Continued Fractions

Manuel Eberl

March 25, 2024

Abstract

This article provides a formalisation of continued fractions of real numbers and their basic properties. It also contains a proof of the classic result that the irrational numbers with periodic continued fraction expansions are precisely the quadratic irrationals, i. e. real numbers that fulfil a non-trivial quadratic equation $ax^2 + bx + c = 0$ with integer coefficients.

Particular attention is given to the continued fraction expansion of $\sqrt{D}$ for a non-square natural number D. Basic results about the length and structure of its period are provided, along with an executable algorithm to compute the period (and from it, the entire expansion).

This is then also used to provide a fairly efficient, executable, and fully formalised algorithm to compute solutions to Pell's equation $x^2 - Dy^2 = 1$. The performance is sufficiently good to find the solution to Archimedes's cattle problem in less than a second on a typical computer. This involves the value $D = 410286423278424$, for which the solution has over 200000 decimals.

Lastly, a derivation of the continued fraction expansions of Euler's number e and an executable function to compute continued fraction expansions using interval arithmetic is also provided.

Contents

1 Continued Fractions

theory *Continued-Fractions*
imports
Complex-Main
Coinductive.Lazy-LList
Coinductive.Coinductive-Nat
HOL−Number-Theory.Fib
HOL−Library.BNF-Corec
Coinductive.Coinductive-Stream
begin

1.1 Auxiliary results

coinductive *linfinite* :: *'a llist* ⇒ *bool* **where**
linfinite xs ⟹ *linfinite* (*LCons x xs*)

lemma *llength-llist-of-stream* [*simp*]: *llength* (*llist-of-stream xs*) = ∞
⟨*proof*⟩

lemma *linfinite-conv-llength*: *linfinite xs* ⟷ *llength xs* = ∞
⟨*proof*⟩

definition *lnth-default* :: *'a* ⇒ *'a llist* ⇒ *nat* ⇒ *'a* **where**
lnth-default dflt xs n = (*if n* < *llength xs then lnth xs n else dflt*)

lemma *lnth-default-code* [*code*]:
lnth-default dflt xs n =
(*if lnull xs then dflt else if n* = *0 then lhd xs else lnth-default dflt* (*ltl xs*) (*n* − *1*))
⟨*proof*⟩

lemma *enat-le-iff*:
enat n ≤ *m* ⟷ *m* = ∞ ∨ (∃ *m'*. *m* = *enat m'* ∧ *n* ≤ *m'*)
⟨*proof*⟩

lemma *enat-less-iff*:
enat n < *m* ⟷ *m* = ∞ ∨ (∃ *m'*. *m* = *enat m'* ∧ *n* < *m'*)
⟨*proof*⟩

lemma *real-of-int-divide-in-Ints-iff*:
real-of-int a / *real-of-int b* ∈ ℤ ⟷ *b dvd a* ∨ *b* = *0*
⟨*proof*⟩

lemma *frac-add-of-nat*: *frac* (*of-nat y* + *x*) = *frac x*
⟨*proof*⟩

lemma *frac-add-of-int*: *frac* (*of-int y* + *x*) = *frac x*
⟨*proof*⟩

lemma *frac-fraction*: *frac* (*real-of-int a* / *real-of-int b*) = (*a mod b*) / *b*
⟨*proof*⟩

lemma *Suc-fib-ge*: *Suc* (*fib n*) ≥ *n*
⟨*proof*⟩

lemma *fib-ge*: *fib n* ≥ *n* − *1*
⟨*proof*⟩

lemma *frac-diff-of-nat-right* [*simp*]: *frac* (*x* − *of-nat y*) = *frac x*
⟨*proof*⟩

lemma *funpow-cycle*:
assumes *m* > *0*
assumes (*f* ^^ *m*) *x* = *x*
shows (*f* ^^ *k*) *x* = (*f* ^^ (*k mod m*)) *x*
⟨*proof*⟩

lemma *of-nat-ge-1-iff*: *of-nat n* ≥ (*1* :: *'a* :: *linordered-semidom*) ⟷ *n* > *0*
⟨*proof*⟩

lemma *not-frac-less-0*: ¬*frac x* < *0*
⟨*proof*⟩

lemma *frac-le-1*: *frac x* ≤ *1*
⟨*proof*⟩

lemma *divide-in-Rats-iff1*:
(*x*::*real*) ∈ ℚ ⟹ *x* ≠ *0* ⟹ *x* / *y* ∈ ℚ ⟷ *y* ∈ ℚ
⟨*proof*⟩

lemma *divide-in-Rats-iff2*:
(*y*::*real*) ∈ ℚ ⟹ *y* ≠ *0* ⟹ *x* / *y* ∈ ℚ ⟷ *x* ∈ ℚ
⟨*proof*⟩

lemma *add-in-Rats-iff1*: *x* ∈ ℚ ⟹ *x* + *y* ∈ ℚ ⟷ *y* ∈ ℚ
⟨*proof*⟩

lemma *add-in-Rats-iff2*: *y* ∈ ℚ ⟹ *x* + *y* ∈ ℚ ⟷ *x* ∈ ℚ
⟨*proof*⟩

lemma *diff-in-Rats-iff1*: *x* ∈ ℚ ⟹ *x* − *y* ∈ ℚ ⟷ *y* ∈ ℚ
⟨*proof*⟩

lemma *diff-in-Rats-iff2*: *y* ∈ ℚ ⟹ *x* − *y* ∈ ℚ ⟷ *x* ∈ ℚ
⟨*proof*⟩

lemma *frac-in-Rats-iff* [*simp*]: *frac x* ∈ ℚ ⟷ *x* ∈ ℚ
⟨*proof*⟩

lemma *filterlim-sequentially-shift*:
filterlim $(\lambda n.\ f\ (n + m))\ F$ *sequentially* $\longleftrightarrow$ *filterlim* $f\ F$ *sequentially*
⟨*proof*⟩

1.2 Bounds on alternating decreasing sums

lemma *alternating-decreasing-sum-bounds*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m \leq n\ \bigwedge k.\ k \in \{m..n\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
defines $S \equiv (\lambda m.\ (\sum k{=}m..n.\ (-1)\ \hat{}\ k * f\ k))$
shows *if even* m *then* $S\ m \in \{0..f\ m\}$ *else* $S\ m \in \{-f\ m..0\}$
⟨*proof*⟩

lemma *alternating-decreasing-sum-bounds′*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m < n\ \bigwedge k.\ k \in \{m..n-1\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n-1\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
defines $S \equiv (\lambda m.\ (\sum k{=}m..<n.\ (-1)\ \hat{}\ k * f\ k))$
shows *if even* m *then* $S\ m \in \{0..f\ m\}$ *else* $S\ m \in \{-f\ m..0\}$
⟨*proof*⟩

lemma *alternating-decreasing-sum-upper-bound*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m \leq n\ \bigwedge k.\ k \in \{m..n\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $(\sum k{=}m..n.\ (-1)\ \hat{}\ k * f\ k) \leq f\ m$
⟨*proof*⟩

lemma *alternating-decreasing-sum-upper-bound′*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m < n\ \bigwedge k.\ k \in \{m..n-1\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n-1\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $(\sum k{=}m..<n.\ (-1)\ \hat{}\ k * f\ k) \leq f\ m$
⟨*proof*⟩

lemma *abs-alternating-decreasing-sum-upper-bound*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m \leq n\ \bigwedge k.\ k \in \{m..n\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $|(\sum k{=}m..n.\ (-1)\ \hat{}\ k * f\ k)| \leq f\ m$ (**is** *abs ?S* $\leq$ -)
⟨*proof*⟩

lemma *abs-alternating-decreasing-sum-upper-bound′*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring,\ ring\text{-}1\}$
assumes $m < n\ \bigwedge k.\ k \in \{m..n-1\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n-1\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $|(\sum k{=}m..<n.\ (-1)\ \hat{}\ k * f\ k)| \leq f\ m$

⟨*proof*⟩

lemma *abs-alternating-decreasing-sum-lower-bound*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring, ring\text{-}1\}$
assumes $m < n$ $\bigwedge k.\ k \in \{m..n\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $|(\sum k{=}m..n.\ (-1)\ \hat{}\ k * f\ k)| \geq f\ m - f\ (Suc\ m)$
⟨*proof*⟩

lemma *abs-alternating-decreasing-sum-lower-bound'*:
fixes $f :: nat \Rightarrow 'a :: \{linordered\text{-}ring, ring\text{-}1\}$
assumes $m{+}1 < n$ $\bigwedge k.\ k \in \{m..n\} \Longrightarrow f\ k \geq 0$
$\bigwedge k.\ k \in \{m..<n\} \Longrightarrow f\ (Suc\ k) \leq f\ k$
shows $|(\sum k{=}m..<n.\ (-1)\ \hat{}\ k * f\ k)| \geq f\ m - f\ (Suc\ m)$
⟨*proof*⟩

lemma *alternating-decreasing-suminf-bounds*:
assumes $\bigwedge k.\ f\ k \geq (0 :: real)$ $\bigwedge k.\ f\ (Suc\ k) \leq f\ k$
$f \longrightarrow 0$
shows $(\sum k.\ (-1)\ \hat{}\ k * f\ k) \in \{f\ 0 - f\ 1..f\ 0\}$
⟨*proof*⟩

lemma
assumes $\bigwedge k.\ k \geq m \Longrightarrow f\ k \geq (0 :: real)$
$\bigwedge k.\ k \geq m \Longrightarrow f\ (Suc\ k) \leq f\ k\ f \longrightarrow 0$
defines $S \equiv (\sum k.\ (-1)\ \hat{}\ (k + m) * f\ (k + m))$
shows *summable-alternating-decreasing*: *summable* $(\lambda k.\ (-1)\ \hat{}\ (k + m) * f\ (k + m))$
and *alternating-decreasing-suminf-bounds'*:
if even m then $S \in \{f\ m - f\ (Suc\ m)\ ..\ f\ m\}$
else $S \in \{-f\ m..f\ (Suc\ m) - f\ m\}$ (**is** *?th1*)
and *abs-alternating-decreasing-suminf*:
$abs\ S \in \{f\ m - f\ (Suc\ m)..f\ m\}$ (**is** *?th2*)
⟨*proof*⟩

lemma
assumes $\bigwedge k.\ k \geq m \Longrightarrow f\ k \geq (0 :: real)$
$\bigwedge k.\ k \geq m \Longrightarrow f\ (Suc\ k) < f\ k\ f \longrightarrow 0$
defines $S \equiv (\sum k.\ (-1)\ \hat{}\ (k + m) * f\ (k + m))$
shows *alternating-decreasing-suminf-bounds-strict'*:
if even m then $S \in \{f\ m - f\ (Suc\ m)<..<f\ m\}$
else $S \in \{-f\ m<..<f\ (Suc\ m) - f\ m\}$ (**is** *?th1*)
and *abs-alternating-decreasing-suminf-strict*:
$abs\ S \in \{f\ m - f\ (Suc\ m)<..<f\ m\}$ (**is** *?th2*)
⟨*proof*⟩

datatype *cfrac* = *CFrac int nat llist*

quickcheck-generator *cfrac constructors*: *CFrac*

lemma *type-definition-cfrac′*:
type-definition (λ*x*. *case x of CFrac a b* ⇒ (*a*, *b*)) (λ(*x*,*y*). *CFrac x y*) *UNIV*
⟨*proof*⟩

setup-lifting *type-definition-cfrac′*

lift-definition *cfrac-of-int* :: *int* ⇒ *cfrac* **is**
λ*n*. (*n*, *LNil*) ⟨*proof*⟩

lemma *cfrac-of-int-code* [*code*]: *cfrac-of-int n* = *CFrac n LNil*
⟨*proof*⟩

lift-definition *cfrac-of-stream* :: *int stream* ⇒ *cfrac* **is**
λ*xs*. (*shd xs*, *llist-of-stream* (*smap* (λ*x*. *nat* (*x* − *1*)) (*stl xs*))) ⟨*proof*⟩

instantiation *cfrac* :: *zero*
begin
definition *zero-cfrac* **where** *0* = *cfrac-of-int 0*
instance ⟨*proof*⟩
end

instantiation *cfrac* :: *one*
begin
definition *one-cfrac* **where** *1* = *cfrac-of-int 1*
instance ⟨*proof*⟩
end

lift-definition *cfrac-tl* :: *cfrac* ⇒ *cfrac* **is**
λ(-, *bs*) ⇒ *case bs of LNil* ⇒ (*1*, *LNil*) | *LCons b bs′* ⇒ (*int b* + *1*, *bs′*) ⟨*proof*⟩

lemma *cfrac-tl-code* [*code*]:
cfrac-tl (*CFrac a bs*) =
(*case bs of LNil* ⇒ *CFrac 1 LNil* | *LCons b bs′* ⇒ *CFrac* (*int b* + *1*) *bs′*)
⟨*proof*⟩

definition *cfrac-drop* :: *nat* ⇒ *cfrac* ⇒ *cfrac* **where**
cfrac-drop n c = (*cfrac-tl* ^^ *n*) *c*

lemma *cfrac-drop-Suc-right*: *cfrac-drop* (*Suc n*) *c* = *cfrac-drop n* (*cfrac-tl c*)
⟨*proof*⟩

lemma *cfrac-drop-Suc-left*: *cfrac-drop* (*Suc n*) *c* = *cfrac-tl* (*cfrac-drop n c*)
⟨*proof*⟩

lemma *cfrac-drop-add*: *cfrac-drop* (*m* + *n*) *c* = *cfrac-drop m* (*cfrac-drop n c*)
⟨*proof*⟩

lemma *cfrac-drop-0* [*simp*]: *cfrac-drop 0* = (λ*x*. *x*)
⟨*proof*⟩

lemma *cfrac-drop-1* [*simp*]: *cfrac-drop 1* = *cfrac-tl*
⟨*proof*⟩

lift-definition *cfrac-length* :: *cfrac* ⇒ *enat* **is**
λ(-, *bs*) ⇒ *llength bs* ⟨*proof*⟩

lemma *cfrac-length-code* [*code*]: *cfrac-length* (*CFrac a bs*) = *llength bs*
⟨*proof*⟩

lemma *cfrac-length-tl* [*simp*]: *cfrac-length* (*cfrac-tl c*) = *cfrac-length c* − *1*
⟨*proof*⟩

lemma *enat-diff-Suc-right* [*simp*]: *m* − *enat* (*Suc n*) = *m* − *n* − *1*
⟨*proof*⟩

lemma *cfrac-length-drop* [*simp*]: *cfrac-length* (*cfrac-drop n c*) = *cfrac-length c* − *n*
⟨*proof*⟩

lemma *cfrac-length-of-stream* [*simp*]: *cfrac-length* (*cfrac-of-stream xs*) = ∞
⟨*proof*⟩

lift-definition *cfrac-nth* :: *cfrac* ⇒ *nat* ⇒ *int* **is**
λ(*a* :: *int*, *bs* :: *nat llist*). λ(*n* :: *nat*).
if n = *0 then a*
else if n ≤ *llength bs then int* (*lnth bs* (*n* − *1*)) + *1 else 1* ⟨*proof*⟩

lemma *cfrac-nth-code* [*code*]:
cfrac-nth (*CFrac a bs*) *n* = (*if n* = *0 then a else lnth-default 0 bs* (*n* − *1*) + *1*)
⟨*proof*⟩

lemma *cfrac-nth-nonneg* [*simp*, *intro*]: *n* > *0* ⟹ *cfrac-nth c n* ≥ *0*
⟨*proof*⟩

lemma *cfrac-nth-nonzero* [*simp*]: *n* > *0* ⟹ *cfrac-nth c n* ≠ *0*
⟨*proof*⟩

lemma *cfrac-nth-pos*[*simp*, *intro*]: *n* > *0* ⟹ *cfrac-nth c n* > *0*
⟨*proof*⟩

lemma *cfrac-nth-ge-1*[*simp*, *intro*]: *n* > *0* ⟹ *cfrac-nth c n* ≥ *1*
⟨*proof*⟩

lemma *cfrac-nth-not-less-1*[*simp*, *intro*]: *n* > *0* ⟹ ¬*cfrac-nth c n* < *1*
⟨*proof*⟩

lemma *cfrac-nth-tl* [*simp*]: *cfrac-nth* (*cfrac-tl c*) *n* = *cfrac-nth c* (*Suc n*)

⟨*proof*⟩

lemma *cfrac-nth-drop* [*simp*]: *cfrac-nth* (*cfrac-drop n c*) *m* = *cfrac-nth c* (*m* + *n*)
⟨*proof*⟩

lemma *cfrac-nth-0-of-int* [*simp*]: *cfrac-nth* (*cfrac-of-int n*) *0* = *n*
⟨*proof*⟩

lemma *cfrac-nth-gt0-of-int* [*simp*]: *m* > *0* ⟹ *cfrac-nth* (*cfrac-of-int n*) *m* = *1*
⟨*proof*⟩

lemma *cfrac-nth-of-stream*:
assumes *sset* (*stl xs*) ⊆ {*0*<..}
shows *cfrac-nth* (*cfrac-of-stream xs*) *n* = *snth xs n*
⟨*proof*⟩

lift-definition *cfrac* :: (*nat* ⇒ *int*) ⇒ *cfrac* **is**
λ*f*. (*f 0*, *inf-llist* (λ*n*. *nat* (*f* (*Suc n*) − *1*))) ⟨*proof*⟩

definition *is-cfrac* :: (*nat* ⇒ *int*) ⇒ *bool* **where** *is-cfrac f* ⟷ (∀ *n*>*0*. *f n* > *0*)

lemma *cfrac-nth-cfrac* [*simp*]:
assumes *is-cfrac f*
shows *cfrac-nth* (*cfrac f*) *n* = *f n*
⟨*proof*⟩

lemma *llength-eq-infty-lnth*: *llength b* = ∞ ⟹ *inf-llist* (*lnth b*) = *b*
⟨*proof*⟩

lemma *cfrac-cfrac-nth* [*simp*]: *cfrac-length c* = ∞ ⟹ *cfrac* (*cfrac-nth c*) = *c*
⟨*proof*⟩

lemma *cfrac-length-cfrac* [*simp*]: *cfrac-length* (*cfrac f*) = ∞
⟨*proof*⟩

lift-definition *cfrac-of-list* :: *int list* ⇒ *cfrac* **is**
λ*xs*. *if xs* = [] *then* (*0*, *LNil*) *else* (*hd xs*, *llist-of* (*map* (λ*n*. *nat n* − *1*) (*tl xs*)))
⟨*proof*⟩

lemma *cfrac-length-of-list* [*simp*]: *cfrac-length* (*cfrac-of-list xs*) = *length xs* − *1*
⟨*proof*⟩

lemma *cfrac-of-list-Nil* [*simp*]: *cfrac-of-list* [] = *0*
⟨*proof*⟩

lemma *cfrac-nth-of-list* [*simp*]:
assumes *n* < *length xs* **and** ∀ *i*∈{*0*<..<*length xs*}. *xs* ! *i* > *0*

shows *cfrac-nth* (*cfrac-of-list xs*) *n* = *xs* ! *n*
⟨*proof*⟩

primcorec *cfrac-of-real-aux* :: *real* ⇒ *nat llist* **where**
cfrac-of-real-aux x =
(*if x* ∈ {*0*<..<*1*} *then LCons* (*nat* ⌊*1*/*x*⌋ − *1*) (*cfrac-of-real-aux* (*frac* (*1*/*x*))) *else LNil*)

lemma *cfrac-of-real-aux-code* [*code*]:
cfrac-of-real-aux x =
(*if x* > *0* ∧ *x* < *1 then LCons* (*nat* ⌊*1*/*x*⌋ − *1*) (*cfrac-of-real-aux* (*frac* (*1*/*x*))) *else LNil*)
⟨*proof*⟩

lemma *cfrac-of-real-aux-LNil* [*simp*]: *x* ∉ {*0*<..<*1*} ⟹ *cfrac-of-real-aux x* = *LNil*
⟨*proof*⟩

lemma *cfrac-of-real-aux-0* [*simp*]: *cfrac-of-real-aux 0* = *LNil*
⟨*proof*⟩

lemma *cfrac-of-real-aux-eq-LNil-iff* [*simp*]: *cfrac-of-real-aux x* = *LNil* ⟷ *x* ∉ {*0*<..<*1*}
⟨*proof*⟩

lemma *lnth-cfrac-of-real-aux*:
assumes *n* < *llength* (*cfrac-of-real-aux x*)
shows *lnth* (*cfrac-of-real-aux x*) (*Suc n*) = *lnth* (*cfrac-of-real-aux* (*frac* (*1*/*x*))) *n*
⟨*proof*⟩

lift-definition *cfrac-of-real* :: *real* ⇒ *cfrac* **is**
λ*x*. (⌊*x*⌋, *cfrac-of-real-aux* (*frac x*)) ⟨*proof*⟩

lemma *cfrac-of-real-code* [*code*]: *cfrac-of-real x* = *CFrac* ⌊*x*⌋ (*cfrac-of-real-aux* (*frac x*))
⟨*proof*⟩

lemma *eq-epred-iff*: *m* = *epred n* ⟷ *m* = *0* ∧ *n* = *0* ∨ *n* = *eSuc m*
⟨*proof*⟩

lemma *epred-eq-iff*: *epred n* = *m* ⟷ *m* = *0* ∧ *n* = *0* ∨ *n* = *eSuc m*
⟨*proof*⟩

lemma *epred-less*: *n* > *0* ⟹ *n* ≠ ∞ ⟹ *epred n* < *n*
⟨*proof*⟩

lemma *cfrac-nth-of-real-0* [*simp*]:

cfrac-nth (*cfrac-of-real x*) *0* = ⌊*x*⌋
⟨*proof*⟩

lemma *frac-eq-0* [*simp*]: *x* ∈ ℤ ⟹ *frac x* = *0*
⟨*proof*⟩

lemma *cfrac-tl-of-real*:
assumes *x* ∉ ℤ
shows *cfrac-tl* (*cfrac-of-real x*) = *cfrac-of-real* (*1* / *frac x*)
⟨*proof*⟩

lemma *cfrac-nth-of-real-Suc*:
assumes *x* ∉ ℤ
shows *cfrac-nth* (*cfrac-of-real x*) (*Suc n*) = *cfrac-nth* (*cfrac-of-real* (*1* / *frac x*)) *n*
⟨*proof*⟩

fun *conv* :: *cfrac* ⇒ *nat* ⇒ *real* **where**
conv c 0 = *real-of-int* (*cfrac-nth c 0*)
| *conv c* (*Suc n*) = *real-of-int* (*cfrac-nth c 0*) + *1* / *conv* (*cfrac-tl c*) *n*

The numerator and denominator of a convergent:

fun *conv-num* :: *cfrac* ⇒ *nat* ⇒ *int* **where**
conv-num c 0 = *cfrac-nth c 0*
| *conv-num c* (*Suc 0*) = *cfrac-nth c 1* ∗ *cfrac-nth c 0* + *1*
| *conv-num c* (*Suc* (*Suc n*)) = *cfrac-nth c* (*Suc* (*Suc n*)) ∗ *conv-num c* (*Suc n*) + *conv-num c n*

fun *conv-denom* :: *cfrac* ⇒ *nat* ⇒ *int* **where**
conv-denom c 0 = *1*
| *conv-denom c* (*Suc 0*) = *cfrac-nth c 1*
| *conv-denom c* (*Suc* (*Suc n*)) = *cfrac-nth c* (*Suc* (*Suc n*)) ∗ *conv-denom c* (*Suc n*) + *conv-denom c n*

lemma *conv-num-rec*:
n ≥ *2* ⟹ *conv-num c n* = *cfrac-nth c n* ∗ *conv-num c* (*n* − *1*) + *conv-num c* (*n* − *2*)
⟨*proof*⟩

lemma *conv-denom-rec*:
n ≥ *2* ⟹ *conv-denom c n* = *cfrac-nth c n* ∗ *conv-denom c* (*n* − *1*) + *conv-denom c* (*n* − *2*)
⟨*proof*⟩

fun *conv′* :: *cfrac* ⇒ *nat* ⇒ *real* ⇒ *real* **where**
conv′ c 0 z = *z*
| *conv′ c* (*Suc n*) *z* = *conv′ c n* (*real-of-int* (*cfrac-nth c n*) + *1* / *z*)

Occasionally, it can be useful to extend the domain of *conv-num* and *conv-denom* to -1 and -2.

definition *conv-num-int* :: *cfrac* ⇒ *int* ⇒ *int* **where**
conv-num-int c n = (*if n* = −*1 then 1 else if n* < *0 then 0 else conv-num c* (*nat n*))

definition *conv-denom-int* :: *cfrac* ⇒ *int* ⇒ *int* **where**
conv-denom-int c n = (*if n* = −*2 then 1 else if n* < *0 then 0 else conv-denom c* (*nat n*))

lemma *conv-num-int-rec*:
assumes *n* ≥ *0*
shows *conv-num-int c n* = *cfrac-nth c* (*nat n*) ∗ *conv-num-int c* (*n* − *1*) + *conv-num-int c* (*n* − *2*)
⟨*proof*⟩

lemma *conv-denom-int-rec*:
assumes *n* ≥ *0*
shows *conv-denom-int c n* = *cfrac-nth c* (*nat n*) ∗ *conv-denom-int c* (*n* − *1*) + *conv-denom-int c* (*n* − *2*)
⟨*proof*⟩

The number $[a_0;\ a_1,\ a_2,\ \ldots]$ that the infinite continued fraction converges to:

definition *cfrac-lim* :: *cfrac* ⇒ *real* **where**
cfrac-lim c =
(*case cfrac-length c of* ∞ ⇒ *lim* (*conv c*) | *enat l* ⇒ *conv c l*)

lemma *cfrac-lim-code* [*code*]:
cfrac-lim c =
(*case cfrac-length c of enat l* ⇒ *conv c l*
| - ⇒ *Code.abort* (*STR ''Cannot compute infinite continued fraction''*) (λ-. *cfrac-lim c*))
⟨*proof*⟩

definition *cfrac-remainder* **where** *cfrac-remainder c n* = *cfrac-lim* (*cfrac-drop n c*)

lemmas *conv'-Suc-right* = *conv'.simps*(*2*)

lemma *conv'-Suc-left*:
assumes *z* > *0*
shows *conv' c* (*Suc n*) *z* =
real-of-int (*cfrac-nth c 0*) + *1* / *conv'* (*cfrac-tl c*) *n z*
⟨*proof*⟩

lemmas [*simp del*] = *conv'.simps*(*2*)

lemma *conv'-left-induct*:

assumes $\bigwedge c.\ P\ c\ 0\ z\ \bigwedge c\ n.\ P\ (cfrac\text{-}tl\ c)\ n\ z \Longrightarrow P\ c\ (Suc\ n)\ z$
shows $P\ c\ n\ z$
⟨*proof*⟩

lemma *enat-less-diff-conv* [*simp*]:
assumes $a = \infty \vee b < \infty \vee c < \infty$
shows $a < c - (b :: enat) \longleftrightarrow a + b < c$
⟨*proof*⟩

lemma *conv-eq-conv'*: $conv\ c\ n = conv'\ c\ n\ (cfrac\text{-}nth\ c\ n)$
⟨*proof*⟩

lemma *conv-num-pos'*:
assumes $cfrac\text{-}nth\ c\ 0 > 0$
shows $conv\text{-}num\ c\ n > 0$
⟨*proof*⟩

lemma *conv-num-nonneg*: $cfrac\text{-}nth\ c\ 0 \geq 0 \Longrightarrow conv\text{-}num\ c\ n \geq 0$
⟨*proof*⟩

lemma *conv-num-pos*:
$cfrac\text{-}nth\ c\ 0 \geq 0 \Longrightarrow n > 0 \Longrightarrow conv\text{-}num\ c\ n > 0$
⟨*proof*⟩

lemma *conv-denom-pos* [*simp*, *intro*]: $conv\text{-}denom\ c\ n > 0$
⟨*proof*⟩

lemma *conv-denom-not-nonpos* [*simp*]: $\neg conv\text{-}denom\ c\ n \leq 0$
⟨*proof*⟩

lemma *conv-denom-not-neg* [*simp*]: $\neg conv\text{-}denom\ c\ n < 0$
⟨*proof*⟩

lemma *conv-denom-nonzero* [*simp*]: $conv\text{-}denom\ c\ n \neq 0$
⟨*proof*⟩

lemma *conv-denom-nonneg* [*simp*, *intro*]: $conv\text{-}denom\ c\ n \geq 0$
⟨*proof*⟩

lemma *conv-num-int-neg1* [*simp*]: $conv\text{-}num\text{-}int\ c\ (-1) = 1$
⟨*proof*⟩

lemma *conv-num-int-neg* [*simp*]: $n < 0 \Longrightarrow n \neq -1 \Longrightarrow conv\text{-}num\text{-}int\ c\ n = 0$
⟨*proof*⟩

lemma *conv-num-int-of-nat* [*simp*]: $conv\text{-}num\text{-}int\ c\ (int\ n) = conv\text{-}num\ c\ n$
⟨*proof*⟩

lemma *conv-num-int-nonneg* [*simp*]: $n \geq 0 \Longrightarrow conv\text{-}num\text{-}int\ c\ n = conv\text{-}num\ c$

(*nat n*)
⟨*proof*⟩

lemma *conv-denom-int-neg2* [*simp*]: *conv-denom-int c* (−*2*) = *1*
⟨*proof*⟩

lemma *conv-denom-int-neg* [*simp*]: *n* < *0* ⟹ *n* ≠ −*2* ⟹ *conv-denom-int c n* = *0*
⟨*proof*⟩

lemma *conv-denom-int-of-nat* [*simp*]: *conv-denom-int c* (*int n*) = *conv-denom c n*
⟨*proof*⟩

lemma *conv-denom-int-nonneg* [*simp*]: *n* ≥ *0* ⟹ *conv-denom-int c n* = *conv-denom c* (*nat n*)
⟨*proof*⟩

lemmas *conv-Suc* [*simp del*] = *conv.simps*(*2*)

lemma *conv'-gt-1*:
assumes *cfrac-nth c 0* > *0 x* > *1*
shows *conv' c n x* > *1*
⟨*proof*⟩

lemma *enat-eq-iff*: *a* = *enat b* ⟷ (∃ *a'*. *a* = *enat a'* ∧ *a'* = *b*)
⟨*proof*⟩

lemma *eq-enat-iff*: *enat a* = *b* ⟷ (∃ *b'*. *b* = *enat b'* ∧ *a* = *b'*)
⟨*proof*⟩

lemma *enat-diff-one* [*simp*]: *enat a* − *1* = *enat* (*a* − *1*)
⟨*proof*⟩

lemma *conv'-eqD*:
assumes *conv' c n x* = *conv' c' n x x* > *1 m* < *n*
shows *cfrac-nth c m* = *cfrac-nth c' m*
⟨*proof*⟩

context
fixes *c* :: *cfrac* **and** *h k*
defines *h* ≡ *conv-num c* **and** *k* ≡ *conv-denom c*
begin

lemma *conv'-num-denom-aux*:
assumes *z*: *z* > *0*
shows *conv' c* (*Suc* (*Suc n*)) *z* ∗ (*z* ∗ *k* (*Suc n*) + *k n*) =
(*z* ∗ *h* (*Suc n*) + *h n*)
⟨*proof*⟩

lemma *conv'-num-denom*:
assumes $z > 0$
shows $conv'\ c\ (Suc\ (Suc\ n))\ z =$
$(z * h\ (Suc\ n) + h\ n)\ /\ (z * k\ (Suc\ n) + k\ n)$
⟨*proof*⟩

lemma *conv-num-denom*: $conv\ c\ n = h\ n\ /\ k\ n$
⟨*proof*⟩

lemma *conv'-num-denom'*:
assumes $z > 0$ **and** $n \geq 2$
shows $conv'\ c\ n\ z = (z * h\ (n - 1) + h\ (n - 2))\ /\ (z * k\ (n - 1) + k\ (n - 2))$
⟨*proof*⟩

lemma *conv'-num-denom-int*:
assumes $z > 0$
shows $conv'\ c\ n\ z =$
$(z * conv\text{-}num\text{-}int\ c\ (int\ n - 1) + conv\text{-}num\text{-}int\ c\ (int\ n - 2))\ /$
$(z * conv\text{-}denom\text{-}int\ c\ (int\ n - 1) + conv\text{-}denom\text{-}int\ c\ (int\ n - 2))$
⟨*proof*⟩

lemma *conv-nonneg*: $cfrac\text{-}nth\ c\ 0 \geq 0 \Longrightarrow conv\ c\ n \geq 0$
⟨*proof*⟩

lemma *conv-pos*:
assumes $cfrac\text{-}nth\ c\ 0 > 0$
shows $conv\ c\ n > 0$
⟨*proof*⟩

lemma *conv-num-denom-prod-diff*:
$k\ n * h\ (Suc\ n) - k\ (Suc\ n) * h\ n = (-1)\ \hat{}\ n$
⟨*proof*⟩

lemma *conv-num-denom-prod-diff'*:
$k\ (Suc\ n) * h\ n - k\ n * h\ (Suc\ n) = (-1)\ \hat{}\ Suc\ n$
⟨*proof*⟩

lemma
fixes $n :: int$
assumes $n \geq -2$
shows *conv-num-denom-int-prod-diff*:
$conv\text{-}denom\text{-}int\ c\ n * conv\text{-}num\text{-}int\ c\ (n + 1) -$
$conv\text{-}denom\text{-}int\ c\ (n + 1) * conv\text{-}num\text{-}int\ c\ n = (-1)\ \hat{}\ (nat\ (n + 2))$
(**is** *?th1*)
and *conv-num-denom-int-prod-diff'*:
$conv\text{-}denom\text{-}int\ c\ (n + 1) * conv\text{-}num\text{-}int\ c\ n -$
$conv\text{-}denom\text{-}int\ c\ n * conv\text{-}num\text{-}int\ c\ (n + 1) = (-1)\ \hat{}\ (nat\ (n + 3))$

(**is** *?th2*)
⟨*proof*⟩

lemma *coprime-conv-num-denom*: *coprime* (*h n*) (*k n*)
⟨*proof*⟩

lemma *coprime-conv-num-denom-int*:
assumes $n \geq -2$
shows *coprime* (*conv-num-int c n*) (*conv-denom-int c n*)
⟨*proof*⟩

lemma *mono-conv-num*:
assumes *cfrac-nth c 0* $\geq$ *0*
shows *mono h*
⟨*proof*⟩

lemma *mono-conv-denom*: *mono k*
⟨*proof*⟩

lemma *conv-num-leI*: *cfrac-nth c 0* $\geq$ *0* $\Longrightarrow$ $m \leq n \Longrightarrow$ *h m* $\leq$ *h n*
⟨*proof*⟩

lemma *conv-denom-leI*: $m \leq n \Longrightarrow$ *k m* $\leq$ *k n*
⟨*proof*⟩

lemma *conv-denom-lessI*:
assumes $m < n$ *1* $<$ *n*
shows *k m* $<$ *k n*
⟨*proof*⟩

lemma *conv-num-lower-bound*:
assumes *cfrac-nth c 0* $\geq$ *0*
shows *h n* $\geq$ *fib n* ⟨*proof*⟩

lemma *conv-denom-lower-bound*: *k n* $\geq$ *fib* (*Suc n*)
⟨*proof*⟩

lemma *conv-diff-eq*: *conv c* (*Suc n*) $-$ *conv c n* $=$ (-1) ^ *n* / (*k n* $*$ *k* (*Suc n*))
⟨*proof*⟩

lemma *conv-telescope*:
assumes $m \leq n$
shows *conv c m* $+$ $(\sum i{=}m..{<}n.$ (-1) ^ *i* / (*k i* $*$ *k* (*Suc i*))) $=$ *conv c n*
⟨*proof*⟩

lemma *fib-at-top*: *filterlim fib at-top at-top*
⟨*proof*⟩

lemma *conv-denom-at-top*: *filterlim k at-top at-top*

⟨*proof*⟩

lemma
shows *summable-conv-telescope*:
summable (λ*i*. (−*1*) ^ *i* / (*k i* ∗ *k* (*Suc i*))) (**is** *?th1*)
and *cfrac-remainder-bounds*:
|(∑ *i*. (−*1*) ^ (*i* + *m*) / (*k* (*i* + *m*) ∗ *k* (*Suc i* + *m*)))| ∈
{*1*/(*k m* ∗ (*k m* + *k* (*Suc m*))) <..< *1* / (*k m* ∗ *k* (*Suc m*))} (**is** *?th2*)
⟨*proof*⟩

lemma *convergent-conv*: *convergent* (*conv c*)
⟨*proof*⟩

lemma *LIMSEQ-cfrac-lim*: *cfrac-length c* = ∞ ⟹ *conv c* ⟶ *cfrac-lim c*
⟨*proof*⟩

lemma *cfrac-lim-nonneg*:
assumes *cfrac-nth c 0* ≥ *0*
shows *cfrac-lim c* ≥ *0*
⟨*proof*⟩

lemma *sums-cfrac-lim-minus-conv*:
assumes *cfrac-length c* = ∞
shows (λ*i*. (−*1*) ^ (*i* + *m*) / (*k* (*i* + *m*) ∗ *k* (*Suc i* + *m*))) *sums* (*cfrac-lim c* − *conv c m*)
⟨*proof*⟩

lemma *cfrac-lim-minus-conv-upper-bound*:
assumes *m* ≤ *cfrac-length c*
shows |*cfrac-lim c* − *conv c m*| ≤ *1* / (*k m* ∗ *k* (*Suc m*))
⟨*proof*⟩

lemma *cfrac-lim-minus-conv-lower-bound*:
assumes *m* < *cfrac-length c*
shows |*cfrac-lim c* − *conv c m*| ≥ *1* / (*k m* ∗ (*k m* + *k* (*Suc m*)))
⟨*proof*⟩

lemma *cfrac-lim-minus-conv-bounds*:
assumes *m* < *cfrac-length c*
shows |*cfrac-lim c* − *conv c m*| ∈ {*1* / (*k m* ∗ (*k m* + *k* (*Suc m*)))..*1* / (*k m* ∗ *k* (*Suc m*))}
⟨*proof*⟩

end

lemma *conv-pos′*:
assumes *n* > *0 cfrac-nth c 0* ≥ *0*
shows *conv c n* > *0*

⟨*proof*⟩

lemma *conv-in-Rats* [*intro*]: *conv c n* ∈ ℚ
⟨*proof*⟩

lemma
assumes *0 < z1 z1* ≤ *z2*
shows *conv′-even-mono*: *even n* ⟹ *conv′ c n z1* ≤ *conv′ c n z2*
and *conv′-odd-mono*: *odd n* ⟹ *conv′ c n z1* ≥ *conv′ c n z2*
⟨*proof*⟩

lemma
shows *conv-even-mono*: *even n* ⟹ *n* ≤ *m* ⟹ *conv c n* ≤ *conv c m*
and *conv-odd-mono*: *odd n* ⟹ *n* ≤ *m* ⟹ *conv c n* ≥ *conv c m*
⟨*proof*⟩

lemma
assumes *m* ≤ *cfrac-length c*
shows *conv-le-cfrac-lim*: *even m* ⟹ *conv c m* ≤ *cfrac-lim c*
and *conv-ge-cfrac-lim*: *odd m* ⟹ *conv c m* ≥ *cfrac-lim c*
⟨*proof*⟩

lemma *cfrac-lim-ge-first*: *cfrac-lim c* ≥ *cfrac-nth c 0*
⟨*proof*⟩

lemma *cfrac-lim-pos*: *cfrac-nth c 0 > 0* ⟹ *cfrac-lim c > 0*
⟨*proof*⟩

lemma *conv′-eq-iff*:
assumes *0* ≤ *z1* ∨ *0* ≤ *z2*
shows *conv′ c n z1* = *conv′ c n z2* ⟷ *z1* = *z2*
⟨*proof*⟩

lemma *conv-even-mono-strict*:
assumes *even n n < m*
shows *conv c n < conv c m*
⟨*proof*⟩

lemma *conv-odd-mono-strict*:
assumes *odd n n < m*
shows *conv c n > conv c m*
⟨*proof*⟩

lemma *conv-less-cfrac-lim*:
assumes *even n n < cfrac-length c*
shows *conv c n < cfrac-lim c*
⟨*proof*⟩

lemma *conv-gt-cfrac-lim*:

assumes *odd n* $n <$ *cfrac-length c*
shows *conv c n* $>$ *cfrac-lim c*
⟨*proof*⟩

lemma *conv-neq-cfrac-lim*:
assumes $n <$ *cfrac-length c*
shows *conv c n* $\neq$ *cfrac-lim c*
⟨*proof*⟩

lemma *conv-ge-first*: *conv c n* $\geq$ *cfrac-nth c 0*
⟨*proof*⟩

definition *cfrac-is-zero* :: *cfrac* $\Rightarrow$ *bool* **where** *cfrac-is-zero c* $\longleftrightarrow$ *c = 0*

lemma *cfrac-is-zero-code* [*code*]: *cfrac-is-zero* (*CFrac n xs*) $\longleftrightarrow$ *lnull xs* $\wedge$ *n = 0*
⟨*proof*⟩

definition *cfrac-is-int* **where** *cfrac-is-int c* $\longleftrightarrow$ *cfrac-length c = 0*

lemma *cfrac-is-int-code* [*code*]: *cfrac-is-int* (*CFrac n xs*) $\longleftrightarrow$ *lnull xs*
⟨*proof*⟩

lemma *cfrac-length-of-int* [*simp*]: *cfrac-length* (*cfrac-of-int n*) *= 0*
⟨*proof*⟩

lemma *cfrac-is-int-of-int* [*simp*, *intro*]: *cfrac-is-int* (*cfrac-of-int n*)
⟨*proof*⟩

lemma *cfrac-is-int-iff*: *cfrac-is-int c* $\longleftrightarrow$ ($\exists$ *n. c = cfrac-of-int n*)
⟨*proof*⟩

lemma *cfrac-lim-reduce*:
assumes $\neg$*cfrac-is-int c*
shows *cfrac-lim c = cfrac-nth c 0 + 1 / cfrac-lim* (*cfrac-tl c*)
⟨*proof*⟩

lemma *cfrac-lim-tl*:
assumes $\neg$*cfrac-is-int c*
shows *cfrac-lim* (*cfrac-tl c*) *= 1 / (cfrac-lim c − cfrac-nth c 0)*
⟨*proof*⟩

lemma *cfrac-remainder-Suc′*:
assumes $n <$ *cfrac-length c*
shows *cfrac-remainder c* (*Suc n*) $*$ (*cfrac-remainder c n − cfrac-nth c n*) *= 1*
⟨*proof*⟩

lemma *cfrac-remainder-Suc*:
assumes $n < \textit{cfrac-length}\ c$
shows $\textit{cfrac-remainder}\ c\ (\textit{Suc}\ n) = 1\ /\ (\textit{cfrac-remainder}\ c\ n - \textit{cfrac-nth}\ c\ n)$
⟨*proof*⟩

lemma *cfrac-remainder-0* [*simp*]: $\textit{cfrac-remainder}\ c\ 0 = \textit{cfrac-lim}\ c$
⟨*proof*⟩

context
fixes $c\ h\ k\ x$
defines $h \equiv \textit{conv-num}\ c$ **and** $k \equiv \textit{conv-denom}\ c$ **and** $x \equiv \textit{cfrac-remainder}\ c$
begin

lemma *cfrac-lim-eq-num-denom-remainder-aux*:
assumes $\textit{Suc}\ (\textit{Suc}\ n) \leq \textit{cfrac-length}\ c$
shows $\textit{cfrac-lim}\ c * (k\ (\textit{Suc}\ n) * x\ (\textit{Suc}\ (\textit{Suc}\ n)) + k\ n) = h\ (\textit{Suc}\ n) * x\ (\textit{Suc}\ (\textit{Suc}\ n)) + h\ n$
⟨*proof*⟩

lemma *cfrac-remainder-nonneg*: $\textit{cfrac-nth}\ c\ n \geq 0 \Longrightarrow \textit{cfrac-remainder}\ c\ n \geq 0$
⟨*proof*⟩

lemma *cfrac-remainder-pos*: $\textit{cfrac-nth}\ c\ n > 0 \Longrightarrow \textit{cfrac-remainder}\ c\ n > 0$
⟨*proof*⟩

lemma *cfrac-lim-eq-num-denom-remainder*:
assumes $\textit{Suc}\ (\textit{Suc}\ n) < \textit{cfrac-length}\ c$
shows $\textit{cfrac-lim}\ c = (h\ (\textit{Suc}\ n) * x\ (\textit{Suc}\ (\textit{Suc}\ n)) + h\ n)\ /\ (k\ (\textit{Suc}\ n) * x\ (\textit{Suc}\ (\textit{Suc}\ n)) + k\ n)$
⟨*proof*⟩

lemma *abs-diff-successive-convs*:
shows $|\textit{conv}\ c\ (\textit{Suc}\ n) - \textit{conv}\ c\ n| = 1\ /\ (k\ n * k\ (\textit{Suc}\ n))$
⟨*proof*⟩

lemma *conv-denom-plus2-ratio-ge*: $k\ (\textit{Suc}\ (\textit{Suc}\ n)) \geq 2 * k\ n$
⟨*proof*⟩

end

lemma *conv'-cfrac-remainder*:
assumes $n < \textit{cfrac-length}\ c$
shows $\textit{conv'}\ c\ n\ (\textit{cfrac-remainder}\ c\ n) = \textit{cfrac-lim}\ c$
⟨*proof*⟩

lemma *cfrac-lim-rational* [*intro*]:
assumes $\textit{cfrac-length}\ c < \infty$
shows $\textit{cfrac-lim}\ c \in \mathbb{Q}$

⟨*proof*⟩

lemma *linfinite-cfrac-of-real-aux*:
$x \notin \mathbb{Q} \Longrightarrow x \in \{0<..<1\} \Longrightarrow$ *linfinite* (*cfrac-of-real-aux* x)
⟨*proof*⟩

lemma *cfrac-length-of-real-irrational*:
assumes $x \notin \mathbb{Q}$
shows *cfrac-length* (*cfrac-of-real* x) $= \infty$
⟨*proof*⟩

lemma *cfrac-length-of-real-reduce*:
assumes $x \notin \mathbb{Z}$
shows *cfrac-length* (*cfrac-of-real* x) = *eSuc* (*cfrac-length* (*cfrac-of-real* (*1 / frac* x)))
⟨*proof*⟩

lemma *cfrac-length-of-real-int* [*simp*]: $x \in \mathbb{Z} \Longrightarrow$ *cfrac-length* (*cfrac-of-real* x) = *0*
⟨*proof*⟩

lemma *conv-cfrac-of-real-le-ge*:
assumes $n \leq$ *cfrac-length* (*cfrac-of-real* x)
shows *if even n then conv* (*cfrac-of-real* x) $n \leq x$ *else conv* (*cfrac-of-real* x) $n \geq x$
⟨*proof*⟩

lemma *cfrac-lim-of-real* [*simp*]: *cfrac-lim* (*cfrac-of-real* x) $= x$
⟨*proof*⟩

lemma *Ints-add-left-cancel*: $x \in \mathbb{Z} \Longrightarrow x + y \in \mathbb{Z} \longleftrightarrow y \in \mathbb{Z}$
⟨*proof*⟩

lemma *Ints-add-right-cancel*: $y \in \mathbb{Z} \Longrightarrow x + y \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

lemma *cfrac-of-real-conv'*:
fixes $m\ n$ :: *nat*
assumes $x > 1$ $m < n$
shows *cfrac-nth* (*cfrac-of-real* (*conv'* $c\ n\ x$)) $m =$ *cfrac-nth* $c\ m$
⟨*proof*⟩

lemma *cfrac-lim-irrational*:
assumes [*simp*]: *cfrac-length* $c = \infty$
shows *cfrac-lim* $c \notin \mathbb{Q}$
⟨*proof*⟩

lemma *cfrac-infinite-iff*: *cfrac-length* $c = \infty \longleftrightarrow$ *cfrac-lim* $c \notin \mathbb{Q}$
⟨*proof*⟩

lemma *cfrac-lim-rational-iff*: *cfrac-lim c* ∈ ℚ ⟷ *cfrac-length c* ≠ ∞
⟨*proof*⟩

lemma *cfrac-of-real-infinite-iff* [*simp*]: *cfrac-length* (*cfrac-of-real x*) = ∞ ⟷ *x* ∉ ℚ
⟨*proof*⟩

lemma *cfrac-remainder-rational-iff* [*simp*]:
cfrac-remainder c n ∈ ℚ ⟷ *cfrac-length c* < ∞
⟨*proof*⟩

lift-definition *cfrac-cons* :: *int* ⇒ *cfrac* ⇒ *cfrac* **is**
λ*a bs*. *case bs of* (*b*, *bs*) ⇒ *if b* ≤ *0 then* (*1*, *LNil*) *else* (*a*, *LCons* (*nat* (*b* − *1*)) *bs*) ⟨*proof*⟩

lemma *cfrac-nth-cons*:
assumes *cfrac-nth x 0* ≥ *1*
shows *cfrac-nth* (*cfrac-cons a x*) *n* = (*if n* = *0 then a else cfrac-nth x* (*n* − *1*))
⟨*proof*⟩

lemma *cfrac-length-cons* [*simp*]:
assumes *cfrac-nth x 0* ≥ *1*
shows *cfrac-length* (*cfrac-cons a x*) = *eSuc* (*cfrac-length x*)
⟨*proof*⟩

lemma *cfrac-tl-cons* [*simp*]:
assumes *cfrac-nth x 0* ≥ *1*
shows *cfrac-tl* (*cfrac-cons a x*) = *x*
⟨*proof*⟩

lemma *cfrac-cons-tl*:
assumes ¬*cfrac-is-int x*
shows *cfrac-cons* (*cfrac-nth x 0*) (*cfrac-tl x*) = *x*
⟨*proof*⟩

1.3 Non-canonical continued fractions

As we will show later, every irrational number has a unique continued fraction expansion. Every rational number x, however, has two different expansions: The canonical one ends with some number n (which is not equal to 1 unless $x = 1$) and a non-canonical one which ends with $n - 1, 1$.

We now define this non-canonical expansion analogously to the canonical one before and show its characteristic properties:

- The length of the non-canonical expansion is one greater than that of the canonical one.
- If the expansion is infinite, the non-canonical and the canonical one

coincide.

- The coefficients of the expansions are all equal except for the last two. The last coefficient of the non-canonical expansion is always 1, and the second to last one is the last of the canonical one minus 1.

lift-definition *cfrac-canonical* :: *cfrac* ⇒ *bool* **is**
  λ(*x*, *xs*). ¬*lfinite xs* ∨ *lnull xs* ∨ *llast xs* ≠ *0* ⟨*proof*⟩

lemma *cfrac-canonical* [*code*]:
  cfrac-canonical (*CFrac x xs*) ⟷ *lnull xs* ∨ *llast xs* ≠ *0* ∨ ¬*lfinite xs*
  ⟨*proof*⟩

lemma *cfrac-canonical-iff*:
  cfrac-canonical c ⟷
    cfrac-length c ∈ {*0*, ∞} ∨ *cfrac-nth c* (*the-enat* (*cfrac-length c*)) ≠ *1*
⟨*proof*⟩

lemma *llast-cfrac-of-real-aux-nonzero*:
  assumes *lfinite* (*cfrac-of-real-aux x*) ¬*lnull* (*cfrac-of-real-aux x*)
  shows *llast* (*cfrac-of-real-aux x*) ≠ *0*
  ⟨*proof*⟩

lemma *cfrac-canonical-of-real* [*intro*]: *cfrac-canonical* (*cfrac-of-real x*)
  ⟨*proof*⟩

primcorec *cfrac-of-real-alt-aux* :: *real* ⇒ *nat llist* **where**
  cfrac-of-real-alt-aux x =
    (*if x* ∈ {*0*<..<*1*} *then*
      if 1 / *x* ∈ ℤ *then*
        LCons (*nat* ⌊*1*/*x*⌋ − *2*) (*LCons 0 LNil*)
      else LCons (*nat* ⌊*1*/*x*⌋ − *1*) (*cfrac-of-real-alt-aux* (*frac* (*1*/*x*)))
     else LNil)

lemma *cfrac-of-real-aux-alt-LNil* [*simp*]: *x* ∉ {*0*<..<*1*} ⟹ *cfrac-of-real-alt-aux x* = *LNil*
  ⟨*proof*⟩

lemma *cfrac-of-real-aux-alt-0* [*simp*]: *cfrac-of-real-alt-aux 0* = *LNil*
  ⟨*proof*⟩

lemma *cfrac-of-real-aux-alt-eq-LNil-iff* [*simp*]: *cfrac-of-real-alt-aux x* = *LNil* ⟷ *x* ∉ {*0*<..<*1*}
  ⟨*proof*⟩

lift-definition *cfrac-of-real-alt* :: *real* ⇒ *cfrac* **is**
  λ*x*. *if x* ∈ ℤ *then* (⌊*x*⌋ − *1*, *LCons 0 LNil*) *else* (⌊*x*⌋, *cfrac-of-real-alt-aux* (*frac x*)) ⟨*proof*⟩

lemma *cfrac-tl-of-real-alt*:
assumes $x \notin \mathbb{Z}$
shows *cfrac-tl* (*cfrac-of-real-alt x*) = *cfrac-of-real-alt* (*1* / *frac x*)
⟨*proof*⟩

lemma *cfrac-nth-of-real-alt-Suc*:
assumes $x \notin \mathbb{Z}$
shows *cfrac-nth* (*cfrac-of-real-alt x*) (*Suc n*) = *cfrac-nth* (*cfrac-of-real-alt* (*1* / *frac x*)) *n*
⟨*proof*⟩

lemma *cfrac-nth-gt0-of-real-int* [*simp*]:
m > *0* ⟹ *cfrac-nth* (*cfrac-of-real* (*of-int n*)) *m* = *1*
⟨*proof*⟩

lemma *cfrac-nth-0-of-real-alt-int* [*simp*]:
cfrac-nth (*cfrac-of-real-alt* (*of-int n*)) *0* = *n* − *1*
⟨*proof*⟩

lemma *cfrac-nth-gt0-of-real-alt-int* [*simp*]:
m > *0* ⟹ *cfrac-nth* (*cfrac-of-real-alt* (*of-int n*)) *m* = *1*
⟨*proof*⟩

lemma *llength-cfrac-of-real-alt-aux*:
assumes $x \in \{0<..<1\}$
shows *llength* (*cfrac-of-real-alt-aux x*) = *eSuc* (*llength* (*cfrac-of-real-aux x*))
⟨*proof*⟩

lemma *cfrac-length-of-real-alt*:
cfrac-length (*cfrac-of-real-alt x*) = *eSuc* (*cfrac-length* (*cfrac-of-real x*))
⟨*proof*⟩

lemma *cfrac-of-real-alt-aux-eq-regular*:
assumes $x \in \{0<..<1\}$ *llength* (*cfrac-of-real-aux x*) = ∞
shows *cfrac-of-real-alt-aux x* = *cfrac-of-real-aux x*
⟨*proof*⟩

lemma *cfrac-of-real-alt-irrational* [*simp*]:
assumes $x \notin \mathbb{Q}$
shows *cfrac-of-real-alt x* = *cfrac-of-real x*
⟨*proof*⟩

lemma *cfrac-nth-of-real-alt-0*:
cfrac-nth (*cfrac-of-real-alt x*) *0* = (*if x* ∈ ℤ *then* ⌊*x*⌋ − *1 else* ⌊*x*⌋)
⟨*proof*⟩

lemma *cfrac-nth-of-real-alt*:
fixes *n* :: *nat* **and** *x* :: *real*
defines *c* ≡ *cfrac-of-real x*

defines $c' \equiv$ *cfrac-of-real-alt* x
defines $l \equiv$ *cfrac-length* c
shows *cfrac-nth* c' $n =$
$($*if enat* $n = l$ *then*
cfrac-nth c $n - 1$
else if enat $n = l + 1$ *then*
1
else
cfrac-nth c $n)$
⟨*proof*⟩

lemma *cfrac-of-real-length-eq-0-iff*: *cfrac-length* (*cfrac-of-real* x) $= 0 \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

lemma *conv'-cong*:
assumes $(\bigwedge k.\ k < n \Longrightarrow$ *cfrac-nth* c $k =$ *cfrac-nth* c' $k)$ $n = n'$ $x = y$
shows *conv'* c n $x =$ *conv'* c' n' y
⟨*proof*⟩

lemma *conv-cong*:
assumes $(\bigwedge k.\ k \leq n \Longrightarrow$ *cfrac-nth* c $k =$ *cfrac-nth* c' $k)$ $n = n'$
shows *conv* c $n =$ *conv* c' n'
⟨*proof*⟩

lemma *conv'-cfrac-of-real-alt*:
assumes *enat* $n \leq$ *cfrac-length* (*cfrac-of-real* x)
shows *conv'* (*cfrac-of-real-alt* x) n $y =$ *conv'* (*cfrac-of-real* x) n y
⟨*proof*⟩

lemma *cfrac-lim-of-real-alt* [*simp*]: *cfrac-lim* (*cfrac-of-real-alt* x) $= x$
⟨*proof*⟩

lemma *cfrac-eqI*:
assumes *cfrac-length* $c =$ *cfrac-length* c' **and** $\bigwedge n.$ *cfrac-nth* c $n =$ *cfrac-nth* c' n
shows $c = c'$
⟨*proof*⟩

lemma *cfrac-eq-0I*:
assumes *cfrac-lim* $c = 0$ *cfrac-nth* c $0 \geq 0$
shows $c = 0$
⟨*proof*⟩

lemma *cfrac-eq-1I*:
assumes *cfrac-lim* $c = 1$ *cfrac-nth* c $0 \neq 0$
shows $c = 1$
⟨*proof*⟩

lemma *cfrac-coinduct* [*coinduct type*: *cfrac*]:
assumes R *c1 c2*

assumes *IH*: $\bigwedge$*c1 c2. R c1 c2* $\Longrightarrow$
cfrac-is-int c1 = cfrac-is-int c2 $\wedge$
cfrac-nth c1 0 = cfrac-nth c2 0 $\wedge$
(¬*cfrac-is-int c1* $\longrightarrow$ ¬*cfrac-is-int c2* $\longrightarrow$ *R* (*cfrac-tl c1*) (*cfrac-tl c2*))
shows *c1 = c2*
⟨*proof*⟩

lemma *cfrac-nth-0-cases*:
cfrac-nth c 0 = $\lfloor$*cfrac-lim c*$\rfloor$ $\vee$ *cfrac-nth c 0* = $\lfloor$*cfrac-lim c*$\rfloor$ − *1* $\wedge$ *cfrac-tl c* = *1*
⟨*proof*⟩

lemma *cfrac-length-1* [*simp*]: *cfrac-length 1 = 0*
⟨*proof*⟩

lemma *cfrac-nth-1* [*simp*]: *cfrac-nth 1 m = 1*
⟨*proof*⟩

lemma *cfrac-lim-1* [*simp*]: *cfrac-lim 1 = 1*
⟨*proof*⟩

lemma *cfrac-nth-0-not-int*:
assumes *cfrac-lim c* $\notin$ $\mathbb{Z}$
shows *cfrac-nth c 0* = $\lfloor$*cfrac-lim c*$\rfloor$
⟨*proof*⟩

lemma *cfrac-of-real-cfrac-lim-irrational*:
assumes *cfrac-lim c* $\notin$ $\mathbb{Q}$
shows *cfrac-of-real* (*cfrac-lim c*) = *c*
⟨*proof*⟩

lemma *cfrac-eqI-first*:
assumes ¬*cfrac-is-int c* ¬*cfrac-is-int c′*
assumes *cfrac-nth c 0 = cfrac-nth c′ 0* **and** *cfrac-tl c = cfrac-tl c′*
shows *c = c′*
⟨*proof*⟩

lemma *cfrac-is-int-of-real-iff*: *cfrac-is-int* (*cfrac-of-real x*) $\longleftrightarrow$ *x* $\in$ $\mathbb{Z}$
⟨*proof*⟩

lemma *cfrac-not-is-int-of-real-alt*: ¬*cfrac-is-int* (*cfrac-of-real-alt x*)
⟨*proof*⟩

lemma *cfrac-tl-of-real-alt-of-int* [*simp*]: *cfrac-tl* (*cfrac-of-real-alt* (*of-int n*)) = *1*
⟨*proof*⟩

lemma *cfrac-is-intI*:
assumes *cfrac-nth c 0* $\geq$ $\lfloor$*cfrac-lim c*$\rfloor$ **and** *cfrac-lim c* $\in$ $\mathbb{Z}$

shows *cfrac-is-int c*
⟨*proof*⟩

lemma *cfrac-eq-of-intI*:
assumes *cfrac-nth c 0* ≥ ⌊*cfrac-lim c*⌋ **and** *cfrac-lim c* ∈ ℤ
shows *c* = *cfrac-of-int* ⌊*cfrac-lim c*⌋
⟨*proof*⟩

lemma *cfrac-lim-of-int* [*simp*]: *cfrac-lim* (*cfrac-of-int n*) = *of-int n*
⟨*proof*⟩

lemma *cfrac-of-real-of-int* [*simp*]: *cfrac-of-real* (*of-int n*) = *cfrac-of-int n*
⟨*proof*⟩

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

lemma *cfrac-int-cases*:
assumes *cfrac-lim c* = *of-int n*
shows *c* = *cfrac-of-int n* ∨ *c* = *cfrac-of-real-alt* (*of-int n*)
⟨*proof*⟩

lemma *cfrac-cases*:
c ∈ {*cfrac-of-real* (*cfrac-lim c*), *cfrac-of-real-alt* (*cfrac-lim c*)}
⟨*proof*⟩

lemma *cfrac-lim-eq-iff*:
assumes *cfrac-length c* = ∞ ∨ *cfrac-length c′* = ∞
shows *cfrac-lim c* = *cfrac-lim c′* ⟷ *c* = *c′*
⟨*proof*⟩

lemma *floor-cfrac-remainder*:
assumes *cfrac-length c* = ∞
shows ⌊*cfrac-remainder c n*⌋ = *cfrac-nth c n*
⟨*proof*⟩

1.4 Approximation properties

In this section, we will show that convergents of the continued fraction expansion of a number x are good approximations of x, and in a certain sense, the reverse holds as well.

lemma *sgn-of-int*:
sgn (*of-int x* :: *'a* :: {*linordered-idom*}) = *of-int* (*sgn x*)
⟨*proof*⟩

lemma *conv-ge-one*: *cfrac-nth c 0* > *0* ⟹ *conv c n* ≥ *1*
⟨*proof*⟩

context

fixes c h k
defines $h \equiv$ *conv-num* c **and** $k \equiv$ *conv-denom* c
begin

lemma *abs-diff-le-abs-add*:
fixes x y :: *real*
assumes $x \geq 0 \wedge y \geq 0 \vee x \leq 0 \wedge y \leq 0$
shows $|x - y| \leq |x + y|$
⟨*proof*⟩

lemma *abs-diff-less-abs-add*:
fixes x y :: *real*
assumes $x > 0 \wedge y > 0 \vee x < 0 \wedge y < 0$
shows $|x - y| < |x + y|$
⟨*proof*⟩

lemma *abs-diff-le-imp-same-sign*:
assumes $|x - y| \leq d$ $d < |y|$
shows *sgn* $x =$ *sgn* (y::*real*)
⟨*proof*⟩

lemma *conv-nonpos*:
assumes *cfrac-nth* c $0 < 0$
shows *conv* c $n \leq 0$
⟨*proof*⟩

lemma *cfrac-lim-nonpos*:
assumes *cfrac-nth* c $0 < 0$
shows *cfrac-lim* $c \leq 0$
⟨*proof*⟩

lemma *conv-num-nonpos*:
assumes *cfrac-nth* c $0 < 0$
shows h $n \leq 0$
⟨*proof*⟩

lemma *conv-best-approximation-aux*:
cfrac-lim $c \geq 0 \wedge h$ $n \geq 0 \vee$ *cfrac-lim* $c \leq 0 \wedge h$ $n \leq 0$
⟨*proof*⟩

lemma *conv-best-approximation-ex*:
fixes a b :: *int* **and** x :: *real*
assumes $n \leq$ *cfrac-length* c
assumes $0 < b$ **and** $b \leq k$ n **and** *coprime* a b **and** $n > 0$
assumes $(a, b) \neq (h$ n, k $n)$
assumes $\neg$(*cfrac-length* $c = 1 \wedge n = 0$)
assumes *Suc* $n \neq$ *cfrac-length* $c \vee$ *cfrac-canonical* c
defines $x \equiv$ *cfrac-lim* c
shows $|k$ $n * x - h$ $n| < |b * x - a|$

⟨*proof*⟩

lemma *conv-best-approximation-ex-weak*:
fixes *a b* :: *int* **and** *x* :: *real*
assumes $n \leq$ *cfrac-length c*
assumes $0 < b$ **and** $b < k$ (*Suc n*) **and** *coprime a b*
defines $x \equiv$ *cfrac-lim c*
shows $|k\ n * x - h\ n| \leq |b * x - a|$
⟨*proof*⟩

lemma *cfrac-canonical-reduce*:
cfrac-canonical c $\longleftrightarrow$
cfrac-is-int c $\vee$ $\neg$*cfrac-is-int c* $\wedge$ *cfrac-tl c* $\neq 1 \wedge$ *cfrac-canonical* (*cfrac-tl c*)
⟨*proof*⟩

lemma *cfrac-nth-0-conv-floor*:
assumes *cfrac-canonical c* $\vee$ *cfrac-length c* $\neq 1$
shows *cfrac-nth c 0* $= \lfloor$*cfrac-lim c*$\rfloor$
⟨*proof*⟩

lemma *conv-best-approximation-ex-nat*:
fixes *a b* :: *nat* **and** *x* :: *real*
assumes $n \leq$ *cfrac-length c* $0 < b$ $b < k$ (*Suc n*) *coprime a b*
shows $|k\ n *$ *cfrac-lim c* $- h\ n| \leq |b *$ *cfrac-lim c* $- a|$
⟨*proof*⟩

lemma *abs-mult-nonneg-left*:
assumes $x \geq (0 :: 'a ::$ {*ordered-ab-group-add-abs*, *idom-abs-sgn*}$)$
shows $x * |y| = |x * y|$
⟨*proof*⟩

Any convergent of the continued fraction expansion of x is a best approximation of x, i.e. there is no other number with a smaller denominator that approximates it better.

lemma *conv-best-approximation*:
fixes *a b* :: *int* **and** *x* :: *real*
assumes $n \leq$ *cfrac-length c*
assumes $0 < b$ **and** $b < k\ n$ **and** *coprime a b*
defines $x \equiv$ *cfrac-lim c*
shows $|x -$ *conv c n*$| \leq |x - a\ /\ b|$
⟨*proof*⟩

lemma *conv-denom-partition*:
assumes $y > 0$
shows $\exists!n.\ y \in \{k\ n..<k\ (Suc\ n)\}$
⟨*proof*⟩

A fraction that approximates a real number x sufficiently well (in a certain sense) is a convergent of its continued fraction expansion.

lemma *frac-is-convergentI*:
fixes $a\ b :: int$ **and** $x :: real$
defines $x \equiv cfrac\text{-}lim\ c$
assumes $b > 0$ **and** $coprime\ a\ b$ **and** $|x - a\ /\ b| < 1\ /\ (2 * b^2)$
shows $\exists n.\ enat\ n \leq cfrac\text{-}length\ c \wedge (a, b) = (h\ n, k\ n)$
⟨*proof*⟩

end

1.5 Efficient code for convergents

function *conv-gen* :: $(nat \Rightarrow int) \Rightarrow int \times int \times nat \Rightarrow nat \Rightarrow int$ **where**
$conv\text{-}gen\ c\ (a, b, n)\ N =$
$(if\ n > N\ then\ b\ else\ conv\text{-}gen\ c\ (b, b * c\ n + a, Suc\ n)\ N)$
⟨*proof*⟩
termination ⟨*proof*⟩

lemmas [*simp del*] = *conv-gen.simps*

lemma *conv-gen-aux-simps* [*simp*]:
$n > N \Longrightarrow conv\text{-}gen\ c\ (a, b, n)\ N = b$
$n \leq N \Longrightarrow conv\text{-}gen\ c\ (a, b, n)\ N = conv\text{-}gen\ c\ (b, b * c\ n + a, Suc\ n)\ N$
⟨*proof*⟩

lemma *conv-num-eq-conv-gen-aux*:
$Suc\ n \leq N \Longrightarrow conv\text{-}num\ c\ n = b * cfrac\text{-}nth\ c\ n + a \Longrightarrow$
$conv\text{-}num\ c\ (Suc\ n) = conv\text{-}num\ c\ n * cfrac\text{-}nth\ c\ (Suc\ n) + b \Longrightarrow$
$conv\text{-}num\ c\ N = conv\text{-}gen\ (cfrac\text{-}nth\ c)\ (a, b, n)\ N$
⟨*proof*⟩

lemma *conv-denom-eq-conv-gen-aux*:
$Suc\ n \leq N \Longrightarrow conv\text{-}denom\ c\ n = b * cfrac\text{-}nth\ c\ n + a \Longrightarrow$
$conv\text{-}denom\ c\ (Suc\ n) = conv\text{-}denom\ c\ n * cfrac\text{-}nth\ c\ (Suc\ n) + b \Longrightarrow$
$conv\text{-}denom\ c\ N = conv\text{-}gen\ (cfrac\text{-}nth\ c)\ (a, b, n)\ N$
⟨*proof*⟩

lemma *conv-num-code* [*code*]: $conv\text{-}num\ c\ n = conv\text{-}gen\ (cfrac\text{-}nth\ c)\ (0, 1, 0)\ n$
⟨*proof*⟩

lemma *conv-denom-code* [*code*]: $conv\text{-}denom\ c\ n = conv\text{-}gen\ (cfrac\text{-}nth\ c)\ (1, 0, 0)\ n$
⟨*proof*⟩

definition *conv-num-fun* **where** $conv\text{-}num\text{-}fun\ c = conv\text{-}gen\ c\ (0, 1, 0)$
definition *conv-denom-fun* **where** $conv\text{-}denom\text{-}fun\ c = conv\text{-}gen\ c\ (1, 0, 0)$

lemma
assumes *is-cfrac c*
shows *conv-num-fun-eq*: $conv\text{-}num\text{-}fun\ c\ n = conv\text{-}num\ (cfrac\ c)\ n$

and *conv-denom-fun-eq*: *conv-denom-fun c n* = *conv-denom* (*cfrac c*) *n*
⟨*proof*⟩

1.6 Computing the continued fraction expansion of a rational number

function *cfrac-list-of-rat* :: *int* × *int* ⇒ *int list* **where**
cfrac-list-of-rat (*a*, *b*) =
(*if b* = *0 then* [*0*]
else a div b # (*if a mod b* = *0 then* [] *else cfrac-list-of-rat* (*b*, *a mod b*)))
⟨*proof*⟩
termination
⟨*proof*⟩

lemmas [*simp del*] = *cfrac-list-of-rat.simps*

lemma *cfrac-list-of-rat-correct*:
(*let xs* = *cfrac-list-of-rat* (*a*, *b*); *c* = *cfrac-of-real* (*a* / *b*)
in length xs = *cfrac-length c* + *1* ∧ (∀ *i*<*length xs*. *xs* ! *i* = *cfrac-nth c i*))
⟨*proof*⟩

lemma *conv-num-cong*:
assumes (⋀*k*. *k* ≤ *n* ⟹ *cfrac-nth c k* = *cfrac-nth c*′ *k*) *n* = *n*′
shows *conv-num c n* = *conv-num c*′ *n*
⟨*proof*⟩

lemma *conv-denom-cong*:
assumes (⋀*k*. *k* ≤ *n* ⟹ *cfrac-nth c k* = *cfrac-nth c*′ *k*) *n* = *n*′
shows *conv-denom c n* = *conv-denom c*′ *n*′
⟨*proof*⟩

lemma *cfrac-lim-diff-le*:
assumes ∀ *k*≤*Suc n*. *cfrac-nth c1 k* = *cfrac-nth c2 k*
assumes *n* ≤ *cfrac-length c1 n* ≤ *cfrac-length c2*
shows |*cfrac-lim c1* − *cfrac-lim c2*| ≤ *2* / (*conv-denom c1 n* ∗ *conv-denom c1* (*Suc n*))
⟨*proof*⟩

lemma *of-int-leI*: *n* ≤ *m* ⟹ (*of-int n* :: *'a* :: *linordered-idom*) ≤ *of-int m*
⟨*proof*⟩

lemma *cfrac-lim-diff-le'*:
assumes ∀ *k*≤*Suc n*. *cfrac-nth c1 k* = *cfrac-nth c2 k*
assumes *n* ≤ *cfrac-length c1 n* ≤ *cfrac-length c2*
shows |*cfrac-lim c1* − *cfrac-lim c2*| ≤ *2* / (*fib* (*n*+*1*) ∗ *fib* (*n*+*2*))
⟨*proof*⟩

end

2 Quadratic Irrationals

theory *Quadratic-Irrationals*
imports
Continued-Fractions
HOL−Computational-Algebra.Computational-Algebra
HOL−Library.Discrete
Coinductive.Coinductive-Stream
begin

lemma *snth-cycle*:
assumes $xs \neq []$
shows $snth\ (cycle\ xs)\ n = xs\ !\ (n\ mod\ length\ xs)$
⟨*proof*⟩

2.1 Basic results on rationality of square roots

lemma *inverse-in-Rats-iff* [*simp*]: $inverse\ (x :: real) \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
⟨*proof*⟩

lemma *nonneg-sqrt-nat-or-irrat*:
assumes $x \hat{} 2 = real\ a$ **and** $x \geq 0$
shows $x \in \mathbb{N} \vee x \notin \mathbb{Q}$
⟨*proof*⟩

A square root of a natural number is either an integer or irrational.

corollary *sqrt-nat-or-irrat*:
assumes $x \hat{} 2 = real\ a$
shows $x \in \mathbb{Z} \vee x \notin \mathbb{Q}$
⟨*proof*⟩

corollary *sqrt-nat-or-irrat′*:
$sqrt\ (real\ a) \in \mathbb{N} \vee sqrt\ (real\ a) \notin \mathbb{Q}$
⟨*proof*⟩

The square root of a natural number n is again a natural number iff n *is a perfect square*.

corollary *sqrt-nat-iff-is-square*:
$sqrt\ (real\ n) \in \mathbb{N} \longleftrightarrow is\text{-}square\ n$
⟨*proof*⟩

corollary *irrat-sqrt-nonsquare*: $\neg is\text{-}square\ n \Longrightarrow sqrt\ (real\ n) \notin \mathbb{Q}$
⟨*proof*⟩

lemma *sqrt-of-nat-in-Rats-iff*: $sqrt\ (real\ n) \in \mathbb{Q} \longleftrightarrow is\text{-}square\ n$
⟨*proof*⟩

lemma *Discrete-sqrt-altdef*: $Discrete.sqrt\ n = nat\ \lfloor sqrt\ n \rfloor$
⟨*proof*⟩

2.2 Definition of quadratic irrationals

Irrational real numbers x that satisfy a quadratic equation $ax^2 + bx + c = 0$ with a, b, c not all equal to 0 are called *quadratic irrationals*. These are of the form $p + q\sqrt{d}$ for rational numbers p, q and a positive integer d.

inductive *quadratic-irrational* :: *real* ⇒ *bool* **where**
  $x \notin \mathbb{Q} \Longrightarrow$ *real-of-int a* ∗ *x* ^ *2* + *real-of-int b* ∗ *x* + *real-of-int c* = *0* ⟹
    $a \neq 0 \vee b \neq 0 \vee c \neq 0 \Longrightarrow$ *quadratic-irrational x*

lemma *quadratic-irrational-sqrt* [*intro*]:
  assumes ¬*is-square n*
  shows *quadratic-irrational* (*sqrt* (*real n*))
  ⟨*proof*⟩

lemma *quadratic-irrational-uminus* [*intro*]:
  assumes *quadratic-irrational x*
  shows *quadratic-irrational* (−*x*)
  ⟨*proof*⟩

lemma *quadratic-irrational-uminus-iff* [*simp*]:
  quadratic-irrational (−*x*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-plus-int* [*intro*]:
  assumes *quadratic-irrational x*
  shows *quadratic-irrational* (*x* + *of-int n*)
  ⟨*proof*⟩

lemma *quadratic-irrational-plus-int-iff* [*simp*]:
  quadratic-irrational (*x* + *of-int n*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-minus-int-iff* [*simp*]:
  quadratic-irrational (*x* − *of-int n*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-plus-nat-iff* [*simp*]:
  quadratic-irrational (*x* + *of-nat n*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-minus-nat-iff* [*simp*]:
  quadratic-irrational (*x* − *of-nat n*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-plus-1-iff* [*simp*]:
  quadratic-irrational (*x* + *1*) ⟷ *quadratic-irrational x*
  ⟨*proof*⟩

lemma *quadratic-irrational-minus-1-iff* [*simp*]:

quadratic-irrational $(x - 1) \longleftrightarrow$ *quadratic-irrational* x
⟨*proof*⟩

lemma *quadratic-irrational-plus-numeral-iff* [*simp*]:
quadratic-irrational $(x +$ *numeral* $n) \longleftrightarrow$ *quadratic-irrational* x
⟨*proof*⟩

lemma *quadratic-irrational-minus-numeral-iff* [*simp*]:
quadratic-irrational $(x -$ *numeral* $n) \longleftrightarrow$ *quadratic-irrational* x
⟨*proof*⟩

lemma *quadratic-irrational-inverse*:
assumes *quadratic-irrational* x
shows *quadratic-irrational* (*inverse* x)
⟨*proof*⟩

lemma *quadratic-irrational-inverse-iff* [*simp*]:
quadratic-irrational (*inverse* x) $\longleftrightarrow$ *quadratic-irrational* x
⟨*proof*⟩

lemma *quadratic-irrational-cfrac-remainder-iff*:
quadratic-irrational (*cfrac-remainder* c n) $\longleftrightarrow$ *quadratic-irrational* (*cfrac-lim* c)
⟨*proof*⟩

2.3 Real solutions of quadratic equations

For the next result, we need some basic properties of real solutions to quadratic equations.

lemma *quadratic-equation-reals*:
fixes a b c :: *real*
defines $f \equiv (\lambda x.\ a * x\ \hat{}\ 2 + b * x + c)$
defines *discr* $\equiv (b\hat{}2 - 4 * a * c)$
shows $\{x.\ f\ x = 0\} =$
(*if* $a = 0$ *then*
(*if* $b = 0$ *then if* $c = 0$ *then UNIV else* $\{\}$ *else* $\{-c/b\}$)
else if discr ≥ 0 *then* $\{(-b + sqrt\ discr) / (2 * a), (-b - sqrt\ discr) / (2 * a)\}$
else $\{\}$) (**is** *?th1*)
⟨*proof*⟩

lemma *finite-quadratic-equation-solutions-reals*:
fixes a b c :: *real*
defines *discr* $\equiv (b\hat{}2 - 4 * a * c)$
shows *finite* $\{x.\ a * x\ \hat{}\ 2 + b * x + c = 0\} \longleftrightarrow a \neq 0 \vee b \neq 0 \vee c \neq 0$
⟨*proof*⟩

lemma *card-quadratic-equation-solutions-reals*:
fixes a b c :: *real*
defines *discr* $\equiv (b\hat{}2 - 4 * a * c)$

shows *card* $\{x.\ a * x\ \widehat{}\ 2 + b * x + c = 0\} =$
(*if* $a = 0$ *then*
(*if* $b = 0$ *then* 0 *else* 1)
else if discr ≥ 0 *then if discr* $= 0$ *then* 1 *else* 2 *else* 0) (**is** *?th1*)
⟨*proof*⟩

lemma *card-quadratic-equation-solutions-reals-le-2*:
card $\{x :: real.\ a * x\ \widehat{}\ 2 + b * x + c = 0\} \leq 2$
⟨*proof*⟩

lemma *quadratic-equation-solution-rat-iff*:
fixes *a b c* :: *int* **and** *x y* :: *real*
defines $f \equiv (\lambda x{::}real.\ a * x\ \widehat{}\ 2 + b * x + c)$
defines *discr* $\equiv$ *nat* $(b\ \widehat{}\ 2 - 4 * a * c)$
assumes $a \neq 0$ $f\ x = 0$
shows $x \in \mathbb{Q} \longleftrightarrow$ *is-square discr*
⟨*proof*⟩

2.4 Periodic continued fractions and quadratic irrationals

We now show the main result: A positive irrational number has a periodic continued fraction expansion iff it is a quadratic irrational.

In principle, this statement naturally also holds for negative numbers, but the current formalisation of continued fractions only supports non-negative numbers. It also holds for rational numbers in some sense, since their continued fraction expansion is finite to begin with.

theorem *periodic-cfrac-imp-quadratic-irrational*:
assumes [*simp*]: *cfrac-length* $c = \infty$
and *period*: $l > 0$ $\bigwedge k.\ k \geq N \Longrightarrow$ *cfrac-nth* $c\ (k + l) =$ *cfrac-nth* $c\ k$
shows *quadratic-irrational* (*cfrac-lim* c)
⟨*proof*⟩

lift-definition *pperiodic-cfrac* :: *nat list* ⇒ *cfrac* **is**
$\lambda xs.$ *if* $xs = []$ *then* (0, *LNil*) *else*
(*int* (*hd xs*), *llist-of-stream* (*cycle* (*map* $(\lambda n.\ n - 1)$ (*tl xs* @ [*hd xs*])))) ⟨*proof*⟩

definition *periodic-cfrac* :: *int list* ⇒ *int list* ⇒ *cfrac* **where**
periodic-cfrac xs ys = *cfrac-of-stream* (*Stream.shift xs* (*Stream.cycle ys*))

lemma *periodic-cfrac-Nil* [*simp*]: *pperiodic-cfrac* [] = 0
⟨*proof*⟩

lemma *cfrac-length-pperiodic-cfrac* [*simp*]:
$xs \neq [] \Longrightarrow$ *cfrac-length* (*pperiodic-cfrac xs*) $= \infty$
⟨*proof*⟩

lemma *cfrac-nth-pperiodic-cfrac*:

assumes $xs \neq []$ **and** $0 \notin set\ xs$
shows *cfrac-nth* (*pperiodic-cfrac xs*) $n = xs\ !\ (n\ mod\ length\ xs)$
⟨*proof*⟩

definition *pperiodic-cfrac-info* :: *nat list* ⇒ *int* × *int* × *int***where**
pperiodic-cfrac-info xs =
(*let* l = *length xs*;
h = *conv-num-fun* ($\lambda n.\ xs\ !\ n$);
k = *conv-denom-fun* ($\lambda n.\ xs\ !\ n$);
$A = k\ (l - 1)$;
$B = h\ (l - 1) - (if\ l = 1\ then\ 0\ else\ k\ (l - 2))$;
$C = (if\ l = 1\ then\ -1\ else\ -h\ (l - 2))$
in $(B\hat{}2-4*A*C,\ B,\ 2 * A)$)

lemma *conv-gen-cong*:
assumes $\forall\, k{\in}\{n..N\}.\ f\ k = f'\ k$
shows *conv-gen f* $(a,b,n)\ N$ = *conv-gen f'* $(a,b,n)\ N$
⟨*proof*⟩

lemma
assumes $\forall\, k{\leq}n.\ c\ k = cfrac\text{-}nth\ c'\ k$
shows *conv-num-fun-eq'*: *conv-num-fun c n* = *conv-num c' n*
and *conv-denom-fun-eq'*: *conv-denom-fun c n* = *conv-denom c' n*
⟨*proof*⟩

lemma *gcd-minus-commute-left*: *gcd* $(a - b :: 'a :: ring\text{-}gcd)\ c$ = *gcd* $(b - a)\ c$
⟨*proof*⟩

lemma *gcd-minus-commute-right*: *gcd c* $(a - b :: 'a :: ring\text{-}gcd)$ = *gcd c* $(b - a)$
⟨*proof*⟩

lemma *periodic-cfrac-info-aux*:
fixes *D E F* :: *int*
assumes *pperiodic-cfrac-info xs* = (*D*, *E*, *F*)
assumes $xs \neq []$ $0 \notin set\ xs$
shows *cfrac-lim* (*pperiodic-cfrac xs*) = (*sqrt D* + *E*) / *F*
and $D > 0$ **and** $F > 0$
⟨*proof*⟩

We can now compute surd representations for (purely) periodic continued fractions, e.g. $[1, 1, 1, \ldots] = \frac{\sqrt{5}+1}{2}$:

value *pperiodic-cfrac-info* [*1*]

We can now compute surd representations for periodic continued fractions, e.g. $[\overline{1, 1, 1, 1, 6}] = \frac{\sqrt{13}+3}{4}$:

value *pperiodic-cfrac-info* [*1,1,1,1,6*]

With a little bit of work, one could also easily derive from this a version for non-purely periodic continued fraction.

Next, we show that any quadratic irrational has a periodic continued fraction expansion.

theorem *quadratic-irrational-imp-periodic-cfrac*:
assumes *quadratic-irrational* (*cfrac-lim e*)
obtains *N l* **where** $l > 0$ **and** $\bigwedge n\ m.\ n \geq N \Longrightarrow$ *cfrac-nth e* $(n + m * l) =$ *cfrac-nth e n*
and *cfrac-remainder e* $(N + l) =$ *cfrac-remainder e N*
and *cfrac-length e* $= \infty$
⟨*proof*⟩

theorem *periodic-cfrac-iff-quadratic-irrational*:
assumes $x \notin \mathbb{Q}$ $x \geq 0$
shows *quadratic-irrational x* $\longleftrightarrow$
$(\exists N\ l.\ l > 0 \wedge (\forall n{\geq}N.$ *cfrac-nth* (*cfrac-of-real x*) $(n + l) =$
cfrac-nth (*cfrac-of-real x*) $n))$
⟨*proof*⟩

The following result can e.g. be used to show that a number is *not* a quadratic irrational.

lemma *quadratic-irrational-cfrac-nth-range-finite*:
assumes *quadratic-irrational* (*cfrac-lim e*)
shows *finite* (*range* (*cfrac-nth e*))
⟨*proof*⟩

end

3 The continued fraction expansion of e

theory *E-CFrac*
imports
HOL−Analysis.Analysis
Continued-Fractions
Quadratic-Irrationals
begin

lemma *fact-real-at-top*: *filterlim* (*fact* :: *nat* ⇒ *real*) *at-top at-top*
⟨*proof*⟩

lemma *filterlim-div-nat-at-top*:
assumes *filterlim f at-top F m* > 0
shows *filterlim* ($\lambda x.$ *f x div m* :: *nat*) *at-top F*
⟨*proof*⟩

The continued fraction expansion of e has the form $[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, \ldots]$:

definition *e-cfrac* **where**
e-cfrac = *cfrac* ($\lambda n.$ *if n* $= 0$ *then 2 else if n mod 3* $= 2$ *then 2* $*$ (*Suc n div 3*) *else 1*)

lemma *cfrac-nth-e*:
cfrac-nth e-cfrac n = (*if n* = *0 then 2 else if n mod 3* = *2 then 2* ∗ (*Suc n div 3*) *else 1*)
⟨*proof*⟩

lemma *cfrac-length-e* [*simp*]: *cfrac-length e-cfrac* = ∞
⟨*proof*⟩

The formalised proof follows the one from Proof Wiki [2].

context
fixes *A B C* :: *nat* ⇒ *real* **and** *p q* :: *nat* ⇒ *int* **and** *a* :: *nat* ⇒ *int*
defines *A* ≡ (λ*n*. *integral* {*0..1*} (λ*x*. *exp x* ∗ *x* ^ *n* ∗ (*x* − *1*) ^ *n* / *fact n*))
and *B* ≡ (λ*n*. *integral* {*0..1*} (λ*x*. *exp x* ∗ *x* ^ *Suc n* ∗ (*x* − *1*) ^ *n* / *fact n*))
and *C* ≡ (λ*n*. *integral* {*0..1*} (λ*x*. *exp x* ∗ *x* ^ *n* ∗ (*x* − *1*) ^ *Suc n* / *fact n*))
and *p* ≡ (λ*n*. *if n* ≤ *1 then 1 else conv-num e-cfrac* (*n* − *2*))
and *q* ≡ (λ*n*. *if n* = *0 then 1 else if n* = *1 then 0 else conv-denom e-cfrac* (*n* − *2*))
and *a* ≡ (λ*n*. *if n mod 3* = *2 then 2* ∗ (*Suc n div 3*) *else 1*)
begin

lemma
assumes *n* ≥ *2*
shows *p-rec*: *p n* = *a* (*n* − *2*) ∗ *p* (*n* − *1*) + *p* (*n* − *2*) (**is** *?th1*)
and *q-rec*: *q n* = *a* (*n* − *2*) ∗ *q* (*n* − *1*) + *q* (*n* − *2*) (**is** *?th2*)
⟨*proof*⟩

lemma
assumes *n* ≥ *1*
shows *p-rec0*: *p* (*3* ∗ *n*) = *p* (*3* ∗ *n* − *1*) + *p* (*3* ∗ *n* − *2*)
and *q-rec0*: *q* (*3* ∗ *n*) = *q* (*3* ∗ *n* − *1*) + *q* (*3* ∗ *n* − *2*)
⟨*proof*⟩

lemma
assumes *n* ≥ *1*
shows *p-rec1*: *p* (*3* ∗ *n* + *1*) = *2* ∗ *int n* ∗ *p* (*3* ∗ *n*) + *p* (*3* ∗ *n* − *1*)
and *q-rec1*: *q* (*3* ∗ *n* + *1*) = *2* ∗ *int n* ∗ *q* (*3* ∗ *n*) + *q* (*3* ∗ *n* − *1*)
⟨*proof*⟩

lemma *p-rec2*: *p* (*3* ∗ *n* + *2*) = *p* (*3* ∗ *n* + *1*) + *p* (*3* ∗ *n*)
and *q-rec2*: *q* (*3* ∗ *n* + *2*) = *q* (*3* ∗ *n* + *1*) + *q* (*3* ∗ *n*)
⟨*proof*⟩

lemma *A-0*: *A 0* = *exp 1* − *1* **and** *B-0*: *B 0* = *1* **and** *C-0*: *C 0* = *2* − *exp 1*
⟨*proof*⟩

lemma *A-bound*: *norm* (*A n*) ≤ *exp 1* / *fact n*
⟨*proof*⟩

lemma *B-bound*: *norm* (*B n*) ≤ *exp 1* / *fact n*

⟨*proof*⟩

lemma *C-bound*: *norm* (*C n*) ≤ *exp 1* / *fact n*
⟨*proof*⟩

lemma *A-Suc*: *A* (*Suc n*) = −*B n* − *C n*
⟨*proof*⟩

lemma *B-Suc*: *B* (*Suc n*) = −*2* ∗ *Suc n* ∗ *A* (*Suc n*) + *C n*
⟨*proof*⟩

lemma *C-Suc*: *C n* = *B n* − *A n*
⟨*proof*⟩

lemma *unfold-add-numeral*: *c* ∗ *n* + *numeral b* = *Suc* (*c* ∗ *n* + *pred-numeral b*)
⟨*proof*⟩

lemma *ABC*:
A n = *q* (*3* ∗ *n*) ∗ *exp 1* − *p* (*3* ∗ *n*) ∧
B n = *p* (*Suc* (*3* ∗ *n*)) − *q* (*Suc* (*3* ∗ *n*)) ∗ *exp 1* ∧
C n = *p* (*Suc* (*Suc* (*3* ∗ *n*))) − *q* (*Suc* (*Suc* (*3* ∗ *n*))) ∗ *exp 1*
⟨*proof*⟩

lemma *q-pos*: *q n* > *0* **if** *n* ≠ *1*
⟨*proof*⟩

lemma *conv-diff-exp-bound*: *norm* (*exp 1* − *p n* / *q n*) ≤ *exp 1* / *fact* (*n div 3*)
⟨*proof*⟩

theorem *e-cfrac*: *cfrac-lim e-cfrac* = *exp 1*
⟨*proof*⟩

corollary *e-cfrac-altdef*: *e-cfrac* = *cfrac-of-real* (*exp 1*)
⟨*proof*⟩

This also provides us with a nice proof that e is not rational and not a quadratic irrational either.

corollary *exp1-irrational*: (*exp 1* :: *real*) ∉ ℚ
⟨*proof*⟩

corollary *exp1-not-quadratic-irrational*: ¬*quadratic-irrational* (*exp 1* :: *real*)
⟨*proof*⟩

end
end

4 Continued fraction expansions for square roots of naturals

theory *Sqrt-Nat-Cfrac*
imports
Quadratic-Irrationals
HOL−Library.While-Combinator
HOL−Library.IArray
begin

In this section, we shall explore the continued fraction expansion of $\sqrt{D}$, where D is a natural number.

lemma *butlast-nth* [*simp*]: $n < length\ xs - 1 \Longrightarrow butlast\ xs\ !\ n = xs\ !\ n$
⟨*proof*⟩

The following is the length of the period in the continued fraction expansion of $\sqrt{D}$ for a natural number D.

definition *sqrt-nat-period-length* :: *nat* ⇒ *nat* **where**
sqrt-nat-period-length D =
(*if is-square D then 0*
else (*LEAST l*. *l* > *0* ∧ (∀ *n*. *cfrac-nth* (*cfrac-of-real* (*sqrt D*)) (*Suc n* + *l*) =
cfrac-nth (*cfrac-of-real* (*sqrt D*)) (*Suc n*))))

Next, we define a more workable representation for the continued fraction expansion of $\sqrt{D}$ consisting of the period length, the natural number $\lfloor\sqrt{D}\rfloor$, and the content of the period.

definition *sqrt-cfrac-info* :: *nat* ⇒ *nat* × *nat* × *nat list* **where**
sqrt-cfrac-info D =
(*sqrt-nat-period-length D*, *Discrete.sqrt D*,
map (λ*n*. *nat* (*cfrac-nth* (*cfrac-of-real* (*sqrt D*)) (*Suc n*))) [*0*..<*sqrt-nat-period-length D*])

lemma *sqrt-nat-period-length-square* [*simp*]: *is-square D* ⟹ *sqrt-nat-period-length D* = *0*
⟨*proof*⟩

definition *sqrt-cfrac* :: *nat* ⇒ *cfrac*
where *sqrt-cfrac D* = *cfrac-of-real* (*sqrt* (*real D*))

context
fixes *D D'* :: *nat*
defines *D'* ≡ *nat* ⌊*sqrt D*⌋
begin

A number $\alpha = \frac{\sqrt{D}+p}{q}$ for $p,\ q \in \mathbb{N}$ is called a *reduced quadratic surd* if $\alpha > 1$ and $bar\alpha \in (-1; 0)$, where $\bar{\alpha}$ denotes the conjugate $\frac{-\sqrt{D}+p}{q}$.
It is furthermore called *associated* to D if q divides $D - p^2$.

definition *red-assoc* :: *nat* × *nat* ⇒ *bool* **where**
red-assoc = (λ(*p*, *q*).
q > *0* ∧ *q dvd* (*D* − *p*²) ∧ (*sqrt D* + *p*) / *q* > *1* ∧ (−*sqrt D* + *p*) / *q* ∈ {−*1*<..<*0*})

The following two functions convert between a surd represented as a pair of natural numbers and the actual real number and its conjugate:

definition *surd-to-real* :: *nat* × *nat* ⇒ *real*
where *surd-to-real* = (λ(*p*, *q*). (*sqrt D* + *p*) / *q*)

definition *surd-to-real-cnj* :: *nat* × *nat* ⇒ *real*
where *surd-to-real-cnj* = (λ(*p*, *q*). (−*sqrt D* + *p*) / *q*)

The next function performs a single step in the continued fraction expansion of $\sqrt{D}$.

definition *sqrt-remainder-step* :: *nat* × *nat* ⇒ *nat* × *nat* **where**
sqrt-remainder-step = (λ(*p*, *q*). *let X* = (*p* + *D'*) *div q*; *p'* = *X* ∗ *q* − *p in* (*p'*, (*D* − *p'*²) *div q*))

If we iterate this step function starting from the surd $\frac{1}{\sqrt{D}-\lfloor\sqrt{D}\rfloor}$, we get the entire expansion.

definition *sqrt-remainder-surd* :: *nat* ⇒ *nat* × *nat*
where *sqrt-remainder-surd* = (λ*n*. (*sqrt-remainder-step* ^^ *n*) (*D'*, *D* − *D'*²))

context
fixes *sqrt-cfrac-nth* :: *nat* ⇒ *nat* **and** *l*
assumes *nonsquare*: ¬*is-square D*
defines *sqrt-cfrac-nth* ≡ (λ*n*. *case sqrt-remainder-surd n of* (*p*, *q*) ⇒ (*D'* + *p*) *div q*)
defines *l* ≡ *sqrt-nat-period-length D*
begin

lemma *D'-pos*: *D'* > *0*
⟨*proof*⟩

lemma *D'-sqr-less-D*: *D'*² < *D*
⟨*proof*⟩

lemma *red-assoc-imp-irrat*:
assumes *red-assoc pq*
shows *surd-to-real pq* ∉ ℚ
⟨*proof*⟩

lemma *surd-to-real-cnj-irrat*:
assumes *red-assoc pq*
shows *surd-to-real-cnj pq* ∉ ℚ
⟨*proof*⟩

lemma *surd-to-real-nonneg* [*intro*]: *surd-to-real pq* $\geq$ *0*
⟨*proof*⟩

lemma *surd-to-real-pos* [*intro*]: *red-assoc pq* $\Longrightarrow$ *surd-to-real pq* $>$ *0*
⟨*proof*⟩

lemma *surd-to-real-nz* [*simp*]: *red-assoc pq* $\Longrightarrow$ *surd-to-real pq* $\neq$ *0*
⟨*proof*⟩

lemma *surd-to-real-cnj-nz* [*simp*]: *red-assoc pq* $\Longrightarrow$ *surd-to-real-cnj pq* $\neq$ *0*
⟨*proof*⟩

lemma *red-assoc-step*:
assumes *red-assoc pq*
defines $X \equiv (D' + fst\ pq)\ div\ snd\ pq$
defines $pq' \equiv$ *sqrt-remainder-step pq*
shows *red-assoc pq′*
$surd\text{-}to\text{-}real\ pq' = 1\ /\ frac\ (surd\text{-}to\text{-}real\ pq)$
$surd\text{-}to\text{-}real\text{-}cnj\ pq' = 1\ /\ (surd\text{-}to\text{-}real\text{-}cnj\ pq - X)$
$X > 0\ X * snd\ pq \leq 2 * D'\ X = nat\ \lfloor surd\text{-}to\text{-}real\ pq \rfloor$
$X = nat\ \lfloor -1\ /\ surd\text{-}to\text{-}real\text{-}cnj\ pq' \rceil$
⟨*proof*⟩

lemma *red-assoc-denom-2D*:
assumes *red-assoc* (p, q)
defines $X \equiv (D' + p)\ div\ q$
assumes $X > D'$
shows $q = 1$
⟨*proof*⟩

lemma *red-assoc-denom-1*:
assumes *red-assoc* $(p, 1)$
shows $p = D'$
⟨*proof*⟩

lemma *red-assoc-begin*:
red-assoc $(D', D - D'^2)$
$surd\text{-}to\text{-}real\ (D', D - D'^2) = 1\ /\ frac\ (sqrt\ D)$
$surd\text{-}to\text{-}real\text{-}cnj\ (D', D - D'^2) = -1\ /\ (sqrt\ D + D')$
⟨*proof*⟩

lemma *cfrac-remainder-surd-to-real*:
assumes *red-assoc pq*
shows $cfrac\text{-}remainder\ (cfrac\text{-}of\text{-}real\ (surd\text{-}to\text{-}real\ pq))\ n =$
$surd\text{-}to\text{-}real\ ((sqrt\text{-}remainder\text{-}step\ ^{\frown\frown}\ n)\ pq)$
⟨*proof*⟩

lemma *red-assoc-step′* [*intro*]: *red-assoc pq* $\Longrightarrow$ *red-assoc* (*sqrt-remainder-step pq*)
⟨*proof*⟩

lemma *red-assoc-steps* [*intro*]: *red-assoc pq* $\Longrightarrow$ *red-assoc* ((*sqrt-remainder-step* ^^ *n*) *pq*)
⟨*proof*⟩

lemma *floor-sqrt-less-sqrt*: $D' < \mathit{sqrt}\ D$
⟨*proof*⟩

lemma *red-assoc-bounds*:
assumes *red-assoc pq*
shows $pq \in (\mathit{SIGMA}\ p{:}\{0{<}..D'\}.\ \{\mathit{Suc}\ D' - p..D' + p\})$
⟨*proof*⟩

lemma *surd-to-real-cnj-eq-iff*:
assumes *red-assoc pq red-assoc pq'*
shows *surd-to-real-cnj pq* = *surd-to-real-cnj pq'* $\longleftrightarrow$ *pq* = *pq'*
⟨*proof*⟩

lemma *red-assoc-sqrt-remainder-surd* [*intro*]: *red-assoc* (*sqrt-remainder-surd n*)
⟨*proof*⟩

lemma *surd-to-real-sqrt-remainder-surd*:
surd-to-real (*sqrt-remainder-surd n*) = *cfrac-remainder* (*cfrac-of-real* (*sqrt D*)) (*Suc n*)
⟨*proof*⟩

lemma *sqrt-cfrac*: *sqrt-cfrac-nth n* = *cfrac-nth* (*cfrac-of-real* (*sqrt D*)) (*Suc n*)
⟨*proof*⟩

lemma *sqrt-cfrac-pos*: *sqrt-cfrac-nth k* > *0*
⟨*proof*⟩

lemma *snd-sqrt-remainder-surd-pos*: *snd* (*sqrt-remainder-surd n*) > *0*
⟨*proof*⟩

lemma
shows *period-nonempty*: $l > 0$
and *period-length-le-aux*: $l \le D' * (D' + 1)$
and *sqrt-remainder-surd-periodic*: $\bigwedge n.$ *sqrt-remainder-surd n* = *sqrt-remainder-surd* (*n mod l*)
and *sqrt-cfrac-periodic*: $\bigwedge n.$ *sqrt-cfrac-nth n* = *sqrt-cfrac-nth* (*n mod l*)
and *sqrt-remainder-surd-smallest-period*:
$\bigwedge n.\ n \in \{0{<}..{<}l\} \Longrightarrow$ *sqrt-remainder-surd n* $\neq$ *sqrt-remainder-surd 0*
and *snd-sqrt-remainder-surd-gt-1*: $\bigwedge n.\ n < l - 1 \Longrightarrow$ *snd* (*sqrt-remainder-surd n*) > *1*
and *sqrt-cfrac-le*: $\bigwedge n.\ n < l - 1 \Longrightarrow$ *sqrt-cfrac-nth n* $\le D'$
and *sqrt-remainder-surd-last*: *sqrt-remainder-surd* $(l - 1) = (D', 1)$
and *sqrt-cfrac-last*: *sqrt-cfrac-nth* $(l - 1) = 2 * D'$

and *sqrt-cfrac-palindrome*: $\bigwedge n.\ n < l - 1 \Longrightarrow$ *sqrt-cfrac-nth* $(l - n - 2) =$ *sqrt-cfrac-nth* n
and *sqrt-cfrac-smallest-period*:
$\bigwedge l'.\ l' > 0 \Longrightarrow (\bigwedge k.$ *sqrt-cfrac-nth* $(k + l') =$ *sqrt-cfrac-nth* $k) \Longrightarrow l' \geq l$
⟨*proof*⟩

theorem *cfrac-sqrt-periodic*:
cfrac-nth (*cfrac-of-real* (*sqrt* D)) (*Suc* n) =
cfrac-nth (*cfrac-of-real* (*sqrt* D)) (*Suc* (n *mod* l))
⟨*proof*⟩

theorem *cfrac-sqrt-le*: $n \in \{0<..<l\} \Longrightarrow$ *cfrac-nth* (*cfrac-of-real* (*sqrt* D)) $n \leq D'$
⟨*proof*⟩

theorem *cfrac-sqrt-last*: *cfrac-nth* (*cfrac-of-real* (*sqrt* D)) $l = 2 * D'$
⟨*proof*⟩

theorem *cfrac-sqrt-palindrome*:
assumes $n \in \{0<..<l\}$
shows *cfrac-nth* (*cfrac-of-real* (*sqrt* D)) $(l - n) =$ *cfrac-nth* (*cfrac-of-real* (*sqrt* D)) n
⟨*proof*⟩

lemma *sqrt-cfrac-info-palindrome*:
assumes *sqrt-cfrac-info* $D = (a,\ b,\ cs)$
shows *rev* (*butlast* cs) = *butlast* cs
⟨*proof*⟩

lemma *sqrt-cfrac-info-last*:
assumes *sqrt-cfrac-info* $D = (a,\ b,\ cs)$
shows *last* $cs = 2 *$ *Discrete.sqrt* D
⟨*proof*⟩

The following lemmas allow us to compute the period of the expansion of the square root:

lemma *while-option-sqrt-cfrac*:
defines $step' \equiv (\lambda(as,\ pq).\ ((D' +$ *fst* $pq)$ *div snd* $pq\ \#\ as,$ *sqrt-remainder-step* $pq))$
defines $b \equiv (\lambda(\text{-},\ pq).$ *snd* $pq \neq 1)$
defines *initial* $\equiv ([] :: nat\ list,\ (D',\ D - D'^2))$
shows *while-option* b $step'$ *initial* =
Some (*rev* (*map sqrt-cfrac-nth* $[0..<l-1]$), $(D',\ 1)$)
⟨*proof*⟩

lemma *while-option-sqrt-cfrac-info*:
defines $step' \equiv (\lambda(as,\ pq).\ ((D' +$ *fst* $pq)$ *div snd* $pq\ \#\ as,$ *sqrt-remainder-step* $pq))$
defines $b \equiv (\lambda(\text{-},\ pq).$ *snd* $pq \neq 1)$
defines *initial* $\equiv ([],\ (D',\ D - D'^2))$

shows *sqrt-cfrac-info D* =
(*case while-option b step' initial of*
Some (*as*, -) ⇒ (*Suc* (*length as*), *D'*, *rev* ((2 * *D'*) # *as*)))
⟨*proof*⟩

end
end

lemma *sqrt-nat-period-length-le*: *sqrt-nat-period-length D* ≤ *nat* ⌊*sqrt D*⌋ * (*nat* ⌊*sqrt D*⌋ + 1)
⟨*proof*⟩

lemma *sqrt-nat-period-length-0-iff* [*simp*]:
sqrt-nat-period-length D = 0 ⟷ *is-square D*
⟨*proof*⟩

lemma *sqrt-nat-period-length-pos-iff* [*simp*]:
sqrt-nat-period-length D > 0 ⟷ ¬*is-square D*
⟨*proof*⟩

lemma *sqrt-cfrac-info-code* [*code*]:
sqrt-cfrac-info D =
(*let D'* = *Discrete.sqrt D*
in *if D'*² = *D then* (0, *D'*, [])
else
case while-option
(λ(-, *pq*). *snd pq* ≠ 1)
(λ(*as*, (*p*, *q*)). *let X* = (*p* + *D'*) *div q*; *p'* = *X* * *q* − *p*
in (*X* # *as*, *p'*, (*D* − *p'*²) *div q*))
([], *D'*, *D* − *D'*²)
of Some (*as*, -) ⇒ (*Suc* (*length as*), *D'*, *rev* ((2 * *D'*) # *as*)))
⟨*proof*⟩

lemma *sqrt-nat-period-length-code* [*code*]:
sqrt-nat-period-length D = *fst* (*sqrt-cfrac-info D*)
⟨*proof*⟩

For efficiency reasons, it is often better to use an array instead of a list:

definition *sqrt-cfrac-info-array* **where**
sqrt-cfrac-info-array D = (*case sqrt-cfrac-info D of* (*a*, *b*, *c*) ⇒ (*a*, *b*, *IArray c*))

lemma *fst-sqrt-cfrac-info-array* [*simp*]: *fst* (*sqrt-cfrac-info-array D*) = *sqrt-nat-period-length D*
⟨*proof*⟩

lemma *snd-sqrt-cfrac-info-array* [*simp*]: *fst* (*snd* (*sqrt-cfrac-info-array D*)) = *Discrete.sqrt D*
⟨*proof*⟩

definition *cfrac-sqrt-nth* :: *nat* × *nat* × *nat iarray* ⇒ *nat* ⇒ *nat* **where**
cfrac-sqrt-nth info n =
(*case info of* (*l*, *a0*, *as*) ⇒ *if n* = *0 then a0 else as* !! ((*n* − *1*) *mod l*))

lemma *cfrac-sqrt-nth*:
assumes ¬*is-square D*
shows *cfrac-nth* (*cfrac-of-real* (*sqrt D*)) *n* =
int (*cfrac-sqrt-nth* (*sqrt-cfrac-info-array D*) *n*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *sqrt-cfrac-code* [*code*]:
sqrt-cfrac D =
(*let info* = *sqrt-cfrac-info-array D*;
(*l*, *a0*, -) = *info*
in if l = *0 then cfrac-of-int* (*int a0*) *else cfrac* (*cfrac-sqrt-nth info*))
⟨*proof*⟩

As a test, we determine the continued fraction expansion of $\sqrt{129}$, which is $[11; \overline{2, 1, 3, 1, 6, 1, 3, 1, 2, 22}]$ (a period length of 10):

value *let info* = *sqrt-cfrac-info-array 129 in info*
value *sqrt-nat-period-length 129*

We can also compute convergents of $\sqrt{129}$ and observe that the difference between the square of the convergents and 129 vanishes quickly::

value *map* (*conv* (*sqrt-cfrac 129*)) [*0..<10*]
value *map* (λ*n*. |*conv* (*sqrt-cfrac 129*) *n* ^ *2* − *129*|) [*0..<20*]

end

5 Lifting solutions of Pell's Equation

theory *Pell-Lifting*
imports *Pell.Pell Pell.Pell-Algorithm*
begin

5.1 Auxiliary material

lemma (**in** *pell*) *snth-pell-solutions*: *snth* (*pell-solutions D*) *n* = *nth-solution n*
⟨*proof*⟩

definition *square-squarefree-part-nat* :: *nat* ⇒ *nat* × *nat* **where**
square-squarefree-part-nat n = (*square-part n*, *squarefree-part n*)

lemma *prime-factorization-squarefree-part*:
assumes *x* ≠ *0*
shows *prime-factorization* (*squarefree-part x*) =
mset-set {*p* ∈ *prime-factors x*. *odd* (*multiplicity p x*)} (**is** *?lhs* = *?rhs*)

⟨*proof*⟩

lemma *squarefree-part-nat*:
squarefree-part (*n* :: *nat*) = (∏ {*p* ∈ *prime-factors n*. *odd* (*multiplicity p n*)})
⟨*proof*⟩

lemma *prime-factorization-square-part*:
assumes *x* ≠ *0*
shows *prime-factorization* (*square-part x*) =
(∑ *p* ∈ *prime-factors x*. *replicate-mset* (*multiplicity p x div 2*) *p*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *prod-mset-sum*: *prod-mset* (*sum f A*) = (∏ *x*∈*A*. *prod-mset* (*f x*))
⟨*proof*⟩

lemma *square-part-nat*:
assumes *n* > *0*
shows *square-part* (*n* :: *nat*) = (∏ *p* ∈ *prime-factors n*. *p* ^ (*multiplicity p n div 2*))
⟨*proof*⟩

lemma *square-squarefree-part-nat-code* [*code*]:
square-squarefree-part-nat n = (*if n* = *0 then* (*0*, *1*)
else let ps = *prime-factorization n*
in ((∏ *p*∈*set-mset ps*. *p* ^ (*count ps p div 2*)),
∏ (*Set.filter* (λ*p*. *odd* (*count ps p*)) (*set-mset ps*))))
⟨*proof*⟩

lemma *square-part-nat-code* [*code-unfold*]:
square-part (*n* :: *nat*) = (*if n* = *0 then 0*
else let ps = *prime-factorization n in* (∏ *p*∈*set-mset ps*. *p* ^ (*count ps p div 2*)))
⟨*proof*⟩

lemma *squarefree-part-nat-code* [*code-unfold*]:
squarefree-part (*n* :: *nat*) = (*if n* = *0 then 1*
else let ps = *prime-factorization n in* (∏ (*Set.filter* (λ*p*. *odd* (*count ps p*)) (*set-mset ps*))))
⟨*proof*⟩

lemma *is-nth-power-mult-nth-powerD*:
assumes *is-nth-power n* (*a* ∗ *b* ^ *n*) *b* > *0 n* > *0*
shows *is-nth-power n* (*a*::*nat*)
⟨*proof*⟩

lemma (**in** *pell*) *fund-sol-eq-fstI*:
assumes *nontriv-solution* (*x*, *y*)
assumes ⋀*x′ y′*. *nontriv-solution* (*x′*, *y′*) ⟹ *x* ≤ *x′*

shows *fund-sol* = (x, y)
⟨*proof*⟩

lemma (**in** *pell*) *fund-sol-eqI-fst'*:
assumes *nontriv-solution xy*
assumes $\bigwedge x'\ y'$. *nontriv-solution* $(x', y') \Longrightarrow$ *fst xy* $\leq x'$
shows *fund-sol* = *xy*
⟨*proof*⟩

lemma (**in** *pell*) *fund-sol-eq-sndI*:
assumes *nontriv-solution* (x, y)
assumes $\bigwedge x'\ y'$. *nontriv-solution* $(x', y') \Longrightarrow y \leq y'$
shows *fund-sol* = (x, y)
⟨*proof*⟩

lemma (**in** *pell*) *fund-sol-eqI-snd'*:
assumes *nontriv-solution xy*
assumes $\bigwedge x'\ y'$. *nontriv-solution* $(x', y') \Longrightarrow$ *snd xy* $\leq y'$
shows *fund-sol* = *xy*
⟨*proof*⟩

5.2 The lifting mechanism

The solutions of Pell's equations for parameters D and $a^2\ D$ stand in correspondence to one another: every solution (x, y) for parameter D can be lowered to a solution (x, ay) for $a^2\ D$, and every solution of the form (x, ay) for parameter $a^2\ D$ can be lifted to a solution (x, y) for parameter D.

locale *pell-lift* = *pell* +
fixes *a D'* :: *nat*
assumes *nz*: $a > 0$
defines $D' \equiv D * a^2$
begin

lemma *nonsquare-D'*: ¬*is-square D'*
⟨*proof*⟩

definition *lift-solution* :: *nat* × *nat* ⇒ *nat* × *nat* **where**
lift-solution = $(\lambda(x, y).\ (x, y\ div\ a))$

definition *lower-solution* :: *nat* × *nat* ⇒ *nat* × *nat* **where**
lower-solution = $(\lambda(x, y).\ (x, y * a))$

definition *liftable-solution* :: *nat* × *nat* ⇒ *bool* **where**
liftable-solution = $(\lambda(x, y).\ a\ dvd\ y)$

sublocale *lift*: *pell D'*
⟨*proof*⟩

lemma *lift-solution-iff*: *lift.solution xy* $\longleftrightarrow$ *solution* (*lower-solution xy*)
⟨*proof*⟩

lemma *lift-solution*:
assumes *solution xy liftable-solution xy*
shows *lift.solution* (*lift-solution xy*)
⟨*proof*⟩

In particular, the fundamental solution for $a^2\ D$ is the smallest liftable solution for D:

lemma *lift-fund-sol*:
assumes $\bigwedge n.\ 0 < n \Longrightarrow n < m \Longrightarrow \neg$*liftable-solution* (*nth-solution n*)
assumes *liftable-solution* (*nth-solution m*) $m > 0$
shows *lift.fund-sol* = *lift-solution* (*nth-solution m*)
⟨*proof*⟩

end

5.3 Accelerated computation of the fundamental solution for non-squarefree inputs

Solving Pell's equation for some D of the form $a^2\ D'$ can be done by solving it for D' and then lifting the solution. Thus, if D is not squarefree, we can compute its squarefree decomposition $a^2\ D'$ with D' squarefree and thus speed up the computation (since D' is smaller than D).

The squarefree decomposition can only be computed (according to current knowledge in mathematics) through the prime decomposition. However, given how big the solutions are for even moderate values of D, it is usually worth doing it if D is not squarefree.

lemma *squarefree-part-of-square* [*simp*]:
assumes *is-square* ($x :: 'a ::$ {*factorial-semiring*, *normalization-semidom-multiplicative*})
assumes $x \neq 0$
shows *squarefree-part x* = *unit-factor x*
⟨*proof*⟩

lemma *squarefree-part-1-imp-square*:
assumes *squarefree-part x* = *1*
shows *is-square x*
⟨*proof*⟩

definition *find-fund-sol-fast* **where**
find-fund-sol-fast D =
(*let* (*a*, *D'*) = *square-squarefree-part-nat D*
in
if D' = *0* ∨ *D'* = *1 then* (*0*, *0*)
else if a = *1 then pell.fund-sol D*

else map-prod id ($\lambda y.$ *y div a*)
(*shd* (*sdrop-while* (λ(-, *y*). *y* = *0* $\vee$ $\neg$*a dvd y*) (*pell-solutions D'*))))

lemma *find-fund-sol-fast*: *find-fund-sol D* = *find-fund-sol-fast D*
⟨*proof*⟩

end

6 The Connection between the continued fraction expansion of square roots and Pell's equation

theory *Pell-Continued-Fraction*
imports
Sqrt-Nat-Cfrac
Pell.Pell-Algorithm
Polynomial-Factorization.Prime-Factorization
Pell-Lifting
begin

lemma *irrational-times-int-eq-intD*:
assumes *p* $*$ *real-of-int a* = *real-of-int b*
assumes *p* $\notin$ $\mathbb{Q}$
shows *a* = *0* $\wedge$ *b* = *0*
⟨*proof*⟩

The solutions to Pell's equation for some non-square D are linked to the continued fraction expansion of $\sqrt{D}$, which we shall show here.

context
fixes *D* :: *nat* **and** *c h k P Q l*
assumes *nonsquare*: $\neg$*is-square D*
defines *c* $\equiv$ *cfrac-of-real* (*sqrt D*)
defines *h* $\equiv$ *conv-num c* **and** *k* $\equiv$ *conv-denom c*
defines *P* $\equiv$ *fst* $\circ$ *sqrt-remainder-surd D* **and** *Q* $\equiv$ *snd* $\circ$ *sqrt-remainder-surd D*
defines *l* $\equiv$ *sqrt-nat-period-length D*
begin

interpretation *pell D*
⟨*proof*⟩

lemma *cfrac-length-infinite* [*simp*]: *cfrac-length c* = ∞
⟨*proof*⟩

lemma *conv-num-denom-pell*:
h 0 ^ *2* − *D* $*$ *k 0* ^ *2* < *0*
m > *0* $\Longrightarrow$ *h m* ^ *2* − *D* $*$ *k m* ^ *2* = (−*1*) ^ *Suc m* $*$ *Q m*
⟨*proof*⟩

Every non-trivial solution to Pell's equation is a convergent in the expansion

of $\sqrt{D}$:

theorem *pell-solution-is-conv*:
  assumes $x^2 = Suc\ (D * y^2)$ **and** $y > 0$
  shows $(int\ x,\ int\ y) \in range\ (\lambda n.\ (conv\text{-}num\ c\ n,\ conv\text{-}denom\ c\ n))$
⟨*proof*⟩

Let l be the length of the period in the continued fraction expansion of $\sqrt{D}$ and let h_i and k_i be the numerator and denominator of the i-th convergent. Then the non-trivial solutions of Pell's equation are exactly the pairs of the form (h_{lm-1}, k_{lm-1}) for any m such that lm is even.

lemma *nontriv-solution-iff-conv-num-denom*:
  nontriv-solution $(x,\ y) \longleftrightarrow$
    $(\exists\, m{>}0.\ int\ x = h\ (l * m - 1) \wedge int\ y = k\ (l * m - 1) \wedge even\ (l * m))$
⟨*proof*⟩

Consequently, the fundamental solution is (h_n, k_n) where $n = l - 1$ if l is even and $n = 2l - 1$ otherwise:

lemma *fund-sol-conv-num-denom*:
  defines $n \equiv if\ even\ l\ then\ l - 1\ else\ 2 * l - 1$
  shows $fund\text{-}sol = (nat\ (h\ n),\ nat\ (k\ n))$
⟨*proof*⟩

end

The following algorithm computes the fundamental solution (or the dummy result $(0,\ 0)$ if D is a square) fairly quickly by computing the continued fraction expansion of $\sqrt{D}$ and then computing the fundamental solution as the appropriate convergent.

lemma *find-fund-sol-code* [*code*]:
  find-fund-sol D =
    (*let info* = *sqrt-cfrac-info-array D*;
        l = *fst info*
     in *if l* = 0 *then* (0, 0) *else*
          let
            c = *cfrac-sqrt-nth info*;
            n = *if even l then l* − 1 *else* 2 ∗ *l* − 1
        in
          (*nat* (*conv-num-fun c n*), *nat* (*conv-denom-fun c n*)))
⟨*proof*⟩

lemma *find-nth-solution-square* [*simp*]: *is-square D* ⟹ *find-nth-solution D n* = (0, 0)
  ⟨*proof*⟩

lemma *fst-find-fund-sol-eq-0-iff* [*simp*]: *fst* (*find-fund-sol D*) = 0 ⟷ *is-square D*
⟨*proof*⟩

Arbitrary solutions can now be computed as powers of the fundamental solution.

lemma *find-nth-solution-code* [*code*]:
find-nth-solution D n =
(*let xy* = *find-fund-sol D*
in if fst xy = *0 then* (*0*, *0*) *else efficient-pell-power D xy n*)
⟨*proof*⟩

lemma *nth-solution-code* [*code*]:
pell.nth-solution D n =
(*let info* = *sqrt-cfrac-info-array D*;
l = *fst info*
in if l = *0 then*
Code.abort (*STR ''nth-solution is undefined for perfect square parameter.''*)
(λ-. *pell.nth-solution D n*)
else
let
c = *cfrac-sqrt-nth info*;
m = *if even l then l* − *1 else 2* ∗ *l* − *1*;
fund-sol = (*nat* (*conv-num-fun c m*), *nat* (*conv-denom-fun c m*))
in
efficient-pell-power D fund-sol n)
⟨*proof*⟩

lemma *fund-sol-code* [*code*]:
pell.fund-sol D = (*let info* = *sqrt-cfrac-info-array D*;
l = *fst info*
in if l = *0 then*
Code.abort (*STR ''fund-sol is undefined for perfect square parameter.''*)
(λ-. *pell.fund-sol D*)
else
let
c = *cfrac-sqrt-nth info*;
n = *if even l then l* − *1 else 2* ∗ *l* − *1*
in
(*nat* (*conv-num-fun c n*), *nat* (*conv-denom-fun c n*)))
⟨*proof*⟩

end

7 Tests for Continued Fractions of Square Roots and Pell's Equation

theory *Pell-Continued-Fraction-Tests*
imports
Pell.Efficient-Discrete-Sqrt
HOL−Library.Code-Lazy
HOL−Library.Code-Target-Numeral

```
  Pell-Continued-Fraction
  Pell-Lifting
begin

code-lazy-type stream


lemma lnth-code [code]:
 lnth xs 0 = (if lnull xs then undefined (0 :: nat) else lhd xs)
 lnth xs (Suc n) = (if lnull xs then undefined (Suc n) else lnth (ltl xs) n)
  ⟨proof⟩

value let c = sqrt-cfrac 1339 in map (cfrac-nth c) [0..<30]


fun arg-max-list where
  arg-max-list - [] = undefined
| arg-max-list f (x # xs) =
    foldl (λ(x, y) x'. let y' = f x' in if y' > y then (x', y') else (x, y)) (x, f x) xs


value [code] sqrt-cfrac-info 17
value [code] sqrt-cfrac-info 1339
value [code] sqrt-cfrac-info 121
value [code] sqrt-nat-period-length 410286423278424
```

For which number $D < 100000$ does $\sqrt{D}$ have the longest period?

```
value [code] arg-max-list sqrt-nat-period-length [0..<100000]
```

7.1 Fundamental solutions of Pell's equation

```
value [code] pell.fund-sol 12
value [code] pell.fund-sol 13
value [code] pell.fund-sol 61
value [code] pell.fund-sol 661
value [code] pell.fund-sol 6661
value [code] pell.fund-sol 4729494
```

Project Euler problem #66: For which $D < 1000$ does Pell's equation have the largest fundamental solution?

```
value [code] arg-max-list (fst ∘ find-fund-sol) [0..<1001]
```

The same for $D < 100000$:

```
value [code] arg-max-list (fst ∘ find-fund-sol) [0..<100000]
```

The solution to the next example, which is at the core of Archimedes' cattle problem, is so big that termifying the result takes extremely long. Therefore, we simply compute the number of decimal digits in the result instead.

fun *log10-aux* :: *nat* ⇒ *nat* ⇒ *nat* **where**
log10-aux acc n =
(*if n* ≥ *10000000000 then log10-aux* (*acc* + *10*) (*n div 10000000000*)
else if n = *0 then acc else log10-aux* (*Suc acc*) (*n div 10*))

definition *log10* **where** *log10* = *log10-aux 0*

value [*code*] *map-prod log10 log10* (*pell.fund-sol 410286423278424*)

Factoring out the square factor 9314^2 does yield a significant speed-up in this case:

value [*code*] *map-prod log10 log10* (*find-fund-sol-fast 410286423278424*)

7.2 Tests for other operations

value [*code*] *pell.nth-solution 13 100*
value [*code*] *pell.nth-solution 4729494 3*

value [*code*] *stake 10* (*pell-solutions 13*)
value [*code*] *stake 10* (*pell-solutions 61*)

value [*code*] *pell.nth-solution 23 8*

end

8 Computing continued fraction expansions through interval arithmetic

theory *Continued-Fraction-Approximation*
imports
Complex-Main
HOL−Decision-Procs.Approximation
Coinductive.Coinductive-List
HOL−Library.Code-Lazy
HOL−Library.Code-Target-Numeral
Continued-Fractions
keywords *approximate-cfrac* :: *diag*
begin

The approximation package allows us to compute an enclosing interval for a given real constant. From this, we are able to compute an initial fragment of the continued fraction expansion of the number.

The algorithm essentially works by computing the continued fraction expansion of the lower and upper bound simultaneously and stopping when the results start to diverge.

This algorithm terminates because the lower and upper bounds, being rational numbers, have a finite continued fraction expansion.

definition *float-to-rat* :: *float* ⇒ *int* × *int* **where**
float-to-rat f = (*if exponent f* ≥ *0 then*
(*mantissa f* ∗ *2* ^ *nat* (*exponent f*), *1*) *else* (*mantissa f*, *2* ^ *nat* (−*exponent f*)))

lemma *float-to-rat*: *fst* (*float-to-rat f*) / *snd* (*float-to-rat f*) = *real-of-float f*
⟨*proof*⟩

lemma *snd-float-to-rat-pos* [*simp*]: *snd* (*float-to-rat f*) > *0*
⟨*proof*⟩

function *cfrac-from-approx* :: *int* × *int* ⇒ *int* × *int* ⇒ *int list* **where**
cfrac-from-approx (*nl*, *dl*) (*nu*, *du*) =
(*if nl* = *0* ∨ *nu* = *0* ∨ *dl* = *0* ∨ *du* = *0 then* []
else let l = *nl div dl*; *u* = *nu div du*
in if l ≠ *u then* []
else l # (*let m* = *nl mod dl in if m* = *0 then* [] *else*
cfrac-from-approx (*du*, *nu mod du*) (*dl*, *m*)))
⟨*proof*⟩
termination ⟨*proof*⟩

lemmas [*simp del*] = *cfrac-from-approx.simps*

lemma *cfrac-from-approx-correct*:
assumes *x* ∈ {*fst l* / *snd l*..*fst u* / *snd u*} **and** *snd l* > *0* **and** *snd u* > *0*
assumes *i* < *length* (*cfrac-from-approx l u*)
shows *cfrac-nth* (*cfrac-of-real x*) *i* = *cfrac-from-approx l u* ! *i*
⟨*proof*⟩

definition *cfrac-from-approx′* :: *float* ⇒ *float* ⇒ *int list* **where**
cfrac-from-approx′ l u = *cfrac-from-approx* (*float-to-rat l*) (*float-to-rat u*)

lemma *cfrac-from-approx′-correct*:
assumes *x* ∈ {*real-of-float l*..*real-of-float u*}
assumes *i* < *length* (*cfrac-from-approx′ l u*)
shows *cfrac-nth* (*cfrac-of-real x*) *i* = *cfrac-from-approx′ l u* ! *i*
⟨*proof*⟩

definition *approx-cfrac* :: *nat* ⇒ *floatarith* ⇒ *int list* **where**
approx-cfrac prec e =
(*case approx′ prec e* [] *of*
None ⇒ []
| *Some ivl* ⇒ *cfrac-from-approx′* (*lower ivl*) (*upper ivl*))

⟨*ML*⟩

Now let us do some experiments:

value *let prec* = *34*; *c* = *cfrac-from-approx′* (*lb-pi prec*) (*ub-pi prec*) *in c*

value *let prec = 34; c = cfrac-from-approx′ (lb-pi prec) (ub-pi prec)*
in map (λn. (conv-num-fun ((!) c) n, conv-denom-fun ((!) c) n)) [0..<length c]

approximate-cfrac *prec: 200 pi*
approximate-cfrac *ln 2*
approximate-cfrac *exp 1*
approximate-cfrac *sqrt 129*
approximate-cfrac *(sqrt 13 + 3) / 4*
approximate-cfrac *arctan 1*

approximate-cfrac *123 / 97*
value *cfrac-list-of-rat (123, 97)*

end

References

[1] A. Khinchin and H. Eagle. *Continued Fractions*. Dover books on mathematics. Dover Publications, 1997.

[2] Proof Wiki.