ 
  assumes  $x^2 + y^2 = n$ 
  shows  $y \leq n$ 
  using sum-of-squares-nat-bound [of  $y\ x$ ] assms by (simp add: add.commute)

lemma is-singleton-conv-Ex1:
  is-singleton  $A \longleftrightarrow (\exists!x. x \in A)$ 
proof
  assume is-singleton  $A$ 
  thus  $\exists!x. x \in A$ 
    by (auto elim!: is-singletonE)
next
  assume  $\exists!x. x \in A$ 
  thus is-singleton  $A$ 
    by (metis equals0D is-singletonI')
qed

lemma the-elemI:
  assumes is-singleton  $A$ 
  shows the-elem  $A \in A$ 
  using assms by (elim is-singletonE) auto

lemma prime-square-sum-nat-decomp-code-aux:
  assumes prime  $p\ p = 2 \vee [p = 1] \pmod{4}$ 
  defines  $z \equiv \text{the-elem } (\text{Set.filter } (\lambda(x,y). x^2 + y^2 = p) (\text{SIGMA } x:\{0..p\}. \{x..p\}))$ 
  shows prime-square-sum-nat-decomp  $p = z$ 
proof -
  let  $?A = \text{Set.filter } (\lambda(x,y). x^2 + y^2 = p) (\text{SIGMA } x:\{0..p\}. \{x..p\})$ 
  have eq:  $?A = \{(x,y). x \leq y \wedge x^2 + y^2 = p\}$ 
    using sum-of-squares-nat-bound [of - -  $p$ ] sum-of-squares-nat-bound' [of - -  $p$ ]
  by auto
  have  $z: z \in \text{Set.filter } (\lambda(x,y). x^2 + y^2 = p) (\text{SIGMA } x:\{0..p\}. \{x..p\})$ 

```

```

unfolding z-def eq using prime-sum-of-squares-unique[OF assms(1,2)]
by (intro the-elemI) (simp add: is-singleton-conv-Ex1)
have prime-square-sum-nat-decomp p = (fst z, snd z)
using z by (intro prime-square-sum-nat-decomp-eqI[OF assms(1)]) auto
also have ... = z
by simp
finally show ?thesis .
qed

```

```

lemma prime-square-sum-nat-decomp-code [code]:
  prime-square-sum-nat-decomp p =
    (if prime p  $\wedge$  (p = 2  $\vee$  [p = 1] (mod 4))
     then the-elem (Set.filter ( $\lambda(x,y). x^2 + y^2 = p$ ) (SIGMA x:{0..p}. {x..p}))
     else (0, 0))
using prime-square-sum-nat-decomp-code-aux[of p]
by (auto simp: prime-square-sum-nat-decomp-def)

```

1.9 Executable factorisation of Gaussian integers

Lastly, we use all of the above to give an executable (albeit not very efficient) factorisation algorithm for Gaussian integers based on factorisation of regular integers. Note that we will only compute the set of prime factors without multiplicity, but given that, it would be fairly easy to determine the multiplicity as well.

First, we need the following function that computes the Gaussian integer factors of a \mathbb{Z} -prime p :

```

definition factor-gauss-int-prime-nat :: nat  $\Rightarrow$  gauss-int list where
  factor-gauss-int-prime-nat p =
    (if p = 2 then [1 + iz]
     else if [p = 3] (mod 4) then [of-nat p]
     else case prime-square-sum-nat-decomp p of
          (x, y)  $\Rightarrow$  [of-nat x + iz * of-nat y, of-nat y + iz * of-nat x])

```

```

lemma factor-gauss-int-prime-nat-correct:
assumes prime p
shows set (factor-gauss-int-prime-nat p) = prime-factors (of-nat p)
using prime-mod-4-cases[OF assms]
proof (elim disjE)
assume p = 2
thus ?thesis
by (auto simp: prime-factorization-2-gauss-int factor-gauss-int-prime-nat-def)
next
assume *: [p = 3] (mod 4)
with assms have prime (of-nat p :: gauss-int)
by (intro prime-gauss-int-of-nat)
thus ?thesis using assms *
by (auto simp: prime-factorization-prime factor-gauss-int-prime-nat-def cong-def)
next

```

```

assume *: [p = 1] (mod 4)
then interpret noninert-gauss-int-prime p
  using ⟨prime p⟩ by unfold-locales
define z where z = prime-square-sum-nat-decomp p
define x y where x = fst z and y = snd z
have xy: x ^ 2 + y ^ 2 = p x ≤ y
  using prime-square-sum-nat-decomp-correct[of p] * assms
  by (auto simp: x-def y-def z-def)
from xy have xy-signs: x > 0 y > 0
  using prime-1-mod-4-sum-of-squares-unique-aux[of p x y] assms * by auto
have norms: gauss-int-norm (of-nat x + iZ * of-nat y) = p
  gauss-int-norm (of-nat y + iZ * of-nat x) = p
  using xy by (auto simp: gauss-int-norm-def nat-add-distrib nat-power-eq)
have prime: prime (of-nat x + iZ * of-nat y) prime (of-nat y + iZ * of-nat x)
  using norms xy-signs ⟨prime p⟩ unfolding prime-def normalized-gauss-int-iff
  by (auto intro!: prime-gauss-int-norm-imp-prime-elem)

have normalize ((of-nat x + iZ * of-nat y) * (of-nat y + iZ * of-nat x)) = of-nat
p
proof –
  have (of-nat x + iZ * of-nat y) * (of-nat y + iZ * of-nat x) = (iZ * of-nat p ::
gauss-int)
  by (subst xy(1) [symmetric]) (auto simp: gauss-int-eq-iff power2-eq-square)
  also have normalize ... = of-nat p
  by simp
  finally show ?thesis .
qed
hence prime-factorization (of-nat p) =
  prime-factorization (prod-mset {#of-nat x + iZ * of-nat y, of-nat y + iZ *
of-nat x#})
  using assms xy by (subst prime-factorization-unique) (auto simp: gauss-int-eq-iff)
  also have ... = {#of-nat x + iZ * of-nat y, of-nat y + iZ * of-nat x#}
  using prime by (subst prime-factorization-prod-mset-primes) auto
finally have prime-factors (of-nat p) = {of-nat x + iZ * of-nat y, of-nat y + iZ
* of-nat x}
  by simp
  also have ... = set (factor-gauss-int-prime-nat p)
  using * unfolding factor-gauss-int-prime-nat-def case-prod-unfold
  by (auto simp: cong-def x-def y-def z-def)
  finally show ?thesis ..
qed

```

Next, we lift this to compute the prime factorisation of any integer in the Gaussian integers:

definition prime-factors-gauss-int-of-nat :: nat ⇒ gauss-int set **where**
 prime-factors-gauss-int-of-nat n = (if n = 0 then {} else
 (⋃ p∈prime-factors n. set (factor-gauss-int-prime-nat p)))

lemma prime-factors-gauss-int-of-nat-correct:

```

    prime-factors-gauss-int-of-nat n = prime-factors (of-nat n)
proof (cases n = 0)
  case False
    from False have [simp]: n > 0 by auto
    have prime-factors (of-nat n :: gauss-int) =
      prime-factors (of-nat (prod-mset (prime-factorization n)))
      by (subst prod-mset-prime-factorization-nat [symmetric]) auto
    also have ... = prime-factors (prod-mset (image-mset of-nat (prime-factorization
n)))
      by (subst of-nat-prod-mset) auto
    also have ... = (⋃ p ∈ prime-factors n. prime-factors (of-nat p))
      by (subst prime-factorization-prod-mset) auto
    also have ... = (⋃ p ∈ prime-factors n. set (factor-gauss-int-prime-nat p))
      by (intro SUP-cong refl factor-gauss-int-prime-nat-correct [symmetric]) auto
    finally show ?thesis by (simp add: prime-factors-gauss-int-of-nat-def)
qed (auto simp: prime-factors-gauss-int-of-nat-def)

```

We can now use this to factor any Gaussian integer by computing a factorisation of its norm and removing all the prime divisors that do not actually divide it.

definition *prime-factors-gauss-int* :: *gauss-int* ⇒ *gauss-int set* **where**
prime-factors-gauss-int z = (if z = 0 then {}
else *Set.filter* (λp. p dvd z) (*prime-factors-gauss-int-of-nat* (*gauss-int-norm* z)))

lemma *prime-factors-gauss-int-correct* [code-unfold]: *prime-factors* z = *prime-factors-gauss-int* z

```

proof (cases z = 0)
  case [simp]: False
    define n where n = gauss-int-norm z
    from False have [simp]: n > 0 by (auto simp: n-def)

    have prime-factors-gauss-int z = Set.filter (λp. p dvd z) (prime-factors (of-nat
n))
      by (simp add: prime-factors-gauss-int-of-nat-correct prime-factors-gauss-int-def
n-def)
    also have of-nat n = z * gauss-cnj z
      by (simp add: n-def self-mult-gauss-cnj)
    also have prime-factors ... = prime-factors z ∪ prime-factors (gauss-cnj z)
      by (subst prime-factors-product) auto
    also have Set.filter (λp. p dvd z) ... = prime-factors z
      by (auto simp: in-prime-factors-iff)
    finally show ?thesis by simp
qed (auto simp: prime-factors-gauss-int-def)

```

end

theory *Gaussian-Integers-Test*
imports
Gaussian-Integers

Polynomial-Factorization.Prime-Factorization
HOL-Library.Code-Target-Numeral
begin

Lastly, we apply our factorisation algorithm to some simple examples:

```
value (1234 + 5678 * iZ) mod (321 + 654 * iZ)
value prime-factors (1 + 3 * iZ)
value prime-factors (4830 + 1610 * iZ)
```

end

1.10 Sums of two squares

theory *Gaussian-Integers-Sums-Of-Two-Squares*
imports *Gaussian-Integers*
begin

As an application, we can now easily prove that a positive natural number is the sum of two squares if and only if all prime factors congruent 3 modulo 4 have even multiplicity.

inductive *sum-of-2-squares-nat* :: *nat* ⇒ *bool* **where**
sum-of-2-squares-nat ($a^2 + b^2$)

lemma *sum-of-2-squares-nat-altdef*: *sum-of-2-squares-nat* n ⇔ $n \in \text{range } \text{gauss-int-norm}$
proof (*safe elim!*: *sum-of-2-squares-nat.cases*)

```
fix a b :: nat
have  $a^2 + b^2 = \text{gauss-int-norm } (\text{of-nat } a + i_{\mathbb{Z}} * \text{of-nat } b)$ 
  by (auto simp: gauss-int-norm-def nat-add-distrib nat-power-eq)
thus  $a^2 + b^2 \in \text{range } \text{gauss-int-norm}$  by blast
```

next

```
fix z :: gauss-int
have gauss-int-norm z =  $\text{nat } |\text{Re } z|^2 + \text{nat } |\text{Im } z|^2$ 
  by (auto simp: gauss-int-norm-def nat-add-distrib simp flip: nat-power-eq)
thus sum-of-2-squares-nat (gauss-int-norm z)
  by (auto intro: sum-of-2-squares-nat.intros)
```

qed

lemma *sum-of-2-squares-nat-gauss-int-norm* [*intro*]: *sum-of-2-squares-nat* (*gauss-int-norm* z)
by (*auto simp: sum-of-2-squares-nat-altdef*)

lemma *sum-of-2-squares-nat-0* [*simp, intro*]: *sum-of-2-squares-nat* 0
and *sum-of-2-squares-nat-1* [*simp, intro*]: *sum-of-2-squares-nat* 1
and *sum-of-2-squares-nat-Suc-0* [*simp, intro*]: *sum-of-2-squares-nat* (*Suc* 0)
and *sum-of-2-squares-nat-2* [*simp, intro*]: *sum-of-2-squares-nat* 2
using *sum-of-2-squares-nat.intros*[of 0 0] *sum-of-2-squares-nat.intros*[of 0 1]
sum-of-2-squares-nat.intros[of 1 1] **by** (*simp-all add: numeral-2-eq-2*)

lemma *sum-of-2-squares-nat-mult* [intro]:
assumes *sum-of-2-squares-nat x sum-of-2-squares-nat y*
shows *sum-of-2-squares-nat (x * y)*
proof –
from *assms* **obtain** *z1 z2* **where** *x = gauss-int-norm z1 y = gauss-int-norm z2*
by (*auto simp: sum-of-2-squares-nat-altdef*)
hence *x * y = gauss-int-norm (z1 * z2)*
by (*simp add: gauss-int-norm-mult*)
thus *?thesis* **by** *auto*
qed

lemma *sum-of-2-squares-nat-power* [intro]:
assumes *sum-of-2-squares-nat m*
shows *sum-of-2-squares-nat (m ^ n)*
using *assms* **by** (*induction n*) *auto*

lemma *sum-of-2-squares-nat-prod* [intro]:
assumes $\bigwedge x. x \in A \implies \text{sum-of-2-squares-nat } (f x)$
shows *sum-of-2-squares-nat ($\prod_{x \in A}. f x$)*
using *assms* **by** (*induction A rule: infinite-finite-induct*) *auto*

lemma *sum-of-2-squares-nat-prod-mset* [intro]:
assumes $\bigwedge x. x \in\# A \implies \text{sum-of-2-squares-nat } x$
shows *sum-of-2-squares-nat (prod-mset A)*
using *assms* **by** (*induction A*) *auto*

lemma *sum-of-2-squares-nat-necessary*:
assumes *sum-of-2-squares-nat n n > 0*
assumes *prime p [p = 3] (mod 4)*
shows *even (multiplicity p n)*
proof –
define *k* **where** *k = multiplicity p n*
from *assms* **obtain** *z* **where** *z: gauss-int-norm z = n*
by (*auto simp: sum-of-2-squares-nat-altdef*)
from *assms* **and** *z* **have** [*simp*]: *z ≠ 0*
by *auto*
have *prime'*: *prime (of-nat p :: gauss-int)*
using *assms* *prime-gauss-int-of-nat* **by** *blast*
have [*simp*]: *multiplicity (of-nat p) (gauss-cnj z) = multiplicity (of-nat p) z*
using *multiplicity-gauss-cnj[of of-nat p z]* **by** *simp*
have *multiplicity (of-nat p) (of-nat n :: gauss-int) =*
*multiplicity (of-nat p) (z * gauss-cnj z)*
using *z* **by** (*simp add: self-mult-gauss-cnj*)
also **have** $\dots = 2 * \text{multiplicity } (of\text{-nat } p) z$
using *prime'* **by** (*subst prime-elem-multiplicity-mult-distrib*) *auto*
finally **have** *multiplicity p n = 2 * multiplicity (of-nat p) z*
by (*subst (asm) multiplicity-gauss-int-of-nat*)
thus *?thesis* **by** *auto*
qed

lemma *sum-of-2-squares-nat-sufficient*:

fixes $n :: \text{nat}$

assumes $n > 0$

assumes $\bigwedge p. p \in \text{prime-factors } n \implies [p = 3] \pmod{4} \implies \text{even } (\text{multiplicity } p \ n)$

shows *sum-of-2-squares-nat* n

proof –

define $P2$ **where** $P2 = \{p \in \text{prime-factors } n. [p = 1] \pmod{4}\}$

define $P3$ **where** $P3 = \{p \in \text{prime-factors } n. [p = 3] \pmod{4}\}$

from $\langle n > 0 \rangle$ **have** $n = (\prod_{p \in \text{prime-factors } n} p^{\text{multiplicity } p \ n})$

by (*subst prime-factorization-nat*) *auto*

also have $\dots = (\prod_{p \in \{2\} \cup P2 \cup P3} p^{\text{multiplicity } p \ n})$

using *prime-mod-4-cases*

by (*intro prod.mono-neutral-left*)

(*auto simp: P2-def P3-def in-prime-factors-iff not-dvd-imp-multiplicity-0*)

also have $\dots = (\prod_{p \in \{2\} \cup P2} p^{\text{multiplicity } p \ n}) * (\prod_{p \in P3} p^{\text{multiplicity } p \ n})$

by (*intro prod.union-disjoint*) (*auto simp: P2-def P3-def cong-def*)

also have $(\prod_{p \in \{2\} \cup P2} p^{\text{multiplicity } p \ n}) = 2^{\text{multiplicity } 2 \ n} * (\prod_{p \in P2} p^{\text{multiplicity } p \ n})$

by (*subst prod.union-disjoint*) (*auto simp: P2-def cong-def*)

also have $(\prod_{p \in P3} p^{\text{multiplicity } p \ n}) = (\prod_{p \in P3} (p^2)^{\text{multiplicity } p \ n \ \text{div } 2})$

proof (*intro prod.cong refl*)

fix $p :: \text{nat}$ **assume** $p \in P3$

have $(p^2)^{\text{multiplicity } p \ n \ \text{div } 2} = p^{2 * (\text{multiplicity } p \ n \ \text{div } 2)}$

by (*simp add: power-mult*)

also have $\text{even } (\text{multiplicity } p \ n)$

using *assms p* **by** (*auto simp: P3-def*)

hence $2 * (\text{multiplicity } p \ n \ \text{div } 2) = \text{multiplicity } p \ n$

by *simp*

finally show $p^{\text{multiplicity } p \ n} = (p^2)^{\text{multiplicity } p \ n \ \text{div } 2}$

by *simp*

qed

finally have $n = 2^{\text{multiplicity } 2 \ n} * (\prod_{p \in P2} p^{\text{multiplicity } p \ n}) * (\prod_{p \in P3} p^{2 * (\text{multiplicity } p \ n \ \text{div } 2)})$

also have *sum-of-2-squares-nat* \dots

proof (*intro sum-of-2-squares-nat-mult sum-of-2-squares-nat-prod; rule sum-of-2-squares-nat-power*)

fix $p :: \text{nat}$ **assume** $p \in P2$

with *prime-cong-1-mod-4-gauss-int-norm-exists*[*of p*] **show** *sum-of-2-squares-nat* p

by (*auto simp: P2-def*)

next

fix $p :: \text{nat}$ **assume** $p \in P3$

have *sum-of-2-squares-nat* (*gauss-int-norm* (*of-nat p*)) **..**

also have *gauss-int-norm* (*of-nat p*) = p^2

by *simp*

finally show *sum-of-2-squares-nat* $(p \wedge 2)$.
qed *auto*
finally show *?thesis* .
qed

theorem *sum-of-2-squares-nat-iff*:
sum-of-2-squares-nat $n \longleftrightarrow$
 $n = 0 \vee (\forall p \in \text{prime-factors } n. [p = 3] \pmod{4} \longrightarrow \text{even } (\text{multiplicity } p \ n))$
using *sum-of-2-squares-nat-necessary*[*of* n] *sum-of-2-squares-nat-sufficient*[*of* n]
by *auto*

end

1.11 Primitive Pythagorean triples

theory *Gaussian-Integers-Pythagorean-Triples*
imports *Gaussian-Integers*
begin

In this section, we derive Euclid's formula for primitive Pythagorean triples using Gaussian integers, following Stillwell [1].

definition *prim-pyth-triple* $:: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
prim-pyth-triple $x \ y \ z \longleftrightarrow x > 0 \wedge y > 0 \wedge \text{coprime } x \ y \wedge x^2 + y^2 = z^2$

lemma *prim-pyth-triple-commute*: *prim-pyth-triple* $x \ y \ z \longleftrightarrow \text{prim-pyth-triple } y \ x \ z$
by (*simp add: prim-pyth-triple-def coprime-commute add-ac conj-ac*)

lemma *prim-pyth-triple-aux*:
fixes $u \ v :: \text{nat}$
assumes $v \leq u$
shows $(2 * u * v) \wedge 2 + (u \wedge 2 - v \wedge 2) \wedge 2 = (u \wedge 2 + v \wedge 2) \wedge 2$
proof –
have $\text{int } ((2 * u * v) \wedge 2 + (u \wedge 2 - v \wedge 2) \wedge 2) =$
 $(2 * \text{int } u * \text{int } v) \wedge 2 + (\text{int } u \wedge 2 - \text{int } v \wedge 2) \wedge 2$
using *assms* **by** (*simp add: of-nat-diff*)
also have $\dots = (\text{int } u \wedge 2 + \text{int } v \wedge 2) \wedge 2$
by (*simp add: power2-eq-square algebra-simps*)
also have $\dots = \text{int } ((u \wedge 2 + v \wedge 2) \wedge 2)$
by *simp*
finally show *?thesis*
by (*simp only: of-nat-eq-iff*)
qed

lemma *prim-pyth-tripleI1*:
assumes $0 < v < u \text{ coprime } u \ v \neg(\text{odd } u \wedge \text{odd } v)$
shows *prim-pyth-triple* $(2 * u * v) \ (u^2 - v^2) \ (u^2 + v^2)$
proof –
have $v \wedge 2 < u \wedge 2$

using *assms* by (intro power-strict-mono) auto
 hence $\neg u^2 < v^2$ by *linarith*

from *assms* have coprime (int u) (int v²)
 by auto
 hence coprime (int u) (int u * int u + (-(int v²)))
 unfolding coprime-iff-gcd-eq-1 by (subst gcd-add-mult) auto
 also have int u * int u + (-(int v²)) = int (u² - v²)
 using $\langle v < u \rangle$ by (simp add: of-nat-diff flip: power2-eq-square)
 finally have coprime1: coprime u (u² - v²)
 by auto

from *assms* have coprime (int v) (int u²)
 by (auto simp: coprime-commute)
 hence coprime (int v) ((-int v) * int v + int u²)
 unfolding coprime-iff-gcd-eq-1 by (subst gcd-add-mult) auto
 also have (-int v) * int v + int u² = int (u² - v²)
 using $\langle v < u \rangle$ by (simp add: of-nat-diff flip: power2-eq-square)
 finally have coprime2: coprime v (u² - v²)
 by auto

have $(2 * u * v)^2 + (u^2 - v^2)^2 = (u^2 + v^2)^2$
 using $\langle v < u \rangle$ by (intro prim-pyth-triple-aux) auto
 moreover have coprime (2 * u * v) (u² - v²)
 using *assms* $\langle \neg u^2 < v^2 \rangle$ coprime1 coprime2 by auto
 ultimately show ?thesis using *assms* $\langle v^2 < u^2 \rangle$
 by (simp add: prim-pyth-triple-def)

qed

lemma prim-pyth-tripleI2:
 assumes $0 < v < u$ coprime u v $\neg(\text{odd } u \wedge \text{odd } v)$
 shows prim-pyth-triple (u² - v²) (2 * u * v) (u² + v²)
 using prim-pyth-tripleI1[OF *assms*] by (simp add: prim-pyth-triple-commute)

lemma primitive-pythagorean-tripleE-int:
 assumes $z^2 = x^2 + y^2$
 assumes coprime x y
 obtains u v :: int
 where coprime u v and $\neg(\text{odd } u \wedge \text{odd } v)$
 and $x = 2 * u * v \wedge y = u^2 - v^2 \vee x = u^2 - v^2 \wedge y = 2 * u * v$

proof -

have $\neg(\text{even } x \wedge \text{even } y)$
 using not-coprimeI[of 2 x y] $\langle \text{coprime } x \ y \rangle$ by auto
 moreover have $\neg(\text{odd } x \wedge \text{odd } y)$

proof safe

assume odd x odd y
 hence $[x^2 + y^2 = 1 + 1] \pmod 4$
 by (intro cong-add odd-square-cong-4-int)
 hence $[z^2 = 2] \pmod 4$

```

    by (simp add: assms)
  moreover have  $[z^2 = 0] \pmod{4} \vee [z^2 = 1] \pmod{4}$ 
    using even-square-cong-4-int[of z] odd-square-cong-4-int[of z]
    by (cases even z) auto
  ultimately show False
    by (auto simp: cong-def)
qed
ultimately have  $\text{even } y \iff \text{odd } x$ 
  by blast

have  $\text{even } z \iff \text{even } (z^2)$ 
  by auto
also have  $\text{even } (z^2) \iff \text{even } (x^2 + y^2)$ 
  by (subst assms(1)) auto
finally have  $\text{odd } z$ 
  by (cases even x) (auto simp:  $\langle \text{even } y \iff \neg \text{even } x \rangle$ )

define t where  $t = \text{of-int } x + i_{\mathbb{Z}} * \text{of-int } y$ 
from assms have t-mult-cnj:  $t * \text{gauss-cnj } t = \text{of-int } z^2$ 
  by (simp add: t-def power2-eq-square algebra-simps flip: of-int-mult of-int-add)

have gauss-int-norm  $t = z^2$ 
  by (simp add: gauss-int-norm-def t-def assms)
with  $\langle \text{coprime } x \ y \rangle$  and  $\langle \text{odd } z \rangle$  have coprime t (gauss-cnj t)
  by (intro coprime-self-gauss-cnj)
  (auto simp: t-def gauss-int-norm-def assms(1) [symmetric] even-nat-iff)
moreover have is-square (t * gauss-cnj t)
  by (subst t-mult-cnj) auto
hence is-nth-power-upto-unit 2 (t * gauss-cnj t)
  by (auto intro: is-nth-power-upto-unit-base)
ultimately have is-nth-power-upto-unit 2 t
  by (rule is-nth-power-upto-unit-mult-coprimeD1)
then obtain a b where  $ab: \text{is-unit } a \ a * t = b^2$ 
  by (auto simp: is-nth-power-upto-unit-def is-nth-power-def)
from ab(1) have  $a \in \{1, -1, i_{\mathbb{Z}}, -i_{\mathbb{Z}}\}$ 
  by (auto simp: is-unit-gauss-int-iff)
then obtain  $u \ v :: \text{int}$  where  $\text{Re } Z \ t = 2 * u * v \wedge \text{Im } Z \ t = u^2 - v^2 \vee$ 
 $\text{Im } Z \ t = 2 * u * v \wedge \text{Re } Z \ t = u^2 - v^2$ 

proof safe
  assume [simp]:  $a = 1$ 
  have  $\text{Re } Z \ t = \text{Re } Z \ b^2 - \text{Im } Z \ b^2 \ \text{Im } Z \ t = 2 * \text{Re } Z \ b * \text{Im } Z \ b$  using ab(2)
    by (auto simp: gauss-int-eq-iff power2-eq-square)
  thus ?thesis using that by blast
next
  assume [simp]:  $a = -1$ 
  have  $\text{Re } Z \ t = \text{Im } Z \ b^2 - (-\text{Re } Z \ b)^2 \ \text{Im } Z \ t = 2 * \text{Im } Z \ b * (-\text{Re } Z \ b)$ 
using ab(2)
  by (auto simp: gauss-int-eq-iff power2-eq-square algebra-simps)
  thus ?thesis using that by blast

```

```

next
  assume [simp]: a = iZ
  hence ImZ t = ImZ b ^ 2 - ReZ b ^ 2 ReZ t = 2 * ImZ b * ReZ b using
ab(2)
  by (auto simp: gauss-int-eq-iff power2-eq-square algebra-simps)
  thus ?thesis using that by blast
next
  assume [simp]: a = -iZ
  hence ImZ t = (-ReZ b) ^ 2 - ImZ b ^ 2 ReZ t = 2 * (-ReZ b) * ImZ b
using ab(2)
  by (auto simp: gauss-int-eq-iff power2-eq-square algebra-simps)
  thus ?thesis using that by blast
qed
also have ReZ t = x
  by (simp add: t-def)
also have ImZ t = y
  by (simp add: t-def)
finally have xy: x = 2 * u * v ∧ y = u2 - v2 ∨ x = u2 - v2 ∧ y = 2 * u * v
  by blast

have not-both-odd: ¬(odd u ∧ odd v)
proof safe
  assume odd u odd v
  hence even x even y
    using xy by auto
  with ⟨coprime x y⟩ show False
    by auto
qed

have coprime u v
proof (rule coprimeI)
  fix d assume d dvd u d dvd v
  hence d dvd (u2 - v2) d dvd 2 * u * v
    by (auto simp: power2-eq-square)
  with xy have d dvd x d dvd y
    by auto
  with ⟨coprime x y⟩ show is-unit d
    using not-coprimeI by blast
qed
with xy not-both-odd show ?thesis
  using that[of u v] by blast
qed

lemma prim-pyth-tripleE:
  assumes prim-pyth-triple x y z
  obtains u v :: nat
  where 0 < v and v < u and coprime u v and ¬(odd u ∧ odd v) and z = u2 +
v2
  and x = 2 * u * v ∧ y = u2 - v2 ∨ x = u2 - v2 ∧ y = 2 * u * v

```

proof –
have *: $(\text{int } z) \wedge 2 = (\text{int } x) \wedge 2 + (\text{int } y) \wedge 2$ *coprime* $(\text{int } x) (\text{int } y)$
using *assms* **by** $(\text{auto simp flip: of-nat-power of-nat-add simp: prim-pyth-triple-def})$
obtain $u v$
where $uv: \text{coprime } u v \neg(\text{odd } u \wedge \text{odd } v)$
 $\text{int } x = 2 * u * v \wedge \text{int } y = u^2 - v^2 \vee \text{int } x = u^2 - v^2 \wedge \text{int } y = 2$
 $* u * v$
using *primitive-pythagorean-tripleE-int[OF *]* **by** *metis*
define $u' v'$ **where** $u' = \text{nat } |u|$ **and** $v' = \text{nat } |v|$

have **: $a = 2 * u' * v'$ **if** $\text{int } a = 2 * u * v$ **for** a
proof –
from *that* **have** $\text{nat } |\text{int } a| = \text{nat } |2 * u * v|$
by *(simp only:)*
thus $a = 2 * u' * v'$
by *(simp add: u'-def v'-def abs-mult nat-mult-distrib)*
qed

have ***: $a = u' \wedge 2 - v' \wedge 2$ $v' \leq u'$ **if** $\text{int } a = u \wedge 2 - v \wedge 2$ **for** a
proof –
have $v \wedge 2 \leq v \wedge 2 + \text{int } a$
by *simp*
also have $\dots = u \wedge 2$
using *that* **by** *simp*
finally have $|v| \leq |u|$
using *abs-le-square-iff* **by** *blast*
thus $v' \leq u'$
by *(simp add: v'-def u'-def)*

from *that* **have** $u \wedge 2 = v \wedge 2 + \text{int } a$
by *simp*
hence $\text{nat } |u \wedge 2| = \text{nat } |v \wedge 2 + \text{int } a|$
by *(simp only:)*
also have $\text{nat } |u \wedge 2| = u' \wedge 2$
by *(simp add: u'-def flip: nat-power-eq)*
also have $\text{nat } |v \wedge 2 + \text{int } a| = v' \wedge 2 + a$
by *(simp add: nat-add-distrib v'-def flip: nat-power-eq)*
finally show $a = u' \wedge 2 - v' \wedge 2$
by *simp*
qed

have *eq*: $x = 2 * u' * v' \wedge y = u'^2 - v'^2 \vee x = u'^2 - v'^2 \wedge y = 2 * u' * v'$
and $v' \leq u'$
using *uv(3) **[of x] **[of y] ***[of x] ***[of y]* **by** *blast+*
moreover have *coprime* $u' v'$
using $\langle \text{coprime } u v \rangle$
by *(auto simp: u'-def v'-def)*
moreover have $\neg(\text{odd } u' \wedge \text{odd } v')$
using *uv(2)* **by** *(auto simp: u'-def v'-def)*
moreover have $v' \neq u' v' > 0$

```

    using ⟨coprime u' v'⟩ eq assms by (auto simp: prim-pyth-triple-def)
  moreover from this have v' < u'
    using ⟨v' ≤ u'⟩ by auto
  moreover have z = u2 + v2
  proof -
    from assms have z2 = x2 + y2
      by (simp add: prim-pyth-triple-def)
    also have ... = (2 * u' * v')2 + (u'2 - v'2)2
      using eq by (auto simp: add-ac)
    also have ... = (u'2 + v'2)2
      by (intro prim-pyth-triple-aux) fact
    finally show ?thesis by simp
  qed
  ultimately show ?thesis using that[of v' u'] by metis
  qed

theorem prim-pyth-triple-iff:
  prim-pyth-triple x y z ↔
    (∃ u v. 0 < v ∧ v < u ∧ coprime u v ∧ ¬(odd u ∧ odd v) ∧
      (x = 2 * u * v ∧ y = u2 - v2 ∨ x = u2 - v2 ∧ y = 2 * u * v) ∧ z =
      u2 + v2)
  (is - ↔ ?rhs)
proof
  assume prim-pyth-triple x y z
  from prim-pyth-tripleE[OF this] show ?rhs by metis
next
  assume ?rhs
  then obtain u v where uv: 0 < v v < u coprime u v ¬(odd u ∧ odd v) z = u2
  + v2 and
    eq: x = 2 * u * v ∧ y = u2 - v2 ∨ x = u2 - v2 ∧ y = 2 * u
  * v
    by metis
  thus prim-pyth-triple x y z
    using uv prim-pyth-tripleI1[OF uv(1-4)] prim-pyth-tripleI2[OF uv(1-4)]
  uv(5) eq by auto
  qed
end

theory Gaussian-Integers-Everything
imports
  Gaussian-Integers
  Gaussian-Integers-Test
  Gaussian-Integers-Sums-Of-Two-Squares
  Gaussian-Integers-Pythagorean-Triples
begin

end

```

References

- [1] J. Stillwell. *The Gaussian integers*, pages 101–116. Springer New York, New York, NY, 2003.