Gaussian Integers

Manuel Eberl

March 17, 2025

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

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 $x\ y$ :: $'a$ :: *factorial-semiring*
  assumes $x \neq 0$ $y \neq 0$
  shows *coprime* $x\ y \longleftrightarrow$ *prime-factors* $x \cap$ *prime-factors* $y = \{\}$
⟨*proof*⟩

lemma *product-dvd-irreducibleD*:
  fixes $a\ b\ x$ :: $'a$ :: *algebraic-semidom*
  assumes *irreducible* x
  assumes $a * b$ *dvd* x
  shows a *dvd* $1 \vee b$ *dvd* 1
⟨*proof*⟩

lemma *prime-elem-mult-dvdI*:
  assumes *prime-elem* p p *dvd* c b *dvd* c $\neg p$ *dvd* b
  shows $p * b$ *dvd* c
⟨*proof*⟩

lemma *prime-elem-power-mult-dvdI*:
  fixes p :: $'a$ :: *algebraic-semidom*
  assumes *prime-elem* p $p \mathbin{\widehat{}} n$ *dvd* c b *dvd* c $\neg p$ *dvd* b
  shows $p \mathbin{\widehat{}} n * b$ *dvd* c
⟨*proof*⟩

lemma *prime-mod-4-cases*:
  fixes p :: *nat*
  assumes *prime* p
  shows $p = 2 \vee [p = 1]\ (mod\ 4) \vee [p = 3]\ (mod\ 4)$
⟨*proof*⟩

lemma *of-nat-prod-mset*: *of-nat* (*prod-mset* A) = *prod-mset* (*image-mset* *of-nat* A)
  ⟨*proof*⟩

lemma *multiplicity-0-left* [*simp*]: *multiplicity* $0\ x = 0$
  ⟨*proof*⟩

lemma *is-unit-power* [*intro*]: *is-unit* $x \Longrightarrow$ *is-unit* $(x \mathbin{\widehat{}} n)$
  ⟨*proof*⟩

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

assumes $n > 0$
shows a ^ n *dvd* b ^ $n \longleftrightarrow a$ *dvd* b
⟨*proof*⟩

lemma *multiplicity-power-power*:
fixes p :: *'a* :: {*factorial-semiring*, *algebraic-semidom*}
assumes $n > 0$
shows *multiplicity* (p ^ n) (x ^ n) = *multiplicity* p x
⟨*proof*⟩

lemma *even-square-cong-4-int*:
‹$[x^2 = 0]$ (*mod 4*)› **if** ‹*even* x› **for** x :: *int*
⟨*proof*⟩

lemma *even-square-cong-4-nat*:
‹$[x^2 = 0]$ (*mod 4*)› **if** ‹*even* x› **for** x :: *nat*
⟨*proof*⟩

lemma *odd-square-cong-4-int*:
‹$[x^2 = 1]$ (*mod 4*)› **if** ‹*odd* x› **for** x :: *int*
⟨*proof*⟩

lemma *odd-square-cong-4-nat*:
‹$[x^2 = 1]$ (*mod 4*)› **if** ‹*odd* x› **for** x :: *nat*
⟨*proof*⟩

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. *is-unit* u $\wedge$ *is-nth-power* n ($u * x$))

lemma *is-nth-power-upto-unit-base*: *is-nth-power* n $x \Longrightarrow$ *is-nth-power-upto-unit* n x
⟨*proof*⟩

lemma *is-nth-power-upto-unitI*:
assumes *normalize* (x ^ n) = *normalize* y
shows *is-nth-power-upto-unit* n y
⟨*proof*⟩

lemma *is-nth-power-upto-unit-conv-multiplicity*:
fixes x :: *'a* :: *factorial-semiring*
assumes $n > 0$
shows *is-nth-power-upto-unit* n $x \longleftrightarrow$ ($\forall$ p. *prime* $p \longrightarrow n$ *dvd* *multiplicity* p x)
⟨*proof*⟩

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

lemma *is-nth-power-upto-unit-unit* [*simp*, *intro*]:
assumes *is-unit x*
shows *is-nth-power-upto-unit n x*
⟨*proof*⟩

lemma *is-nth-power-upto-unit-1-left* [*simp*, *intro*]: *is-nth-power-upto-unit 1 x*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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 $\mathrm{i}_{\mathbb{Z}}$:

primcorec *gauss-i* **where**
ReZ gauss-i = *0*
| *ImZ gauss-i* = *1*

lemma *gauss-int-eq-iff*: *x* = *y* ⟷ *ReZ x* = *ReZ y* ∧ *ImZ x* = *ImZ y*
⟨*proof*⟩

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

primcorec *gauss2complex* **where**
Re (*gauss2complex z*) = *of-int* (*ReZ z*)
| *Im* (*gauss2complex z*) = *of-int* (*ImZ z*)

declare [[*coercion gauss2complex*]]

lemma *gauss2complex-eq-iff* [*simp*]: *gauss2complex z* = *gauss2complex u* ⟷ *z* = *u*
⟨*proof*⟩

Gaussian integers also have conjugates, just like complex numbers:

primcorec *gauss-cnj* **where**
ReZ (*gauss-cnj z*) = *ReZ z*
| *ImZ* (*gauss-cnj z*) = −*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**
ReZ zero-gauss-int = *0*
| *ImZ zero-gauss-int* = *0*

primcorec *one-gauss-int* **where**
ReZ one-gauss-int = *1*
| *ImZ one-gauss-int* = *0*

primcorec *uminus-gauss-int* **where**
ReZ (*uminus-gauss-int x*) = −*ReZ x*
| *ImZ* (*uminus-gauss-int x*) = −*ImZ x*

primcorec *plus-gauss-int* **where**
ReZ (*plus-gauss-int x y*) = *ReZ x* + *ReZ y*
| *ImZ* (*plus-gauss-int x y*) = *ImZ x* + *ImZ y*

primcorec *minus-gauss-int* **where**
ReZ (*minus-gauss-int x y*) = *ReZ x* − *ReZ y*
| *ImZ* (*minus-gauss-int x y*) = *ImZ x* − *ImZ y*

primcorec *times-gauss-int* **where**
ReZ (*times-gauss-int x y*) = *ReZ x* * *ReZ y* − *ImZ x* * *ImZ y*
| *ImZ* (*times-gauss-int x y*) = *ReZ x* * *ImZ y* + *ImZ x* * *ReZ y*

instance
⟨*proof*⟩

end

lemma *gauss-i-times-i* [*simp*]: $\mathrm{i}_{\mathbb{Z}}$ * $\mathrm{i}_{\mathbb{Z}}$ = (−*1* :: *gauss-int*)
and *gauss-cnj-i* [*simp*]: *gauss-cnj* $\mathrm{i}_{\mathbb{Z}}$ = −$\mathrm{i}_{\mathbb{Z}}$
⟨*proof*⟩

lemma *gauss-cnj-eq-0-iff* [*simp*]: *gauss-cnj z = 0* $\longleftrightarrow$ *z = 0*
⟨*proof*⟩

lemma *gauss-cnj-eq-self*: *Im z = 0* $\Longrightarrow$ *gauss-cnj z = z*
and *gauss-cnj-eq-minus-self*: *Re z = 0* $\Longrightarrow$ *gauss-cnj z* = −*z*
⟨*proof*⟩

lemma *ReZ-of-nat* [*simp*]: *ReZ* (*of-nat n*) = *of-nat n*
and *ImZ-of-nat* [*simp*]: *ImZ* (*of-nat n*) = *0*
⟨*proof*⟩

lemma *ReZ-of-int* [*simp*]: *ReZ* (*of-int n*) = *n*
and *ImZ-of-int* [*simp*]: *ImZ* (*of-int n*) = *0*
⟨*proof*⟩

lemma *ReZ-numeral* [*simp*]: *ReZ* (*numeral n*) = *numeral n*
and *ImZ-numeral* [*simp*]: *ImZ* (*numeral n*) = *0*
⟨*proof*⟩

lemma *gauss2complex-0* [*simp*]: *gauss2complex 0 = 0*
and *gauss2complex-1* [*simp*]: *gauss2complex 1 = 1*
and *gauss2complex-i* [*simp*]: *gauss2complex* $\mathrm{i}_{\mathbb{Z}}$ = 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*)
⟨*proof*⟩

lemma *gauss2complex-of-nat* [*simp*]: *gauss2complex* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

lemma *gauss2complex-eq-0-iff* [*simp*]: *gauss2complex x = 0* $\longleftrightarrow$ *x = 0*
and *gauss2complex-eq-1-iff* [*simp*]: *gauss2complex x = 1* $\longleftrightarrow$ *x = 1*
and *zero-eq-gauss2complex-iff* [*simp*]: *0 = gauss2complex x* $\longleftrightarrow$ *x = 0*
and *one-eq-gauss2complex-iff* [*simp*]: *1 = gauss2complex x* $\longleftrightarrow$ *x = 1*
⟨*proof*⟩

lemma *gauss-i-times-gauss-i-times* [*simp*]: $\mathrm{i}_{\mathbb{Z}}$ ∗ ($\mathrm{i}_{\mathbb{Z}}$ ∗ *x*) = (−*x* :: *gauss-int*)
⟨*proof*⟩

lemma *gauss-i-neq-0* [*simp*]: $\mathrm{i}_{\mathbb{Z}} \neq 0$ $0 \neq \mathrm{i}_{\mathbb{Z}}$
and *gauss-i-neq-1* [*simp*]: $\mathrm{i}_{\mathbb{Z}} \neq 1$ $1 \neq \mathrm{i}_{\mathbb{Z}}$
and *gauss-i-neq-of-nat* [*simp*]: $\mathrm{i}_{\mathbb{Z}} \neq$ *of-nat n of-nat n* $\neq \mathrm{i}_{\mathbb{Z}}$
and *gauss-i-neq-of-int* [*simp*]: $\mathrm{i}_{\mathbb{Z}} \neq$ *of-int n of-int n* $\neq \mathrm{i}_{\mathbb{Z}}$

and *gauss-i-neq-numeral* [*simp*]: $\mathrm{i}_{\mathbb{Z}} \neq$ *numeral m numeral m* $\neq \mathrm{i}_{\mathbb{Z}}$
⟨*proof*⟩

lemma *gauss-cnj-0* [*simp*]: *gauss-cnj 0* = *0*
and *gauss-cnj-1* [*simp*]: *gauss-cnj 1* = *1*
and *gauss-cnj-cnj* [*simp*]: *gauss-cnj* (*gauss-cnj z*) = *z*
and *gauss-cnj-uminus* [*simp*]: *gauss-cnj* (−*a*) = −*gauss-cnj a*
and *gauss-cnj-add* [*simp*]: *gauss-cnj* (*a* + *b*) = *gauss-cnj a* + *gauss-cnj b*
and *gauss-cnj-diff* [*simp*]: *gauss-cnj* (*a* − *b*) = *gauss-cnj a* − *gauss-cnj b*
and *gauss-cnj-mult* [*simp*]: *gauss-cnj* (*a* ∗ *b*) = *gauss-cnj a* ∗ *gauss-cnj b*
and *gauss-cnj-of-nat* [*simp*]: *gauss-cnj* (*of-nat n1*) = *of-nat n1*
and *gauss-cnj-of-int* [*simp*]: *gauss-cnj* (*of-int n2*) = *of-int n2*
and *gauss-cnj-numeral* [*simp*]: *gauss-cnj* (*numeral n3*) = *numeral n3*
⟨*proof*⟩

lemma *gauss-cnj-power* [*simp*]: *gauss-cnj* (*a* ^ *n*) = *gauss-cnj a* ^ *n*
⟨*proof*⟩

lemma *gauss-cnj-sum* [*simp*]: *gauss-cnj* (*sum f A*) = ($\sum$ *x*∈*A*. *gauss-cnj* (*f x*))
⟨*proof*⟩

lemma *gauss-cnj-prod* [*simp*]: *gauss-cnj* (*prod f A*) = ($\prod$ *x*∈*A*. *gauss-cnj* (*f x*))
⟨*proof*⟩

lemma *of-nat-dvd-of-nat*:
assumes *a dvd b*
shows *of-nat a dvd* (*of-nat b* :: *'a* :: *comm-semiring-1*)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *of-int-dvd-of-int-gauss-int-iff*:
(*of-int m* :: *gauss-int*) *dvd of-int n* ⟷ *m dvd n*
⟨*proof*⟩

lemma *of-nat-dvd-of-nat-gauss-int-iff*:
(*of-nat m* :: *gauss-int*) *dvd of-nat n* ⟷ *m dvd n*
⟨*proof*⟩

lemma *gauss-cnj-dvd*:
assumes *a dvd b*
shows *gauss-cnj a dvd gauss-cnj b*
⟨*proof*⟩

lemma *gauss-cnj-dvd-iff*: *gauss-cnj a dvd gauss-cnj b* ⟷ *a dvd b*
⟨*proof*⟩

lemma *gauss-cnj-dvd-left-iff*: *gauss-cnj a dvd b* ⟷ *a dvd gauss-cnj b*
⟨*proof*⟩

lemma *gauss-cnj-dvd-right-iff*: *a dvd gauss-cnj b* ⟷ *gauss-cnj a dvd b*
⟨*proof*⟩

instance *gauss-int* :: *idom*
⟨*proof*⟩

instance *gauss-int* :: *ring-char-0*
⟨*proof*⟩

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 = $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 0 (−*1*) = $-\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 1 0 = *1*
Gauss-Int 1 1 = *1* + $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 1 (−*1*) = *1* − $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*1*) *0* = −*1*
Gauss-Int (−*1*) *1* = −*1* + $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*1*) (−*1*) = −*1* − $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (*numeral b*) *0* = *numeral b*
Gauss-Int (−*numeral b*) *0* = −*numeral b*
Gauss-Int (*numeral b*) *1* = *numeral b* + $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*numeral b*) *1* = −*numeral b* + $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (*numeral b*) (−*1*) = *numeral b* − $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*numeral b*) (−*1*) = −*numeral b* − $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 0 (*numeral b*) = *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 0 (−*numeral b*) = −*numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 1 (*numeral b*) = *1* + *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int 1 (−*numeral b*) = *1* − *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*1*) (*numeral b*) = −*1* + *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (−*1*) (−*numeral b*) = −*1* − *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$
Gauss-Int (*numeral a*) (*numeral b*) = *numeral a* + *numeral b* ∗ $\mathrm{i}_{\mathbb{Z}}$

$Gauss\text{-}Int\ (numeral\ a)\ (-numeral\ b) = numeral\ a - numeral\ b * i_{\mathbb{Z}}$
$Gauss\text{-}Int\ (-numeral\ a)\ (numeral\ b) = -numeral\ a + numeral\ b * i_{\mathbb{Z}}$
$Gauss\text{-}Int\ (-numeral\ a)\ (-numeral\ b) = -numeral\ a - numeral\ b * i_{\mathbb{Z}}$
⟨*proof*⟩

value $i_{\mathbb{Z}}$ ^ *3*
value *2* ∗ (*3* + $i_{\mathbb{Z}}$)
value (*2* + $i_{\mathbb{Z}}$) ∗ (*2* − $i_{\mathbb{Z}}$)

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* $i_{\mathbb{Z}}$ = *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′*)
⟨*proof*⟩

lemma *gauss-int-norm-uminus* [*simp*]: *gauss-int-norm* (−*z*) = *gauss-int-norm z*
⟨*proof*⟩

lemma *gauss-int-norm-eq-0-iff* [*simp*]: *gauss-int-norm z* = *0* ⟷ *z* = *0*
⟨*proof*⟩

lemma *gauss-int-norm-pos-iff* [*simp*]: *gauss-int-norm z* > *0* ⟷ *z* ≠ *0*
⟨*proof*⟩

lemma *real-gauss-int-norm*: *real* (*gauss-int-norm z*) = *norm* (*gauss2complex z*) ^ *2*
⟨*proof*⟩

lemma *gauss-int-norm-mult*: *gauss-int-norm* (*z* ∗ *u*) = *gauss-int-norm z* ∗ *gauss-int-norm u*
⟨*proof*⟩

lemma *self-mult-gauss-cnj*: *z* ∗ *gauss-cnj z* = *of-nat* (*gauss-int-norm z*)
⟨*proof*⟩

lemma *gauss-cnj-mult-self*: *gauss-cnj z* $*$ *z* = *of-nat* (*gauss-int-norm z*)
⟨*proof*⟩

lemma *self-plus-gauss-cnj*: *z* + *gauss-cnj z* = *of-int* (*2* $*$ *ReZ z*)
and *self-minus-gauss-cnj*: *z* − *gauss-cnj z* = *of-int* (*2* $*$ *ImZ z*) $*$ $\mathrm{i}_{\mathbb{Z}}$
⟨*proof*⟩

lemma *gauss-int-norm-dvd-mono*:
assumes *a dvd b*
shows *gauss-int-norm a dvd gauss-int-norm b*
⟨*proof*⟩

lemma *gauss-int-norm-power*: *gauss-int-norm* (*x* ^ *n*) = *gauss-int-norm x* ^ *n*
⟨*proof*⟩

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

lemma *is-unit-gauss-int-iff*: *x dvd 1* ⟷ *x* ∈ {*1*, −*1*, $\mathrm{i}_{\mathbb{Z}}$, $-\mathrm{i}_{\mathbb{Z}}$ :: *gauss-int*}
and *is-unit-gauss-int-iff′*: *x dvd 1* ⟷ *gauss-int-norm x* = *1*
⟨*proof*⟩

lemma *is-unit-gauss-i* [*simp*, *intro*]: (*gauss-i* :: *gauss-int*) *dvd 1*
⟨*proof*⟩

lemma *gauss-int-norm-eq-Suc-0-iff*: *gauss-int-norm x* = *Suc 0* ⟷ *x dvd 1*
⟨*proof*⟩

lemma *is-unit-gauss-cnj* [*intro*]: *z dvd 1* ⟹ *gauss-cnj z dvd 1*
⟨*proof*⟩

lemma *is-unit-gauss-cnj-iff* [*simp*]: *gauss-cnj z dvd 1* ⟷ *z dvd 1*
⟨*proof*⟩

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* ⇒ *gauss-int* **where**
ReZ (*round-complex z*) = *round* (*Re z*)
| *ImZ* (*round-complex z*) = *round* (*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*)) ^ *2* ≤ *1* / *2*
⟨*proof*⟩

lemma *dist-round-complex-le*: *dist z* (*gauss2complex* (*round-complex z*)) ≤ *sqrt 2 / 2*
⟨*proof*⟩

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 ±i, each association class (other than *0*) has four representatives, one in each quadrant. We simply define the on 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*⟩

end

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

definition *unit-factor-gauss-int* :: *gauss-int* ⇒ *gauss-int* **where**
  unit-factor-gauss-int z =
    (*if z* = *0 then 0 else*
     if ImZ z ≥ *0* ∧ *ReZ z* > *0 then 1*
     else if ReZ z ≤ *0* ∧ *ImZ z* > *0 then* $\mathrm{i}_{\mathbb{Z}}$
     else if ImZ z ≤ *0* ∧ *ReZ z* < *0 then* −*1*
     else −$\mathrm{i}_{\mathbb{Z}}$)

instance ⟨*proof*⟩

end

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

definition *normalize-gauss-int* :: *gauss-int* ⇒ *gauss-int* **where**
  normalize-gauss-int z =

(*if* $z = 0$ *then* 0 *else*
if $ImZ\ z \geq 0 \wedge ReZ\ z > 0$ *then* z
else if $ReZ\ z \leq 0 \wedge ImZ\ z > 0$ *then* $-i_{\mathbb{Z}} * z$
else if $ImZ\ z \leq 0 \wedge ReZ\ z < 0$ *then* $-z$
else $i_{\mathbb{Z}} * z$)

instance ⟨*proof*⟩

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′*
⟨*proof*⟩

lemma *normalize-gauss-i* [*simp*]: *normalize* $i_{\mathbb{Z}} = 1$
⟨*proof*⟩

lemma *gauss-int-norm-normalize* [*simp*]: *gauss-int-norm* (*normalize x*) = *gauss-int-norm x*
⟨*proof*⟩

lemma *normalized-gauss-int*:
assumes *normalize* $z = z$
shows $ReZ\ z \geq 0$ $ImZ\ z \geq 0$
⟨*proof*⟩

lemma *normalized-gauss-int′*:
assumes *normalize* $z = z$ $z \neq 0$
shows $ReZ\ z > 0$ $ImZ\ z \geq 0$
⟨*proof*⟩

lemma *normalized-gauss-int-iff*:
normalize $z = z \longleftrightarrow z = 0 \vee ReZ\ z > 0 \wedge ImZ\ z \geq 0$
⟨*proof*⟩

instantiation *gauss-int* :: *idom-modulo*
begin

definition *modulo-gauss-int* :: *gauss-int* ⇒ *gauss-int* ⇒ *gauss-int* **where**
modulo-gauss-int a b = *a* − *a div b* ∗ *b*

instance ⟨*proof*⟩

end

lemma *gauss-int-norm-mod-less-aux*:
  assumes [*simp*]: $b \neq 0$
  shows $2 * gauss\text{-}int\text{-}norm\ (a\ mod\ b) \leq gauss\text{-}int\text{-}norm\ b$
⟨*proof*⟩

lemma *gauss-int-norm-mod-less*:
  assumes [*simp*]: $b \neq 0$
  shows $gauss\text{-}int\text{-}norm\ (a\ mod\ b) < gauss\text{-}int\text{-}norm\ b$
⟨*proof*⟩

lemma *gauss-int-norm-dvd-imp-le*:
  assumes $b \neq 0$
  shows $gauss\text{-}int\text{-}norm\ a \leq gauss\text{-}int\text{-}norm\ (a * b)$
⟨*proof*⟩

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

definition *euclidean-size-gauss-int* :: *gauss-int* ⇒ *nat* **where**
  [*simp*]: *euclidean-size-gauss-int* = *gauss-int-norm*

instance ⟨*proof*⟩

end

instance *gauss-int* :: *normalization-euclidean-semiring* ⟨*proof*⟩

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

definition *gcd-gauss-int* :: *gauss-int* ⇒ *gauss-int* ⇒ *gauss-int* **where**
  gcd-gauss-int ≡ *normalization-euclidean-semiring-class.gcd*
definition *lcm-gauss-int* :: *gauss-int* ⇒ *gauss-int* ⇒ *gauss-int* **where**
  lcm-gauss-int ≡ *normalization-euclidean-semiring-class.lcm*
definition *Gcd-gauss-int* :: *gauss-int set* ⇒ *gauss-int* **where**
  Gcd-gauss-int ≡ *normalization-euclidean-semiring-class.Gcd*
definition *Lcm-gauss-int* :: *gauss-int set* ⇒ *gauss-int* **where**
  Lcm-gauss-int ≡ *normalization-euclidean-semiring-class.Lcm*

instance
  ⟨*proof*⟩

end

lemma *gcd-gauss-cnj*: *gcd* (*gauss-cnj x*) (*gauss-cnj y*) = *normalize* (*gauss-cnj* (*gcd x y*))
⟨*proof*⟩

lemma *gcd-gauss-cnj-left*: *gcd* (*gauss-cnj x*) *y* = *normalize* (*gauss-cnj* (*gcd x* (*gauss-cnj*

y)))
⟨*proof*⟩

lemma *gcd-gauss-cnj-right*: *gcd x* (*gauss-cnj y*) = *normalize* (*gauss-cnj* (*gcd* (*gauss-cnj x*) *y*))
⟨*proof*⟩

lemma *multiplicity-gauss-cnj*: *multiplicity* (*gauss-cnj a*) (*gauss-cnj b*) = *multiplicity a b*
⟨*proof*⟩

lemma *multiplicity-gauss-int-of-nat*:
multiplicity (*of-nat a*) (*of-nat b* :: *gauss-int*) = *multiplicity a b*
⟨*proof*⟩

lemma *gauss-int-dvd-same-norm-imp-associated*:
assumes *z1 dvd z2 gauss-int-norm z1* = *gauss-int-norm z2*
shows *normalize z1* = *normalize z2*
⟨*proof*⟩

lemma *gcd-of-int-gauss-int*: *gcd* (*of-int a* :: *gauss-int*) (*of-int b*) = *of-int* (*gcd a b*)
⟨*proof*⟩

lemma *coprime-of-int-gauss-int*: *coprime* (*of-int a* :: *gauss-int*) (*of-int b*) = *coprime a b*
⟨*proof*⟩

lemma *gcd-of-nat-gauss-int*: *gcd* (*of-nat a* :: *gauss-int*) (*of-nat b*) = *of-nat* (*gcd a b*)
⟨*proof*⟩

lemma *coprime-of-nat-gauss-int*: *coprime* (*of-nat a* :: *gauss-int*) (*of-nat b*) = *coprime a b*
⟨*proof*⟩

lemma *gauss-cnj-dvd-self-iff*: *gauss-cnj z dvd z* ⟷ *ReZ z* = *0* ∨ *ImZ z* = *0* ∨ |*ReZ z*| = |*ImZ z*|
⟨*proof*⟩

lemma *self-dvd-gauss-cnj-iff*: *z dvd gauss-cnj z* ⟷ *ReZ z* = *0* ∨ *ImZ z* = *0* ∨ |*ReZ z*| = |*ImZ z*|
⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

lemma *prime-elem-gauss-cnj-iff* [*simp*]: *prime-elem* (*gauss-cnj z*) ⟷ *prime-elem z*
  ⟨*proof*⟩

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*: $2 = -\mathrm{i}_{\mathbb{Z}} * (1 + \mathrm{i}_{\mathbb{Z}})$ ^ 2
  ⟨*proof*⟩

lemma *prime-elem-one-plus-i-gauss-int*: *prime-elem* (1 + $\mathrm{i}_{\mathbb{Z}}$)
  ⟨*proof*⟩

lemma *prime-one-plus-i-gauss-int*: *prime* (1 + $\mathrm{i}_{\mathbb{Z}}$)
  ⟨*proof*⟩

lemma *prime-factorization-2-gauss-int*:
  prime-factorization (*2* :: *gauss-int*) = {#*1* + $\mathrm{i}_{\mathbb{Z}}$, *1* + $\mathrm{i}_{\mathbb{Z}}$#}
⟨*proof*⟩

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* ≠ *3*] (*mod 4*)
⟨*proof*⟩

lemma *prime-elem-gauss-int-of-nat*:
  fixes *n* :: *nat*
  assumes *prime*: *prime n* **and** [*n* = *3*] (*mod 4*)
  shows *prime-elem* (*of-nat n* :: *gauss-int*)
⟨*proof*⟩

theorem *prime-gauss-int-of-nat*:
  fixes *n* :: *nat*
  assumes *prime*: *prime n* **and** [*n* = *3*] (*mod 4*)
  shows *prime* (*of-nat n* :: *gauss-int*)
  ⟨*proof*⟩

1.6.3 Non-inert primes

Any 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* :: *nat* **where** *x* ≤ (*n* − *1*) *div 2* **and** [*x* ^ *2* + *1* = *0*] (*mod n*)
⟨*proof*⟩

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

locale *noninert-gauss-int-prime* =
  fixes *p* :: *nat*
  assumes *prime-p*: *prime p* **and** *cong-1-p*: [*p* = *1*] (*mod 4*)
begin

lemma *p-gt-2*: *p* > *2* **and** *odd-p*: *odd p*
⟨*proof*⟩

-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* ≤ (*p* − *1*) *div 2* [*x* ^ *2* + *1* = *0*] (*mod p*)
⟨*proof*⟩

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

lemma *not-prime*: ¬*prime-elem* (*of-nat p* :: *gauss-int*)
⟨*proof*⟩

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*⟩

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*⟩

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 = $i_{\mathbb{Z}}$ *∗ gauss-cnj q1*
ReZ q1 > 0 ImZ q1 > 0 ReZ q1 > 0 ImZ q2 > 0
⟨*proof*⟩

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 :: nat*
assumes *prime p [p = 1] (mod 4)*
shows *∃ z. gauss-int-norm z = p ∧ ReZ z > 0 ∧ ImZ z > 0*
⟨*proof*⟩

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 :: gauss-int*
assumes *prime x*
obtains
(one-plus-i) x = 1 + $i_{\mathbb{Z}}$
| *(cong-3-mod-4) p* **where** *x = 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)*
ReZ x > 0 ImZ x > 0 ReZ x ≠ ImZ x
⟨*proof*⟩

lemma *prime-gauss-int-norm-squareD*:
fixes *z :: gauss-int*
assumes *prime z gauss-int-norm z = p ^ 2*
shows *prime p ∧ z = of-nat p*
⟨*proof*⟩

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*⟩

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* :: *gauss-int*
defines $n \equiv$ *gauss-int-norm z*
shows *prime-elem z* $\longleftrightarrow$ *prime n* $\vee$ $(\exists p.\ n = p$ ^ $2 \wedge$ *prime p* $\wedge$ $[p = 3]$ $(mod\ 4))$
⟨*proof*⟩

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 $1 + \mathrm{i}_{\mathbb{Z}}$ in an integer c is simply twice the $\mathbb{Z}$-multiplicity of 2 in c:

lemma *multiplicity-prime-1-plus-i-aux*: *multiplicity* $(1 + \mathrm{i}_{\mathbb{Z}})$ (*of-nat c*) $= 2 *$ *multiplicity 2 c*
⟨*proof*⟩

Tha 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*⟩

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* $p \neq$ *ReZ p*
shows *multiplicity p* (*of-nat c*) $=$ *multiplicity* (*gauss-int-norm p*) *c*
⟨*proof*⟩

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-cnj p*) *z*
⟨*proof*⟩

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

lemma *multiplicity-prime-1-plus-i*: *multiplicity* (*1* + $\mathrm{i}_{\mathbb{Z}}$) *z* = *multiplicity 2* (*gauss-int-norm z*)
⟨*proof*⟩

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* (*ReZ z*) (*ImZ z*) **and** *odd* (*gauss-int-norm z*)
  shows *coprime z* (*gauss-cnj z*)
⟨*proof*⟩

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* :: *nat*
  assumes *prime p* [*p* = *1*] (*mod 4*) *x* ^ *2* + *y* ^ *2* = *p*
  shows *x* > *0* ∧ *y* > *0* ∧ *x* ≠ *y*
⟨*proof*⟩

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* :: *nat*
  assumes *prime p* [*p* = *1*] (*mod 4*)
  shows ∃!(*x*,*y*). *x* ≤ *y* ∧ *x* ^ *2* + *y* ^ *2* = *p*
⟨*proof*⟩

lemma *two-sum-of-squares-nat-iff*: (*x* :: *nat*) ^ *2* + *y* ^ *2* = *2* ⟷ *x* = *1* ∧ *y* = *1*
⟨*proof*⟩

lemma *prime-sum-of-squares-unique*:
  fixes *p* :: *nat*
  assumes *prime p p* = *2* ∨ [*p* = *1*] (*mod 4*)

shows $\exists!(x,y).\ x \leq y \wedge x \mathbin{\hat{}} 2 + y \mathbin{\hat{}} 2 = p$
⟨*proof*⟩

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* ⇒ *nat* × *nat* **where**
prime-square-sum-nat-decomp p =
(*if prime p* ∧ (*p* = *2* ∨ [*p* = *1*] (*mod 4*))
then THE (*x*,*y*). *x* ≤ *y* ∧ *x* ^ *2* + *y* ^ *2* = *p else* (*0*, *0*))

lemma *prime-square-sum-nat-decomp-eqI*:
assumes *prime p x* ^ *2* + *y* ^ *2* = *p x* ≤ *y*
shows *prime-square-sum-nat-decomp p* = (*x*, *y*)
⟨*proof*⟩

lemma *prime-square-sum-nat-decomp-correct*:
assumes *prime p p* = *2* ∨ [*p* = *1*] (*mod 4*)
defines *z* ≡ *prime-square-sum-nat-decomp p*
shows *fst z* ^ *2* + *snd z* ^ *2* = *p fst z* ≤ *snd z*
⟨*proof*⟩

lemma *sum-of-squares-nat-bound*:
fixes *x y n* :: *nat*
assumes *x* ^ *2* + *y* ^ *2* = *n*
shows *x* ≤ *n*
⟨*proof*⟩

lemma *sum-of-squares-nat-bound'*:
fixes *x y n* :: *nat*
assumes *x* ^ *2* + *y* ^ *2* = *n*
shows *y* ≤ *n*
⟨*proof*⟩

lemma *is-singleton-conv-Ex1*:
is-singleton A ⟷ (∃!*x*. *x* ∈ *A*)
⟨*proof*⟩

lemma *the-elemI*:
assumes *is-singleton A*
shows *the-elem A* ∈ *A*
⟨*proof*⟩

lemma *prime-square-sum-nat-decomp-code-aux*:
assumes *prime p p* = *2* ∨ [*p* = *1*] (*mod 4*)
defines *z* ≡ *the-elem* (*Set.filter* (λ(*x*,*y*). *x* ^ *2* + *y* ^ *2* = *p*) (*SIGMA x*:{*0*..*p*}.
{*x*..*p*}))
shows *prime-square-sum-nat-decomp p* = *z*
⟨*proof*⟩

lemma *prime-square-sum-nat-decomp-code* [*code*]:
prime-square-sum-nat-decomp p =
(*if prime p* ∧ (*p* = *2* ∨ [*p* = *1*] (*mod 4*))
then the-elem (*Set.filter* (λ(*x*,*y*). *x* ^*2* + *y* ^*2* = *p*) (*SIGMA x*:{*0*..*p*}. {*x*..*p*}))
else (*0*, *0*))
⟨*proof*⟩

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 ℤ-prime *p*:

definition *factor-gauss-int-prime-nat* :: *nat* ⇒ *gauss-int list* **where**
factor-gauss-int-prime-nat p =
(*if p* = *2 then* [*1* + $\mathrm{i}_{\mathbb{Z}}$]
else if [*p* = *3*] (*mod 4*) *then* [*of-nat p*]
else case prime-square-sum-nat-decomp p of
(*x*, *y*) ⇒ [*of-nat x* + $\mathrm{i}_{\mathbb{Z}}$ ∗ *of-nat y*, *of-nat y* + $\mathrm{i}_{\mathbb{Z}}$ ∗ *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*)
⟨*proof*⟩

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*⟩

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 (λ*p*. *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*⟩

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* ∗ $i_{\mathbb{Z}}$) *mod* (*321* + *654* ∗ $i_{\mathbb{Z}}$)
value *prime-factors* (*1* + *3* ∗ $i_{\mathbb{Z}}$)
value *prime-factors* (*4830* + *1610* ∗ $i_{\mathbb{Z}}$)

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* ∈ *range gauss-int-norm*
⟨*proof*⟩

lemma *sum-of-2-squares-nat-gauss-int-norm* [*intro*]: *sum-of-2-squares-nat* (*gauss-int-norm z*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *sum-of-2-squares-nat-power* [*intro*]:
  assumes *sum-of-2-squares-nat m*
  shows *sum-of-2-squares-nat* (*m* ^ *n*)
  ⟨*proof*⟩

lemma *sum-of-2-squares-nat-prod* [*intro*]:
  assumes ⋀*x*. *x* ∈ *A* ⟹ *sum-of-2-squares-nat* (*f x*)
  shows *sum-of-2-squares-nat* (∏*x*∈*A*. *f x*)
  ⟨*proof*⟩

lemma *sum-of-2-squares-nat-prod-mset* [*intro*]:
  assumes ⋀*x*. *x* ∈# *A* ⟹ *sum-of-2-squares-nat x*
  shows *sum-of-2-squares-nat* (*prod-mset A*)
  ⟨*proof*⟩

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*⟩

lemma *sum-of-2-squares-nat-sufficient*:
  fixes *n* :: *nat*
  assumes *n* > *0*
  assumes ⋀*p*. *p* ∈ *prime-factors n* ⟹ [*p* = *3*] (*mod 4*) ⟹ *even* (*multiplicity p n*)
  shows *sum-of-2-squares-nat n*
⟨*proof*⟩

theorem *sum-of-2-squares-nat-iff*:
  sum-of-2-squares-nat n ⟷
    n = *0* ∨ (∀ *p*∈*prime-factors n*. [*p* = *3*] (*mod 4*) ⟶ *even* (*multiplicity p n*))
  ⟨*proof*⟩

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* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
  prim-pyth-triple x y z ⟷ *x* > *0* ∧ *y* > *0* ∧ *coprime x y* ∧ $x^2 + y^2 = z^2$

lemma *prim-pyth-triple-commute*: *prim-pyth-triple x y z* ⟷ *prim-pyth-triple y x z*

⟨*proof*⟩

lemma *prim-pyth-triple-aux*:
fixes $u\ v$ :: *nat*
assumes $v \le u$
shows $(2 * u * v)\ \hat{}\ 2 + (u\ \hat{}\ 2 - v\ \hat{}\ 2)\ \hat{}\ 2 = (u\ \hat{}\ 2 + v\ \hat{}\ 2)\ \hat{}\ 2$
⟨*proof*⟩

lemma *prim-pyth-tripleI1*:
assumes $0 < v$ $v < u$ *coprime u v* $\neg(odd\ u \wedge odd\ v)$
shows *prim-pyth-triple* $(2 * u * v)$ $(u^2 - v^2)$ $(u^2 + v^2)$
⟨*proof*⟩

lemma *prim-pyth-tripleI2*:
assumes $0 < v$ $v < u$ *coprime u v* $\neg(odd\ u \wedge odd\ v)$
shows *prim-pyth-triple* $(u^2 - v^2)$ $(2 * u * v)$ $(u^2 + v^2)$
⟨*proof*⟩

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

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** $\neg(odd\ u \wedge odd\ v)$ **and** $z = u^2 + v^2$
and $x = 2 * u * v \wedge y = u^2 - v^2 \ \vee\ x = u^2 - v^2 \wedge y = 2 * u * v$
⟨*proof*⟩

theorem *prim-pyth-triple-iff*:
prim-pyth-triple x y z $\longleftrightarrow$
$(\exists u\ v.\ 0 < v \wedge v < u \wedge coprime\ u\ v \wedge \neg(odd\ u \wedge odd\ v) \wedge$
$(x = 2 * u * v \wedge y = u^2 - v^2 \ \vee\ x = u^2 - v^2 \wedge y = 2 * u * v) \wedge z = u^2 + v^2)$
(**is** - $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

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.