# Continued Fractions

### Manuel Eberl

#### March 17, 2025

#### 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	Cor	tinued Fractions 3
	1.1	Auxiliary results
	1.2	Bounds on alternating decreasing sums
	1.3	Non-canonical continued fractions
	1.4	Approximation properties
	1.5	Efficient code for convergents
	1.6	Computing the continued fraction expansion of a rational
		number
<b>2</b>	Quadratic Irrationals 83	
	2.1	Basic results on rationality of square roots
	2.2	Definition of quadratic irrationals
	2.3	Real solutions of quadratic equations
	2.4	Periodic continued fractions and quadratic irrationals 91
3	The	e continued fraction expansion of $e$ 105
4	Continued fraction expansions for square roots of naturals 112	
5	Lift	ing solutions of Pell's Equation 135
	5.1	Auxiliary material
	5.2	The lifting mechanism
	5.3	Accelerated computation of the fundamental solution for non-
		squarefree inputs
6	The Connection between the continued fraction expansion	
	of s	quare roots and Pell's equation143
7	Tests for Continued Fractions of Square Roots and Pell's	
	Equ	nation 152
	7.1	Fundamental solutions of Pell's equation
	7.2	Tests for other operations
8	Computing continued fraction expansions through interval	
	arit	hmetic 154

## **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 \Rightarrow bool where
  linfinite \ xs \implies linfinite \ (LCons \ x \ xs)
lemma llength-llist-of-stream [simp]: llength (llist-of-stream xs) = \infty
  by (simp add: not-lfinite-llength)
lemma linfinite-conv-llength: linfinite xs \leftrightarrow llength xs = \infty
proof
  assume linfinite xs
  thus llength xs = \infty
  proof (coinduction arbitrary: xs rule: enat-coinduct2)
   fix xs :: 'a llist
   assume llength xs \neq 0 linfinite xs
   thus (\exists xs':: a \ llist. epred \ (llength \ xs) = llength \ xs' \land epred \ \infty = \infty \land linfinite
xs') \lor
             epred (llength xs) = epred \infty
     by (intro disjI1 exI[of - ltl xs]) (auto simp: linfinite.simps[of xs])
  \mathbf{next}
   fix xs :: a llist assume linfinite xs thus (llength xs = 0) \longleftrightarrow (\infty = (0::enat))
     by (subst (asm) linfinite.simps) auto
  qed
next
  assume llength xs = \infty
  thus linfinite xs
  proof (coinduction arbitrary: xs)
   case linfinite
   thus \exists xsa x.
            xs = LCons \ x \ xsa \ \land
            ((\exists xs. xsa = xs \land llength xs = \infty) \lor
             linfinite xsa)
     by (cases xs) (auto simp: eSuc-eq-infinity-iff)
  qed
qed
```

**definition** lnth-default ::  $a \Rightarrow a$  llist  $\Rightarrow nat \Rightarrow a$  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 (induction n arbitrary: xs)
 case \theta
 thus ?case
   by (cases xs) (auto simp: lnth-default-def simp flip: zero-enat-def)
\mathbf{next}
 case (Suc n)
 show ?case
 proof (cases xs)
   case LNil
   thus ?thesis
     by (auto simp: lnth-default-def)
 next
   case (LCons x xs')
   thus ?thesis
     by (auto simp: lnth-default-def Suc-ile-eq)
 qed
\mathbf{qed}
lemma enat-le-iff:
  enat n \leq m \leftrightarrow m = \infty \lor (\exists m'. m = enat m' \land n \leq m')
 by (cases m) auto
lemma enat-less-iff:
  enat \ n < m \longleftrightarrow m = \infty \lor (\exists m'. \ m = enat \ m' \land n < m')
 by (cases m) auto
lemma real-of-int-divide-in-Ints-iff:
 real-of-int a \mid real-of-int \ b \in \mathbb{Z} \longleftrightarrow b \ dvd \ a \lor b = 0
proof safe
 assume real-of-int a \mid real-of-int \ b \in \mathbb{Z} b \neq 0
 then obtain n where real-of-int a / real-of-int b = real-of-int n
   by (auto simp: Ints-def)
 hence real-of-int b * real-of-int n = real-of-int a
   using \langle b \neq 0 \rangle by (auto simp: field-simps)
 also have real-of-int b * real-of-int n = real-of-int (b * n)
   by simp
 finally have b * n = a
   by linarith
 thus b dvd a
   by auto
\mathbf{qed} \ auto
```

```
lemma frac-add-of-nat: frac (of-nat y + x) = frac x
unfolding frac-def by simp
```

**lemma** frac-add-of-int: frac (of-int y + x) = frac x unfolding *frac-def* by *simp* **lemma** frac-fraction: frac (real-of-int  $a / real-of-int b) = (a \mod b) / b$ proof have frac  $(a / b) = frac ((a \mod b + b * (a \dim b)) / b)$ by (subst mod-mult-div-eq) auto also have  $(a \mod b + b * (a \dim b)) / b = of-int (a \dim b) + a \mod b / b$ unfolding of-int-add by (subst add-divide-distrib) auto also have  $frac \ldots = frac (a \mod b / b)$ by (rule frac-add-of-int) also have  $\ldots = a \mod b / b$ **by** (*simp add: floor-divide-of-int-eq frac-def*) finally show ?thesis . qed lemma Suc-fib-ge: Suc (fib n)  $\geq n$ **proof** (*induction n rule: fib.induct*) case (3 n)show ?case **proof** (cases n < 2) case True thus ?thesis by (cases n) auto next case False hence Suc (Suc (Suc n))  $\leq$  Suc n + n by simp also have  $\ldots \leq Suc (fib (Suc n)) + Suc (fib n)$ by (intro add-mono 3) also have  $\ldots = Suc (Suc (fib (Suc (Suc n))))$ by simp finally show ?thesis by (simp only: Suc-le-eq) qed qed auto lemma fib-ge: fib  $n \ge n - 1$ using Suc-fib-ge[of n] by simp **lemma** frac-diff-of-nat-right [simp]: frac (x - of-nat y) = frac x**using** floor-diff-of-int[of x int y] **by** (simp add: frac-def) **lemma** of-nat-ge-1-iff: of-nat  $n \ge (1 :: 'a :: linordered-semidom) \longleftrightarrow n > 0$ using of-nat-le-iff[of 1 n] unfolding of-nat-1 by auto **lemma** not-frac-less-0:  $\neg$ frac x < 0**by** (*simp add: frac-def not-less*) lemma frac-le-1: frac  $x \leq 1$ unfolding frac-def by linarith

**lemma** *divide-in-Rats-iff1*:  $(x::real) \in \mathbb{Q} \Longrightarrow x \neq 0 \Longrightarrow x \mid y \in \mathbb{Q} \longleftrightarrow y \in \mathbb{Q}$ **proof** safe assume  $*: x \in \mathbb{Q} \ x \neq 0 \ x / y \in \mathbb{Q}$ from \*(1,3) have  $x / (x / y) \in \mathbb{Q}$ by (rule Rats-divide) also from \* have x / (x / y) = y by simp finally show  $y \in \mathbb{Q}$ . qed (auto intro: Rats-divide) **lemma** *divide-in-Rats-iff2*:  $(y::real) \in \mathbb{Q} \Longrightarrow y \neq 0 \Longrightarrow x \mid y \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$ **proof** safe assume  $*: y \in \mathbb{Q} \ y \neq 0 \ x \ / \ y \in \mathbb{Q}$ from \*(3,1) have  $x / y * y \in \mathbb{Q}$ by (rule Rats-mult) also from \* have x / y \* y = x by simp finally show  $x \in \mathbb{Q}$ . **qed** (*auto intro: Rats-divide*) **lemma** add-in-Rats-iff1:  $x \in \mathbb{Q} \implies x + y \in \mathbb{Q} \iff y \in \mathbb{Q}$ using Rats-diff [of x + y x] by auto lemma add-in-Rats-iff2:  $y \in \mathbb{Q} \implies x + y \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$ using Rats-diff [of x + y y] by auto lemma diff-in-Rats-iff1:  $x \in \mathbb{Q} \implies x - y \in \mathbb{Q} \iff y \in \mathbb{Q}$ using Rats-diff [of x x - y] by auto lemma diff-in-Rats-iff2:  $y \in \mathbb{Q} \Longrightarrow x - y \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$ using Rats-add[of x - y y] by auto **lemma** frac-in-Rats-iff [simp]: frac  $x \in \mathbb{Q} \leftrightarrow x \in \mathbb{Q}$ **by** (*simp add: frac-def diff-in-Rats-iff2*) **lemma** *filterlim-sequentially-shift*: filterlim ( $\lambda n. f(n+m)$ ) F sequentially  $\leftrightarrow$  filterlim f F sequentially **proof** (*induction* m) case (Suc m) have filterlim ( $\lambda n. f (n + Suc m)$ ) F at-top  $\leftrightarrow$ filterlim ( $\lambda n. f$  (Suc n + m)) F at-top by simp also have  $\ldots \longleftrightarrow$  filterlim  $(\lambda n. f (n + m)) F$  at-top by (rule filterlim-sequentially-Suc) also have  $\ldots \iff$  filterlim f F at-top by (rule Suc.IH) finally show ?case . qed simp-all

#### **1.2** Bounds on alternating decreasing sums

**lemma** alternating-decreasing-sum-bounds: fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m \leq n \bigwedge k$ .  $k \in \{m..n\} \Longrightarrow f k \geq 0$  $\bigwedge k. \ k \in \{m.. < n\} \Longrightarrow f \ (Suc \ k) \le f \ k$ defines  $S \equiv (\lambda m. (\sum k = m..n. (-1) \land k * f k))$ **shows** if even m then  $S m \in \{0.., f m\}$  else  $S m \in \{-f m ... 0\}$ using assms(1)**proof** (*induction rule: inc-induct*) case (step m') have  $[simp]: -a \leq b \iff a + b \geq (0 :: 'a)$  for  $a \ b$ **by** (*metis le-add-same-cancel1 minus-add-cancel*) have [simp]:  $S m' = (-1) \widehat{m'} * f m' + S (Suc m')$ using step.hyps unfolding S-def **by** (subst sum.atLeast-Suc-atMost) simp-all from step.hyps have nonneg:  $f m' \ge 0$ by (intro assms) auto **from** step.hyps **have** mono:  $f(Suc m') \leq fm'$ by (intro assms) auto show ?case **proof** (cases even m') case True hence  $0 \leq f$  (Suc m') + S (Suc m') using step.IH by simp also note mono finally show ?thesis using True step.IH by auto  $\mathbf{next}$ case False with step. IH have S (Suc m')  $\leq f$  (Suc m') by simp also note mono finally show ?thesis using step.IH False by auto qed qed (insert assms, auto) **lemma** alternating-decreasing-sum-bounds': fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m < n \land k$ .  $k \in \{m..n-1\} \Longrightarrow f k \ge 0$  $\bigwedge k. \ k \in \{m..{<}n{-}1\} \Longrightarrow f \ (Suc \ k) \leq f \ k$ defines  $S \equiv (\lambda m. (\sum k = m.. < n. (-1) \land k * f k))$ **shows** if even m then  $S m \in \{0.., f m\}$  else  $S m \in \{-f m ... 0\}$ **proof** (cases n) case  $\theta$ thus ?thesis using assms by auto

next

**case** (Suc n') **hence** if even m then  $(\sum k=m..n-1. (-1) \land k * f k) \in \{0..f m\}$ else  $(\sum k=m..n-1. (-1) \land k * f k) \in \{-f m..0\}$ 

using assms by (intro alternating-decreasing-sum-bounds) auto

also have  $(\sum k=m..n-1, (-1) \land k * f k) = S m$ unfolding S-def by (intro sum.cong) (auto simp: Suc) finally show ?thesis . qed **lemma** alternating-decreasing-sum-upper-bound: fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m \leq n \bigwedge k$ .  $k \in \{m...n\} \Longrightarrow f k \geq 0$ using alternating-decreasing-sum-bounds [of m n f, OF assms] assms(1) by (auto split: if-splits intro: order.trans[OF - assms(2)]) **lemma** alternating-decreasing-sum-upper-bound': fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m < n \land k. k \in \{m..n-1\} \Longrightarrow f k \ge 0$  $\bigwedge k. \ k \in \{m.. < n-1\} \Longrightarrow f \ (Suc \ k) \le f \ k$ shows  $(\sum k=m..< n. (-1) \land k * f k) \le f m$ using alternating-decreasing-sum-bounds' [of m n f, OF assms] assms(1) by (auto split: if-splits intro: order.trans[OF - assms(2)]) **lemma** abs-alternating-decreasing-sum-upper-bound: fixes  $f :: nat \Rightarrow 'a :: \{linordered\text{-ring}, ring\text{-}1\}$ assumes  $m \le n \bigwedge k$ .  $k \in \{m..n\} \Longrightarrow f k \ge 0$ **using** alternating-decreasing-sum-bounds of m n f, OF assms] **by** (*auto split: if-splits simp: minus-le-iff*) lemma abs-alternating-decreasing-sum-upper-bound': fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m < n \land k. k \in \{m..n-1\} \Longrightarrow f k \ge 0$  $\bigwedge k. \ k \in \{m.. < n-1\} \Longrightarrow f \ (Suc \ k) \le f \ k$ shows  $|(\sum k = m ... < n. (-1) \land k * f k)| \le f m$ using alternating-decreasing-sum-bounds' of m n f, OF assms by (auto split: if-splits simp: minus-le-iff) **lemma** *abs-alternating-decreasing-sum-lower-bound*: fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m < n \land k. k \in \{m..n\} \Longrightarrow f k \ge 0$ proof – have  $(\sum k=m..n. (-1) \ \widehat{}\ k * f k) = (\sum k \in insert \ m \ \{m < ..n\}. (-1) \ \widehat{}\ k * f k)$ using assms by (intro sum.cong) auto also have ... =  $(-1) \ \widehat{}\ m * f \ m + (\sum k \in \{m < ... n\}. \ (-1) \ \widehat{}\ k * f \ k)$ by auto also have  $(\sum k \in \{m < ...n\}, (-1) \land k * f k) = (\sum k \in \{m ... < n\}, (-1) \land Suc k * f$  $(Suc \ k))$ 

by (intro sum.reindex-bij-witness[of - Suc  $\lambda i$ . i - 1]) auto also have  $(-1)^m * f m + ... = (-1)^m * f m - (\sum k \in \{m.. < n\}. (-1)^k *$ f (Suc k)) **by** (simp add: sum-negf) also have  $|...| \ge |(-1) \hat{m} * f m| - |(\sum k \in \{m.. < n\}, (-1) \hat{k} * f (Suc k))|$ **by** (rule abs-triangle-ineq2) also have  $|(-1)\hat{m} * f m| = f m$ using assms by (cases even m) auto finally have  $f m - |\sum k = m \dots \langle n . (-1) \land k * f (Suc k)|$   $\leq |\sum k = m \dots n \dots (-1) \land k * f k|$ . moreover have  $f m - |(\sum k \in \{m \dots < n\}, (-1) \land k * f (Suc k))| \geq f m - f (Suc k)|$ m) using assms by (intro diff-mono abs-alternating-decreasing-sum-upper-bound') autoultimately show ?thesis by (rule order.trans[rotated]) qed lemma abs-alternating-decreasing-sum-lower-bound': fixes  $f :: nat \Rightarrow 'a :: \{linordered-ring, ring-1\}$ assumes  $m+1 < n \land k. k \in \{m..n\} \Longrightarrow f k \ge 0$  $\bigwedge k. \ k \in \{m..{<}n\} \Longrightarrow f \ (Suc \ k) \leq f \ k$ shows  $|(\sum k=m..<n.(-1) \ k * f k)| \ge f m - f (Suc m)$ **proof** (cases n) case  $\theta$ thus ?thesis using assms by auto next case (Suc n') hence  $|(\sum k=m..n-1, (-1) \cap k * f k)| \ge f m - f (Suc m)$ using assms by (intro abs-alternating-decreasing-sum-lower-bound) auto also have  $(\sum k=m..n-1. (-1) \land k * f k) = (\sum k=m..< n. (-1) \land k * f k)$ by (intro sum.cong) (auto simp: Suc) finally show ?thesis . qed **lemma** alternating-decreasing-suminf-bounds: assumes  $\bigwedge k. f k \ge (0 :: real) \bigwedge k. f (Suc k) \le f k$  $f \xrightarrow{f} 0$ shows  $(\sum k. (-1) \land k * f k) \in \{f \ 0 - f \ 1..f \ 0\}$ proof – have summable  $(\lambda k. (-1) \land k * f k)$ by (intro summable-Leibniz' assms) hence lim:  $(\lambda n. \sum k \le n. (-1) \land k * f k) \longrightarrow (\sum k. (-1) \land k * f k)$ **by** (*auto dest: summable-LIMSEQ'*) have bounds:  $(\sum k=0..n. (-1) \land k * f k) \in \{f \ 0 - f \ 1..f \ 0\}$  $\mathbf{if}\ n > \theta \ \mathbf{for}\ n$ using alternating-decreasing-sum-bounds of 1 n f assms that **by** (subst sum.atLeast-Suc-atMost) auto **note** [simp] = atLeast0AtMost**note** [intro!] = eventually-mono[OF eventually-gt-at-top[of 0]]

from lim have  $(\sum k. (-1) \ \hat{k} * f k) \ge f \ 0 - f \ 1$ by (rule tendsto-lowerbound) (insert bounds, auto) moreover from lim have  $(\sum k. (-1) \ \hat{k} * f k) \le f \ 0$ by (rule tendsto-upperbound) (use bounds in auto) ultimately show ?thesis by simp qed

#### lemma

assumes  $\bigwedge k. \ k \ge m \Longrightarrow f \ k \ge (0 :: real)$  $\bigwedge k. \ k \ge m \Longrightarrow f \ (Suc \ k) \le f \ k \ f \longrightarrow 0$ defines  $S \equiv (\sum k. (-1) \widehat{(k+m)} * f (k+m))$ **shows** summable-alternating-decreasing: summable  $(\lambda k. (-1) \uparrow (k+m) * f (k+m))$ + m))alternating-decreasing-suminf-bounds': and if even m then  $S \in \{f m - f (Suc m) \dots f m\}$ else  $S \in \{-f m..f (Suc m) - f m\}$  (is ?th1) abs-alternating-decreasing-suminf: and abs  $S \in \{f m - f (Suc m) ... f m\}$  (is ?th2) proof – have summable: summable  $(\lambda k. (-1) \uparrow k * f (k + m))$ using assms by (intro summable-Leibniz') (auto simp: filterlim-sequentially-shift) thus summable  $(\lambda k. (-1) \uparrow (k+m) * f (k+m))$ by (subst add.commute) (auto simp: power-add mult.assoc intro: summable-mult) have  $S = (\sum k. (-1) \ \widehat{}\ m * ((-1) \ \widehat{}\ k * f \ (k + m)))$  $\mathbf{by}~(simp~add:~S\text{-}def~power\text{-}add~mult\text{-}ac)$ **also have** ... =  $(-1)^{-1} m * (\sum k. (-1)^{-1} k * f (k + m))$ using summable by (rule suminf-mult) finally have  $S = (-1) \ \hat{m} * (\sum k. (-1) \ \hat{k} * f \ (k+m))$ . moreover have  $(\sum k. (-1) \ \hat{k} * f \ (k+m)) \in \{f \ (0+m) - f \ (1+m) \ .. \ f \ (0+m)\}$ using assms **by** (*intro alternating-decreasing-suminf-bounds*) (auto simp: filterlim-sequentially-shift) ultimately show ?th1 by (auto split: if-splits) thus ?th2 using assms(2)[of m] by (auto split: if-splits)  $\mathbf{qed}$ 

#### lemma

assumes  $\bigwedge k. \ k \ge m \Longrightarrow f \ k \ge (0 :: real)$   $\bigwedge k. \ k \ge m \Longrightarrow f \ (Suc \ k) < f \ k \ f \longrightarrow 0$ defines  $S \equiv (\sum k. \ (-1) \ (k + m) * f \ (k + m))$ 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 -

define S' where  $S' = (\sum k. (-1) (k + Suc (Suc m)) * f (k + Suc (Suc m)))$ 

have  $(\lambda k. (-1) \ (k + m) * f \ (k + m))$  sums S using assms unfolding S-def by (intro summable-sums summable-Leibniz' summable-alternating-decreasing) (auto simp: less-eq-real-def) from sums-split-initial-segment[OF this, of 2] have S': S' = S - (-1) \ m \* (f m - f \ (Suc m)) by (simp-all add: sums-iff S'-def algebra-simps lessThan-nat-numeral) have if even (Suc (Suc m)) then S'  $\in$  {f (Suc (Suc m)) - f (Suc (Suc (Suc m))) else S'  $\in$  {- f (Suc (Suc m))..f (Suc (Suc (Suc m))) - f (Suc (Suc m))} unfolding S'-def using assms by (intro alternating-decreasing-suminf-bounds') (auto simp: less-eq-real-def) thus ?th1 using assms(2)[of Suc m] assms(2)[of Suc (Suc m)] unfolding S' by (auto simp: algebra-simps) thus ?th2 using assms(2)[of m] by (auto split: if-splits) qed

datatype cfrac = CFrac int nat llist

quickcheck-generator cfrac constructors: CFrac

```
lemma type-definition-cfrac':
```

type-definition ( $\lambda x$ . case x of CFrac a  $b \Rightarrow (a, b)$ ) ( $\lambda(x,y)$ . CFrac x y) UNIV by (auto simp: type-definition-def split: cfrac.splits)

setup-lifting type-definition-cfrac'

```
lift-definition cfrac-of-int :: int \Rightarrow cfrac is \lambda n. (n, LNil).
```

**lemma** cfrac-of-int-code [code]: cfrac-of-int n = CFrac n LNil**by** (*auto simp*: cfrac-of-int-def)

**lift-definition** cfrac-of-stream :: int stream  $\Rightarrow$  cfrac is  $\lambda xs. (shd xs, llist-of-stream (smap (<math>\lambda x. nat (x - 1)) (stl xs)))$ .

instantiation cfrac :: zerobegin definition zero-cfrac where 0 = cfrac-of-int 0instance .. end

```
instantiation cfrac :: one
begin
definition one-cfrac where 1 = cfrac-of-int 1
instance ..
end
```

**lift-definition**  $cfrac-tl :: cfrac \Rightarrow cfrac$  is

 $\lambda(-, bs) \Rightarrow case \ bs \ of \ LNil \Rightarrow (1, \ LNil) \mid LCons \ b \ bs' \Rightarrow (int \ b + 1, \ bs')$ .

 $\mathbf{lemma} \ cfrac\mbox{-}tl\mbox{-}code \ [code]:$ 

 $cfrac-tl \ (CFrac \ a \ bs) =$ (case bs of  $LNil \Rightarrow CFrac \ 1 \ LNil \mid LCons \ b \ bs' \Rightarrow CFrac \ (int \ b + 1) \ bs')$ by (auto simp:  $cfrac-tl-def \ split: \ llist.splits)$ 

- **definition** *cfrac-drop* :: *nat*  $\Rightarrow$  *cfrac*  $\Rightarrow$  *cfrac* **where** *cfrac-drop n c* = (*cfrac-tl*  $\frown n$ ) *c*
- **lemma** cfrac-drop-Suc-right: cfrac-drop (Suc n) c = cfrac-drop n (cfrac-tl c)by (simp add: cfrac-drop-def funpow-Suc-right del: funpow.simps)
- **lemma** cfrac-drop-Suc-left: cfrac-drop (Suc n) c = cfrac-tl (cfrac-drop n c) by (simp add: cfrac-drop-def)
- **lemma** cfrac-drop-add: cfrac-drop (m + n) c = cfrac-drop m (cfrac-drop n c)**by** (simp add: cfrac-drop-def funpow-add)
- **lemma** cfrac-drop-0 [simp]: cfrac-drop  $0 = (\lambda x. x)$ **by** (simp add: fun-eq-iff cfrac-drop-def)
- **lemma** cfrac-drop-1 [simp]: cfrac-drop 1 = cfrac-tl **by** (simp add: fun-eq-iff cfrac-drop-def)
- **lift-definition** *cfrac-length* :: *cfrac*  $\Rightarrow$  *enat* is  $\lambda(-, bs) \Rightarrow$  *llength bs*.
- **lemma** cfrac-length-code [code]: cfrac-length (CFrac a bs) = llength bs **by** (simp add: cfrac-length-def)
- **lemma** cfrac-length-tl [simp]: cfrac-length (cfrac-tl c) = cfrac-length c 1by transfer (auto split: llist.splits)
- **lemma** enat-diff-Suc-right [simp]: m enat (Suc n) = m n 1by (auto simp: diff-enat-def enat-1-iff split: enat.splits)

lemma cfrac-length-drop [simp]: cfrac-length (cfrac-drop n c) = cfrac-length c - n

by (induction n) (auto simp: cfrac-drop-def)

**lemma** cfrac-length-of-stream [simp]: cfrac-length (cfrac-of-stream xs) =  $\infty$  by transfer auto

**lift-definition** cfrac-nth :: cfrac  $\Rightarrow$  nat  $\Rightarrow$  int is  $\lambda(a :: int, bs :: nat llist). \lambda(n :: nat).$ if n = 0 then a else if  $n \leq llength$  bs then int (lnth bs (n - 1)) + 1 else 1. **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 have  $n > 0 \longrightarrow enat (n - Suc \ 0) < llength bs \iff enat \ n \leq llength bs$ **by** (*metis Suc-ile-eq Suc-pred*) thus ?thesis by (auto simp: cfrac-nth-def lnth-default-def) qed **lemma** cfrac-nth-nonneg [simp, intro]:  $n > 0 \implies$  cfrac-nth  $c \ n \ge 0$ by transfer auto **lemma** cfrac-nth-nonzero [simp]:  $n > 0 \implies$  cfrac-nth  $c \ n \neq 0$ by transfer (auto split: if-splits) **lemma** cfrac-nth-pos[simp, intro]:  $n > 0 \implies$  cfrac-nth  $c \ n > 0$ **by** transfer auto **lemma** cfrac-nth-ge-1[simp, intro]:  $n > 0 \implies$  cfrac-nth  $c \ n \ge 1$ by transfer auto **lemma** cfrac-nth-not-less-1[simp, intro]:  $n > 0 \implies \neg$  cfrac-nth c n < 1by transfer (auto split: if-splits) **lemma** cfrac-nth-tl [simp]: cfrac-nth (cfrac-tl c) n = cfrac-nth c (Suc n) apply transfer **apply** (auto split: llist.splits nat.splits simp: Suc-ile-eq lnth-LCons enat-0-iff simp flip: zero-enat-def) done **lemma** cfrac-nth-drop [simp]: cfrac-nth (cfrac-drop n c) m = cfrac-nth c (m + n)**by** (*induction n arbitrary: m*) (*auto simp: cfrac-drop-def*) **lemma** cfrac-nth-0-of-int [simp]: cfrac-nth (cfrac-of-int n) 0 = nby transfer auto **lemma** cfrac-nth-qt0-of-int [simp]:  $m > 0 \implies$  cfrac-nth (cfrac-of-int n) m = 1**by** transfer (auto simp: enat-0-iff) **lemma** cfrac-nth-of-stream: assumes sset (stl xs)  $\subseteq \{0 < ..\}$ **shows** cfrac-nth (cfrac-of-stream xs) n = snth xs nusing assms proof (transfer', goal-cases) case (1 xs n)thus ?case **by** (cases xs; cases n) (auto simp: subset-iff) qed

**lift-definition** *cfrac* ::  $(nat \Rightarrow int) \Rightarrow cfrac$  is  $\lambda f. (f \ 0, inf-llist (\lambda n. nat (f (Suc \ n) - 1)))$ .

**definition** *is-cfrac* ::  $(nat \Rightarrow int) \Rightarrow bool$  where *is-cfrac*  $f \leftrightarrow (\forall n > 0, f n > 0)$ 

**lemma** cfrac-nth-cfrac [simp]: assumes is-cfrac f **shows** cfrac-nth (cfrac f) n = f nusing assms unfolding is-cfrac-def by transfer auto **lemma** *llength-eq-infty-lnth*: *llength*  $b = \infty \implies inf-llist (lnth b) = b$ **by** (*simp add: llength-eq-infty-conv-lfinite*) **lemma** cfrac-cfrac-nth [simp]: cfrac-length  $c = \infty \implies$  cfrac (cfrac-nth c) = c by transfer (auto simp: llength-eq-infty-lnth) **lemma** cfrac-length-cfrac [simp]: cfrac-length (cfrac f) =  $\infty$ by transfer auto **lift-definition** *cfrac-of-list* :: *int list*  $\Rightarrow$  *cfrac* **is**  $\lambda xs. if xs = [] then (0, LNil) else (hd xs, llist-of (map (\lambda n. nat n - 1) (tl xs))).$ **lemma** cfrac-length-of-list [simp]: cfrac-length (cfrac-of-list xs) = length xs - 1by transfer (auto simp: zero-enat-def) **lemma** cfrac-of-list-Nil [simp]: cfrac-of-list [] = 0unfolding zero-cfrac-def by transfer auto **lemma** cfrac-nth-of-list [simp]: assumes n < length xs and  $\forall i \in \{0 < ... < length xs\}$ . xs ! i > 0**shows** cfrac-nth (cfrac-of-list xs) n = xs ! nusing assms **proof** (transfer, goal-cases) case (1 n xs)show ?case **proof** (cases n) case (Suc n') with 1 have  $xs \mid n > 0$ using 1 by auto hence int (nat (tl xs ! n') - Suc 0) + 1 = xs ! Suc n'

using 1(1) Suc by (auto simp: nth-tl of-nat-diff)
thus ?thesis
using Suc 1(1) by (auto simp: hd-conv-nth zero-enat-def)
qed (use 1 in <auto simp: hd-conv-nth>)
qed

**primcorec** *cfrac-of-real-aux* :: *real*  $\Rightarrow$  *nat llist* **where** 

 $cfrac-of-real-aux \ x = (if \ x \in \{0 < ... < 1\} \ then \ LCons \ (nat \ \lfloor 1/x \rfloor - 1) \ (cfrac-of-real-aux \ (frac \ (1/x))))$ else LNil)

**lemma** cfrac-of-real-aux-code [code]:

 $cfrac-of-real-aux \ x =$ 

 $(if \ x > 0 \land x < 1 \ then \ LCons \ (nat \ \lfloor 1/x \rfloor - 1) \ (cfrac-of-real-aux \ (frac \ (1/x))))$  else LNil)

**by** (subst cfrac-of-real-aux.code) auto

- **lemma** cfrac-of-real-aux-LNil [simp]:  $x \notin \{0 < ... < 1\} \implies$  cfrac-of-real-aux x = LNilby (subst cfrac-of-real-aux.code) auto
- **lemma** cfrac-of-real-aux-0 [simp]: cfrac-of-real-aux 0 = LNilby (subst cfrac-of-real-aux.code) auto

**lemma** cfrac-of-real-aux-eq-LNil-iff [simp]: cfrac-of-real-aux  $x = LNil \leftrightarrow x \notin \{0 < ... < 1\}$ by (subst cfrac-of-real-aux.code) auto

**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
using assms
apply (induction n arbitrary: x)
apply (subst cfrac-of-real-aux.code)
apply auto []
apply (subst cfrac-of-real-aux.code)
apply (auto)
done</pre>

**lift-definition** *cfrac-of-real* :: *real*  $\Rightarrow$  *cfrac* is  $\lambda x. (|x|, cfrac-of-real-aux (frac x))$ .

**lemma** cfrac-of-real-code [code]: cfrac-of-real  $x = CFrac \lfloor x \rfloor$  (cfrac-of-real-aux (frac x))

**by** (*simp add: cfrac-of-real-def*)

**lemma** eq-epred-iff:  $m = epred \ n \leftrightarrow m = 0 \land n = 0 \lor n = eSuc \ m$ by (cases m; cases n) (auto simp: enat-0-iff enat-eSuc-iff infinity-eq-eSuc-iff)

**lemma** epred-eq-iff: epred  $n = m \leftrightarrow m = 0 \land n = 0 \lor n = eSuc m$ by (cases m; cases n) (auto simp: enat-0-iff enat-eSuc-iff infinity-eq-eSuc-iff)

**lemma** epred-less:  $n > 0 \implies n \neq \infty \implies$  epred n < nby (cases n) (auto simp: enat-0-iff) **lemma** cfrac-nth-of-real-0 [simp]:  $cfrac-nth (cfrac-of-real x) \theta = |x|$ by transfer auto **lemma** frac-eq-0 [simp]:  $x \in \mathbb{Z} \Longrightarrow$  frac x = 0by simp **lemma** *cfrac-tl-of-real*: assumes  $x \notin \mathbb{Z}$ **shows** cfrac-tl (cfrac-of-real x) = cfrac-of-real (1 / frac x)using assms **proof** (*transfer*, *goal-cases*) case (1 x)hence int (nat  $|1 | frac x| - Suc \theta$ ) + 1 = |1 | frac x|**by** (*subst of-nat-diff*) (*auto simp: le-nat-iff frac-le-1*) with  $\langle x \notin \mathbb{Z} \rangle$  show ?case by (subst cfrac-of-real-aux.code) (auto split: llist.splits simp: frac-lt-1) qed **lemma** cfrac-nth-of-real-Suc: assumes  $x \notin \mathbb{Z}$ shows cfrac-nth (cfrac-of-real x) (Suc n) = cfrac-nth (cfrac-of-real (1 / fracx)) nproof have cfrac-nth (cfrac-of-real x) (Suc n) = cfrac-nth (cfrac-tl (cfrac-of-real x)) nby simp also have cfrac-tl (cfrac-of-real x) = cfrac-of-real (1 / frac x) **by** (*simp add: cfrac-tl-of-real assms*)

finally show ?thesis .

```
\mathbf{qed}
```

**fun**  $conv :: cfrac \Rightarrow nat \Rightarrow real$ **where**  $<math>conv \ c \ 0 = real-of-int \ (cfrac-nth \ c \ 0)$  $| \ conv \ c \ (Suc \ n) = real-of-int \ (cfrac-nth \ c \ 0) + 1 \ / \ conv \ (cfrac-tl \ c) \ n$ 

The numerator and denominator of a convergent:

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

 $\begin{array}{l} \textbf{fun conv-denom :: } cfrac \Rightarrow nat \Rightarrow int \textbf{ where} \\ conv-denom \ c \ 0 = 1 \\ | \ conv-denom \ c \ (Suc \ 0) = cfrac-nth \ c \ 1 \\ | \ conv-denom \ c \ (Suc \ (Suc \ n)) = cfrac-nth \ c \ (Suc \ (Suc \ n)) * \ conv-denom \ c \ (Suc \ n) \\ + \ conv-denom \ c \ n \end{array}$ 

lemma conv-num-rec:

 $n \ge 2 \Longrightarrow conv-num \ c \ n = cfrac-nth \ c \ n * conv-num \ c \ (n - 1) + conv-num \ c$ (n - 2)by (cases n; cases n - 1) auto

**lemma** conv-denom-rec:  $n \ge 2 \Longrightarrow$  conv-denom  $c \ n = cfrac$ -nth  $c \ n * conv$ -denom  $c \ (n - 1) + conv$ -denom  $c \ (n - 2)$ **by** (cases n; cases n - 1) auto

**fun**  $conv' :: cfrac \Rightarrow nat \Rightarrow real \Rightarrow real$ **where**  $<math>conv' c \ 0 \ z = z$  $| \ conv' \ c \ (Suc \ n) \ z = conv' \ c \ n \ (real-of-int \ (cfrac-nth \ c \ n) + 1 \ / \ z)$ 

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

**definition** conv-num-int ::  $cfrac \Rightarrow int \Rightarrow int$  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 ::  $cfrac \Rightarrow int \Rightarrow int$  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  $n \ge 0$ shows conv-num-int c n = cfrac-nth c (nat n) \* conv-num-int <math>c (n - 1) + conv-num-int c (n - 2)proof (cases  $n \ge 2$ ) case True define n' where n' = nat (n - 2)have n: n = int (Suc (Suc n')) using True by (simp add: n'-def) show ?thesis by (simp add: n conv-num-int-def nat-add-distrib) qed (use assms in (auto simp: conv-num-int-def >) lemma conv-denom-int-rec: assumes  $n \ge 0$ shows conv-denom-int c n = cfrac-nth c (nat n) \* conv-denom-int c (n - 1)

shows conv-denom-int c n = c/ac-nin c (nar n) \* conv-denom-int c (n + conv-denom-int c (n - 2))proof – consider  $n = 0 | n = 1 | n \ge 2$ using assms by force thus ?thesis proof cases assume  $n \ge 2$ 

```
define n' where n' = nat (n - 2)
have n: n = int (Suc (Suc n'))
using \langle n \geq 2 \rangle by (simp \ add: n'-def)
show ?thesis
by (simp \ add: n \ conv-denom-int-def \ nat-add-distrib)
qed (use \ assms \ in \ \langle auto \ simp: \ conv-denom-int-def \rangle)
```

 $\mathbf{qed}$ 

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

definition  $cfrac-lim :: cfrac \Rightarrow real$  where cfrac-lim c =  $(case cfrac-length c of \infty \Rightarrow lim (conv c) | enat l \Rightarrow conv c l)$ lemma cfrac-lim-code [code]: cfrac-lim c =  $(case cfrac-length c of enat l \Rightarrow conv c l$   $| - \Rightarrow Code.abort (STR "Cannot compute infinite continued fraction") (\lambda-.$ <math>cfrac-lim c))

by (simp add: cfrac-lim-def split: enat.splits)

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

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

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

**lemma** conv'-left-induct: **assumes**  $\bigwedge c$ .  $P \ c \ 0 \ z \ \land c \ n$ .  $P \ (cfrac-tl \ c) \ n \ z \implies P \ c \ (Suc \ n) \ z$  **shows**  $P \ c \ n \ z$ **using** assms by (rule conv.induct)

**lemma** enat-less-diff-conv [simp]:

assumes  $a = \infty \lor b < \infty \lor c < \infty$ **shows**  $a < c - (b :: enat) \leftrightarrow a + b < c$ using assms by (cases a; cases b; cases c) auto **lemma** conv-eq-conv': conv c n = conv' c n (cfrac-nth c n) **proof** (cases n = 0) case False hence cfrac- $nth \ c \ n > 0$  by (auto introl: cfrac-nth-pos) thus ?thesis by (induction c n rule: conv.induct) (simp-all add: conv'-Suc-left) qed simp-all lemma conv-num-pos': assumes cfrac-nth c  $\theta > \theta$ **shows** conv-num  $c \ n > 0$ using assms by (induction n rule: fib.induct) (auto simp: introl: add-pos-nonneg) **lemma** conv-num-nonneg: cfrac-nth c  $0 \ge 0 \Longrightarrow$  conv-num c  $n \ge 0$ **by** (*induction* c n rule: conv-num.induct) (auto simp: intro!: mult-nonneg-nonneg add-nonneg-nonneg *intro*: *cfrac-nth-nonneg*) **lemma** conv-num-pos:  $cfrac-nth \ c \ 0 \ge 0 \implies n > 0 \implies conv-num \ c \ n > 0$ **by** (*induction c n rule: conv-num.induct*) (auto introl: mult-pos-pos mult-nonneg-nonneg add-pos-nonneg conv-num-nonneg cfrac-nth-pos *intro: cfrac-nth-nonneg simp: enat-le-iff*) **lemma** conv-denom-pos [simp, intro]: conv-denom  $c \ n > 0$ **by** (*induction* c n rule: conv-num.induct) (auto introl: add-nonneg-pos mult-nonneg-nonneg cfrac-nth-nonneg *simp*: *enat-le-iff*) **lemma** conv-denom-not-nonpos [simp]:  $\neg$  conv-denom c  $n \leq 0$ using conv-denom-pos[of c n] by linarith **lemma** conv-denom-not-neg [simp]:  $\neg$  conv-denom c n < 0using conv-denom-pos[of c n] by linarith **lemma** conv-denom-nonzero [simp]: conv-denom c  $n \neq 0$ using conv-denom-pos[of c n] by linarith **lemma** conv-denom-nonneg [simp, intro]: conv-denom  $c \ n \ge 0$ using conv-denom-pos[of c n] by linarith **lemma** conv-num-int-neg1 [simp]: conv-num-int c(-1) = 1by (simp add: conv-num-int-def)

- **lemma** conv-num-int-neg [simp]:  $n < 0 \implies n \neq -1 \implies$  conv-num-int c n = 0by (simp add: conv-num-int-def)
- **lemma** conv-num-int-of-nat [simp]: conv-num-int c (int n) = conv-num c nby (simp add: conv-num-int-def)

**lemma** conv-num-int-nonneg [simp]:  $n \ge 0 \implies$  conv-num-int  $c \ n =$  conv-num c (nat n)

- by (simp add: conv-num-int-def)
- **lemma** conv-denom-int-neg2 [simp]: conv-denom-int c(-2) = 1by (simp add: conv-denom-int-def)

lemma conv-denom-int-neg [simp]:  $n < 0 \implies n \neq -2 \implies$  conv-denom-int c n = 0

**by** (*simp add: conv-denom-int-def*)

**lemma** conv-denom-int-of-nat [simp]: conv-denom-int c (int n) = conv-denom c nby (simp add: conv-denom-int-def)

**lemma** conv-denom-int-nonneg [simp]:  $n \ge 0 \implies$  conv-denom-int c n = conv-denom c (nat n)

**by** (*simp add: conv-denom-int-def*)

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

lemma conv'-qt-1: assumes cfrac-nth c 0 > 0 x > 1shows  $conv' c \ n \ x > 1$ using assms **proof** (*induction* n *arbitrary*: c x) case (Suc n c x) from Suc. prems have pos: cfrac-nth c n > 0 using cfrac-nth-pos[of n c] by (cases n = 0) (auto simp: enat-le-iff) have 1 < 1 + 1 / xusing Suc.prems by simp also have  $\ldots \leq c frac \cdot n th \ c \ n + 1 \ / \ x \ using \ pos$ by (intro add-right-mono) (auto simp: of-nat-ge-1-iff) finally show ?case by (subst conv'-Suc-right, intro Suc.IH) (use Suc.prems in (auto simp: enat-le-iff)) qed auto **lemma** enat-eq-iff:  $a = enat \ b \longleftrightarrow (\exists a'. \ a = enat \ a' \land a' = b)$ by (cases a) auto

**lemma** eq-enat-iff: enat  $a = b \iff (\exists b'. b = enat b' \land a = b')$ by (cases b) auto **lemma** enat-diff-one [simp]: enat a - 1 = enat (a - 1)by (cases enat (a - 1)) (auto simp flip: idiff-enat-enat) lemma *conv'-eqD*: assumes  $conv' c \ n \ x = conv' \ c' \ n \ x \ x > 1 \ m < n$ **shows** cfrac-nth c m = cfrac-nth c' musing assms **proof** (induction n arbitrary: m c c') case (Suc n m c c') have gt: conv' (cfrac-tl c) n x > 1 conv' (cfrac-tl c') n x > 1by (rule conv'-gt-1; use Suc.prems in (force intro: cfrac-nth-pos simp: enat-le-iff))+ have eq: cfrac-nth c 0 + 1 / conv' (cfrac-tl c) n x = $cfrac-nth \ c' \ 0 \ + \ 1 \ / \ conv' \ (cfrac-tl \ c') \ n \ x$ using Suc.prems by (subst (asm) (1 2) conv'-Suc-left) auto **hence**  $| cfrac-nth \ c \ 0 + 1 \ / \ conv' \ (cfrac-tl \ c) \ n \ x | =$ |cfrac-nth c' 0 + 1 / conv' (cfrac-tl c') n x|**by** (simp only:) also from qt have floor (cfrac-nth c 0 + 1 / conv' (cfrac-tl c) n x) = cfrac-nth  $c \ \theta$ by (intro floor-unique) auto also from gt have floor (cfrac-nth c' 0 + 1 / conv' (cfrac-tl c') n x) = cfrac-nth  $c' \theta$ by (intro floor-unique) auto finally have [simp]: cfrac-nth  $c \ 0 = c$ frac-nth  $c' \ 0$  by simp show ?case **proof** (cases m) case (Suc m') from eq and gt have conv' (cfrac-tl c) n x = conv' (cfrac-tl c') n xby simp hence cfrac-nth (cfrac-tl c) m' = cfrac-nth (cfrac-tl c') m'using Suc.prems by (intro Suc.IH[of cfrac-tl c cfrac-tl c']) (auto simp: o-def Suc enat-le-iff) with Suc show ?thesis by simp **qed** simp-all qed simp-all context fixes c :: cfrac and h kdefines  $h \equiv conv$ -num c and  $k \equiv conv$ -denom c

```
defir
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)

using z
```

**proof** (*induction* n *arbitrary*: z) case  $\theta$ hence  $1 + z * cfrac-nth \ c \ 1 > 0$ **by** (*intro add-pos-nonneg*) (*auto simp: cfrac-nth-nonneg*) with 0 show ?case by (auto simp add: h-def k-def field-simps conv'-Suc-right max-def not-le)  $\mathbf{next}$ case (Suc n) have [simp]: h (Suc (Suc n)) = cfrac-nth c (n+2) \* h (n+1) + h n by (simp add: h-def) have [simp]: k (Suc (Suc n)) = cfrac-nth c (n+2) \* k (n+1) + k n **by** (*simp add: k-def*) define z' where  $z' = cfrac \cdot nth c (n+2) + 1 / z$ from  $\langle z > 0 \rangle$  have z' > 0**by** (*auto simp*: z'-def intro!: add-nonneg-pos cfrac-nth-nonneg) have z \* real-of-int (h (Suc (Suc n))) + real-of-int (h (Suc n)) =z \* (z' \* h (Suc n) + h n)using  $\langle z > 0 \rangle$  by (simp add: algebra-simps z'-def) also have  $\ldots = z * (conv' c (Suc (Suc n)) z' * (z' * k (Suc n) + k n))$ using  $\langle z' > 0 \rangle$  by (subst Suc.IH [symmetric]) auto also have  $\ldots = conv' c (Suc (Suc n)) z *$ (z \* k (Suc (Suc n)) + k (Suc n))unfolding z'-def using  $\langle z > 0 \rangle$ by (subst (2) conv'-Suc-right) (simp add: algebra-simps) finally show ?case .. qed lemma *conv'-num-denom*: assumes  $z > \theta$ shows conv' c (Suc (Suc n)) z =(z \* h (Suc n) + h n) / (z \* k (Suc n) + k n)proof have z \* real-of-int (k (Suc n)) + real-of-int (k n) > 0using assms by (intro add-pos-nonneg mult-pos-pos) (auto simp: k-def) with conv'-num-denom-aux[of z n] assms show ?thesis **by** (*simp add: divide-simps*) qed lemma conv-num-denom: conv c n = h n / k nproof – consider  $n = 0 \mid n = Suc \mid n$  where  $n = Suc (Suc \mid n)$ using not0-implies-Suc by blast thus ?thesis **proof** cases assume  $n = Suc \ \theta$ thus ?thesis by (auto simp: h-def k-def field-simps max-def conv-Suc)

 $\mathbf{next}$ 

```
fix m assume [simp]: n = Suc (Suc m)
   have conv c \ n = conv' \ c \ (Suc \ (Suc \ m)) \ (cfrac-nth \ c \ (Suc \ (Suc \ m)))
    by (subst conv-eq-conv') simp-all
   also have \ldots = h n / k n
    by (subst conv'-num-denom) (simp-all add: h-def k-def)
   finally show ?thesis .
 qed (auto simp: h-def k-def)
qed
lemma conv'-num-denom':
 assumes z > 0 and n \ge 2
 shows conv' c \ n \ z = (z * h \ (n - 1) + h \ (n - 2)) / (z * k \ (n - 1) + k \ (n - 2))
2))
 using assms conv'-num-denom[of z n - 2]
 by (auto simp: eval-nat-numeral Suc-diff-Suc)
lemma conv'-num-denom-int:
 assumes z > \theta
 shows conv' c n z =
          (z * conv-num-int c (int n - 1) + conv-num-int c (int n - 2)) /
          (z * conv-denom-int c (int n - 1) + conv-denom-int c (int n - 2))
proof –
 consider n = 0 \mid n = 1 \mid n \ge 2 by force
 thus ?thesis
 proof cases
   case 1
   thus ?thesis using conv-num-int-neg1 by auto
 next
   case 2
   thus ?thesis using assms by (auto simp: conv'-Suc-right field-simps)
 \mathbf{next}
   case 3
   thus ?thesis using conv'-num-denom'[OF assms(1), of nat n]
    by (auto simp: nat-diff-distrib h-def k-def)
 qed
qed
lemma conv-nonneg: cfrac-nth c 0 \ge 0 \Longrightarrow conv c n \ge 0
 by (subst conv-num-denom)
   (auto introl: divide-nonneg-nonneg conv-num-nonneg simp: h-def k-def)
lemma conv-pos:
 assumes cfrac-nth c \ 0 > 0
 shows
         conv \ c \ n > 0
proof -
 have conv c n = h n / k n
   using assms by (intro conv-num-denom)
 also from assms have \ldots > 0 unfolding h-def k-def
   by (intro divide-pos-pos) (auto intro!: conv-num-pos')
```

finally show ?thesis . qed

```
lemma conv-num-denom-prod-diff:
 k n * h (Suc n) - k (Suc n) * h n = (-1) \widehat{} n
 by (induction c n rule: conv-num.induct)
    (auto simp: k-def h-def algebra-simps)
lemma conv-num-denom-prod-diff':
 k (Suc n) * h n - k n * h (Suc n) = (-1) \cap Suc n
 by (induction c n rule: conv-num.induct)
    (auto simp: k-def h-def algebra-simps)
lemma
 fixes n :: int
 assumes n > -2
 shows conv-num-denom-int-prod-diff:
         conv-denom-int c n * conv-num-int c (n + 1) - 
           conv-denom-int c (n + 1) * conv-num-int c n = (-1) \land (nat (n + 2))
(is ?th1)
 and
         conv-num-denom-int-prod-diff':
         conv-denom-int c (n + 1) * conv-num-int c n - conv
           conv-denom-int c n * \text{conv-num-int } c (n + 1) = (-1) \widehat{} (nat (n + 3))
(is ?th2)
proof -
 from assms consider n = -2 \mid n = -1 \mid n \ge 0 by force
 thus ?th1 using conv-num-denom-prod-diff[of nat n]
   by cases (auto simp: h-def k-def nat-add-distrib)
 moreover from assms have nat (n + 3) = Suc (nat (n + 2)) by (simp add:
nat-add-distrib)
 ultimately show ?th2 by simp
qed
lemma coprime-conv-num-denom: coprime (h \ n) \ (k \ n)
proof (cases n)
 case [simp]: (Suc m)
 {
   fix d :: int
   assume d \ dvd \ h \ n and d \ dvd \ k \ n
   hence abs d dvd abs (k n * h (Suc n) - k (Suc n) * h n)
    by simp
   also have \ldots = 1
    by (subst conv-num-denom-prod-diff) auto
   finally have is-unit d by simp
 }
 thus ?thesis by (rule coprimeI)
qed (auto simp: h-def k-def)
```

**lemma** coprime-conv-num-denom-int:

```
assumes n \geq -2
 shows coprime (conv-num-int c n) (conv-denom-int c n)
proof -
 from assms consider n = -2 \mid n = -1 \mid n \ge 0 by force
  thus ?thesis by cases (insert coprime-conv-num-denom[of nat n], auto simp:
h-def k-def)
\mathbf{qed}
lemma mono-conv-num:
 assumes cfrac-nth c 0 \ge 0
 shows mono h
proof (rule incseq-SucI)
 show h \ n \le h (Suc n) for n
 proof (cases n)
   case \theta
   have 1 * cfrac-nth \ c \ 0 + 1 \le cfrac-nth \ c \ (Suc \ 0) * cfrac-nth \ c \ 0 + 1
     using assms by (intro add-mono mult-right-mono) auto
   thus ?thesis using assms by (simp add: le-Suc-eq Suc-le-eq h-def 0)
 \mathbf{next}
   case (Suc m)
   have 1 * h (Suc m) + 0 \leq cfrac-nth c (Suc (Suc m)) * h (Suc m) + h m
     using assms
     by (intro add-mono mult-right-mono)
       (auto simp: Suc-le-eq h-def intro!: conv-num-nonneg)
   with Suc show ?thesis by (simp add: h-def)
 qed
qed
lemma mono-conv-denom: mono k
proof (rule incseq-SucI)
 show k n \leq k (Suc n) for n
 proof (cases n)
   case \theta
   thus ?thesis by (simp add: le-Suc-eq Suc-le-eq k-def)
 \mathbf{next}
   case (Suc m)
   have 1 * k (Suc m) + 0 \leq cfrac-nth c (Suc (Suc m)) * k (Suc m) + k m
     by (intro add-mono mult-right-mono) (auto simp: Suc-le-eq k-def)
   with Suc show ?thesis by (simp add: k-def)
 qed
qed
lemma conv-num-leI: cfrac-nth c 0 \ge 0 \implies m \le n \implies h \ m \le h \ n
 using mono-conv-num by (auto simp: mono-def)
lemma conv-denom-leI: m \leq n \Longrightarrow k \ m \leq k \ n
 using mono-conv-denom by (auto simp: mono-def)
```

**lemma** conv-denom-lessI:

```
assumes m < n \ 1 < n
 shows k m < k n
proof (cases n)
 case [simp]: (Suc n')
 show ?thesis
 proof (cases n')
   case [simp]: (Suc n'')
   from assms have k m \leq 1 * k n' + 0
    by (auto intro: conv-denom-leI simp: less-Suc-eq)
   also have \ldots \leq cfrac \cdot n + k n' + 0
    using assms by (intro add-mono mult-mono) (auto simp: Suc-le-eq k-def)
   also have \ldots < cfrac-nth \ c \ n * k \ n' + k \ n'' unfolding k-def
    by (intro add-strict-left-mono conv-denom-pos assms)
   also have \ldots = k \ n \ by \ (simp \ add: k-def)
   finally show ?thesis .
 qed (insert assms, auto simp: k-def)
qed (insert assms, auto)
lemma conv-num-lower-bound:
 assumes cfrac-nth c \theta \geq \theta
 shows h n \ge fib n unfolding h-def
 using assms
proof (induction c n rule: conv-denom.induct)
 case (3 c n)
 hence conv-num c (Suc (Suc n)) \geq 1 * int (fib (Suc n)) + int (fib n)
   using 3.prems unfolding conv-num.simps
   by (intro add-mono mult-mono 3.IH) auto
 thus ?case by simp
qed auto
lemma conv-denom-lower-bound: k \ n \ge fib (Suc n)
 unfolding k-def
proof (induction c n rule: conv-denom.induct)
 case (3 c n)
 hence conv-denom c (Suc (Suc n)) \geq 1 * int (fib (Suc (Suc n))) + int (fib (Suc
n))
   using 3.prems unfolding conv-denom.simps
   by (intro add-mono mult-mono 3.IH) auto
 thus ?case by simp
qed (auto simp: Suc-le-eq)
lemma conv-diff-eq: conv c (Suc n) - conv c n = (-1) \ \hat{n} / (k \ n * k \ (Suc \ n))
proof –
 have pos: k n > 0 k (Suc n) > 0 unfolding k-def
   by (intro conv-denom-pos)+
 have conv \ c \ (Suc \ n) - conv \ c \ n =
        (k n * h (Suc n) - k (Suc n) * h n) / (k n * k (Suc n))
    using pos by (subst (1 2) conv-num-denom) (simp add: conv-num-denom
field-simps)
```

also have k n \* h (Suc n) – k (Suc n) \*  $h n = (-1) \cap n$ by (rule conv-num-denom-prod-diff) finally show ?thesis by simp qed

lemma conv-telescope: assumes  $m \leq n$ shows conv c m + ( $\sum i=m..<n.(-1) \hat{i} / (k \ i * k \ (Suc \ i))$ ) = conv c n proof have ( $\sum i=m..<n.(-1) \hat{i} / (k \ i * k \ (Suc \ i))$ ) = ( $\sum i=m..<n. \ conv \ c \ (Suc \ i) - \ conv \ c \ i$ ) by (simp add: conv-diff-eq assms del: conv.simps) also have conv c m + ... = conv c n using assms by (induction rule: dec-induct) simp-all finally show ?thesis . qed

#### $\mathbf{qed}$

**lemma** conv-denom-at-top: filterlim k at-top at-top **proof** (rule filterlim-at-top-mono) **show** filterlim ( $\lambda n$ . int (fib (Suc n))) at-top at-top **by** (rule filterlim-compose[OF filterlim-int-sequentially]) (simp add: fib-at-top filterlim-sequentially-Suc) **show** eventually ( $\lambda n$ . fib (Suc n)  $\leq k$  n) at-top **by** (intro always-eventually conv-denom-lower-bound allI) **qed** 

#### lemma

shows summable-conv-telescope: summable  $(\lambda i. (-1) \cap i / (k \ i * k \ (Suc \ i)))$  (is ?th1) and cfrac-remainder-bounds:  $|(\sum i. (-1) \cap (i + m) / (k \ (i + m) * k \ (Suc \ i + m)))| \in \{1/(k \ m * (k \ m + k \ (Suc \ m))) < ... < 1 / (k \ m * k \ (Suc \ m))\}$  (is ?th2) proof – have  $[simp]: k \ n > 0 \ k \ n \ge 0 \ \neg k \ n = 0$  for nby  $(auto \ simp: \ k-def)$ have k-rec:  $k \ (Suc \ (Suc \ n)) = cfrac$ -nth  $c \ (Suc \ (Suc \ n)) * k \ (Suc \ n) + k \ n$  for nby  $(simp \ add: \ k-def)$ have  $[simp]: \ a + b = 0 \iff a = 0 \land b = 0$  if  $a \ge 0 \ b \ge 0$  for  $a \ b :: real$ using that by linarith define g where  $g = (\lambda i. inverse (real-of-int (k i * k (Suc i))))$ 

{ fix m :: nathave filterlim ( $\lambda n. k n$ ) at-top at-top and filterlim ( $\lambda n. k$  (Suc n)) at-top at-top by (force simp: filterlim-sequentially-Suc intro: conv-denom-at-top)+ hence lim:  $q \longrightarrow 0$ **unfolding** *g*-*def of*-*int*-*mult* by (intro tendsto-inverse-0-at-top filterlim-at-top-mult-at-top *filterlim-compose*[OF *filterlim-real-of-int-at-top*]) from lim have A: summable  $(\lambda n. (-1) \cap (n+m) * g (n+m))$  unfolding g-def **by** (*intro summable-alternating-decreasing*) (auto intro!: conv-denom-leI mult-nonneg-nonneg) have 1 / (k m \* (real-of-int (k (Suc m)) + k m / 1)) <1 / (k m \* (k (Suc m) + k m / cfrac-nth c (m+2)))by (intro divide-left-mono mult-left-mono add-left-mono mult-pos-pos add-pos-pos divide-pos-pos) (auto simp: of-nat-ge-1-iff) also have  $\ldots = g m - g (Suc m)$ **by** (*simp add: g-def k-rec field-simps add-pos-pos*) finally have let  $1 / (k m * (real-of-int (k (Suc m)) + k m / 1)) \leq g m - g$  $(Suc \ m)$  by simphave \*:  $|(\sum i. (-1) \cap (i + m) * g (i + m))| \in \{g m - g (Suc m) < ... < g m\}$ using *lim* unfolding *g*-def by (intro abs-alternating-decreasing-suminf-strict) (auto intro!: conv-denom-lessI) also from le have  $\ldots \subseteq \{1 \mid (k \ m \ast (k \ (Suc \ m) + k \ m)) < \ldots < g \ m\}$ by (subst greater Than Less Than-subset eq-greater Than Less Than) autofinally have  $B: |\sum i. (-1) (i + m) * g (i + m)| \in ...$ . note A B $\mathbf{B} = this$ from AB(1)[of 0] show ?th1 by (simp add: field-simps g-def) from AB(2)[of m] show ?th2 by (simp add: g-def divide-inverse add-ac) qed **lemma** convergent-conv: convergent (conv c) proof -

have convergent (λn. conv c 0 + (∑i<n. (-1) ^i / (k i \* k (Suc i)))) using summable-conv-telescope by (intro convergent-add convergent-const) (simp-all add: summable-iff-convergent) also have ... = conv c by (rule ext, subst (2) conv-telescope [of 0, symmetric]) (simp-all add: atLeast0LessThan) finally show ?thesis . ged

**lemma** LIMSEQ-cfrac-lim: cfrac-length  $c = \infty \implies conv \ c \longrightarrow cfrac-lim \ c$ 

using convergent-conv by (auto simp: convergent-LIMSEQ-iff cfrac-lim-def)

**lemma** cfrac-lim-nonneg: assumes cfrac-nth c  $0 \ge 0$ **shows** cfrac-lim  $c \ge 0$ **proof** (cases cfrac-length c) case infinity have conv c - $\longrightarrow cfrac-lim c$ **by** (rule LIMSEQ-cfrac-lim) fact thus ?thesis **by** (*rule tendsto-lowerbound*) (auto intro!: conv-nonneg always-eventually assms) next **case** (enat l)thus *?thesis* using assms **by** (*auto simp: cfrac-lim-def conv-nonneq*) qed **lemma** sums-cfrac-lim-minus-conv: assumes cfrac-length  $c = \infty$ shows  $(\lambda i. (-1) (i + m) / (k (i + m) * k (Suc i + m)))$  sums (cfrac-lim c  $conv \ c \ m$ ) proof – have  $(\lambda n. \ conv \ c \ (n + m) - \ conv \ c \ m) \longrightarrow cfrac-lim \ c - \ conv \ c \ m$ by (auto introl: tendsto-diff LIMSEQ-cfrac-lim simp: filterlim-sequentially-shift assms) also have  $(\lambda n. \ conv \ c \ (n + m) - \ conv \ c \ m) =$  $(\lambda n. (\sum i=0 + m..< n + m. (-1) \hat{i} / (k i * k (Suc i))))$ **by** (subst conv-telescope [of m, symmetric]) simp-all also have ... =  $(\lambda n. (\sum i < n. (-1) \cap (i+m) / (k (i+m) * k (Suc i+m))))$ by (subst sum.shift-bounds-nat-ivl) (simp-all add: atLeast0LessThan) finally show ?thesis unfolding sums-def. qed **lemma** cfrac-lim-minus-conv-upper-bound: assumes m < c frac-length cshows  $|cfrac-lim c - conv c m| \le 1 / (k m * k (Suc m))$ **proof** (cases cfrac-length c) case infinity have cfrac-lim c - conv c m =  $(\sum i. (-1) (i + m) / (k (i + m) * k (Suc i + m)))$ + m)))using sums-cfrac-lim-minus-conv infinity by (simp add: sums-iff) **also note** *cfrac-remainder-bounds*[*of m*] finally show ?thesis by simp  $\mathbf{next}$ **case** [simp]:  $(enat \ l)$ show ?thesis **proof** (cases l = m) case True

thus ?thesis by (auto simp: cfrac-lim-def k-def) next  ${\bf case} \ {\it False}$ let  $?S = (\sum i = m ... < l. (-1) \cap i * (1 / real-of-int (k i * k (Suc i))))$ have  $[simp]: k n \ge 0 k n > 0$  for n **by** (*simp-all add*: *k-def*) hence cfrac-lim c - conv c m = conv c l - conv c m**by** (*simp add: cfrac-lim-def*) also have  $\ldots = ?S$ using assms by (subst conv-telescope [symmetric, of m]) auto finally have cfrac-lim  $c - conv \ c \ m = ?S$ . moreover have  $|?S| \leq 1$  / real-of-int  $(k \ m * k \ (Suc \ m))$ unfolding of-int-mult using assms False  $\mathbf{by}~(intro~abs-alternating-decreasing-sum-upper-bound'~divide-nonneg-nonneg$ *frac-le mult-mono*) (simp-all add: conv-denom-leI del: conv-denom.simps) ultimately show ?thesis by simp qed qed **lemma** cfrac-lim-minus-conv-lower-bound: assumes m < cfrac-length cshows  $|cfrac-lim c - conv c m| \ge 1 / (k m * (k m + k (Suc m)))$ **proof** (cases cfrac-length c) case infinity have cfrac-lim c - conv c m =  $(\sum i. (-1) \hat{(i + m)} / (k (i + m) * k (Suc i)))$ (+ m)))using sums-cfrac-lim-minus-conv infinity by (simp add: sums-iff) **also note** *cfrac-remainder-bounds*[*of m*] finally show ?thesis by simp  $\mathbf{next}$  $\mathbf{case}~[simp]:~(enat~l)$ let  $S = (\sum i=m..<l. (-1) \ i * (1 / real-of-int (k i * k (Suc i))))$ have  $[simp]: k n \ge 0 k n > 0$  for n **by** (*simp-all add: k-def*) hence cfrac-lim c - conv c m = conv c l - conv c m**by** (*simp add: cfrac-lim-def*) also have  $\ldots = ?S$ using assms by (subst conv-telescope [symmetric, of m]) (auto simp: split: enat.splits) finally have cfrac-lim  $c - conv \ c \ m = ?S$ . moreover have  $|?S| \geq 1 / (k m * (k m + k (Suc m)))$ **proof** (cases  $m < cfrac-length \ c - 1$ ) case False hence [simp]: m = l - 1 and l > 0 using assms **by** (*auto simp: not-less*) have  $1 / (k m * (k m + k (Suc m))) \le 1 / (k m * k (Suc m))$ unfolding *of-int-mult* 

by (intro divide-left-mono mult-mono mult-pos-pos) (auto introl: add-pos-pos) also from  $\langle l > 0 \rangle$  have  $\{m.. < l\} = \{m\}$  by *auto* hence 1 /  $(k \ m * k \ (Suc \ m)) = |?S|$ by simp finally show ?thesis.  $\mathbf{next}$ case True with assms have less: m < l - 1by *auto* have k m + k (Suc m) > 0  $\mathbf{by} \ (intro \ add$ -pos-pos) (auto simp: k-def) hence  $1 / (k m * (k m + k (Suc m))) \le 1 / (k m * k (Suc m)) - 1 / (k (Suc m))$ m) \* k (Suc (Suc m))) **by** (*simp add: divide-simps*) (*auto simp: k-def algebra-simps*) also have  $\ldots < |?S|$ unfolding *of-int-mult* using *less* by (intro abs-alternating-decreasing-sum-lower-bound' divide-nonneg-nonneg *frac-le mult-mono*) (simp-all add: conv-denom-leI del: conv-denom.simps) finally show ?thesis . ged ultimately show ?thesis by simp qed **lemma** cfrac-lim-minus-conv-bounds: assumes m < c frac-length cshows  $|cfrac-lim c - conv c m| \in \{1 / (k m * (k m + k (Suc m)))...1 / (k m m)\}$ \* k (Suc m))using cfrac-lim-minus-conv-lower-bound[of m] cfrac-lim-minus-conv-upper-bound[of m] assms

by auto

#### $\mathbf{end}$

**lemma** conv-pos': **assumes** n > 0 cfrac-nth  $c \ 0 \ge 0$  **shows** conv  $c \ n > 0$ **using** assms by (cases n) (auto simp: conv-Suc intro!: add-nonneq-pos conv-pos)

**lemma** conv-in-Rats [intro]: conv  $c \ n \in \mathbb{Q}$ **by** (induction  $c \ n \ rule: conv.induct$ ) (auto simp: conv-Suc o-def)

#### lemma

assumes  $0 < z1 \ z1 \le z2$ shows  $conv'-even-mono: even n \Longrightarrow conv' c \ n \ z1 \le conv' c \ n \ z2$ and  $conv'-odd-mono: odd \ n \Longrightarrow conv' c \ n \ z1 \ge conv' c \ n \ z2$ proof let  $?P = (\lambda n \ (f::nat \Rightarrow real \Rightarrow real).$ 

if even n then  $f n z 1 \leq f n z 2$  else  $f n z 1 \geq f n z 2$ ) have ?P n (conv' c) using assms **proof** (*induction* n *arbitrary*:  $z1 \ z2$ ) case (Suc n) note z12 = Suc.prems**consider**  $n = 0 \mid even \mid n > 0 \mid odd \mid n$  by force thus ?case proof cases assume  $n = \theta$ thus ?thesis using Suc by (simp add: conv'-Suc-right field-simps)  $\mathbf{next}$ assume n: even n n > 0with Suc.IH have IH:  $conv' c \ n \ z1 \le conv' c \ n \ z2$ if  $0 < z1 \ z1 \le z2$  for  $z1 \ z2$  using that by auto show ?thesis using Suc.prems n z12 by (auto simp: conv'-Suc-right field-simps intro!: IH add-pos-nonneg mult-nonneg-nonneg) next assume n: odd nhence [simp]: n > 0 by (auto introl: Nat.gr0I) from *n* and Suc.IH have IH: conv' c  $n z_1 \ge conv' c n z_2$ if  $0 < z1 \ z1 \le z2$  for  $z1 \ z2$  using that by auto show ?thesis using Suc.prems n **by** (*auto simp: conv'-Suc-right field-simps* intro!: IH add-pos-nonneg mult-nonneg-nonneg) qed qed auto thus even  $n \Longrightarrow conv' c \ n \ z1 \le conv' c \ n \ z2$ odd  $n \Longrightarrow conv' c \ n \ z1 \ge conv' c \ n \ z2$  by auto qed lemma **shows** conv-even-mono: even  $n \implies n \le m \implies conv \ c \ n \le conv \ c \ m$ and conv-odd-mono: odd  $n \implies n \le m \implies conv \ c \ n \ge conv \ c \ m$ proof – assume even n have A: conv c n < conv c (Suc (Suc n)) if even n for n **proof** (cases  $n = \theta$ ) case False with  $\langle even n \rangle$  show ?thesis by (auto simp add: conv-eq-conv' conv'-Suc-right intro: conv'-even-mono) **qed** (*auto simp: conv-Suc*) have B: conv c  $n \leq conv c$  (Suc n) if even n for n **proof** (cases n = 0)  $\mathbf{case} \ \mathit{False}$ with (even n) show ?thesis by (auto simp add: conv-eq-conv' conv'-Suc-right intro: conv'-even-mono) **qed** (*auto simp*: *conv-Suc*)

show conv c  $n \leq conv c m$  if  $n \leq m$  for m using that **proof** (*induction m rule: less-induct*) case (less m) from  $(n \leq m)$  consider  $m = n \mid even \mid m > n \mid odd \mid m > n$ by *force* thus ?case proof cases assume m: even m m > nwith (even n) have  $m': m - 2 \ge n$  by presburger with m have conv c  $n \leq conv c (m - 2)$ by (intro less.IH) auto also have  $\ldots \leq conv \ c \ (Suc \ (m-2)))$ using m m' by (intro A) auto also have Suc (Suc (m - 2)) = musing *m* by *presburger* finally show ?thesis . next assume m: odd m m > nhence conv c  $n \leq conv c (m - 1)$  $\mathbf{by}~(\mathit{intro~less.IH})~\mathit{auto}$ also have  $\ldots \leq conv \ c \ (Suc \ (m-1))$ using m by (intro B) auto also have Suc (m - 1) = musing m by simpfinally show ?thesis . **qed** simp-all ged  $\mathbf{next}$ assume odd nhave A: conv c  $n \ge conv c$  (Suc (Suc n)) if odd n for n using that by (auto simp add: conv-eq-conv' conv'-Suc-right odd-pos intro!: conv'-odd-mono) have B: conv c  $n \ge conv c$  (Suc n) if odd n for n using that by (auto simp add: conv-eq-conv' conv'-Suc-right odd-pos intro!: conv'-odd-mono) show conv c  $n \ge conv c m$  if  $n \le m$  for m using that **proof** (*induction m rule: less-induct*) case (less m) from  $\langle n \leq m \rangle$  consider  $m = n \mid even \mid m > n \mid odd \mid m \mid n > n$ by *force* thus ?case **proof** cases assume m: odd m m > nwith (odd n) have  $m': m - 2 \ge n m \ge 2$  by presburger+ from m and (odd n) have m = Suc (Suc (m - 2)) by presburger also have conv c ...  $\leq$  conv c (m - 2)using m m' by (intro A) auto

```
also have \ldots \leq conv \ c \ n
      using m m' by (intro less.IH) auto
     finally show ?thesis .
   \mathbf{next}
     assume m: even m m > n
     from m have m = Suc (m - 1) by presburger
     also have conv c ... \leq conv c (m - 1)
       using m by (intro B) auto
     also have \ldots \leq conv \ c \ n
       using m by (intro less.IH) auto
     finally show ?thesis .
   qed simp-all
 qed
qed
lemma
 assumes m \leq c frac-length c
 shows conv-le-cfrac-lim: even m \Longrightarrow conv \ c \ m \le cfrac-lim \ c
   and conv-ge-cfrac-lim: odd m \Longrightarrow conv \ c \ m \ge cfrac-lim \ c
proof –
 have if even m then conv c m \leq cfrac-lim c else conv c m \geq cfrac-lim c
 proof (cases cfrac-length c)
   case [simp]: infinity
   show ?thesis
   proof (cases even m)
     case True
     have eventually (\lambda i. conv c m \leq conv c i) at-top
     using eventually-ge-at-top[of m] by eventually-elim (rule conv-even-mono[OF
True])
     hence conv c m \leq cfrac-lim c
      by (intro tendsto-lowerbound[OF LIMSEQ-cfrac-lim]) auto
     thus ?thesis using True by simp
   next
     {\bf case} \ {\it False}
     have eventually (\lambda i. conv c m \ge conv c i) at-top
     using eventually-ge-at-top[of m] by eventually-elim (rule conv-odd-mono[OF
False])
     hence conv c m \ge cfrac-lim c
      by (intro tendsto-upperbound[OF LIMSEQ-cfrac-lim]) auto
     thus ?thesis using False by simp
   qed
  \mathbf{next}
   case [simp]: (enat \ l)
   show ?thesis
     using conv-even-mono[of m \ l \ c] conv-odd-mono[of m \ l \ c] assms
     by (auto simp: cfrac-lim-def)
 qed
 thus even m \Longrightarrow conv \ c \ m \le cfrac-lim \ c and odd \ m \Longrightarrow conv \ c \ m \ge cfrac-lim \ c
   by auto
```

#### $\mathbf{qed}$

```
lemma cfrac-lim-ge-first: cfrac-lim c \ge cfrac-nth c \ 0
 using conv-le-cfrac-lim of 0 c by (auto simp: less-eq-enat-def split: enat.splits)
lemma cfrac-lim-pos: cfrac-nth c \ 0 > 0 \Longrightarrow cfrac-lim c > 0
 by (rule less-le-trans[OF - cfrac-lim-ge-first]) auto
lemma conv'-eq-iff:
 assumes 0 \le z1 \lor 0 \le z2
 shows conv' c n z1 = conv' c n z2 \leftrightarrow z1 = z2
proof
 assume conv' c \ n \ z1 = conv' c \ n \ z2
 thus z1 = z2 using assms
 proof (induction n arbitrary: z1 \ z2)
   case (Suc n)
   show ?case
   proof (cases n = 0)
    case True
     thus ?thesis using Suc by (auto simp: conv'-Suc-right)
   \mathbf{next}
     case False
     have conv' c n (real-of-int (cfrac-nth c n) + 1 / z1) =
             conv' c \ n \ (real-of-int \ (cfrac-nth \ c \ n) + 1 \ / \ z2) \ using \ Suc.prems
      by (simp add: conv'-Suc-right)
     hence real-of-int (cfrac-nth c n) + 1 / z1 = real-of-int (cfrac-nth c n) + 1
/ z2
      by (rule Suc.IH)
         (insert Suc.prems False, auto introl: add-nonneg-pos add-nonneg-nonneg)
     with Suc. prems show z1 = z2 by simp
   qed
 qed auto
qed auto
lemma conv-even-mono-strict:
 assumes even n n < m
 shows conv c n < conv c m
proof (cases m = n + 1)
 case [simp]: True
 show ?thesis
 proof (cases n = \theta)
   case True
   thus ?thesis using assms by (auto simp: conv-Suc)
 next
   \mathbf{case} \ \mathit{False}
   hence conv' c n (real-of-int (cfrac-nth c n)) \neq
          conv' c n (real-of-int (cfrac-nth c n) + 1 / real-of-int (cfrac-nth c (Suc
n)))
    by (subst conv'-eq-iff) auto
```

```
with assms have conv c n \neq conv c m
    by (auto simp: conv-eq-conv' conv'-eq-iff conv'-Suc-right field-simps)
   moreover from assms have conv c n \leq conv c m
    by (intro conv-even-mono) auto
   ultimately show ?thesis by simp
 qed
\mathbf{next}
 case False
 show ?thesis
 proof (cases n = 0)
   case True
   thus ?thesis using assms
    by (cases m) (auto simp: conv-Suc conv-pos)
 next
   case False
   have 1 + real-of-int (cfrac-nth c (n+1)) * cfrac-nth c (n+2) > 0
    by (intro add-pos-nonneg) auto
   with assms have conv c n \neq conv c (Suc (Suc n))
    unfolding conv-eq-conv' conv'-Suc-right using False
    by (subst conv'-eq-iff) (auto simp: field-simps)
   moreover from assms have conv c \ n \leq conv \ c \ (Suc \ (Suc \ n))
    by (intro conv-even-mono) auto
   ultimately have conv c \ n < conv \ c \ (Suc \ (Suc \ n)) by simp
   also have \ldots \leq conv \ c \ m \ using \ assms \ \langle m \neq n + 1 \rangle
    by (intro conv-even-mono) auto
   finally show ?thesis .
 ged
qed
lemma conv-odd-mono-strict:
 assumes odd n n < m
 shows conv c n > conv c m
proof (cases m = n + 1)
 case [simp]: True
 from assms have n > 0 by (intro Nat.gr0I) auto
 hence conv' c n (real-of-int (cfrac-nth c n)) \neq
      conv' c n (real-of-int (cfrac-nth c n) + 1 / real-of-int (cfrac-nth c (Suc n)))
   by (subst conv'-eq-iff) auto
 hence conv \ c \ n \neq conv \ c \ m
   by (simp add: conv-eq-conv' conv'-Suc-right)
 moreover from assms have conv c n \ge conv c m
   by (intro conv-odd-mono) auto
 ultimately show ?thesis by simp
next
 case False
 from assms have n > 0 by (intro Nat.gr0I) auto
 have 1 + real-of-int (cfrac-nth c (n+1)) * cfrac-nth c (n+2) > 0
   by (intro add-pos-nonneg) auto
```
```
with assms \langle n > 0 \rangle have conv c n \neq conv c (Suc (Suc n))
   unfolding conv-eq-conv' conv'-Suc-right
   by (subst conv'-eq-iff) (auto simp: field-simps)
 moreover from assms have conv c n \ge conv c (Suc (Suc n))
   by (intro conv-odd-mono) auto
 ultimately have conv c \ n > conv \ c \ (Suc \ (Suc \ n)) by simp
 moreover have conv c (Suc (Suc n)) \geq conv c m using assms False
   by (intro conv-odd-mono) auto
 ultimately show ?thesis by linarith
qed
lemma conv-less-cfrac-lim:
 assumes even n n < cfrac-length c
 shows conv c \ n < cfrac-lim \ c
proof (cases cfrac-length c)
 case (enat l)
 with assms show ?thesis by (auto simp: cfrac-lim-def conv-even-mono-strict)
next
 case [simp]: infinity
 from assms have conv c n < conv c (n + 2)
   by (intro conv-even-mono-strict) auto
 also from assms have \ldots \leq cfrac-lim c
   by (intro conv-le-cfrac-lim) auto
 finally show ?thesis .
qed
lemma conv-gt-cfrac-lim:
 assumes odd n n < cfrac-length c
 shows conv \ c \ n > cfrac-lim \ c
proof (cases cfrac-length c)
 case (enat l)
 with assms show ?thesis by (auto simp: cfrac-lim-def conv-odd-mono-strict)
next
 case [simp]: infinity
 from assms have cfrac-lim c \leq conv \ c \ (n+2)
   by (intro conv-ge-cfrac-lim) auto
 also from assms have \ldots < conv \ c \ n
   by (intro conv-odd-mono-strict) auto
 finally show ?thesis .
qed
lemma conv-neq-cfrac-lim:
 assumes n < cfrac-length c
 shows conv c n \neq cfrac-lim c
 using conv-gt-cfrac-lim[OF - assms] conv-less-cfrac-lim[OF - assms]
 by (cases even n) auto
lemma conv-ge-first: conv c n \ge cfrac-nth \ c \ 0
```

```
using conv-even-mono[of 0 \ n \ c] by simp
```

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  $xs \land n = 0$ **unfolding** cfrac-is-zero-def lnull-def zero-cfrac-def cfrac-of-int-def **by** (auto simp: cfrac-length-def)

```
definition cfrac-is-int where cfrac-is-int c \leftrightarrow cfrac-length c = 0
lemma cfrac-is-int-code [code]: cfrac-is-int (CFrac n xs) \leftrightarrow lnull xs
 unfolding cfrac-is-int-def lnull-def by (auto simp: cfrac-length-def)
lemma cfrac-length-of-int [simp]: cfrac-length (cfrac-of-int n) = 0
 by transfer auto
lemma cfrac-is-int-of-int [simp, intro]: cfrac-is-int (cfrac-of-int n)
 unfolding cfrac-is-int-def by simp
lemma cfrac-is-int-iff: cfrac-is-int c \leftrightarrow (\exists n. c = cfrac-of-int n)
proof –
 have c = cfrac \text{-}of \text{-}int (cfrac \text{-}nth \ c \ 0) if cfrac \text{-}is \text{-}int \ c
   using that unfolding cfrac-is-int-def by transfer auto
  thus ?thesis
   by auto
qed
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 (cases cfrac-length c)
 case [simp]: infinity
 have \theta < cfrac-nth (cfrac-tl c) \theta
   by simp
 also have \ldots < cfrac-lim (cfrac-tl c)
   by (rule cfrac-lim-ge-first)
 finally have (\lambda n. real-of-int (cfrac-nth \ c \ 0) + 1 \ / \ conv \ (cfrac-tl \ c) \ n) \longrightarrow
          real-of-int (cfrac-nth \ c \ 0) + 1 \ / \ cfrac-lim \ (cfrac-tl \ c)
   by (intro tendsto-intros LIMSEQ-cfrac-lim) auto
 also have (\lambda n. real-of-int (cfrac-nth c 0) + 1 / conv (cfrac-tl c) n) = conv c \circ
Suc
   by (simp add: o-def conv-Suc)
 finally have *: conv \ c \longrightarrow real-of-int \ (cfrac-nth \ c \ 0) + 1 \ / \ cfrac-lim \ (cfrac-tl
c)
   by (simp add: o-def filterlim-sequentially-Suc)
 show ?thesis
   by (rule tendsto-unique[OF - LIMSEQ-cfrac-lim *]) auto
next
```

```
case [simp]: (enat \ l)
 from assms obtain l' where [simp]: l = Suc l'
   by (cases l) (auto simp: cfrac-is-int-def zero-enat-def)
 thus ?thesis
   by (auto simp: cfrac-lim-def conv-Suc)
\mathbf{qed}
lemma cfrac-lim-tl:
 assumes \neg cfrac-is-int c
 shows cfrac-lim (cfrac-tl c) = 1 / (cfrac-lim c - cfrac-nth c 0)
 using cfrac-lim-reduce[OF assms] by simp
lemma cfrac-remainder-Suc':
 assumes n < c frac-length c
 shows cfrac-remainder c (Suc n) * (cfrac-remainder c n - cfrac-nth c n) = 1
proof -
 have 0 < real-of-int (cfrac-nth c (Suc n)) by simp
 also have cfrac-nth c (Suc n) \leq cfrac-remainder c (Suc n)
   using cfrac-lim-ge-first[of cfrac-drop (Suc n) c]
   by (simp add: cfrac-remainder-def)
 finally have \ldots > 0.
 have cfrac-remainder c (Suc n) = cfrac-lim (cfrac-tl (cfrac-drop n c))
   by (simp add: o-def cfrac-remainder-def cfrac-drop-Suc-left)
 also have \ldots = 1 / (cfrac-remainder c n - cfrac-nth c n) using assms
  by (subst cfrac-lim-tl) (auto simp: cfrac-remainder-def cfrac-is-int-def enat-less-iff
enat-0-iff)
 finally show ?thesis
   using \langle cfrac-remainder c (Suc \ n) > 0 \rangle
   by (auto simp add: cfrac-remainder-def field-simps)
qed
lemma cfrac-remainder-Suc:
 assumes n < cfrac-length c
 shows cfrac-remainder c (Suc n) = 1 / (cfrac-remainder c n - cfrac-nth c n)
proof –
 have cfrac-remainder c (Suc n) = cfrac-lim (cfrac-tl (cfrac-drop n c))
   by (simp add: o-def cfrac-remainder-def cfrac-drop-Suc-left)
 also have \ldots = 1 / (cfrac-remainder c n - cfrac-nth c n) using assms
  by (subst cfrac-lim-tl) (auto simp: cfrac-remainder-def cfrac-is-int-def enat-less-iff)
enat-0-iff)
 finally show ?thesis .
```



**lemma** cfrac-remainder-0 [simp]: cfrac-remainder c 0 = cfrac-lim c **by** (simp add: cfrac-remainder-def)

#### context

fixes c h k xdefines  $h \equiv conv$ -num c and  $k \equiv conv$ -denom c and  $x \equiv cfrac$ -remainder c begin **lemma** cfrac-lim-eq-num-denom-remainder-aux: assumes Suc (Suc n)  $\leq$  cfrac-length c shows  $cfrac-lim \ c * (k \ (Suc \ n) * x \ (Suc \ (Suc \ n)) + k \ n) = h \ (Suc \ n) * x \ (Suc \ n)$  $(Suc \ n)) + h \ n$ using assms **proof** (*induction* n) case  $\theta$ have cfrac-lim  $c \neq c$ frac-nth c 0 using conv-neq-cfrac-lim[of 0 c] 0 by (auto simp: enat-le-iff) **moreover have** cfrac-nth c 1 \* (cfrac-lim c - cfrac-nth  $c 0) \neq 1$ using conv-neq-cfrac-lim[of 1 c] 0 **by** (*auto simp: enat-le-iff conv-Suc field-simps*) ultimately show ?case using assms by (auto simp: cfrac-remainder-Suc divide-simps x-def h-def k-def enat-le-iff) (auto simp: field-simps) next case (Suc n) have less: enat (Suc (Suc n)) < cfrac-length cusing Suc. prems by (cases cfrac-length c) auto have  $*: x (Suc (Suc n)) \neq real-of-int (cfrac-nth c (Suc (Suc n)))$ using conv-neq-cfrac-lim[of 0 cfrac-drop (n+2) c] Suc.prems by (cases cfrac-length c) (auto simp: x-def cfrac-remainder-def) hence  $cfrac-lim \ c * (k \ (Suc \ (Suc \ n)) * x \ (Suc \ (Suc \ (Suc \ n))) + k \ (Suc \ n)) =$  $(cfrac-lim \ c * (k \ (Suc \ n) * x \ (Suc \ (Suc \ n)) + k \ n)) / (x \ (Suc \ (Suc \ n)) - k \ n)) / (x \ (Suc \ (Suc \ n))) / (x \ (Suc \ (Suc \ n))) - (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) - k \ n)) / (x \ (Suc \ (Suc \ n))) / (x \ (Suc \ (Suc \ n))) / (x \ (Suc \ n)) / (x \ (Suc \ n)) / (x \ (Suc \ n))) / (x \ (Suc \ n)) / (x \ (Suc$  $cfrac-nth \ c \ (Suc \ (Suc \ n)))$ unfolding x-def k-def h-def using less **by** (*subst cfrac-remainder-Suc*) (*auto simp: field-simps*) also have cfrac-lim c \* (k (Suc n) \* x (Suc (Suc n)) + k n) =h (Suc n) \* x (Suc (Suc n)) + h n using less by (intro Suc.IH) auto also have (h (Suc n) \* x (Suc (Suc n)) + h n) / (x (Suc (Suc n)) - cfrac-nth)c (Suc (Suc n))) =h (Suc (Suc n)) \* x (Suc (Suc (Suc n))) + h (Suc n) using \*unfolding x-def k-def h-def using less by (subst (3) cfrac-remainder-Suc) (auto simp: field-simps) finally show ?case . qed

**lemma** cfrac-remainder-nonneg: cfrac-nth  $c \ n \ge 0 \implies$  cfrac-remainder  $c \ n \ge 0$ unfolding cfrac-remainder-def by (rule cfrac-lim-nonneg) auto

**lemma** cfrac-remainder-pos: cfrac-nth  $c \ n > 0 \implies$  cfrac-remainder  $c \ n > 0$ unfolding cfrac-remainder-def by (rule cfrac-lim-pos) auto **lemma** cfrac-lim-eq-num-denom-remainder: assumes Suc (Suc n) < cfrac-length cshows cfrac-lim c = (h (Suc n) \* x (Suc (Suc n)) + h n) / (k (Suc n) \* x (Suc n)) + h n) / (k (Suc n) + h n) / (k (Suc n)) + h n) / (k (Suc $(Suc \ n)) + k \ n)$ proof have k (Suc n) \* x (Suc (Suc n)) + k n > 0**by** (*intro add-nonneg-pos mult-nonneg-nonneg*) (auto simp: k-def x-def intro!: conv-denom-pos cfrac-remainder-nonneg) with cfrac-lim-eq-num-denom-remainder-aux[of n] assms show ?thesis **by** (*auto simp add: field-simps h-def k-def x-def*) qed **lemma** *abs-diff-successive-convs*: shows |conv c (Suc n) - conv c n| = 1 / (k n \* k (Suc n))proof – have  $[simp]: k n \neq 0$  for n :: natunfolding k-def using conv-denom-pos[of c n] by auto have conv c (Suc n) - conv c n = h (Suc n) / k (Suc n) - h n / k n **by** (*simp add: conv-num-denom k-def h-def*) also have  $\ldots = (k n * h (Suc n) - k (Suc n) * h n) / (k n * k (Suc n))$ **by** (*simp add: field-simps*) also have k n \* h (Suc n) - k (Suc n) \* h n = (-1) ^ n **unfolding** *h*-def *k*-def **by** (*intro conv*-*num*-*denom*-*prod*-*diff*) finally show ?thesis by (simp add: k-def) qed **lemma** conv-denom-plus2-ratio-ge: k (Suc (Suc n))  $\geq 2 * k n$ proof have  $1 * k n + k n \leq cfrac-nth c (Suc (Suc n)) * k (Suc n) + k n$ by (*intro add-mono mult-mono*) (auto simp: k-def Suc-le-eq intro!: conv-denom-leI) thus ?thesis by (simp add: k-def) qed end **lemma** conv'-cfrac-remainder: assumes n < cfrac-length c**shows** conv' c n (cfrac-remainder c n) = cfrac-lim c using assms **proof** (*induction n arbitrary*: *c*) case (Suc n c) have conv' c (Suc n) (cfrac-remainder c (Suc n)) =  $cfrac-nth \ c \ 0 + 1 \ / \ conv' \ (cfrac-tl \ c) \ n \ (cfrac-remainder \ c \ (Suc \ n))$  ${\bf using} \ Suc.prems$ **by** (*subst conv'-Suc-left*) (*auto intro*!: *cfrac-remainder-pos*) also have cfrac-remainder c (Suc n) = cfrac-remainder (cfrac-tl c) n**by** (*simp add: cfrac-remainder-def cfrac-drop-Suc-right*)

also have conv' (cfrac-tl c)  $n \ldots = cfrac-lim$  (cfrac-tl c)

using Suc.prems by (subst Suc.IH) (auto simp: cfrac-remainder-def enat-less-iff) also have cfrac-nth  $c \ 0 + 1 \ / \ldots = cfrac-lim \ c$ 

using Suc.prems by (intro cfrac-lim-reduce [symmetric]) (auto simp: cfrac-is-int-def) finally show ?case by (simp add: cfrac-remainder-def cfrac-drop-Suc-right) qed auto

 $\begin{array}{l} \textbf{lemma } cfrac-lim-rational \ [intro]: \\ \textbf{assumes } cfrac-length \ c < \infty \\ \textbf{shows } cfrac-lim \ c \in \mathbb{Q} \\ \textbf{using } assms \ \textbf{by } (cases \ cfrac-length \ c) \ (auto \ simp: \ cfrac-lim-def) \\ \end{array}$ 

```
\begin{array}{l} \textbf{lemma } cfrac-length \ of real-irrational:\\ \textbf{assumes } x \notin \mathbb{Q}\\ \textbf{shows } cfrac-length \ (cfrac \ of real \ x) = \infty\\ \textbf{proof } (insert \ assms, \ transfer, \ clarify)\\ \textbf{fix } x :: real \ \textbf{assume } x \notin \mathbb{Q}\\ \textbf{thus } llength \ (cfrac \ of \ real \ aux \ (frac \ x)) = \infty\\ \textbf{using } linfinite \ cfrac \ of \ real \ aux \ (frac \ x)) = \infty\\ \textbf{using } linfinite \ cfrac \ of \ real \ aux \ (frac \ x) \ of \ frac \ x) \ Ints \ subset \ Rats\\ \textbf{by } (auto \ simp: \ linfinite \ conv \ llength \ frac \ lt \ 1)\\ \textbf{ged}\end{array}
```

```
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)))

using assms

by (transfer, subst cfrac-of-real-aux.code) (auto simp: frac-lt-1)
```

**lemma** cfrac-length-of-real-int [simp]:  $x \in \mathbb{Z} \implies$  cfrac-length (cfrac-of-real x) = 0 by transfer auto

```
lemma conv-cfrac-of-real-le-ge:

assumes n \le cfrac-length (cfrac-of-real x)

shows if even n then conv (cfrac-of-real x) n \le x else conv (cfrac-of-real x) n \le x

using assms

proof (induction n arbitrary: x)

case (Suc n x)
```

```
hence [simp]: x \notin \mathbb{Z}
   using Suc by (auto simp: enat-0-iff)
 let ?x' = 1 / frac x
 have enat n \leq cfrac-length (cfrac-of-real (1 / frac x))
  using Suc. prems by (auto simp: cfrac-length-of-real-reduce simp flip: eSuc-enat)
  hence IH: if even n then conv (cfrac-of-real ?x') n \leq ?x' else ?x' \leq conv
(cfrac-of-real ?x') n
   using Suc.prems by (intro Suc.IH) auto
 have remainder-pos: conv (cfrac-of-real ?x') n > 0
   by (rule conv-pos) (auto simp: frac-le-1)
 show ?case
 proof (cases even n)
   case True
   have x \leq real-of-int |x| + frac x
     by (simp add: frac-def)
   also have frac x < 1 / conv (cfrac-of-real ?x') n
     using IH True remainder-pos frac-gt-0-iff of x by (simp add: field-simps)
   finally show ?thesis using True
     by (auto simp: conv-Suc cfrac-tl-of-real)
 \mathbf{next}
   case False
   have real-of-int |x| + 1 / conv (cfrac-of-real ?x') n \leq real-of-int |x| + frac x
     using IH False remainder-pos frac-gt-0-iff [of x] by (simp add: field-simps)
   also have \ldots = x
     by (simp add: frac-def)
   finally show ?thesis using False
     by (auto simp: conv-Suc cfrac-tl-of-real)
 ged
\mathbf{qed} \ auto
lemma cfrac-lim-of-real [simp]: cfrac-lim (cfrac-of-real x) = x
proof (cases cfrac-length (cfrac-of-real x))
 case (enat l)
 hence conv (cfrac-of-real x) l = x
 proof (induction l arbitrary: x)
   case \theta
   hence x \in \mathbb{Z}
     using cfrac-length-of-real-reduce zero-enat-def by fastforce
   thus ?case by (auto elim: Ints-cases)
 next
   case (Suc l x)
   hence [simp]: x \notin \mathbb{Z}
     by (auto simp: enat-0-iff)
   have eSuc (cfrac-length (cfrac-of-real (1 / frac x))) = enat (Suc l)
     using Suc.prems by (auto simp: cfrac-length-of-real-reduce)
   hence conv (cfrac-of-real (1 / frac x)) l = 1 / frac x
     by (intro Suc.IH) (auto simp flip: eSuc-enat)
   thus ?case
     by (simp add: conv-Suc cfrac-tl-of-real frac-def)
```

#### $\mathbf{qed}$

thus ?thesis by (simp add: enat cfrac-lim-def)  $\mathbf{next}$ **case** [simp]: infinity have lim: conv (cfrac-of-real x)  $\longrightarrow$  cfrac-lim (cfrac-of-real x) **by** (*simp add: LIMSEQ-cfrac-lim*) have cfrac-lim (cfrac-of-real x)  $\leq x$ **proof** (*rule tendsto-upperbound*) **show**  $(\lambda n. \ conv \ (cfrac-of-real x) \ (n * 2)) \longrightarrow cfrac-lim \ (cfrac-of-real x)$ by (intro filterlim-compose[OF lim] mult-nat-right-at-top) auto show eventually ( $\lambda n$ . conv (cfrac-of-real x) (n \* 2)  $\leq x$ ) at-top using conv-cfrac-of-real-le-ge[of n \* 2 x for n] by (intro always-eventually) autoqed auto **moreover have** cfrac-lim (cfrac-of-real x)  $\geq x$ **proof** (rule tendsto-lowerbound) **show**  $(\lambda n. \ conv \ (cfrac-of-real x) \ (Suc \ (n * 2))) \longrightarrow cfrac-lim \ (cfrac-of-real x)$ x)by (intro filterlim-compose[OF lim] filterlim-compose[OF filterlim-Suc] mult-nat-right-at-top) auto **show** eventually  $(\lambda n. \ conv \ (cfrac-of-real x) \ (Suc \ (n * 2)) \ge x)$  at-top using conv-cfrac-of-real-le-ge[of Suc (n \* 2) x for n] by (intro always-eventually) autoqed auto ultimately show ?thesis by (rule antisym) qed lemma Ints-add-left-cancel:  $x \in \mathbb{Z} \implies x + y \in \mathbb{Z} \iff y \in \mathbb{Z}$ using Ints-diff [of x + y x] by auto lemma Ints-add-right-cancel:  $y \in \mathbb{Z} \implies x + y \in \mathbb{Z} \iff x \in \mathbb{Z}$ using Ints-diff [of x + y y] by auto **lemma** cfrac-of-real-conv': fixes m n :: natassumes x > 1 m < n**shows** cfrac-nth (cfrac-of-real (conv' c n x)) m = cfrac-nth c musing assms **proof** (*induction* n *arbitrary*: c m) case (Suc n c m) from Suc.prems have gt-1: 1 < conv' (cfrac-tl c) n xby (intro conv'-gt-1) (auto simp: enat-le-iff intro: cfrac-nth-pos) show ?case **proof** (cases m) case  $\theta$ thus ?thesis using gt-1 Suc.prems by (simp add: conv'-Suc-left nat-add-distrib floor-eq-iff) next case (Suc m')

from gt-1 have 1 / conv' (cfrac-tl c)  $n x \in \{0 < .. < 1\}$ by *auto* have 1 / conv' (cfrac-tl c)  $n x \notin \mathbb{Z}$ proof assume 1 / conv' (cfrac-tl c)  $n x \in \mathbb{Z}$ then obtain k :: int where k: 1 / conv' (cfrac-tl c) n x = of-int kby (elim Ints-cases) have real-of-int  $k \in \{0 < .. < 1\}$ using qt-1 by (subst k [symmetric]) auto thus False by auto qed hence not-int: real-of-int (cfrac-nth c 0) + 1 / conv' (cfrac-tl c)  $n x \notin \mathbb{Z}$ by (subst Ints-add-left-cancel) (auto simp: field-simps elim!: Ints-cases) have cfrac-nth (cfrac-of-real (conv' c (Suc n) x)) m =cfrac-nth (cfrac-of-real (of-int (cfrac-nth c 0) + 1 / conv' (cfrac-tl c) nx)) (Suc m')using  $\langle x > 1 \rangle$  by (subst conv'-Suc-left) (auto simp: Suc) also have  $\ldots = cfrac \cdot nth (cfrac \cdot of \cdot real (1 / frac (1 / conv' (cfrac \cdot tl c) n x)))$ m'using  $\langle x \rangle 1 \rangle$  Suc not-int by (subst cfrac-nth-of-real-Suc) (auto simp: *frac-add-of-int*) also have  $1 / conv' (cfrac-tl c) \ n \ x \in \{0 < .. < 1\}$  using gt-1by (auto simp: field-simps) hence frac (1 / conv' (cfrac-tl c) n x) = 1 / conv' (cfrac-tl c) n x**by** (subst frac-eq) auto hence 1 / frac (1 / conv' (cfrac-tl c) n x) = conv' (cfrac-tl c) n x**by** simp also have cfrac-nth (cfrac-of-real ...) m' = cfrac-nth c m using Suc.prems by (subst Suc.IH) (auto simp: Suc enat-le-iff) finally show ?thesis . qed qed simp-all **lemma** cfrac-lim-irrational: assumes [simp]: cfrac-length  $c = \infty$ shows *cfrac-lim*  $c \notin \mathbf{Q}$ proof assume cfrac-lim  $c \in \mathbb{Q}$ then obtain a :: int and b :: nat where ab: b > 0 cfrac-lim c = a / bby (auto simp: Rats-eq-int-div-nat) define h and k where h = conv-num c and k = conv-denom chave filterlim ( $\lambda m$ . conv-denom c (Suc m)) at-top at-top using conv-denom-at-top filterlim-Suc by (rule filterlim-compose) then obtain m where m: conv-denom c (Suc m)  $\geq b + 1$ **by** (*auto simp: filterlim-at-top eventually-at-top-linorder*) have \*: (a \* k m - b \* h m) / (k m \* b) = a / b - h m / k m

using  $\langle b > 0 \rangle$  by (simp add: field-simps k-def)

have |cfrac-lim c - conv c m| = |(a \* k m - b \* h m) / (k m \* b)|**by** (*subst* \*) (*auto simp: ab h-def k-def conv-num-denom*) **also have** ... = |a \* k m - b \* h m| / (k m \* b)**by** (*simp add: k-def*) finally have eq: |cfrac-lim c - conv c m| = of-int |a \* k m - b \* h m| / of-int $(k \ m \ * \ b)$ . have  $|cfrac-lim c - conv c m| * (k m * b) \neq 0$ using conv-neq-cfrac-lim[of m c]  $\langle b > 0 \rangle$  by (auto simp: k-def) also have |cfrac-lim c - conv c m| \* (k m \* b) = of-int |a \* k m - b \* h m|**using**  $\langle b > 0 \rangle$  **by** (subst eq) (auto simp: k-def) finally have  $|a * k m - b * h m| \ge 1$  by linarith hence real-of-int  $|a * k m - b * h m| \ge 1$  by linarith hence 1 / of-int  $(k \ m \ast b) \leq of$ -int  $|a \ast k \ m - b \ast h \ m|$  / real-of-int  $(k \ m \ast b)$ using  $\langle b > 0 \rangle$  by (intro divide-right-mono) (auto simp: k-def) also have  $\ldots = |c frac - lim c - conv c m|$ by (rule eq [symmetric]) also have  $\ldots \leq 1$  / real-of-int (conv-denom c m \* conv-denom c (Suc m)) by (intro cfrac-lim-minus-conv-upper-bound) auto also have  $\ldots = 1 / (real-of-int (k m) * real-of-int (k (Suc m)))$ by (simp add: k-def) also have  $\ldots < 1 / (real-of-int (k m) * real b)$ using  $m \langle b > 0 \rangle$ by (intro divide-strict-left-mono mult-strict-left-mono) (auto simp: k-def) finally show False by simp qed **lemma** cfrac-infinite-iff: cfrac-length  $c = \infty \leftrightarrow$  cfrac-lim  $c \notin \mathbb{Q}$ using cfrac-lim-irrational[of c] cfrac-lim-rational[of c] by auto **lemma** cfrac-lim-rational-iff: cfrac-lim  $c \in \mathbb{Q} \leftrightarrow c$ frac-length  $c \neq \infty$ using cfrac-lim-irrational of c] cfrac-lim-rational of c] by auto **lemma** cfrac-of-real-infinite-iff [simp]: cfrac-length (cfrac-of-real x) =  $\infty \leftrightarrow x \notin$ 0 **by** (*simp add: cfrac-infinite-iff*) **lemma** cfrac-remainder-rational-iff [simp]: cfrac-remainder  $c \ n \in \mathbb{Q} \longleftrightarrow c$ frac-length  $c < \infty$ proof have cfrac-remainder  $c \ n \in \mathbb{Q} \longleftrightarrow$  cfrac-lim (cfrac-drop  $n \ c$ )  $\in \mathbb{Q}$ **by** (*simp add: cfrac-remainder-def*) also have  $\ldots \leftrightarrow cfrac$ -length  $c \neq \infty$ by (cases cfrac-length c) (auto simp add: cfrac-lim-rational-iff) finally show ?thesis by simp qed **lift-definition** cfrac-cons ::  $int \Rightarrow cfrac \Rightarrow cfrac$  is

### $\lambda a \ bs. \ case \ bs \ of \ (b, \ bs) \Rightarrow if \ b \leq 0 \ then \ (1, \ LNil) \ else \ (a, \ LCons \ (nat \ (b-1)))$

bs) .

```
lemma cfrac-nth-cons:
 assumes cfrac-nth x 0 \ge 1
 shows cfrac-nth (cfrac-cons a x) n = (if n = 0 then a else cfrac-nth x (n - 1))
 using assms
proof (transfer, goal-cases)
 case (1 \ x \ a \ n)
 obtain b bs where [simp]: x = (b, bs)
   by (cases x)
 show ?case using 1
   by (cases llength bs) (auto simp: lnth-LCons eSuc-enat le-imp-diff-is-add split:
nat.splits)
qed
lemma cfrac-length-cons [simp]:
 assumes cfrac-nth x \ 0 > 1
 shows cfrac-length (cfrac-cons a x) = eSuc (cfrac-length x)
 using assms by transfer auto
lemma cfrac-tl-cons [simp]:
 assumes cfrac-nth x 0 \ge 1
 shows cfrac-tl (cfrac-cons a x) = x
 using assms by transfer auto
lemma cfrac-cons-tl:
 assumes \neg cfrac\text{-}is\text{-}int x
 shows cfrac-cons (cfrac-nth x \ \theta) (cfrac-tl x) = x
 using assms unfolding cfrac-is-int-def
```

```
by transfer (auto split: llist.splits)
```

# 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*  $\Rightarrow$  *bool* is  $\lambda(x, xs)$ .  $\neg$ *lfinite*  $xs \lor lnull xs \lor llast xs \neq 0$ .

**lemma** cfrac-canonical [code]: cfrac-canonical (CFrac x xs)  $\longleftrightarrow$  lnull  $xs \lor$  llast  $xs \neq 0 \lor \neg$ lfinite xsby (auto simp add: cfrac-canonical-def)

```
lemma cfrac-canonical-iff:
  cfrac-canonical c \leftrightarrow \rightarrow
    cfrac-length c \in \{0, \infty\} \lor cfrac-nth c (the-enat (cfrac-length c)) \neq 1
proof (transfer, clarify, goal-cases)
 case (1 x xs)
 \mathbf{show}~? case
   by (cases llength xs)
      (auto simp: llast-def enat-0 lfinite-conv-llength-enat split: nat.splits)
qed
lemma llast-cfrac-of-real-aux-nonzero:
 assumes lfinite (cfrac-of-real-aux x) \neglnull (cfrac-of-real-aux x)
 shows llast (cfrac-of-real-aux x) \neq 0
 using assms
proof (induction cfrac-of-real-aux x arbitrary: x rule: lfinite-induct)
 case (LCons x)
  from LCons.prems have x \in \{0 < ... < 1\}
   by (subst (asm) cfrac-of-real-aux.code) (auto split: if-splits)
 show ?case
 proof (cases 1 \mid x \in \mathbb{Z})
   case False
   thus ?thesis using LCons
     by (auto simp: llast-LCons frac-lt-1 cfrac-of-real-aux.code[of x])
 \mathbf{next}
   case True
   then obtain n where n: 1 / x = of-int n
     by (elim Ints-cases)
   have 1 / x > 1 using \langle x \in \neg by auto
   with n have n > 1 by simp
   from n have x = 1 / of-int n
     using \langle n > 1 \rangle \langle x \in - \rangle by (simp add: field-simps)
   with \langle n > 1 \rangle show ?thesis
    using LCons cfrac-of-real-aux.code[of x] by (auto simp: llast-LCons frac-lt-1)
 ged
qed auto
```

```
lemma cfrac-canonical-of-real [intro]: cfrac-canonical (cfrac-of-real x)
by (transfer fixing: x) (use llast-cfrac-of-real-aux-nonzero[of frac x] in force)
```

**primcorec** *cfrac-of-real-alt-aux* :: *real*  $\Rightarrow$  *nat llist* **where** 

 $cfrac-of-real-alt-aux \ x = \\ (if \ x \in \{0 < ... < 1\} \ then \\ if \ 1 \ / \ x \in \mathbb{Z} \ then \\ LCons \ (nat \ \lfloor 1/x \rfloor - 2) \ (LCons \ 0 \ LNil) \\ else \ LCons \ (nat \ \lfloor 1/x \rfloor - 1) \ (cfrac-of-real-alt-aux \ (frac \ (1/x))) \\ else \ LNil) \end{aligned}$ 

**lemma** cfrac-of-real-aux-alt-LNil [simp]:  $x \notin \{0 < ... < 1\} \implies$  cfrac-of-real-alt-aux x = LNil

**by** (*subst cfrac-of-real-alt-aux.code*) *auto* 

**lemma** cfrac-of-real-aux-alt-0 [simp]: cfrac-of-real-alt-aux 0 = LNilby (subst cfrac-of-real-alt-aux.code) auto

**lemma** cfrac-of-real-aux-alt-eq-LNil-iff [simp]: cfrac-of-real-alt-aux  $x = LNil \leftrightarrow x \notin \{0 < ... < 1\}$ by (subst cfrac-of-real-alt-aux.code) auto

**lift-definition** cfrac-of-real-alt :: real  $\Rightarrow$  cfrac is  $\lambda x.$  if  $x \in \mathbb{Z}$  then  $(\lfloor x \rfloor - 1, LCons \ 0 \ LNil)$  else  $(\lfloor x \rfloor, cfrac-of-real-alt-aux (frac x))$ .

```
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)
 using assms
proof (transfer, goal-cases)
 case (1 x)
 show ?case
 proof (cases 1 / frac x \in \mathbb{Z})
   case False
   from 1 have int (nat \lfloor 1 / frac x \rfloor - Suc 0) + 1 = \lfloor 1 / frac x \rfloor
     by (subst of-nat-diff) (auto simp: le-nat-iff frac-le-1)
   with False show ?thesis
     using \langle x \notin \mathbb{Z} \rangle
     by (subst cfrac-of-real-alt-aux.code) (auto split: llist.splits simp: frac-lt-1)
 next
   case True
   then obtain n where 1 / frac x = of-int n
     by (auto simp: Ints-def)
   moreover have 1 / frac x > 1
     using 1 by (auto simp: divide-simps frac-lt-1)
   ultimately have 1 / frac x \ge 2
     by simp
   hence int (nat | 1 / frac x | - 2) + 2 = | 1 / frac x |
     by (subst of-nat-diff) (auto simp: le-nat-iff frac-le-1)
   thus ?thesis
     using \langle x \notin \mathbb{Z} \rangle
     by (subst cfrac-of-real-alt-aux.code) (auto split: llist.splits simp: frac-lt-1)
```

qed **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)) nproof have cfrac-nth (cfrac-of-real-alt x) (Suc n) = cfrac-nth (cfrac-tl (cfrac-of-real-alt x)) nby simp also have cfrac-tl (cfrac-of-real-alt x) = cfrac-of-real-alt (1 / frac x) **by** (*simp add: cfrac-tl-of-real-alt assms*) finally show ?thesis .  $\mathbf{qed}$ **lemma** cfrac-nth-gt0-of-real-int [simp]:  $m > 0 \implies cfrac-nth (cfrac-of-real (of-int n)) m = 1$ by transfer (auto simp: lnth-LCons eSuc-def enat-0-iff split: nat.splits) **lemma** cfrac-nth-0-of-real-alt-int [simp]: cfrac-nth (cfrac-of-real-alt (of-int n))  $\theta = n - 1$ by transfer auto **lemma** cfrac-nth-gt0-of-real-alt-int [simp]:  $m > 0 \implies cfrac-nth (cfrac-of-real-alt (of-int n)) m = 1$ by transfer (auto simp: lnth-LCons eSuc-def split: nat.splits) **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)) using assms **proof** (coinduction arbitrary: x rule: enat-coinduct) **case** (Eq-enat x)show ?case **proof** (cases  $1 / x \in \mathbb{Z}$ ) case False with Eq-enat have frac  $(1 / x) \in \{0 < .. < 1\}$ by (auto intro: frac-lt-1) hence  $\exists x'$ . llength (cfrac-of-real-alt-aux (frac (1 / x))) =  $llength (cfrac-of-real-alt-aux x') \land$ llength (cfrac-of-real-aux (frac (1 / x))) = llength (cfrac-of-real-aux x') $\wedge$  $0 < x' \land x' < 1$ by (intro exI[of - frac (1 / x)]) auto thus ?thesis using False Eq-enat by (auto simp: cfrac-of-real-alt-aux.code[of x] cfrac-of-real-aux.code[of x]) $\mathbf{qed} (use \ Eq \text{-enat} \ \mathbf{in} \land auto \ simp: \ cfrac \text{-}of \text{-}real \text{-}alt \text{-}aux. \ code[of x] \ cfrac \text{-}of \text{-}real \text{-}aux. \ code[of x] \ cfrac \text{-}aux. \ code[of x] \ cfrac \text{-}aux. \ code[of x] \ cfrac \text{-}of \text{-}real \text{-}aux. \ code[of x] \ cfrac \text{-}aux. \ code[of x] \ cdrac \ cdrac \ cdrac \ cdrac \$  $x \rightarrow )$ 

qed

#### $\mathbf{qed}$

```
lemma cfrac-length-of-real-alt:
  cfrac-length (cfrac-of-real-alt x) = eSuc (cfrac-length (cfrac-of-real x))
 by transfer (auto simp: llength-cfrac-of-real-alt-aux frac-lt-1)
lemma cfrac-of-real-alt-aux-eq-regular:
 assumes x \in \{0 < ... < 1\} llength (cfrac-of-real-aux x) = \infty
 shows cfrac-of-real-alt-aux \ x = cfrac-of-real-aux \ x
 using assms
proof (coinduction arbitrary: x)
 case (Eq-llist x)
 hence \exists x'. cfrac-of-real-aux (frac (1 / x)) =
       cfrac-of-real-aux x' \wedge
       cfrac-of-real-alt-aux (frac (1 / x)) =
       cfrac-of-real-alt-aux \ x' \land 0 < x' \land x' < 1 \land llength \ (cfrac-of-real-aux \ x') =
\infty
   by (intro exI[of - frac (1 / x)])
      (auto simp: cfrac-of-real-aux.code[of x] cfrac-of-real-alt-aux.code[of x]
                eSuc-eq-infinity-iff frac-lt-1)
  with Eq-llist show ?case
   by (auto simp: eSuc-eq-infinity-iff)
qed
lemma cfrac-of-real-alt-irrational [simp]:
 assumes x \notin \mathbb{Q}
 shows cfrac-of-real-alt x = cfrac-of-real x
proof -
 from assms have cfrac-length (cfrac-of-real x) = \infty
   using cfrac-length-of-real-irrational by blast
  with assms show ?thesis
   by transfer
      (use Ints-subset-Rats in
     (auto intro!: cfrac-of-real-alt-aux-eq-regular simp: frac-lt-1 llength-cfrac-of-real-alt-aux))
qed
lemma cfrac-nth-of-real-alt-0:
```

```
cfrac-nth (cfrac-of-real-alt x) 0 = (if x \in \mathbb{Z} then \lfloor x \rfloor - 1 else \lfloor x \rfloor)
by transfer auto
```

```
lemma cfrac-nth-of-real-alt:

fixes n :: nat and x :: real

defines c \equiv 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)
  unfolding c-def c'-def l-def
proof (induction n arbitrary: x rule: less-induct)
  case (less n)
  consider x \notin \mathbb{Q} \mid x \in \mathbb{Z} \mid n = 0 \ x \in \mathbb{Q} - \mathbb{Z} \mid n' where n = Suc \ n' \ x \in \mathbb{Q} - \mathbb{Z}
   by (cases n) auto
  thus ?case
  proof cases
   assume x \notin \mathbb{Q}
   thus ?thesis
     by (auto simp: cfrac-length-of-real-irrational)
  \mathbf{next}
   assume x \in \mathbb{Z}
   thus ?thesis
     by (auto simp: Ints-def one-enat-def zero-enat-def)
  next
   assume *: n = 0 x \in \mathbb{Q} - \mathbb{Z}
   have enal 0 \neq cfrac-length (cfrac-of-real x) + 1
     using zero-enat-def by auto
   moreover have enalt 0 \neq cfrac-length (cfrac-of-real x)
     using * cfrac-length-of-real-reduce zero-enat-def by auto
   ultimately show ?thesis using *
     by (auto simp: cfrac-nth-of-real-alt-0)
  \mathbf{next}
   fix n' assume *: n = Suc n' x \in \mathbb{Q} - \mathbb{Z}
   from less.IH [of n' 1 / frac x] and * show ?thesis
    by (auto simp: cfrac-nth-of-real-Suc cfrac-nth-of-real-alt-Suc cfrac-length-of-real-reduce
```

 $eSuc-def\ one-enat-def\ enat-0-iff\ split:\ enat.splits)$ 

# $\begin{array}{c} \mathbf{qed} \\ \mathbf{qed} \end{array}$

**lemma** cfrac-of-real-length-eq-0-iff: cfrac-length (cfrac-of-real x) = 0  $\leftrightarrow x \in \mathbb{Z}$ by transfer (auto simp: frac-lt-1)

**lemma** conv'-cong: **assumes**  $(\bigwedge k. \ k < n \implies cfrac-nth \ c \ k = cfrac-nth \ c' \ k) \ n = n' \ x = y$  **shows** conv' c  $n \ x = conv' \ c' \ n' \ y$  **using** assms(1) **unfolding** assms(2,3) [symmetric] **by** (induction n arbitrary: x) (auto simp: conv'-Suc-right)

lemma conv-cong:

assumes  $(\bigwedge k. \ k \le n \implies cfrac \text{-}nth \ c \ k = cfrac \text{-}nth \ c' \ k) \ n = n'$ shows conv c  $n = conv \ c' \ n'$ using assms(1) unfolding assms(2) [symmetric]

by (induction n arbitrary: c c') (auto simp: conv-Suc)

```
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 (cases cfrac-length (cfrac-of-real x))
 case infinity
 thus ?thesis by auto
\mathbf{next}
  case [simp]: (enat l')
  with assms show ?thesis
   by (intro conv'-cong refl; subst cfrac-nth-of-real-alt) (auto simp: one-enat-def)
qed
lemma cfrac-lim-of-real-alt [simp]: cfrac-lim (cfrac-of-real-alt x) = x
proof (cases cfrac-length (cfrac-of-real x))
 case infinity
 thus ?thesis by auto
next
 case (enat \ l)
 thus ?thesis
 proof (induction l arbitrary: x)
   case \theta
   hence x \in \mathbb{Z}
     using cfrac-of-real-length-eq-0-iff zero-enat-def by auto
   thus ?case
    by (auto simp: Ints-def cfrac-lim-def cfrac-length-of-real-alt eSuc-def conv-Suc)
 \mathbf{next}
   case (Suc l x)
   hence *: \neg cfrac\text{-}is\text{-}int (cfrac\text{-}of\text{-}real\text{-}alt x) x \notin \mathbb{Z}
      by (auto simp: cfrac-is-int-def cfrac-length-of-real-alt Ints-def zero-enat-def
eSuc-def)
   hence cfrac-lim (cfrac-of-real-alt x) =
           of-int \lfloor x \rfloor + 1 / cfrac-lim (cfrac-tl (cfrac-of-real-alt x))
     by (subst cfrac-lim-reduce) (auto simp: cfrac-nth-of-real-alt-0)
   also have cfrac-length (cfrac-of-real (1 / frac x)) = l
    using Suc.prems * by (metis cfrac-length-of-real-reduce eSuc-enat eSuc-inject)
   hence 1 / cfrac-lim (cfrac-tl (cfrac-of-real-alt x)) = frac x
     by (subst cfrac-tl-of-real-alt[OF * (2)], subst Suc) (use Suc.prems * in auto)
   also have real-of-int |x| + frac x = x
     by (simp add: frac-def)
   finally show ?case .
 qed
qed
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 (use assms in transfer, safe, goal-cases)
 case (1 \ a \ xs \ b \ ys)
 from 1(2)[of \ 0] show ?case
```

by auto  $\mathbf{next}$ case  $(2 \ a \ xs \ b \ ys)$ define f where  $f = (\lambda xs \ n. \ if \ enat \ (Suc \ n) \le llength \ xs \ then \ int \ (lnth \ xs \ n) +$  $1 \ else \ 1$ ) have  $\forall n. f xs n = f ys n$ using  $2(2)[of Suc \ n \text{ for } n]$  by (auto simp: f-def cong: if-cong) with 2(1) show xs = ys**proof** (coinduction arbitrary: xs ys) **case** (Eq-llist xs ys) show ?case **proof** (cases lnull  $xs \lor$  lnull ys) case False from False have \*: enat (Suc 0)  $\leq$  llength ys using Suc-ile-eq zero-enat-def by auto have *llength* (*ltl* xs) = *llength* (*ltl* ys) using Eq-llist by (cases xs; cases ys) auto moreover have lhd xs = lhd ysusing False \* Eq-llist(1) spec[OF Eq-llist(2), of 0] by (auto simp: f-def lnth-0-conv-lhd) moreover have f (*ltl xs*) n = f (*ltl ys*) n for nusing Eq-llist(1) \* spec[OF Eq-llist(2), of Suc n] by (cases xs; cases ys) (auto simp: f-def Suc-ile-eq split: if-splits) ultimately show ?thesis using False by auto  $\mathbf{next}$ case True thus ?thesis using Eq-llist(1) by auto qed qed qed **lemma** cfrac-eq-01: assumes cfrac-lim c = 0 cfrac-nth  $c \ 0 \ge 0$ shows  $c = \theta$ proof **have** \*: cfrac-is-int c **proof** (rule ccontr) **assume**  $*: \neg cfrac-is-int c$ from \* have conv c  $\theta$  < cfrac-lim c by (intro conv-less-cfrac-lim) (auto simp: cfrac-is-int-def simp flip: zero-enat-def) hence cfrac-nth c  $\theta < \theta$ using assms by simp  ${\bf thus} \ {\it False}$ using assms by simp ged from \* assms have cfrac-nth c  $\theta = \theta$ **by** (*auto simp: cfrac-lim-def cfrac-is-int-def*)

```
from * and this show c = 0
   unfolding zero-cfrac-def cfrac-is-int-def by transfer auto
qed
lemma cfrac-eq-11:
 assumes cfrac-lim c = 1 cfrac-nth c \ 0 \neq 0
 shows c = 1
proof -
 have *: cfrac-is-int c
 proof (rule ccontr)
   assume *: \neg cfrac-is-int c
   from * have conv c 0 < cfrac-lim c
    by (intro conv-less-cfrac-lim) (auto simp: cfrac-is-int-def simp flip: zero-enat-def)
   hence cfrac-nth c \theta < \theta
     using assms by simp
   have cfrac-lim c = real-of-int (cfrac-nth c 0) + 1 / cfrac-lim (cfrac-tl c)
     using * by (subst cfrac-lim-reduce) auto
   also have real-of-int (cfrac-nth c 0) < 0
     using \langle cfrac-nth \ c \ 0 < 0 \rangle by simp
   also have 1 / cfrac-lim (cfrac-tl c) \leq 1
   proof -
     have 1 \leq cfrac \cdot nth (cfrac \cdot tl c) 0
       by auto
     also have \ldots \leq cfrac-lim (cfrac-tl c)
       by (rule cfrac-lim-ge-first)
     finally show ?thesis by simp
   ged
   finally show False
     using assms by simp
 qed
 from * assms have cfrac-nth c 0 = 1
   by (auto simp: cfrac-lim-def cfrac-is-int-def)
 from * and this show c = 1
   unfolding one-cfrac-def cfrac-is-int-def by transfer auto
\mathbf{qed}
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 \land
              cfrac-nth \ c1 \ 0 = cfrac-nth \ c2 \ 0 \ \land
            (\neg cfrac\text{-}is\text{-}int\ c1 \longrightarrow \neg cfrac\text{-}is\text{-}int\ c2 \longrightarrow R\ (cfrac\text{-}tl\ c1)\ (cfrac\text{-}tl\ c2))
 shows c1 = c2
proof (rule cfrac-eqI)
 show cfrac-nth c1 n = cfrac-nth c2 n for n
   using assms(1)
 proof (induction n arbitrary: c1 c2)
```

```
case \theta
   from IH[OF this] show ?case
     by auto
  \mathbf{next}
   case (Suc n)
   thus ?case
     using IH by (metis cfrac-is-int-iff cfrac-nth-0-of-int cfrac-nth-tl)
 qed
\mathbf{next}
 show cfrac-length c1 = cfrac-length c2
   using assms(1)
 proof (coinduction arbitrary: c1 c2 rule: enat-coinduct)
   case (Eq-enat c1 c2)
   \mathbf{show} \ ?case
   proof (cases cfrac-is-int c1)
     case True
     thus ?thesis
       using IH[OF \ Eqenat(1)] by (auto simp: cfrac-is-int-def)
   \mathbf{next}
     case False
      with IH[OF \ Eqenat(1)] have **: \neg cfrac-is-int \ c1 \ R \ (cfrac-tl \ c1) \ (cfrac-tl
c2)
       by auto
     have *: (cfrac-length \ c1 = 0) = (cfrac-length \ c2 = 0)
       using IH[OF \ Eq\ enat(1)] by (auto simp: cfrac-is-int-def)
     show ?thesis
      by (intro conjI impI disjI1 *, rule exI[of - cfrac-tl c1], rule exI[of - cfrac-tl
c2])
          (use ** in (auto simp: epred-conv-minus))
   qed
 qed
qed
lemma cfrac-nth-0-cases:
 cfrac-nth \ c \ 0 = |cfrac-lim \ c| \ \lor \ cfrac-nth \ c \ 0 = |cfrac-lim \ c| - 1 \land cfrac-tl \ c
= 1
proof (cases cfrac-is-int c)
 case True
 hence cfrac-nth \ c \ 0 = |cfrac-lim \ c|
   by (auto simp: cfrac-lim-def cfrac-is-int-def)
 thus ?thesis by blast
\mathbf{next}
 case False
 note not-int = this
 have bounds: 1 / cfrac-lim (cfrac-tl c) \geq 0 \wedge 1 / cfrac-lim (cfrac-tl c) \leq 1
 proof -
   have 1 \leq cfrac \cdot nth (cfrac \cdot tl c) 0
     by simp
   also have \ldots \leq c frac - lim (c frac - tl c)
```

```
by (rule cfrac-lim-ge-first)
   finally show ?thesis
     using False by (auto simp: cfrac-lim-nonneg)
 qed
 thus ?thesis
 proof (cases cfrac-lim (cfrac-tl c) = 1)
   case False
   have |cfrac-lim c| = cfrac-nth c 0 + |1 / cfrac-lim (cfrac-tl c)|
     using not-int by (subst cfrac-lim-reduce) auto
   also have 1 / cfrac-lim (cfrac-tl c) \geq 0 \wedge 1 / cfrac-lim (cfrac-tl c) < 1
     using bounds False by (auto simp: divide-simps)
   hence |1 / cfrac-lim (cfrac-tl c)| = 0
     by linarith
   finally show ?thesis by simp
 \mathbf{next}
   case True
   have cfrac-nth c 0 = \lfloor cfrac-lim \ c \rfloor - 1
     using not-int True by (subst cfrac-lim-reduce) auto
   moreover have cfrac-tl c = 1
     using True by (intro cfrac-eq-11) auto
   ultimately show ?thesis by blast
 qed
qed
lemma cfrac-length-1 [simp]: cfrac-length 1 = 0
 unfolding one-cfrac-def by simp
lemma cfrac-nth-1 [simp]: cfrac-nth 1 m = 1
 unfolding one-cfrac-def by transfer (auto simp: enat-0-iff)
lemma cfrac-lim-1 [simp]: cfrac-lim 1 = 1
 by (auto simp: cfrac-lim-def)
lemma cfrac-nth-0-not-int:
 assumes cfrac-lim c \notin \mathbb{Z}
 shows cfrac-nth c \ 0 = |c frac-lim \ c|
proof –
 have cfrac-tl c \neq 1
 proof
   assume eq: cfrac-tl \ c = 1
   have \neg c frac \text{-} is \text{-} int c
     using assms by (auto simp: cfrac-lim-def cfrac-is-int-def)
   hence cfrac-lim c = of-int |cfrac-nth c \ 0| + 1
     using eq by (subst cfrac-lim-reduce) auto
   hence cfrac-lim c \in \mathbb{Z}
     bv auto
   with assms show False by auto
```

```
qed
  with cfrac-nth-0-cases[of c] show ?thesis by auto
qed
lemma cfrac-of-real-cfrac-lim-irrational:
 assumes cfrac-lim c \notin \mathbf{Q}
 shows cfrac-of-real (cfrac-lim c) = c
proof (rule cfrac-eqI)
  from assess show cfrac-length (cfrac-of-real (cfrac-lim c)) = cfrac-length c
   using cfrac-lim-rational-iff by auto
\mathbf{next}
 fix n
 show cfrac-nth (cfrac-of-real (cfrac-lim c)) n = cfrac-nth c n
   using assms
  proof (induction n arbitrary: c)
   case (0 c)
   thus ?case
     using Ints-subset-Rats by (subst cfrac-nth-0-not-int) auto
  \mathbf{next}
   case (Suc n c)
   from Suc.prems have [simp]: cfrac-lim c \notin \mathbb{Z}
     using Ints-subset-Rats by blast
   have cfrac-nth (cfrac-of-real (cfrac-lim c)) (Suc n) =
         cfrac-nth (cfrac-tl (cfrac-of-real (cfrac-lim c))) n
     by (simp flip: cfrac-nth-tl)
  also have cfrac-tl (cfrac-of-real (cfrac-lim c)) = cfrac-of-real (1 / frac (cfrac-lim
c))
     using Suc.prems Ints-subset-Rats by (subst cfrac-tl-of-real) auto
   also have 1 / frac (cfrac-lim c) = cfrac-lim (cfrac-tl c)
     using Suc.prems by (subst cfrac-lim-tl) (auto simp: frac-def cfrac-is-int-def
cfrac-nth-0-not-int)
   also have cfrac-nth (cfrac-of-real (cfrac-lim (cfrac-tl c))) n = cfrac-nth c (Suc
n)
     using Suc.prems by (subst Suc.IH) (auto simp: cfrac-lim-rational-iff)
   finally show ?case .
 qed
\mathbf{qed}
lemma cfrac-eqI-first:
 assumes \neg cfrac\text{-}is\text{-}int \ c \ \neg cfrac\text{-}is\text{-}int \ c'
 assumes cfrac-nth \ c \ 0 = cfrac-nth \ c' \ 0 and cfrac-tl \ c = cfrac-tl \ c'
 shows c = c'
 using assms unfolding cfrac-is-int-def
 by transfer (auto split: llist.splits)
```

**lemma** cfrac-is-int-of-real-iff: cfrac-is-int (cfrac-of-real x)  $\longleftrightarrow x \in \mathbb{Z}$ unfolding cfrac-is-int-def by transfer (use frac-lt-1 in auto)

**lemma** *cfrac-not-is-int-of-real-alt*:  $\neg$ *cfrac-is-int* (*cfrac-of-real-alt* x)

**unfolding** cfrac-is-int-def by transfer (auto simp: frac-lt-1)

**lemma** cfrac-tl-of-real-alt-of-int [simp]: cfrac-tl (cfrac-of-real-alt (of-int n)) = 1 **unfolding** one-cfrac-def by transfer auto

```
lemma cfrac-is-intI:
 assumes cfrac-nth c 0 \ge | cfrac-lim c| and cfrac-lim c \in \mathbb{Z}
 shows cfrac-is-int c
proof (rule ccontr)
 assume *: \neg cfrac-is-int c
 from * have conv c 0 < cfrac-lim c
  by (intro conv-less-cfrac-lim) (auto simp: cfrac-is-int-def simp flip: zero-enat-def)
 with assms show False
   by (auto simp: Ints-def)
qed
lemma cfrac-eq-of-intI:
 assumes cfrac-nth c 0 \ge \lfloor cfrac-lim \ c \rfloor and cfrac-lim c \in \mathbb{Z}
 shows c = cfrac - of - int | cfrac - lim c |
proof –
  from assms have int: cfrac-is-int c
   by (intro cfrac-is-intI) auto
 have [simp]: cfrac-lim c = cfrac-nth c \ 0
   using int by (simp add: cfrac-lim-def cfrac-is-int-def)
 from int have c = cfrac - of - int (cfrac - nth c 0)
   unfolding cfrac-is-int-def by transfer auto
 also from assms have cfrac-nth c \ 0 = |cfrac-lim \ c|
   using int by auto
 finally show ?thesis .
qed
lemma cfrac-lim-of-int [simp]: cfrac-lim (cfrac-of-int n) = of-int n
 by (simp add: cfrac-lim-def)
lemma cfrac-of-real-of-int [simp]: cfrac-of-real (of-int n) = cfrac-of-int n
 by transfer auto
lemma cfrac-of-real-of-nat [simp]: cfrac-of-real (of-nat n) = cfrac-of-int (int n)
 by transfer auto
lemma cfrac-int-cases:
 assumes cfrac-lim c = of-int n
 shows c = cfrac - of - int \ n \lor c = cfrac - of - real - alt \ (of - int \ n)
proof -
 from cfrac-nth-\theta-cases[of c] show ?thesis
 proof (rule disj-forward)
   assume eq: cfrac-nth c \ 0 = |cfrac-lim \ c|
   have c = cfrac - of - int | cfrac - lim c |
     using assms eq by (intro cfrac-eq-of-intI) auto
```

```
with assms eq show c = cfrac \text{-} of \text{-} int n
     by simp
  \mathbf{next}
   assume *: cfrac-nth c 0 = |cfrac-lim c| - 1 \wedge cfrac-tl c = 1
   have \neg c frac\text{-}is\text{-}int c
     using * by (auto simp: cfrac-is-int-def cfrac-lim-def)
   hence cfrac-length c = eSuc (cfrac-length (cfrac-tl c))
     by (subst cfrac-length-tl; cases cfrac-length c)
        (auto simp: cfrac-is-int-def eSuc-def enat-0-iff split: enat.splits)
   also have cfrac-tl c = 1
     using * by auto
   finally have cfrac-length c = 1
     by (simp add: eSuc-def one-enat-def)
   show c = cfrac - of - real - alt (of - int n)
     by (rule cfrac-eqI-first)
        (use \langle \neg cfrac-is-int c \rangle * assms in \langle auto simp: cfrac-not-is-int-of-real-alt \rangle)
 qed
qed
lemma cfrac-cases:
  c \in \{cfrac-of-real (cfrac-lim c), cfrac-of-real-alt (cfrac-lim c)\}
proof (cases cfrac-length c)
  case infinity
 hence cfrac-lim c \notin \mathbf{Q}
   by (simp add: cfrac-lim-irrational)
 thus ?thesis
   using cfrac-of-real-cfrac-lim-irrational by simp
next
 case (enat l)
 thus ?thesis
 proof (induction l arbitrary: c)
   case (\theta c)
   hence c = cfrac - of - real (cfrac - nth c 0)
     by transfer (auto simp flip: zero-enat-def)
   with 0 show ?case by (auto simp: cfrac-lim-def)
 \mathbf{next}
   case (Suc l c)
   show ?case
   proof (cases cfrac-lim c \in \mathbb{Z})
     case True
     thus ?thesis
       using cfrac-int-cases[of c] by (force simp: Ints-def)
   \mathbf{next}
     case [simp]: False
     have \neg cfrac\text{-}is\text{-}int c
       using Suc.prems by (auto simp: cfrac-is-int-def enat-0-iff)
     show ?thesis
       using cfrac-nth-0-cases[of c]
     proof (elim disjE conjE)
```

**assume** \*: cfrac- $nth \ c \ 0 = |cfrac$ - $lim \ c| - 1 \ cfrac$ - $tl \ c = 1$ hence *cfrac-lim*  $c \in \mathbb{Z}$ **using**  $\langle \neg cfrac\text{-}is\text{-}int c \rangle$  **by** (subst cfrac-lim-reduce) auto thus ?thesis **by** (*auto simp: cfrac-int-cases*)  $\mathbf{next}$ **assume** eq: cfrac-nth  $c \ 0 = |cfrac-lim \ c|$ have cfrac-tl  $c = cfrac-of-real (cfrac-lim (cfrac-tl c)) \lor$  $cfrac-tl \ c = cfrac-of-real-alt \ (cfrac-lim \ (cfrac-tl \ c))$ using Suc.IH[of cfrac-tl c] Suc.prems by auto hence c = cfrac-of-real (cfrac-lim c)  $\lor$ c = c frac - o f - real - alt (c frac - lim c)**proof** (*rule disj-forward*) **assume** eq':  $cfrac-tl \ c = cfrac-of-real (cfrac-lim (cfrac-tl \ c))$ **show** c = cfrac - of - real (cfrac - lim c)by (rule cfrac-eqI-first)  $(use \langle \neg cfrac-is-int c \rangle eq eq' in$ (auto simp: cfrac-is-int-of-real-iff cfrac-tl-of-real cfrac-lim-tl frac-def)) next **assume** eq':  $cfrac-tl \ c = cfrac-of-real-alt \ (cfrac-lim \ (cfrac-tl \ c))$ have eq'': cfrac-nth (cfrac-of-real-alt (cfrac-lim c))  $\theta = |cfrac-lim c|$ using Suc.prems by (subst cfrac-nth-of-real-alt-0) auto show c = cfrac - of - real - alt (cfrac - lim c)**by** (*rule cfrac-eqI-first*)  $(use \langle \neg cfrac\text{-}is\text{-}int c \rangle eq eq' eq'' in$ *(auto simp: cfrac-not-is-int-of-real-alt cfrac-tl-of-real-alt cfrac-lim-tl* frac-def) ged thus ?thesis by simp qed qed qed  $\mathbf{qed}$ **lemma** cfrac-lim-eq-iff: assumes cfrac-length  $c = \infty \lor c$ frac-length  $c' = \infty$ **shows** cfrac-lim c = cfrac-lim  $c' \leftrightarrow c = c'$ proof **assume** \*: cfrac-lim c = cfrac-lim c'**hence** cfrac-of-real (cfrac-lim c) = cfrac-of-real (cfrac-lim c') by (simp only:) thus c = c'using assms \* by (subst (asm) (1 2) cfrac-of-real-cfrac-lim-irrational) (auto simp: cfrac-infinite-iff) qed auto **lemma** floor-cfrac-remainder:

assumes cfrac-length  $c = \infty$ 

**shows**  $\lfloor cfrac-remainder c n \rfloor = cfrac-nth c n$  **by** (metis add.left-neutral assms cfrac-length-drop cfrac-lim-eq-iff idiff-infinity cfrac-lim-of-real cfrac-nth-drop cfrac-nth-of-real-0 cfrac-remainder-def)

## 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:  $sgn (of-int x :: 'a :: \{linordered-idom\}) = of-int (sgn x)$ **by** (*auto simp: sqn-if*) **lemma** conv-ge-one: cfrac-nth  $c \ 0 > 0 \Longrightarrow$  conv  $c \ n \ge 1$ **by** (rule order.trans[OF - conv-ge-first]) auto context fixes c h k**defines**  $h \equiv conv$ -num c and  $k \equiv conv$ -denom cbegin **lemma** *abs-diff-le-abs-add*: fixes x y :: realassumes  $x \ge \theta \land y \ge \theta \lor x \le \theta \land y \le \theta$ shows  $|x - y| \le |x + y|$ using assms by linarith **lemma** *abs-diff-less-abs-add*: fixes x y :: realassumes  $x > 0 \land y > 0 \lor x < 0 \land y < 0$ shows |x - y| < |x + y|using assms by linarith **lemma** *abs-diff-le-imp-same-sign*: assumes  $|x - y| \leq d d < |y|$ **shows** sgn x = sgn (y::real)using assms by (auto simp: sgn-if) lemma conv-nonpos: assumes cfrac-nth c 0 < 0**shows** conv c  $n \leq 0$ **proof** (cases n) case  $\theta$ thus ?thesis using assms by auto  $\mathbf{next}$ case [simp]: (Suc n') have conv c n = real-of-int (cfrac-nth c 0) + 1 / conv (cfrac-tl c) n'**by** (*simp add: conv-Suc*) also have ...  $\leq -1 + 1 / 1$ 

```
using assms by (intro add-mono divide-left-mono) (auto intro!: conv-pos conv-ge-one)
 finally show ?thesis by simp
qed
lemma cfrac-lim-nonpos:
 assumes cfrac-nth c \theta < \theta
 shows cfrac-lim c \leq 0
proof (cases cfrac-length c)
 case infinity
 show ?thesis using LIMSEQ-cfrac-lim[OF infinity]
   by (rule tendsto-upperbound) (use assms in (auto simp: conv-nonpos))
\mathbf{next}
 case (enat l)
 thus ?thesis by (auto simp: cfrac-lim-def conv-nonpos assms)
qed
lemma conv-num-nonpos:
 assumes cfrac-nth c \theta < \theta
 shows h n \leq \theta
proof (induction n rule: fib.induct)
 case 2
 have cfrac-nth c (Suc 0) * cfrac-nth c 0 \leq 1 * cfrac-nth c 0
   using assms by (intro mult-right-mono-neg) auto
 also have \ldots + 1 \leq 0 using assms by auto
 finally show ?case by (auto simp: h-def)
\mathbf{next}
 case (3 n)
 have cfrac-nth c (Suc (Suc n)) *h (Suc n) \leq 0
   using 3 by (simp add: mult-nonneg-nonpos)
 also have \ldots + h \ n \leq 0
   using 3 by simp
 finally show ?case
   by (auto simp: h-def)
qed (use assms in \langle auto \ simp: \ h-def \rangle)
lemma conv-best-approximation-aux:
 cfrac-lim c \ge 0 \land h \ n \ge 0 \lor cfrac-lim c \le 0 \land h \ n \le 0
proof (cases cfrac-nth c \ 0 \ge 0)
 case True
 from True have 0 \leq conv \ c \ 0
   by simp
 also have \ldots \leq cfrac-lim c
   by (rule conv-le-cfrac-lim) (auto simp: enat-0)
 finally have cfrac-lim c \ge 0.
 moreover from True have h \ n \ge 0
   unfolding h-def by (intro conv-num-nonneg)
 ultimately show ?thesis by (simp add: sqn-if)
next
 case False
```

thus ?thesis using cfrac-lim-nonpos conv-num-nonpos[of n] by (auto simp: h-def) qed

```
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 \land n = 0)
 assumes Suc n \neq cfrac-length c \lor cfrac-canonical c
 defines x \equiv c frac - lim c
 shows |k \ n \ * \ x \ - \ h \ n| < |b \ * \ x \ - \ a|
proof (cases |a| = |h| \wedge b = k n)
 case True
 with assms have [simp]: a = -h n
   by (auto simp: abs-if split: if-splits)
 have k n > 0
   by (auto simp: k-def)
 show ?thesis
 proof (cases x = 0)
   case True
   hence c = cfrac-of-real 0 \lor c = cfrac-of-real-alt 0
     unfolding x-def by (metis cfrac-cases empty-iff insert-iff)
   hence False
   proof
     assume c = cfrac - of - real 0
     thus False
      using assms by (auto simp: enat-0-iff h-def k-def)
   \mathbf{next}
     assume [simp]: c = cfrac-of-real-alt 0
     hence n = 0 \lor n = 1
        using assms by (auto simp: cfrac-length-of-real-alt enat-0-iff k-def h-def
eSuc-def)
     thus False
      using assms True
        by (elim disjE) (auto simp: cfrac-length-of-real-alt enat-0-iff k-def h-def
eSuc-def
                               cfrac-nth-of-real-alt one-enat-def split: if-splits)
   ged
   thus ?thesis ..
 \mathbf{next}
   case False
   have h \ n \neq 0
```

using True assms(6) h-def by auto

hence  $x > 0 \land h n > 0 \lor x < 0 \land h n < 0$ 

```
using \langle x \neq 0 \rangle conv-best-approximation-aux[of n] unfolding x-def by auto
hence |real-of-int (k n) * x - real-of-int (h n)| < |real-of-int (k n) * x + real-of-int (h n)|
```

```
using \langle k | n > 0 \rangle
   by (intro abs-diff-less-abs-add) (auto simp: not-le zero-less-mult-iff mult-less-0-iff)
   thus ?thesis using True by auto
 qed
next
 case False
 note * = this
 show ?thesis
 proof (cases n = cfrac-length c)
   case True
   hence x = conv c n
     by (auto simp: cfrac-lim-def x-def split: enat.splits)
   also have \ldots = h n / k n
    by (auto simp: h-def k-def conv-num-denom)
   finally have x: x = h n / k n.
   hence |k n * x - h n| = 0
    by (simp add: k-def)
   also have b * x \neq a
   proof
    assume b * x = a
     hence of-int (h \ n) * of-int b = of-int (k \ n) * (of-int a :: real)
      using assms True by (auto simp: field-simps k-def x)
     hence of-int (h \ n \ast b) = (of-int \ (k \ n \ast a) :: real)
      by (simp only: of-int-mult)
     hence h n * b = k n * a
      by linarith
     hence h \ n = a \land k \ n = b
      using assms by (subst (asm) coprime-crossproduct')
                   (auto simp: h-def k-def coprime-conv-num-denom)
    thus False using True False by simp
   qed
   hence \theta < |b * x - a|
    by simp
   finally show ?thesis .
 next
   case False
   define s where s = (-1) \widehat{} n * (a * k n - b * h n)
   define r where r = (-1) \hat{n} * (b * h (Suc n) - a * k (Suc n))
   have k n \leq k (Suc n)
    unfolding k-def by (intro conv-denom-leI) auto
   have r * h n + s * h (Suc n) =
          (-1) \widehat{Suc} n * a * (k (Suc n) * h n - k n * h (Suc n))
    by (simp add: s-def r-def algebra-simps h-def k-def)
   also have \ldots = a using assms unfolding h-def k-def
    by (subst conv-num-denom-prod-diff') (auto simp: algebra-simps)
   finally have eq1: r * h n + s * h (Suc n) = a.
```

have r \* k n + s \* k (Suc n) =

 $(-1) \ Suc \ n * b * (k \ (Suc \ n) * h \ n - k \ n * h \ (Suc \ n))$ by  $(simp \ add: s-def \ r-def \ algebra-simps \ h-def \ k-def)$ also have  $\dots = b$  using assms unfolding  $h-def \ k-def$ by  $(subst \ conv-num-denom-prod-diff') \ (auto \ simp: \ algebra-simps)$ finally have  $eq2: \ r * k \ n + s * k \ (Suc \ n) = b$ . have  $k \ n < k \ (Suc \ n)$ 

using  $\langle n > 0 \rangle$  by (auto simp: k-def intro: conv-denom-lessI)

have  $r \neq 0$ proof assume r = 0hence a \* k (Suc n) = b \* h (Suc n) by (simp add: r-def) hence abs (a \* k (Suc n)) = abs (h (Suc n) \* b) by (simp only: mult-ac) hence \*: abs (h (Suc n)) = abs  $a \land k$  (Suc n) = bunfolding abs-mult h-def k-def using coprime-conv-num-denom assms by (subst (asm) coprime-crossproduct-int) auto with  $\langle k \ n < k$  (Suc n) $\rangle$  and  $\langle b \leq k \ n \rangle$  show False by auto qed

have  $s \neq 0$ proof assume  $s = \theta$ hence a \* k n = b \* h n by (simp add: s-def) hence abs (a \* k n) = abs (h n \* b) by (simp only: mult-ac) hence  $b = k n \wedge |a| = |h n|$  unfolding abs-mult h-def k-def using coprime-conv-num-denom assms **by** (subst (asm) coprime-crossproduct-int) auto with \* show False by simp qed have r \* k n + s \* k (Suc n) = b by fact also have  $\ldots \in \{0 < \ldots < k (Suc n)\}$  using assms  $\langle k n < k (Suc n) \rangle$  by auto finally have  $*: r * k n + s * k (Suc n) \in ...$ have opposite-signs1:  $r > 0 \land s < 0 \lor r < 0 \land s > 0$ **proof** (cases  $r \ge 0$ ; cases  $s \ge 0$ ) assume  $r \ge 0$   $s \ge 0$ hence  $0 * (k n) + 1 * (k (Suc n)) \le r * k n + s * k (Suc n)$ using  $\langle s \neq 0 \rangle$  by (intro add-mono mult-mono) (auto simp: k-def) with \* show ?thesis by auto  $\mathbf{next}$ assume  $\neg(r \ge 0) \neg(s \ge 0)$ hence r \* k n + s \* k (Suc n)  $\leq 0$ by (intro add-nonpos-nonpos mult-nonpos-nonneg) (auto simp: k-def) with \* show ?thesis by auto

qed (insert  $\langle r \neq 0 \rangle \langle s \neq 0 \rangle$ , auto)

have  $r \neq 1$ proof assume [simp]: r = 1have b = r \* k n + s \* k (Suc n) using  $\langle r * k n + s * k (Suc n) = b \rangle$ ... also have s \* k (Suc n)  $\leq (-1) * k$  (Suc n) using opposite-signs1 by (intro mult-right-mono) (auto simp: k-def) **also have** r \* k n + (-1) \* k (Suc n) = k n - k (Suc n)by simp also have  $\ldots \leq \theta$ **unfolding** k-def **by** (auto introl: conv-denom-leI) finally show False using  $\langle b > 0 \rangle$  by simp qed have enat  $n \leq c frac$ -length c enat (Suc n)  $\leq c frac$ -length cusing assms False by (cases cfrac-length c; simp)+ hence conv c  $n \ge x \land conv c$  (Suc  $n) \le x \lor conv c$   $n \le x \land conv c$  (Suc  $n) \ge$  $\boldsymbol{x}$ using conv-ge-cfrac-lim[of n c] conv-ge-cfrac-lim[of Suc n c]  $conv-le-cfrac-lim[of \ n \ c] \ conv-le-cfrac-lim[of \ Suc \ n \ c] \ assms$ by (cases even n) auto **hence** opposite-signs 2:  $k \ n \ast x - h \ n \ge 0 \land k \ (Suc \ n) \ast x - h \ (Suc \ n) \le 0 \lor$  $k n * x - h n \leq 0 \land k (Suc n) * x - h (Suc n) \geq 0$ using assms conv-denom-pos[of c n] conv-denom-pos[of c Suc n] **by** (*auto simp: k-def h-def conv-num-denom field-simps*) **from** *opposite-signs1 opposite-signs2* **have** *same-signs:*  $r * (k n * x - h n) \ge 0 \land s * (k (Suc n) * x - h (Suc n)) \ge 0 \lor$  $r * (k n * x - h n) \le 0 \land s * (k (Suc n) * x - h (Suc n)) \le 0$ by (auto intro: mult-nonpos-nonneg mult-nonneg-nonpos mult-nonneg-nonneg mult-nonpos-nonpos) show ?thesis **proof** (cases Suc n = cfrac-length c) case True have x: x = h (Suc n) / k (Suc n) using True[symmetric] by (auto simp: cfrac-lim-def h-def k-def conv-num-denom x-def) have  $r \neq -1$ proof assume [simp]: r = -1have r \* k n + s \* k (Suc n) = bby fact also have b < k (Suc n) using  $\langle b \leq k n \rangle$  and  $\langle k n < k (Suc n) \rangle$  by simp finally have (s - 1) \* k (Suc n) < k n **by** (*simp add: algebra-simps*) also have  $k n \leq 1 * k$  (Suc n) **by** (*simp add: k-def conv-denom-leI*)

finally have s < 2by (subst (asm) mult-less-cancel-right) (auto simp: k-def) moreover from *opposite-signs1* have s > 0 by *auto* ultimately have [simp]: s = 1 by simphave  $b = (cfrac-nth \ c \ (Suc \ n) - 1) * k \ n + k \ (n - 1)$ using  $eq2 \langle n > 0 \rangle$  by (cases n) (auto simp: k-def algebra-simps) also have *cfrac-nth* c (Suc n) > 1 proof have cfrac-canonical c using assms True by auto hence *cfrac-nth* c (Suc n)  $\neq 1$ using True[symmetric] by (auto simp: cfrac-canonical-iff enat-0-iff) moreover have cfrac-nth c (Suc n) > 0 by *auto* ultimately show *cfrac-nth* c (Suc n) > 1 by linarith  $\mathbf{qed}$ hence  $(cfrac-nth \ c \ (Suc \ n) - 1) * k \ n + k \ (n - 1) \ge 1 * k \ n + k \ (n - 1)$ by (intro add-mono mult-right-mono) (auto simp: k-def) finally have b > k nusing conv-denom-pos[of c n - 1] unfolding k-def by linarith with assms show False by simp qed with  $\langle r \neq 1 \rangle \langle r \neq 0 \rangle$  have |r| > 1by *auto* from  $\langle s \neq 0 \rangle$  have  $k \ n \ast x \neq h \ n$ using conv-num-denom-prod-diff[of c n] by (auto simp: x field-simps k-def h-def simp flip: of-int-mult) hence 1 \* |k n \* x - h n| < |r| \* |k n \* x - h n|using  $\langle |r| > 1 \rangle$  by (intro mult-strict-right-mono) auto also have  $\ldots = |r| * |k n * x - h n| + 0$  by simp also have ...  $\leq |r * (k n * x - h n)| + |s * (k (Suc n) * x - h (Suc n))|$ **unfolding** abs-mult of-int-abs using conv-denom-pos[of c Suc n]  $\langle s \neq 0 \rangle$ by (intro add-left-mono mult-nonneq-nonneq) (auto simp: field-simps k-def) **also have** ... = |r \* (k n \* x - h n) + s \* (k (Suc n) \* x - h (Suc n))|using same-signs by auto **also have** ... = |(r \* k n + s \* k (Suc n)) \* x - (r \* h n + s \* h (Suc n))|**by** (simp add: algebra-simps) also have  $\ldots = |b * x - a|$ unfolding eq1 eq2 by simp finally show ?thesis by simp  $\mathbf{next}$ case False from assms have Suc n < cfrac-length cusing False (Suc  $n \leq cfrac$ -length c) by force have  $1 * |k n * x - h n| \le |r| * |k n * x - h n|$ using  $\langle r \neq 0 \rangle$  by (intro mult-right-mono) auto

also have  $\ldots = |r| * |k n * x - h n| + 0$  by simp also have  $x \neq h$  (Suc n) / k (Suc n) using conv-neq-cfrac-lim[of Suc n c]  $\langle Suc n < cfrac-length c \rangle$ by (auto simp: conv-num-denom h-def k-def x-def) **hence** |s \* (k (Suc n) \* x - h (Suc n))| > 0**using**  $\langle s \neq 0 \rangle$  by (auto simp: field-simps k-def) also have  $|r| * |k n * x - h n| + \ldots \leq$ |r \* (k n \* x - h n)| + |s \* (k (Suc n) \* x - h (Suc n))|**unfolding** *abs-mult* of-*int-abs* **by** (*intro add-left-mono mult-nonneq-nonneq*) autoalso have ... = |r \* (k n \* x - h n) + s \* (k (Suc n) \* x - h (Suc n))|using same-signs by auto **also have** ... = |(r \* k n + s \* k (Suc n)) \* x - (r \* h n + s \* h (Suc n))|**by** (*simp add: algebra-simps*) also have  $\ldots = |b * x - a|$ unfolding eq1 eq2 by simp finally show ?thesis by simp qed qed qed **lemma** conv-best-approximation-ex-weak: fixes  $a \ b :: int \text{ and } x :: real$ assumes  $n \leq c frac$ -length cassumes 0 < b and b < k (Suc n) and coprime a b defines  $x \equiv c frac$ -lim c **shows**  $|k \ n * x - h \ n| \le |b * x - a|$ **proof** (cases  $|a| = |h| \wedge b = k n$ ) case True note \* = thisshow ?thesis **proof** (cases sgn a = sgn (h n)) case True with \* have [simp]: a = h nby (auto simp: abs-if split: if-splits) thus ?thesis using \* by auto next case False with True have [simp]: a = -h nby (auto simp: abs-if split: if-splits) have  $|real-of-int (k n) * x - real-of-int (h n)| \leq |real-of-int (k n) * x +$ real-of-int (h n)**unfolding** *x*-*def* **using** *conv*-*best*-*approximation*-*aux*[*of n*] by (intro abs-diff-le-abs-add) (auto simp: k-def not-le zero-less-mult-iff) thus ?thesis using True by auto qed next case False note \* = this

show ?thesis **proof** (cases n = cfrac-length c) case True hence x = conv c n**by** (*auto simp: cfrac-lim-def x-def split: enat.splits*) also have  $\ldots = h n / k n$ **by** (*auto simp: h-def k-def conv-num-denom*) finally show ?thesis by (auto simp: k-def)  $\mathbf{next}$ case False define s where  $s = (-1) \ \widehat{}\ n * (a * k n - b * h n)$ define r where r = (-1)  $\widehat{} n * (b * h (Suc n) - a * k (Suc n))$ have r \* h n + s \* h (Suc n) =(-1)  $\cap$  Suc n \* a \* (k (Suc n) \* h n - k n \* h (Suc n))by (simp add: s-def r-def algebra-simps h-def k-def) also have  $\ldots = a$  using assms unfolding h-def k-def **by** (subst conv-num-denom-prod-diff') (auto simp: algebra-simps) finally have eq1: r \* h n + s \* h (Suc n) = a. have r \* k n + s \* k (Suc n) =  $(-1) \cap Suc \ n * b * (k \ (Suc \ n) * h \ n - k \ n * h \ (Suc \ n))$ by (simp add: s-def r-def algebra-simps h-def k-def) also have  $\ldots = b$  using assms unfolding h-def k-def **by** (*subst conv-num-denom-prod-diff*') (*auto simp: algebra-simps*) finally have eq2: r \* k n + s \* k (Suc n) = b. have  $r \neq 0$ proof assume  $r = \theta$ hence a \* k (Suc n) = b \* h (Suc n) by (simp add: r-def) hence abs (a \* k (Suc n)) = abs (h (Suc n) \* b) by (simp only: mult-ac) hence b = k (Suc n) unfolding abs-mult h-def k-def using coprime-conv-num-denom assms **by** (subst (asm) coprime-crossproduct-int) auto with assms show False by simp qed have  $s \neq \theta$ proof assume  $s = \theta$ hence a \* k n = b \* h n by (simp add: s-def) hence abs (a \* k n) = abs (h n \* b) by (simp only: mult-ac) hence  $b = k n \wedge |a| = |h n|$  unfolding abs-mult h-def k-def using coprime-conv-num-denom assms **by** (subst (asm) coprime-crossproduct-int) auto with \* show False by simp qed

have r \* k n + s \* k (Suc n) = b by fact also have  $\ldots \in \{0 < \ldots < k (Suc \ n)\}$  using assms by auto finally have  $*: r * k n + s * k (Suc n) \in ...$ have opposite-signs1:  $r > 0 \land s < 0 \lor r < 0 \land s > 0$ **proof** (cases  $r \ge 0$ ; cases  $s \ge 0$ ) assume  $r \ge 0$   $s \ge 0$ hence  $0 * (k n) + 1 * (k (Suc n)) \le r * k n + s * k (Suc n)$ using  $\langle s \neq 0 \rangle$  by (intro add-mono mult-mono) (auto simp: k-def) with \* show ?thesis by auto  $\mathbf{next}$ assume  $\neg(r \geq 0) \neg(s \geq 0)$ hence r \* k n + s \* k (Suc n)  $\leq 0$ by (intro add-nonpos-nonpos mult-nonpos-nonneg) (auto simp: k-def) with \* show ?thesis by auto qed (insert  $\langle r \neq 0 \rangle \langle s \neq 0 \rangle$ , auto) have enat  $n \leq c frac$ -length c enat (Suc n)  $\leq c frac$ -length cusing assms False by (cases cfrac-length c; simp)+ hence conv c  $n \ge x \land conv c$  (Suc  $n) \le x \lor conv c$   $n \le x \land conv c$  (Suc  $n) \ge$ using conv-ge-cfrac-lim[of n c] conv-ge-cfrac-lim[of Suc n c]  $conv-le-cfrac-lim[of \ n \ c] \ conv-le-cfrac-lim[of \ Suc \ n \ c] \ assms$ by (cases even n) auto **hence** opposite-signs 2:  $k \ n \ast x - h \ n \ge 0 \land k \ (Suc \ n) \ast x - h \ (Suc \ n) \le 0 \lor$  $k n * x - h n \leq 0 \wedge k (Suc n) * x - h (Suc n) \geq 0$ using assms conv-denom-pos[of c n] conv-denom-pos[of c Suc n] **by** (*auto simp: k-def h-def conv-num-denom field-simps*) **from** *opposite-signs1 opposite-signs2* **have** *same-signs:*  $r * (k n * x - h n) \ge 0 \land s * (k (Suc n) * x - h (Suc n)) \ge 0 \lor$  $r * (k n * x - h n) \le 0 \land s * (k (Suc n) * x - h (Suc n)) \le 0$ by (auto intro: mult-nonpos-nonneg mult-nonneg-nonpos mult-nonneg-nonneg *mult-nonpos-nonpos*) have  $1 * |k n * x - h n| \le |r| * |k n * x - h n|$ using  $\langle r \neq 0 \rangle$  by (intro mult-right-mono) auto also have  $\ldots = |r| * |k n * x - h n| + \theta$  by simp also have ...  $\leq |r * (k n * x - h n)| + |s * (k (Suc n) * x - h (Suc n))|$ **unfolding** abs-mult of-int-abs using conv-denom-pos[of c Suc n]  $\langle s \neq 0 \rangle$ 

by (intro add-left-mono mult-nonneg-nonneg) (auto simp: field-simps k-def) **also have** ... = |r \* (k n \* x - h n) + s \* (k (Suc n) \* x - h (Suc n))|using same-signs by auto

also have ... = |(r \* k n + s \* k (Suc n)) \* x - (r \* h n + s \* h (Suc n))|**by** (*simp add: algebra-simps*) also have  $\ldots = |b * x - a|$ 

unfolding eq1 eq2 by simp

x

finally show ?thesis by simp

#### qed qed

```
lemma cfrac-canonical-reduce:
  cfrac-canonical c \leftrightarrow \rightarrow
    cfrac-is-int \ c \lor \neg cfrac-is-int \ c \land cfrac-tl \ c \neq 1 \land cfrac-canonical \ (cfrac-tl \ c)
 unfolding cfrac-is-int-def one-cfrac-def
 by transfer (auto simp: cfrac-canonical-def llast-LCons split: if-splits split: llist.splits)
lemma cfrac-nth-0-conv-floor:
 assumes cfrac-canonical c \lor cfrac-length c \neq 1
 shows cfrac-nth \ c \ 0 = |cfrac-lim \ c|
proof (cases cfrac-is-int c)
 case True
 thus ?thesis
   by (auto simp: cfrac-lim-def cfrac-is-int-def)
next
 case False
 show ?thesis
 proof (cases cfrac-length c = 1)
   case True
   hence cfrac-canonical c using assms by auto
   hence cfrac-tl \ c \neq 1 using False
     by (subst (asm) cfrac-canonical-reduce) auto
   thus ?thesis
     using cfrac-nth-0-cases[of c] by auto
  \mathbf{next}
   case False
   hence cfrac-length c > 1
     using \langle \neg cfrac\text{-}is\text{-}int c \rangle
    by (cases cfrac-length c) (auto simp: cfrac-is-int-def one-enat-def zero-enat-def)
   have cfrac-tl c \neq 1
   proof
     assume cfrac-tl \ c = 1
     have \theta < cfrac-length c - 1
     proof (cases cfrac-length c)
       case [simp]: (enat l)
       have cfrac-length c - 1 = enat (l - 1)
         by auto
       also have \ldots > enat \ \theta
         using \langle cfrac-length \ c > 1 \rangle by (simp \ add: \ one-enat-def)
       finally show ?thesis by (simp add: zero-enat-def)
     qed auto
     also have \ldots = cfrac-length (cfrac-tl c)
       by simp
     also have cfrac-tl \ c = 1
       bv fact
     finally show False by simp
   qed
```
```
thus ?thesis using cfrac-nth-0-cases[of c] by auto
 qed
qed
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| \le |b * cfrac-lim c - a|
 using conv-best-approximation-ex-weak[OF assms(1), of b a] assms by auto
lemma abs-mult-nonneg-left:
 assumes x \ge (0 :: 'a :: \{ordered-ab-group-add-abs, idom-abs-sgn\})
 shows x * |y| = |x * y|
proof -
 from assms have x = |x| by simp
 also have \dots * |y| = |x * y| by (simp add: abs-mult)
 finally show ?thesis .
qed
```

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 c frac-length c
 assumes 0 < b and b < k n and coprime a b
 defines x \equiv cfrac-lim c
 shows |x - conv \ c \ n| \le |x - a \ / \ b|
proof -
 have b < k n by fact
 also have k n \leq k (Suc n)
   unfolding k-def by (intro conv-denom-leI) auto
 finally have *: b < k (Suc n) by simp
 have |x - conv c n| = |k n * x - h n| / k n
   using conv-denom-pos[of c n] assms(1)
   by (auto simp: conv-num-denom field-simps k-def h-def)
 also have \ldots \leq |b * x - a| / k n unfolding x-def using assms *
   by (intro divide-right-mono conv-best-approximation-ex-weak) auto
 also from assms have \ldots \leq |b * x - a| / b
   by (intro divide-left-mono) auto
 also have \dots = |x - a / b| using assms by (simp add: field-simps)
 finally show ?thesis .
qed
lemma conv-denom-partition:
```

```
assumes y > 0
shows \exists !n. y \in \{k \ n.. < k \ (Suc \ n)\}
proof (rule ex-ex11)
from conv-denom-at-top[of c] assms have *: \exists n. k \ n \ge y + 1
```

**by** (*auto simp: k-def filterlim-at-top eventually-at-top-linorder*) define *n* where  $n = (LEAST n. k n \ge y + 1)$ from LeastI-ex[OF \*] have n: k n > y by  $(simp \ add: \ Suc-le-eq \ n-def)$ from n and assms have n > 0 by (intro Nat.gr0I) (auto simp: k-def) have  $k (n - 1) \leq y$ **proof** (rule ccontr) assume  $\neg k (n-1) \leq y$ hence  $k (n - 1) \ge y + 1$  by *auto* hence  $n - 1 \ge n$  unfolding *n*-def by (rule Least-le) with  $\langle n > 0 \rangle$  show False by simp qed with n and  $\langle n > 0 \rangle$  have  $y \in \{k (n-1) .. < k (Suc (n-1))\}$  by auto thus  $\exists n. y \in \{k \ n.. < k \ (Suc \ n)\}$  by blast next fix m nassume  $y \in \{k \ m.. < k \ (Suc \ m)\}\ y \in \{k \ n.. < k \ (Suc \ n)\}\$ thus m = n**proof** (*induction* m n rule: *linorder-wlog*) case (le m n) show m = n**proof** (rule ccontr) assume  $m \neq n$ with le have k (Suc m)  $\leq k$  n unfolding k-def by (intro conv-denom-leI assms) auto with le show False by auto qed qed auto qed

```
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 c frac - lim c
 assumes b > 0 and coprime a b and |x - a / b| < 1 / (2 * b^2)
 shows \exists n. enat n \leq cfrac-length c \land (a, b) = (h n, k n)
proof (cases a = 0)
 case True
 with assms have [simp]: a = 0 b = 1
   by auto
 show ?thesis
 proof (cases x \ 0 :: real rule: linorder-cases)
   case greater
   hence \theta < x x < 1/2
     using assms by auto
   hence x \notin \mathbb{Z}
     by (auto simp: Ints-def)
```

hence *cfrac-nth*  $c \ 0 = |x|$ using assms by (subst cfrac-nth-0-not-int) (auto simp: x-def) also from  $\langle x > 0 \rangle \langle x < 1/2 \rangle$  have ... = 0 by *linarith* finally have  $(a, b) = (h \ \theta, k \ \theta)$ **by** (*auto simp*: *h*-*def k*-*def*) thus ?thesis by (intro exI[of - 0]) (auto simp flip: zero-enat-def) next case less hence  $x < 0 \ x > -1/2$ using assms by auto hence  $x \notin \mathbb{Z}$ by (auto simp: Ints-def) hence not-int:  $\neg cfrac\text{-}is\text{-}int \ c$ **by** (*auto simp: cfrac-is-int-def x-def cfrac-lim-def*) have cfrac-nth c  $\theta = |x|$ using  $\langle x \notin \mathbb{Z} \rangle$  assms by (subst cfrac-nth-0-not-int) (auto simp: x-def) also from  $\langle x < 0 \rangle \langle x > -1/2 \rangle$  have  $\ldots = -1$ by *linarith* finally have [simp]: cfrac-nth c 0 = -1. have  $cfrac-nth \ c \ (Suc \ 0) = cfrac-nth \ (cfrac-tl \ c) \ 0$ by simp have cfrac-lim (cfrac-tl c) = 1 / (x + 1)using not-int by (subst cfrac-lim-tl) (auto simp: x-def) also from  $\langle x < 0 \rangle \langle x > -1/2 \rangle$  have  $\ldots \in \{1 < .. < 2\}$ **by** (*auto simp: divide-simps*) finally have  $*: cfrac-lim (cfrac-tl c) \in \{1 < .. < 2\}$ . have cfrac-nth (cfrac-tl c) 0 = |cfrac-lim (cfrac-tl c)|**using** \* **by** (*subst cfrac-nth-0-not-int*) (*auto simp: Ints-def*) also have  $\ldots = 1$ using \* by (simp, linarith?) finally have  $(a, b) = (h \ 1, k \ 1)$ by (auto simp: h-def k-def) moreover have cfrac-length  $c \geq 1$ using not-int **by** (cases cfrac-length c) (auto simp: cfrac-is-int-def one-enat-def zero-enat-def) ultimately show ?thesis by (intro exI[of - 1]) (auto simp: one-enat-def)  $\mathbf{next}$ case equal show ?thesis using cfrac-nth-0-cases[of c]proof assume cfrac- $nth \ c \ 0 = |cfrac$ - $lim \ c|$ with equal have  $(a, b) = (h \ 0, k \ 0)$ **by** (*simp add: x-def h-def k-def*) thus ?thesis by (intro exI[of - 0]) (auto simp flip: zero-enat-def) next **assume** \*: cfrac-nth c  $0 = |cfrac-lim c| - 1 \land cfrac-tl c = 1$ have [simp]: cfrac-nth c 0 = -1

```
using * equal by (auto simp: x-def)
     from * have \neg cfrac\text{-}is\text{-}int c
      by (auto simp: cfrac-is-int-def cfrac-lim-def floor-minus)
     have cfrac-nth \ c \ 1 = cfrac-nth \ (cfrac-tl \ c) \ 0
       by auto
     also have cfrac-tl c = 1
       using * by auto
     finally have cfrac-nth c 1 = 1
       by simp
     hence (a, b) = (h \ 1, k \ 1)
       by (auto simp: h-def k-def)
     moreover from \langle \neg cfrac\text{-}is\text{-}int c \rangle have cfrac\text{-}length c \geq 1
    by (cases cfrac-length c) (auto simp: one-enat-def zero-enat-def cfrac-is-int-def)
     ultimately show ?thesis
       by (intro exI[of - 1]) (auto simp: one-enat-def)
   qed
 qed
next
 case False
 hence a-nz: a \neq 0 by auto
 have x \neq 0
 proof
   assume [simp]: x = 0
   hence |a| / b < 1 / (2 * b \hat{2})
     using assms by simp
   hence |a| < 1 / (2 * b)
     using assms by (simp add: field-simps power2-eq-square)
   also have \ldots \leq 1 / 2
     using assms by (intro divide-left-mono) auto
   finally have a = 0 by auto
   with \langle a \neq 0 \rangle show False by simp
  \mathbf{qed}
 show ?thesis
 proof (rule ccontr)
   assume no-convergent: \nexists n. enat n \leq cfrac-length c \land (a, b) = (h n, k n)
   from assms have \exists !r. b \in \{k \ r.. < k \ (Suc \ r)\}
     by (intro conv-denom-partition) auto
   then obtain r where r: b \in \{k \ r.. < k \ (Suc \ r)\} by auto
   have k r > \theta
     using conv-denom-pos[of c r] assms by (auto simp: k-def)
   show False
   proof (cases enat r \leq cfrac-length c)
     case False
     then obtain l where l: cfrac-length c = enat l
       by (cases cfrac-length c) auto
     have k \ l \leq k \ r
```

using False l unfolding k-def by (intro conv-denom-leI) auto also have  $\ldots \leq b$ using r by simpfinally have  $b \ge k l$ . have  $x = conv \ c \ l$ **by** (auto simp: x-def cfrac-lim-def l) hence x-eq: x = h l / k lby (auto simp: conv-num-denom h-def k-def) have  $k \ l > 0$ by (simp add: k-def) have b \* k l \* |h l / k l - a / b| < k l / (2\*b)using assms x-eq  $\langle k | > 0 \rangle$  by (auto simp: field-simps power2-eq-square) **also have** b \* k l \* |h l / k l - a / b| = |b \* k l \* (h l / k l - a / b)|using  $\langle b > 0 \rangle \langle k | b > 0 \rangle$  by (subst abs-mult) auto also have  $\ldots = of int |b * h l - a * k l|$ using  $\langle b > 0 \rangle \langle k | b > 0 \rangle$  by (simp add: algebra-simps) **also have** k l / (2 \* b) < 1using  $\langle b \geq k \ l \rangle \ \langle b > 0 \rangle$  by auto finally have a \* k l = b \* h lby linarith moreover have coprime  $(h \ l) \ (k \ l)$ **unfolding** *h*-def *k*-def **by** (simp add: coprime-conv-num-denom) ultimately have (a, b) = (h l, k l)using  $\langle coprime \ a \ b \rangle$  using  $a - nz \ \langle b > 0 \rangle \ \langle k \ l > 0 \rangle$ by (subst (asm) coprime-crossproduct') (auto simp: coprime-commute) with no-convergent and l show False by *auto*  $\mathbf{next}$ case True have k r \* |x - h r / k r| = |k r \* x - h r|using  $\langle k | r > 0 \rangle$  by (simp add: field-simps) **also have** |k r \* x - h r| < |b \* x - a|

using assms r True unfolding x-def by (intro conv-best-approximation-ex-weak) auto

also have ... = b \* |x - a / b|using  $\langle b > 0 \rangle$  by  $(simp \ add: \ field-simps)$ also have ...  $\langle b * (1 / (2 * b^2))$ using  $\langle b > 0 \rangle$  by  $(intro \ mult-strict-left-mono \ assms)$  auto finally have less:  $|x - conv \ c \ r| < 1 / (2 * b * k \ r)$ using  $\langle k \ r > 0 \rangle$  and  $\langle b > 0 \rangle$  and assmsby  $(simp \ add: \ field-simps \ power2-eq-square \ conv-num-denom \ h-def \ k-def)$ have  $|x - a / b| < 1 / (2 * b^2)$  by fact also have ... = 1 / (2 \* b) \* (1 / b)

**by** (*simp add: power2-eq-square*)

also have ...  $\leq 1 / (2 * b) * (|a| / b)$ using a-nz assms by (intro mult-left-mono divide-right-mono) auto also have ... < 1 / 1 \* (|a| / b)using *a*-nz assms **by** (*intro* mult-strict-right-mono divide-left-mono divide-strict-left-mono) autoalso have  $\ldots = |a / b|$  using assms by simp finally have sgn x = sgn (a / b)by (auto simp: sqn-if split: if-splits) hence sgn x = sgn a using assms by (auto simp: sgn-of-int) hence  $a \ge 0 \land x \ge 0 \lor a \le 0 \land x \le 0$ by (auto simp: sgn-if split: if-splits) moreover have  $h \ r \ge 0 \land x \ge 0 \lor h \ r \le 0 \land x \le 0$ using conv-best-approximation-aux[of r] by (auto simp: h-def x-def) ultimately have signs:  $h \ r \ge 0 \land a \ge 0 \lor h \ r \le 0 \land a \le 0$ using  $\langle x \neq \theta \rangle$  by *auto* with no-convergent assms assms True have  $|h r| \neq |a| \lor b \neq k r$ **by** (*auto simp*: *h*-*def k*-*def*) hence  $|h r| * |b| \neq |a| * |k r|$  unfolding h-def k-def using assms coprime-conv-num-denom [of c r] **by** (subst coprime-crossproduct-int) auto hence  $|h r| * b \neq |a| * k r$  using assms by (simp add: k-def) hence  $k r * a - h r * b \neq 0$ using signs by (auto simp: algebra-simps) hence  $|k r * a - h r * b| \ge 1$  by presburger **hence** real-of-int 1 /  $(k \ r \ast b) \le |k \ r \ast a - h \ r \ast b|$  /  $(k \ r \ast b)$ using assms by (intro divide-right-mono, subst of-int-le-iff) (auto simp: k-def) also have  $\ldots = |(real \text{-} of \text{-} int (k r) * a - h r * b) / (k r * b)|$ using assms by (simp add: k-def) also have (real-of-int (k r) \* a - h r \* b) / (k r \* b) = a / b - conv c rusing assms  $\langle k r > 0 \rangle$  by (simp add: h-def k-def conv-num-denom field-simps) **also have** |a / b - conv c r| = |(x - conv c r) - (x - a / b)|**by** (*simp add: algebra-simps*) also have  $\ldots \leq |x - conv \ c \ r| + |x - a \ / \ b|$ **by** (rule abs-triangle-ineq4) also have ... < 1 /  $(2 * b * k r) + 1 / (2 * b^2)$ **by** (*intro add-strict-mono assms less*) finally have k r > busing  $\langle b > 0 \rangle$  and  $\langle k r > 0 \rangle$  by (simp add: power2-eq-square field-simps) with r show False by auto qed qed qed 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-gen c (a, b, n) N = (if n > N then b else conv-gen c (b, b \* c n + a, Suc n) N) by *auto* termination by (relation measure  $(\lambda(-, (-, -, n), N))$ . Suc N - n)) auto **lemmas**  $[simp \ del] = conv-gen.simps$ **lemma** conv-gen-aux-simps [simp]:  $n > N \Longrightarrow conv-qen \ c \ (a, b, n) \ N = b$  $n \leq N \Longrightarrow conv-gen \ c \ (a, \ b, \ n) \ N = conv-gen \ c \ (b, \ b * c \ n + a, \ Suc \ n) \ N$ **by** (subst conv-gen.simps, simp)+ **lemma** conv-num-eq-conv-gen-aux: Suc  $n \leq N \implies conv-num \ c \ n = b * cfrac-nth \ c \ n + a \implies$ conv-num c (Suc n) = conv-num c n \* cfrac-nth c (Suc n) +  $b \Longrightarrow$ conv-num c N = conv-gen (cfrac-nth c) (a, b, n) N **proof** (induction cfrac-nth c (a, b, n) N arbitrary: c a b n rule: conv-gen.induct) case  $(1 \ a \ b \ n \ N \ c)$ show ?case **proof** (cases Suc (Suc n)  $\leq N$ ) case True thus ?thesis by (subst 1) (insert 1.prems, auto)  $\mathbf{next}$ case False thus ?thesis using 1 **by** (*auto simp: not-le less-Suc-eq*) qed qed **lemma** conv-denom-eq-conv-gen-aux: Suc  $n \leq N \Longrightarrow$  conv-denom  $c \ n = b * cfrac-nth \ c \ n + a \Longrightarrow$  $conv-denom \ c \ (Suc \ n) = conv-denom \ c \ n * cfrac-nth \ c \ (Suc \ n) + b \Longrightarrow$ conv-denom c N = conv-gen (cfrac-nth c) (a, b, n) N**proof** (induction cfrac-nth c (a, b, n) N arbitrary: c a b n rule: conv-gen.induct) case  $(1 \ a \ b \ n \ N \ c)$ show ?case **proof** (cases Suc (Suc n)  $\leq N$ ) case True thus ?thesis by (subst 1) (insert 1.prems, auto)  $\mathbf{next}$ case False thus ?thesis using 1 **by** (*auto simp: not-le less-Suc-eq*) qed qed

**lemma** conv-num-code [code]: conv-num  $c \ n = conv-gen \ (cfrac-nth \ c) \ (0, \ 1, \ 0) \ n$ using conv-num-eq-conv-gen-aux[of 0 n c 1 0] by (cases n) simp-all

**lemma** conv-denom-code [code]: conv-denom  $c \ n = conv-gen$  (cfrac-nth c) (1, 0, 0) n

using conv-denom-eq-conv-gen-aux[of  $0 \ n \ c \ 0 \ 1$ ] by (cases n) simp-all

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

### lemma

assumes is-cfrac c shows conv-num-fun-eq: conv-num-fun c n = conv-num (cfrac c) nand conv-denom-fun-eq: conv-denom-fun c n = conv-denom (cfrac c) nproof – from assms have cfrac-nth (cfrac c) = c by (intro ext) simp-all thus conv-num-fun c n = conv-num (cfrac c) n and conv-denom-fun c n = conv-denom (cfrac c) n

 $\mathbf{by} \; (simp-all \; add: \; conv-num-fun-def \; conv-num-code \; conv-denom-fun-def \; conv-denom-code) \\ \mathbf{qed} \;$ 

# 1.6 Computing the continued fraction expansion of a rational number

function cfrac-list-of-rat ::  $int \times int \Rightarrow int$  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)))by auto termination by (relation measure ( $\lambda(a,b)$ . nat (abs b))) (auto simp: abs-mod-less)

**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  $c + 1 \land (\forall i < length xs. xs ! i = cfrac$ -nth c i))proof (induction (a, b) arbitrary: a b rule: cfrac-list-of-rat.induct) case  $(1 \ a \ b)$ show ?case proof (cases b = 0) case True thus ?thesis by (subst cfrac-list-of-rat.simps) (auto simp: one-enat-def) next case False define c where c = cfrac-of-real (a / b)

```
define c' where c' = cfrac \text{-} of \text{-} real (b / (a \mod b))
   define xs' where xs' = (if a mod b = 0 then [] else cfrac-list-of-rat (b, a mod
b))
   define xs where xs = a \ div \ b \ \# \ xs'
   have [simp]: cfrac-nth c 0 = a \ div \ b
     by (auto simp: c-def floor-divide-of-int-eq)
   obtain l where l: cfrac-length c = enat l
     by (cases cfrac-length c) (auto simp: c-def)
   have length xs = l + 1 \land (\forall i < length xs. xs ! i = cfrac-nth c i)
   proof (cases b \ dvd \ a)
     case True
     thus ?thesis using l
       by (auto simp: Let-def xs-def xs'-def c-def of-int-divide-in-Ints one-enat-def
enat-0-iff)
   next
     case False
     have l \neq 0
       using l False cfrac-of-real-length-eq-0-iff [of a / b] \langle b \neq 0 \rangle
     by (auto simp: c-def zero-enat-def real-of-int-divide-in-Ints-iff introl: Nat.gr0I)
     have c': c' = cfrac-tl c
       using False \langle b \neq 0 \rangle unfolding c'-def c-def
     by (subst cfrac-tl-of-real) (auto simp: real-of-int-divide-in-Ints-iff frac-fraction)
     from 1 have enat (length xs') = cfrac-length c' + 1
            and xs': \forall i < length xs'. xs' ! i = cfrac-nth c' i
       using \langle b \neq 0 \rangle \langle \neg b \, dvd \, a \rangle by (auto simp: Let-def xs'-def c'-def)
     have enat (length xs') = cfrac-length c' + 1
       by fact
     also have \ldots = enat \ l - 1 + 1
       using c' l by simp
     also have ... = enat (l - 1 + 1)
       by (metis enat-diff-one one-enat-def plus-enat-simps(1))
     also have l - 1 + 1 = l
       using \langle l \neq 0 \rangle by simp
     finally have [simp]: length xs' = l
       by simp
     from xs' show ?thesis
       by (auto simp: xs-def nth-Cons c' split: nat.splits)
   qed
   thus ?thesis using l False
    by (subst cfrac-list-of-rat.simps) (simp-all add: xs-def xs'-def c-def one-enat-def)
 qed
qed
```

**lemma** conv-num-cong:

```
assumes (\bigwedge k. \ k \leq n \implies cfrac \text{-} nth \ c \ k = cfrac \text{-} nth \ c' \ k) \ n = n'
 shows conv-num c n = conv-num c' n
proof -
 have conv-num c n = conv-num c' n
   using assms(1)
   by (induction n arbitrary: rule: conv-num.induct) simp-all
 thus ?thesis using assms(2)
   by simp
qed
lemma conv-denom-cong:
 assumes (\bigwedge k. \ k \leq n \implies cfrac \cdot nth \ c \ k = cfrac \cdot nth \ c' \ k) \ n = n'
 shows conv-denom c n = conv-denom c' n'
proof -
 have conv-denom c \ n = conv-denom c' \ n
   using assms(1)
   by (induction n arbitrary: rule: conv-denom.induct) simp-all
 thus ?thesis using assms(2)
   by simp
qed
lemma cfrac-lim-diff-le:
 assumes \forall k \leq Suc \ n. \ cfrac-nth \ c1 \ k = cfrac-nth \ c2 \ k
 assumes n \leq cfrac-length c1 n \leq cfrac-length c2
 shows |cfrac-lim c1 - cfrac-lim c2| \le 2 / (conv-denom c1 n * conv-denom c1
(Suc n)
proof –
 define d where d = (\lambda k. \ conv-denom \ c1 \ k)
 have |cfrac-lim c1 - cfrac-lim c2| \le |cfrac-lim c1 - conv c1 n| + |cfrac-lim c2|
- conv c1 n
   by linarith
 also have |cfrac-lim c1 - conv c1 n| \le 1 / (d n * d (Suc n))
   unfolding d-def using assms
   by (intro cfrac-lim-minus-conv-upper-bound) auto
 also have conv c1 n = conv c2 n
   using assms by (intro conv-conq) auto
 also have |cfrac-lim c2 - conv c2 n| \le 1 / (conv-denom c2 n * conv-denom c2
(Suc \ n)
    using assms unfolding d-def by (intro cfrac-lim-minus-conv-upper-bound)
auto
 also have conv-denom c2 n = d n
   unfolding d-def using assms by (intro conv-denom-cong) auto
 also have conv-denom c2 (Suc n) = d (Suc n)
   unfolding d-def using assms by (intro conv-denom-cong) auto
 also have 1 / (d n * d (Suc n)) + 1 / (d n * d (Suc n)) = 2 / (d n * d (Suc n))
n))
   by simp
 finally show ?thesis
   by (simp add: d-def)
```

# qed

**lemma** of-int-leI:  $n \leq m \implies (of\text{-int } n :: 'a :: linordered\text{-idom}) \leq of\text{-int } m$ by simp lemma cfrac-lim-diff-le': **assumes**  $\forall k \leq Suc \ n. \ cfrac-nth \ c1 \ k = cfrac-nth \ c2 \ k$ assumes  $n \leq cfrac$ -length c1  $n \leq cfrac$ -length c2 shows  $|cfrac-lim c1 - cfrac-lim c2| \le 2 / (fib (n+1) * fib (n+2))$ proof have  $|cfrac-lim c1 - cfrac-lim c2| \le 2$  / (conv-denom c1 n \* conv-denom c1  $(Suc \ n))$ by (rule cfrac-lim-diff-le) (use assms in auto) also have  $\ldots \leq 2$  / (int (fib (Suc n)) \* int (fib (Suc (Suc n)))) unfolding of-nat-mult of-int-mult by (intro divide-left-mono mult-mono mult-pos-pos of-int-leI conv-denom-lower-bound) (auto intro!: fib-neq-0-nat simp del: fib.simps) **also have** ... = 2 / (fib (n+1) \* fib (n+2))by simp finally show ?thesis . qed

end

# 2 Quadratic Irrationals

```
theory Quadratic-Irrationals
imports
 Continued-Fractions
 HOL-Computational-Algebra. Computational-Algebra
 HOL-Library.Discrete-Functions
 Coinductive. Coinductive-Stream
begin
lemma snth-cycle:
 assumes xs \neq []
 shows snth (cycle xs) n = xs ! (n \mod length xs)
proof (induction n rule: less-induct)
 case (less n)
 have snth (shift xs (cycle xs)) n = xs ! (n \mod length xs)
 proof (cases n < length xs)
   case True
   thus ?thesis
    by (subst shift-snth-less) auto
 \mathbf{next}
   case False
```

```
have 0 < length xs
using assms by simp
also have \ldots \leq n
```

```
using False by simp
finally have n > 0.
from False have snth (shift xs (cycle xs)) n = snth (cycle xs) (n - length xs)
    by (subst shift-snth-ge) auto
    also have ... = xs ! ((n - length xs) mod length xs)
    using assms <n > 0 > by (intro less) auto
    also have (n - length xs) mod length xs = n mod length xs
    using False by (simp add: mod-if)
    finally show ?thesis .
    qed
    also have shift xs (cycle xs) = cycle xs
    by (rule cycle-decomp [symmetric]) fact
    finally show ?case .
    qed
```

# 2.1 Basic results on rationality of square roots

**lemma** inverse-in-Rats-iff [simp]: inverse  $(x :: real) \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$ by (auto simp: inverse-eq-divide divide-in-Rats-iff1)

```
lemma nonneg-sqrt-nat-or-irrat:
 assumes x \cap 2 = real \ a and x \ge 0
 shows x \in \mathbb{N} \lor x \notin \mathbb{Q}
proof safe
 assume x \notin \mathbb{N} and x \in \mathbb{Q}
 from Rats-abs-nat-div-natE[OF this(2)]
   obtain p q :: nat where q-nz [simp]: q \neq 0 and abs x = p / q and coprime:
coprime p q.
  with \langle x \geq 0 \rangle have x: x = p / q
     by simp
  with assms have real (q \ 2) * real \ a = real \ (p \ 2)
   by (simp add: field-simps)
 also have real (q \ 2) * real \ a = real \ (q \ 2 * a)
   by simp
 finally have p \uparrow 2 = q \uparrow 2 * a
   by (subst (asm) of-nat-eq-iff) auto
 hence q \uparrow 2 dvd p \uparrow 2
   by simp
 hence q \, dvd \, p
   by simp
  with coprime have q = 1
   by auto
  with x and \langle x \notin \mathbb{N} \rangle show False
   by simp
qed
```

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  $x \in \mathbb{Z} \lor x \notin \mathbb{Q}$ **proof** (cases  $x \ge 0$ ) case True with nonneg-sqrt-nat-or-irrat[OF assms this] **show** ?thesis **by** (auto simp: Nats-altdef2)  $\mathbf{next}$ case False from assms have  $(-x) \hat{\ } 2 = real a$ by simp moreover from *False* have  $-x \ge 0$ by simp ultimately have  $-x \in \mathbb{N} \lor -x \notin \mathbb{Q}$ **by** (*rule nonneg-sqrt-nat-or-irrat*) thus ?thesis by (auto simp: Nats-altdef2 minus-in-Ints-iff)  $\mathbf{qed}$ 

**corollary** sqrt-nat-or-irrat': sqrt (real a)  $\in \mathbb{N} \lor$  sqrt (real a)  $\notin \mathbb{Q}$ using nonneg-sqrt-nat-or-irrat[of sqrt a a] by auto

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:  $sqrt (real n) \in \mathbb{N} \iff is$ -square n proof assume  $sqrt (real n) \in \mathbb{N}$ then obtain k where sqrt (real n) = real k by (auto elim!: Nats-cases) hence  $sqrt (real n) \land 2 = real (k \land 2)$  by (simp only: of-nat-power) also have  $sqrt (real n) \land 2 = real n$  by simpfinally have  $n = k \land 2$  by (simp only: of-nat-eq-iff) thus is-square n by blast qed (auto elim!: is-nth-powerE) corollary irrat-sqrt-nonsquare:  $\neg is$ -square  $n \Longrightarrow sqrt (real n) \notin \mathbb{Q}$ using sqrt-nat-or-irrat'[of n] by (auto simp: sqrt-nat-iff-is-square)

**lemma** sqrt-of-nat-in-Rats-iff: sqrt (real n)  $\in \mathbb{Q} \leftrightarrow is$ -square nusing irrat-sqrt-nonsquare[of n] sqrt-nat-iff-is-square[of n] Nats-subset-Rats by blast

**lemma** Discrete-sqrt-altdef: floor-sqrt  $n = nat \lfloor sqrt n \rfloor$  **proof** – **have** real (floor-sqrt  $n \ 2$ )  $\leq sqrt n \ 2$  **by** simp **hence** floor-sqrt  $n \leq sqrt n$  **unfolding** of-nat-power **by** (rule power2-le-imp-le) auto **moreover have** real (Suc (floor-sqrt n) \ 2) > real n unfolding of-nat-less-iff by (rule Suc-floor-sqrt-power2-gt) hence real (floor-sqrt n + 1)  $^2 > sqrt n ^2$ unfolding of-nat-power by simp hence real (floor-sqrt n + 1) > sqrt nby (rule power2-less-imp-less) auto hence floor-sqrt n + 1 > sqrt n by simp ultimately show ?thesis by linarith ged

# 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  $\Rightarrow$  bool where  $x \notin \mathbb{Q} \implies$  real-of-int  $a * x \stackrel{?}{2} +$  real-of-int b \* x + real-of-int  $c = 0 \implies$  $a \neq 0 \lor b \neq 0 \lor c \neq 0 \implies$  quadratic-irrational x

```
lemma quadratic-irrational-sqrt [intro]:
 assumes \neg is-square n
 shows quadratic-irrational (sqrt (real n))
 using irrat-sqrt-nonsquare[OF assms]
 by (intro quadratic-irrational.intros of sqrt n \ 1 \ 0 \ -int \ n) auto
lemma quadratic-irrational-uminus [intro]:
 assumes quadratic-irrational x
 shows quadratic-irrational (-x)
 using assms
proof induction
 case (1 x a b c)
 thus ?case by (intro quadratic-irrational.intros [of -x a - b c]) auto
qed
lemma quadratic-irrational-uninus-iff [simp]:
 quadratic-irrational (-x) \longleftrightarrow quadratic-irrational x
  using quadratic-irrational-uninus[of x] quadratic-irrational-uninus[of -x] by
auto
lemma quadratic-irrational-plus-int [intro]:
 assumes quadratic-irrational x
 shows quadratic-irrational (x + of\text{-int } n)
 using assms
proof induction
 case (1 x a b c)
 define x' where x' = x + of-int n
 define a' b' c' where
    a' = a and b' = b - 2 * of-int n * a and
    c' = a * of-int n \hat{2} - b * of-int n + c
 from 1 have 0 = a * (x' - of int n) \hat{2} + b * (x' - of int n) + c
```

by (simp add: x'-def) also have  $\ldots = a' * x' \hat{\phantom{a}} 2 + b' * x' + c'$ by (simp add: algebra-simps a'-def b'-def c'-def power2-eq-square) finally have  $\ldots = 0$ .. moreover have  $x' \notin \mathbb{Q}$ using 1 by (auto simp: x'-def add-in-Rats-iff2) moreover have  $a' \neq 0 \lor b' \neq 0 \lor c' \neq 0$ using 1 by (auto simp: a'-def b'-def c'-def) ultimately show ?case by (intro quadratic-irrational.intros[of x + of-int n a' b' c']) (auto simp: x'-def) qed

**lemma** quadratic-irrational-plus-int-iff [simp]: quadratic-irrational  $(x + of\text{-int } n) \leftrightarrow$  quadratic-irrational xusing quadratic-irrational-plus-int[of x n] quadratic-irrational-plus-int[of x + of-int n - n] by auto

**lemma** quadratic-irrational-minus-int-iff [simp]: quadratic-irrational  $(x - of\text{-int } n) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff [of x - n] by (simp del: quadratic-irrational-plus-int-iff)

- **lemma** quadratic-irrational-plus-nat-iff [simp]: quadratic-irrational  $(x + of\text{-nat } n) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff[of x int n] by (simp del: quadratic-irrational-plus-int-iff)
- **lemma** quadratic-irrational-minus-nat-iff [simp]: quadratic-irrational  $(x - of\text{-}nat n) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff [of x - int n] by (simp del: quadratic-irrational-plus-int-iff)
- **lemma** quadratic-irrational-plus-1-iff [simp]: quadratic-irrational  $(x + 1) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff[of x 1] by (simp del: quadratic-irrational-plus-int-iff)
- **lemma** quadratic-irrational-minus-1-iff [simp]: quadratic-irrational  $(x - 1) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff[of x - 1] by (simp del: quadratic-irrational-plus-int-iff)
- **lemma** quadratic-irrational-plus-numeral-iff [simp]: quadratic-irrational  $(x + numeral n) \leftrightarrow$  quadratic-irrational x using quadratic-irrational-plus-int-iff[of x numeral n] by (simp del: quadratic-irrational-plus-int-iff)
- **lemma** quadratic-irrational-minus-numeral-iff [simp]: quadratic-irrational  $(x - numeral \ n) \leftrightarrow$  quadratic-irrational x

```
by (simp del: quadratic-irrational-plus-int-iff)
lemma quadratic-irrational-inverse:
 assumes quadratic-irrational x
 shows quadratic-irrational (inverse x)
 using assms
proof induction
 case (1 x a b c)
 from 1 have x \neq 0 by auto
 have \theta = (real-of-int \ a * x^2 + real-of-int \ b * x + real-of-int \ c) / x \ 2
   by (subst 1) simp
  also have \ldots = real-of-int c * (inverse x) ^2 + real-of-int b * inverse x +
real-of-int a
   using \langle x \neq 0 \rangle by (simp add: field-simps power2-eq-square)
 finally have \ldots = 0...
 thus ?case using 1
   by (intro quadratic-irrational.intros[of inverse x \ c \ b \ a]) auto
qed
lemma quadratic-irrational-inverse-iff [simp]:
  quadratic-irrational (inverse x) \longleftrightarrow quadratic-irrational x
  using quadratic-irrational-inverse [of x] quadratic-irrational-inverse [of inverse x]
 by (cases x = 0) auto
lemma quadratic-irrational-cfrac-remainder-iff:
  quadratic-irrational (cfrac-remainder c \ n) \longleftrightarrow quadratic-irrational (cfrac-lim c)
proof (cases cfrac-length c = \infty)
 case False
 thus ?thesis
   by (auto simp: quadratic-irrational.simps)
next
 \mathbf{case}~[simp]:~True
 show ?thesis
 proof (induction n)
   case (Suc n)
   from Suc. prems have cfrac-remainder c (Suc n) =
                       inverse (cfrac-remainder c n - of-int (cfrac-nth c n))
     by (subst cfrac-remainder-Suc) (auto simp: field-simps)
   also have quadratic-irrational \ldots \leftrightarrow quadratic-irrational (cfrac-remainder c
n)
     by simp
   also have \ldots \iff quadratic-irrational (cfrac-lim c)
     by (rule Suc.IH)
   finally show ?case .
  qed auto
qed
```

using quadratic-irrational-plus-int-iff [of x -numeral n]

# 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  $f \equiv (\lambda x. \ a * x \ \widehat{2} + b * x + c)$ defines  $discr \equiv (b^2 - 4 * a * c)$ 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  $\geq 0$  then  $\{(-b + sqrt \ discr) / (2 * a), (-b - sqrt \ discr)\}$ /(2 \* a)else {}) (is ?th1) **proof** (cases a = 0) case [simp]: True **show** ?th1 **proof** (cases  $b = \theta$ ) case [simp]: True hence  $\{x, f x = 0\} = (if c = 0 \text{ then UNIV else } \}$ **by** (*auto simp: f-def*) thus ?th1 by simp  $\mathbf{next}$ case False hence  $\{x. f x = 0\} = \{-c / b\}$  by (auto simp: f-def field-simps) thus ?th1 using False by simp qed  $\mathbf{next}$ case [simp]: False show ?th1 **proof** (cases discr > 0) case True { fix x :: realhave  $f x = a * (x - (-b + sqrt \ discr) / (2 * a)) * (x - (-b - sqrt \ discr) / (2 * a))$ (2 \* a))using True by (simp add: f-def field-simps discr-def power2-eq-square) also have  $\ldots = 0 \iff x \in \{(-b + sqrt \ discr) \ / \ (2 * a), \ (-b - sqrt \ discr)\}$ /(2 \* a)by simp finally have  $f x = 0 \leftrightarrow \dots$ . hence  $\{x, f x = 0\} = \{(-b + sqrt \ discr) / (2 * a), (-b - sqrt \ discr) / (2 * a)\}$  $a)\}$ by blast thus ?th1 using True by simp  $\mathbf{next}$ case False {

fix x :: realassume x: f x = 0have  $0 \leq (x + b / (2 * a)) \hat{} 2$  by simp also have  $f x = a * ((x + b / (2 * a)) ^2 - b ^2 / (4 * a ^2) + c / a)$ **by** (*simp add: field-simps power2-eq-square f-def*) with x have  $(x + b / (2 * a)) \hat{2} - b \hat{2} / (4 * a \hat{2}) + c / a = 0$ by simp hence  $(x + b / (2 * a)) \ 2 = b \ 2 / (4 * a \ 2) - c / a$ **by** (*simp add: algebra-simps*) finally have  $0 \le (b^2 / (4 * a^2) - c / a) * (4 * a^2)$  $\mathbf{by}~(\mathit{intro}~\mathit{mult-nonneg-nonneg})~\mathit{auto}$ also have  $\ldots = b^2 - 4 * a * c$  by (simp add: field-simps power2-eq-square) also have  $\ldots < 0$  using False by (simp add: discr-def) finally have *False* by *simp* } hence  $\{x, f x = 0\} = \{\}$  by *auto* thus ?th1 using False by simp qed qed **lemma** finite-quadratic-equation-solutions-reals: fixes  $a \ b \ c :: real$ defines  $discr \equiv (b^2 - 4 * a * c)$ shows finite  $\{x. \ a * x \ 2 + b * x + c = 0\} \longleftrightarrow a \neq 0 \lor b \neq 0 \lor c \neq 0$ **by** (*subst quadratic-equation-reals*) (auto simp: discr-def card-eq-0-iff infinite-UNIV-char-0 split: if-split) **lemma** card-quadratic-equation-solutions-reals: fixes  $a \ b \ c :: real$ defines  $discr \equiv (b^2 - 4 * a * c)$ **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  $\geq 0$  then if discr = 0 then 1 else 2 else 0) (is ?th1) by (subst quadratic-equation-reals) (auto simp: discr-def card-eq-0-iff infinite-UNIV-char-0 split: if-split)

**lemma** card-quadratic-equation-solutions-reals-le-2: card { $x :: real. \ a * x \ 2 + b * x + c = 0$ }  $\leq 2$ by (subst card-quadratic-equation-solutions-reals) auto **lemma** quadratic-equation-solution-rat-iff: fixes  $a \ b \ c :: int$  and  $x \ y :: real$ defines  $f \equiv (\lambda x :: real. \ a * x \ 2 + b * x + c)$ defines  $discr \equiv nat \ (b \ 2 - 4 * a * c)$ 

assumes  $a \neq 0$  f x = 0shows  $x \in \mathbb{Q} \iff is$ -square discr

proof -

define discr' where discr'  $\equiv$  real-of-int  $(b \ 2 - 4 * a * c)$ from assms have  $x \in \{x, f x = 0\}$  by simp

with  $\langle a \neq 0 \rangle$  have  $discr' \geq 0$  unfolding discr'-def f-def of-nat-diff

by (subst (asm) quadratic-equation-reals) (auto simp: discr-def split: if-splits) hence \*: sqrt (discr') = sqrt (real discr) unfolding of-int-0-le-iff discr-def discr'-def

**by** (*simp add: algebra-simps nat-diff-distrib*)

from  $\langle x \in \{x, f | x = 0\}$  have  $x = (-b + sqrt \ discr) / (2 * a) \lor x = (-b - b)$  $sqrt \ discr) \ / \ (2 \ * \ a)$ 

using  $\langle a \neq 0 \rangle *$  unfolding discr'-def f-def

by (subst (asm) quadratic-equation-reals) (auto split: if-splits)

thus ?thesis using  $\langle a \neq 0 \rangle$ 

by (auto simp: sqrt-of-nat-in-Rats-iff divide-in-Rats-iff2 diff-in-Rats-iff2 diff-in-Rats-iff1) qed

### 2.4Periodic 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:  $l > 0 \ Ak$ .  $k \ge N \Longrightarrow cfrac-nth \ c \ (k+l) = cfrac-nth \ c \ k$ **shows** quadratic-irrational (cfrac-lim c) proof define h' and k' where h' = conv-num-int (cfrac-drop N c) and k' = conv-denom-int (cfrac-drop N c)define x' where x' = cfrac-remainder c Nhave c-pos: cfrac-nth  $c \ n > 0$  if n > N for n proof from assms(1,2) have  $cfrac-nth \ c \ (n+l) > 0$  by autowith assms(3)[OF that] show ?thesis by simp qed have k'-pos: k' n > 0 if  $n \neq -1$   $n \geq -2$  for n using that by (auto simp: k'-def conv-denom-int-def intro!: conv-denom-pos) have k'-nonneq:  $k' n \ge 0$  if  $n \ge -2$  for n using that by (auto simp: k'-def conv-denom-int-def intro!: conv-denom-pos) have cfrac-nth c (n + (N + l)) = cfrac-nth c (n + N) for nusing period(2)[of n + N] by  $(simp \ add: add-ac)$ have cfrac-drop (N + l) c = cfrac-drop N cby (rule cfrac-eqI) (use period(2)[of n + N for n] in (auto simp: algebra-simps)) hence x'-altdef: x' = cfrac-remainder c (N + l)by (simp add: x'-def cfrac-remainder-def) have x'-pos: x' > 0 unfolding x'-def

using *c*-pos by (intro cfrac-remainder-pos) auto

define A where  $A = (k' (int \ l - 1))$ define B where B = k' (int l - 2) – h' (int l - 1) define C where  $C = -(h' (int \ l - 2))$ have pos: (k' (int l - 1) \* x' + k' (int l - 2)) > 0using x'-pos  $\langle l > 0 \rangle$ by (intro add-pos-nonneg mult-pos-pos) (auto intro!: k'-pos k'-nonneg) have cfrac-remainder c N = conv' (cfrac-drop N c) l (cfrac-remainder c (l + l)N))unfolding cfrac-remainder-def cfrac-drop-add by (subst (2) cfrac-remainder-def [symmetric]) (auto simp: conv'-cfrac-remainder) hence  $x' = conv' (cfrac-drop \ N \ c) \ l \ x'$ by (subst (asm) add.commute) (simp only: x'-def [symmetric] x'-altdef [symmetric]) also have ... = (h' (int l - 1) \* x' + h' (int l - 2)) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2)) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2)) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l - 1) \* x' + h' (int l - 2))) / (k' (int l  $k' (int \ l - 2))$ using conv'-num-denom-int[OF x'-pos, of - l] unfolding h'-def k'-def **by** (*simp add: mult-ac*) finally have x' \* (k' (int l - 1) \* x' + k' (int l - 2)) = (h' (int l - 1) \* x' + k' (int l - 2)) $h'(int \ l - 2))$ using pos by (simp add: divide-simps) hence quadratic:  $A * x' \hat{\phantom{a}} 2 + B * x' + C = 0$ by (simp add: algebra-simps power2-eq-square A-def B-def C-def) moreover have  $x' \notin \mathbb{Q}$  unfolding x'-def by auto moreover have A > 0 using  $\langle l > 0 \rangle$  by (auto simp: A-def introl: k'-pos) ultimately have *quadratic-irrational* x' using  $\langle x' \notin \mathbf{O} \rangle$ by (intro quadratic-irrational.intros[of x' A B C]) simp-all thus ?thesis using assms by (simp add: x'-def quadratic-irrational-cfrac-remainder-iff) qed

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

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

**lemma** periodic-cfrac-Nil [simp]: pperiodic-cfrac [] = 0unfolding zero-cfrac-def by transfer auto

**lemma** cfrac-length-pperiodic-cfrac [simp]:  $xs \neq [] \implies cfrac-length (pperiodic-cfrac xs) = \infty$ by transfer auto

**lemma** cfrac-nth-pperiodic-cfrac:

assumes  $xs \neq []$  and  $0 \notin set xs$ **shows** cfrac-nth (pperiodic-cfrac xs)  $n = xs ! (n \mod length xs)$ using assms **proof** (*transfer*, *goal-cases*) case (1 xs n)show ?case **proof** (cases n) case (Suc n') have int (cycle (tl (map  $(\lambda n. n - 1) xs)$  @ [hd (map  $(\lambda n. n - 1) xs$ )]) !! n') + 1 =int (stl (cycle (map  $(\lambda n. n - 1) xs)$ ) !! n') + 1 **by** (subst cycle.sel(2) [symmetric]) (rule refl) also have ... = int (cycle (map  $(\lambda n. n - 1) xs) \parallel n) + 1$ by (simp add: Suc del: cycle.sel) also have  $\ldots = int (xs ! (n mod length xs) - 1) + 1$ by (simp add: snth-cycle  $\langle xs \neq [] \rangle$ ) also have  $xs ! (n \mod length xs) \in set xs$ using  $\langle xs \neq | \rangle$  by (auto simp: set-conv-nth) with 1 have  $xs ! (n \mod length xs) > 0$ by (intro Nat.gr $\theta I$ ) auto hence int  $(xs ! (n \mod length xs) - 1) + 1 = int (xs ! (n \mod length xs))$ by simp finally show ?thesis using Suc 1 by (simp add: hd-conv-nth map-tl) **qed** (use 1 **in** (auto simp: hd-conv-nth)) qed **definition** pperiodic-cfrac-info :: nat list  $\Rightarrow$  int  $\times$  int  $\times$  intwhere 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  $(B^2 - 4 * A * C, B, 2 * A))$ 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 using assms **proof** (induction N - n arbitrary:  $a \ b \ n \ N$ ) case (Suc d n N a b) have conv-gen f(b, b \* f n + a, Suc n) N = conv-gen f'(b, b \* f n + a, Suc n)Nusing Suc(2,3) by (intro Suc) auto moreover have f n = f' nusing bspec[OF Suc.prems, of n] Suc(2) by auto ultimately show ?case

**by** (subst (1 2) conv-gen.simps) auto **qed** (auto simp: conv-gen.simps)

### lemma

assumes  $\forall k \leq n. \ c \ k = cfrac - nth \ c' \ k$ **shows** conv-num-fun-eq': conv-num-fun c n = conv-num c' nand conv-denom-fun-eq':  $conv-denom-fun \ c \ n = conv-denom \ c' \ n$ proof have conv-num c' n = conv-gen (cfrac-nth c') (0, 1, 0) nunfolding conv-num-code .. also have  $\ldots = conv-gen \ c \ (0, \ 1, \ 0) \ n$ unfolding conv-num-fun-def using assms by (intro conv-gen-cong) auto finally show conv-num-fun c n = conv-num c' n**by** (*simp add: conv-num-fun-def*)  $\mathbf{next}$ have conv-denom c' n = conv-qen (cfrac-nth c') (1, 0, 0) n unfolding conv-denom-code .. also have  $\ldots = conv-gen \ c \ (1, \ 0, \ 0) \ n$ unfolding conv-denom-fun-def using assms by (intro conv-gen-cong) auto finally show conv-denom-fun c n = conv-denom c' nby (simp add: conv-denom-fun-def)  $\mathbf{qed}$ 

**lemma** gcd-minus-commute-left: gcd (a - b :: 'a :: ring-gcd) c = gcd (b - a) cby (metis gcd.commute gcd-neg2 minus-diff-eq)

**lemma** gcd-minus-commute-right: gcd c (a - b :: 'a :: ring-gcd) = gcd c (b - a)by (metis gcd-neg2 minus-diff-eq)

**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 > 0proof – define c where c = pperiodic-cfrac xshave [simp]: cfrac-length  $c = \infty$ using assms by (simp add: c-def) define h and k where h = conv-num-int c and k = conv-denom-int c define x where x = cfrac-lim c define l where l = length xsdefine A where  $A = (k \ (int \ l - 1))$ 

define A where  $A = (k \ (int \ l - 1))$ define B where  $B = k \ (int \ l - 2) - h \ (int \ l - 1)$ define C where  $C = -(h \ (int \ l - 2))$ define discr where  $discr = B \ 2 - 4 * A * C$ 

have  $l > \theta$ 

using assms by (simp add: l-def) have c-pos: cfrac-nth c n > 0 for nusing assms by (auto simp: c-def cfrac-nth-pperiodic-cfrac set-conv-nth) have x-pos: x > 0**unfolding** *x*-*def* **by** (*intro cfrac-lim-pos c-pos*) have *h*-pos:  $h \ n > 0$  if n > -2 for nusing that unfolding h-def by (auto simp: conv-num-int-def intro: conv-num-pos' c-pos) have k-pos: k n > 0 if n > -1 for n using that unfolding k-def by (auto simp: conv-denom-int-def) have k-nonneg:  $k \ n \ge 0$  for n **unfolding** k-def by (auto simp: conv-denom-int-def) have pos: (k (int l - 1) \* x + k (int l - 2)) > 0using *x*-pos  $\langle l > 0 \rangle$ by (intro add-pos-nonneq mult-pos-pos) (auto intro!: k-pos k-nonneq) have cfrac-drop l c = cusing assms by (intro cfrac-eqI) (auto simp: c-def cfrac-nth-pperiodic-cfrac l-def) have x = conv' c l (cfrac-remainder c l) **unfolding** *x*-def **by** (rule conv'-cfrac-remainder[symmetric]) auto also have  $\ldots = conv' c \ l x$ unfolding cfrac-remainder-def  $\langle cfrac-drop \ l \ c = c \rangle \ x-def$ . finally have x = conv' c l x. **also have** ... = (h (int l - 1) \* x + h (int l - 2)) / (k (int l - 1) \* x + k) $(int \ l - 2))$ using conv'-num-denom-int[OF x-pos, of - l] unfolding h-def k-def **by** (*simp add: mult-ac*) finally have x \* (k (int l - 1) \* x + k (int l - 2)) = (h (int l - 1) \* x + h) $(int \ l - 2))$ using pos by (simp add: divide-simps) hence quadratic:  $A * x \hat{2} + B * x + C = 0$ by (simp add: algebra-simps power2-eq-square A-def B-def C-def) have A > 0 using  $\langle l > 0 \rangle$  by (auto simp: A-def introl: k-pos) have discr-altdef: discr =  $(k (int l-2) - h (int l-1)) \land 2 + 4 * k (int l-1) *$ h (int l-2) by (simp add: discr-def A-def B-def C-def) have 0 < 0 + 4 \* A \* 1using  $\langle A > \theta \rangle$  by simp also have  $\theta + 4 * A * 1 \leq discr$ **unfolding** discr-altdef A-def using h-pos[of int l - 2]  $\langle l > 0 \rangle$ by (intro add-mono mult-mono order.refl k-nonneg mult-nonneg-nonneg) auto finally have discr > 0. have  $x \in \{x. A * x \hat{2} + B * x + C = 0\}$ using quadratic by simp

hence x-cases:  $x = (-B - sqrt \ discr) / (2 * A) \lor x = (-B + sqrt \ discr) / (2$ \* Aunfolding quadratic-equation-reals of-int-diff using  $\langle A > 0 \rangle$ by (auto split: if-splits simp: discr-def) have  $B \ 2 < discr$ **unfolding** discr-def by (auto introl: mult-pos-pos k-pos h-pos  $\langle l > 0 \rangle$  simp: A-def C-def) hence  $|B| < sqrt \ discr$ using  $\langle discr > 0 \rangle$  by  $(simp \ add: real-less-rsqrt)$ have  $x = (if x \ge 0 \text{ then } (sqrt \operatorname{discr} - B) / (2 * A) \text{ else } -(sqrt \operatorname{discr} + B) / (2 * A) / (2 * A) \text{ else } -(sqrt \operatorname{discr} + B) / (2 * A) /$ \* A))using *x*-cases proof assume x:  $x = (-B - sqrt \ discr) / (2 * A)$ have  $(-B - sqrt \ discr) / (2 * A) < 0$ using  $\langle |B| < sqrt \ discr \rangle \langle A > 0 \rangle$  by (intro divide-neg-pos) auto also note *x*[*symmetric*] finally show ?thesis using x by simp  $\mathbf{next}$ assume x:  $x = (-B + sqrt \ discr) / (2 * A)$ have  $(-B + sqrt \ discr) / (2 * A) > 0$ using  $\langle |B| < sqrt \ discr \rangle \langle A > 0 \rangle$  by (intro divide-pos-pos) auto also note *x*[*symmetric*] finally show ?thesis using x by simp qed also have  $x \ge 0 \iff \text{floor } x \ge 0$ **by** *auto* **also have** floor x = floor (cfrac-lim c)by (simp add: x-def) also have  $\ldots = cfrac \cdot nth \ c \ \theta$ **by** (subst cfrac-nth-0-conv-floor) auto also have  $\ldots = int (hd xs)$ using assms unfolding c-def by (subst cfrac-nth-pperiodic-cfrac) (auto simp: *hd-conv-nth*) finally have x-eq:  $x = (sqrt \ discr - B) / (2 * A)$ by simp

define h' where h' = conv-num-fun  $(\lambda n. int (xs ! n))$ define k' where k' = conv-denom-fun  $(\lambda n. int (xs ! n))$ have num-eq: h' i = h iif i < l for i using that assms unfolding h'-def h-def by (subst conv-num-fun-eq'[where c' = c]) (auto simp: c-def l-def cfrac-nth-pperiodic-cfrac) have denom-eq: k' i = k iif i < l for i using that assms unfolding k'-def k-def

by (subst conv-denom-fun-eq'[where c' = c]) (auto simp: c-def l-def cfrac-nth-pperiodic-cfrac)

have 1:  $h(int \ l - 1) = h'(l - 1)$ by (subst num-eq) (use  $\langle l > 0 \rangle$  in (auto simp: of-nat-diff)) have 2: k (int l - 1) = k'(l - 1)by (subst denom-eq) (use  $\langle l > 0 \rangle$  in  $\langle auto simp: of-nat-diff \rangle$ ) have 3:  $h(int \ l - 2) = (if \ l = 1 \ then \ 1 \ else \ h'(l - 2))$ using  $\langle l > 0 \rangle$  num-eq[of l - 2] by (auto simp: h-def nat-diff-distrib) have 4:  $k (int \ l - 2) = (if \ l = 1 \ then \ 0 \ else \ k' (l - 2))$ using  $\langle l > 0 \rangle$  denom-eq[of l - 2] by (auto simp: k-def nat-diff-distrib) **have** pperiodic-cfrac-info xs =(let A = k (int l - 1);B = h (int l - 1) - (if l = 1 then 0 else k (int l - 2)); $C = (if \ l = 1 \ then \ -1 \ else \ -h \ (int \ l \ -2))$  $in (B^2 - 4 * A * C, B, 2 * A))$ unfolding pperiodic-cfrac-info-def Let-def using  $1 \ 2 \ 3 \ 4 \ \langle l > 0 \rangle$ by (auto simp: num-eq denom-eq h'-def k'-def l-def of-nat-diff) also have ... =  $(B^2 - 4 * A * C, -B, 2 * A)$ by (simp add: Let-def A-def B-def C-def h-def k-def algebra-simps power2-commute) finally have per-eq: pperiodic-cfrac-info xs = (discr, -B, 2 \* A)by (simp add: discr-def) **show** x = (sqrt (real-of-int D) + real-of-int E) / real-of-int Fusing per-eq assms by (simp add: x-eq)

```
show D > 0 F > 0
using assms per-eq (discr > 0) (A > 0) by auto
qed
```

qea

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  $\bigwedge n m. n \ge N \implies cfrac-nth e (n + m * l) = cfrac-nth e n$ and cfrac-remainder e (N + l) = cfrac-remainder e Nand cfrac-length  $e = \infty$ proof -

have [simp]: cfrac-length  $e = \infty$ 

using assms by (auto simp: quadratic-irrational.simps) **note** [intro] = assms(1)define x where x = cfrac-lim efrom assms obtain  $a \ b \ c :: int$  where *nontrivial*:  $a \neq 0 \lor b \neq 0 \lor c \neq 0$  and *root*:  $a * x^2 + b * x + c = 0$  (is ?f x = 0) **by** (*auto simp: quadratic-irrational.simps x-def*) define f where f = ?fdefine h and k where h = conv-num e and k = conv-denom edefine X where X = cfrac-remainder e have [simp]:  $k \ i > 0 \ k \ i \neq 0$  for i using conv-denom-pos[of e i] by (auto simp: k-def) have k-leI:  $k \ i \leq k \ j$  if  $i \leq j$  for  $i \ j$ **by** (*auto simp: k-def intro!: conv-denom-leI that*) have k-nonneq: k n > 0 for n **by** (*auto simp*: k-def) have k-ge-1:  $k n \ge 1$  for nusing k-leI[of 0 n] by (simp add: k-def) define R where R = conv edefine A where  $A = (\lambda n. \ a * h \ (n - 1) \ \hat{2} + b * h \ (n - 1) * k \ (n - 1) + c$  $* k (n - 1) \hat{2}$ define B where  $B = (\lambda n. \ 2 * a * h \ (n - 1) * h \ (n - 2) + b * (h \ (n - 1) * k)$ (n-2) + h(n-2) \* k(n-1) + 2 \* c \* k(n-1) \* k(n-2))define C where  $C = (\lambda n. \ a * h \ (n-2) \ \widehat{2} + b * h \ (n-2) * k \ (n-2) + b + h \ (n-2) + k \ (n$  $c * k (n - 2) \hat{\phantom{a}} 2$ define A' where A' = nat |2 \* |a| \* |x| + |a| + |b||define B' where B' = nat |(3 / 2) \* (2 \* |a| \* |x| + |b|) + 9 / 4 \* |a||have [simp]:  $X n \notin \mathbb{Q}$  for n unfolding X-def by simp from this of 0 have simple  $x \notin \mathbb{Q}$ **unfolding** X-def by (simp add: x-def) have  $a \neq 0$ proof assume a = 0with root and nontrivial have  $x = 0 \lor x = -c / b$ **by** (*auto simp: divide-simps add-eq-0-iff*) hence  $x \in \mathbb{Q}$  by (auto simp del:  $\langle x \notin \mathbb{Q} \rangle$ ) thus False by simp qed have bounds:  $(A \ n, B \ n, C \ n) \in \{-A'..A'\} \times \{-B'..B'\} \times \{-A'..A'\}$ and X-root: A  $n * X n \hat{2} + B n * X n + C n = 0$  if  $n: n \ge 2$  for n proof define n' where n' = n - 2

have n': n = Suc (Suc n') using  $(n \ge 2)$  unfolding n'-def by simp have \*: of-int  $(k (n - Suc \ \theta)) * X n + of-int (k (n - 2)) \neq \theta$ proof assume of-int  $(k (n - Suc \ 0)) * X n + of-int (k (n - 2)) = 0$ hence X n = -k (n - 2) / k (n - 1) by (auto simp: divide-simps mult-ac) also have  $\ldots \in \mathbb{Q}$  by *auto* finally show False by simp qed let ?denom = (k (n - 1) \* X n + k (n - 2))have  $\theta = \theta * ?denom \hat{2}$  by simp also have  $0 * ?denom \ 2 = (a * x \ 2 + b * x + c) * ?denom \ 2$  using root by simp also have  $\ldots = a * (x * ?denom) \land 2 + b * ?denom * (x * ?denom) + c *$ ?denom \* ?denom **by** (simp add: algebra-simps power2-eq-square) **also have** x \* ?denom = h (n - 1) \* X n + h (n - 2)using cfrac-lim-eq-num-denom-remainder-aux[of n - 2 e]  $\langle n \geq 2 \rangle$ by (simp add: numeral-2-eq-2 Suc-diff-Suc x-def k-def h-def X-def) also have  $a * \ldots ?2 + b * ?denom * \ldots + c * ?denom * ?denom = A n * ?denom = A n$  $X n \widehat{\phantom{a}} 2 + B n * X n + C n$ by (simp add: A-def B-def C-def power2-eq-square algebra-simps) finally show  $A n * X n \hat{2} + B n * X n + C n = 0$ .. have *f*-abs-bound:  $|f(R n)| \le (2 * |a| * |x| + |b|) * (1 / (k n * k (Suc n))) +$  $|a| * (1 / (k n * k (Suc n))) ^2$  for n proof – have |f(R n)| = |?f(R n) - ?fx| by (simp add: root f-def) **also have**  $?f(R n) - ?fx = (R n - x) * (2 * a * x + b) + (R n - x) ^2$ \* a **by** (*simp add: power2-eq-square algebra-simps*) also have  $|...| \leq |(R \ n - x) * (2 * a * x + b)| + |(R \ n - x) \widehat{2} * a|$ **by** (*rule abs-triangle-ineq*) also have ... =  $|2 * a * x + b| * |R n - x| + |a| * |R n - x| ^2$ **by** (*simp add: abs-mult*) **also have** ... < |2 \* a \* x + b| \* (1 / (k n \* k (Suc n))) + |a| \* (1 / (k n n)) $* k (Suc n))) ^2$ **unfolding** *x*-def *R*-def **using** cfrac-lim-minus-conv-bounds[of *n* e] by (intro add-mono mult-left-mono power-mono) (auto simp: k-def) **also have**  $|2 * a * x + b| \le 2 * |a| * |x| + |b|$ by (rule order.trans[OF abs-triangle-ineq]) (auto simp: abs-mult) hence |2 \* a \* x + b| \* (1 / (k n \* k (Suc n))) + |a| \* (1 / (k n \* k (Suc n))) $n))) \cap 2 \leq$ ...  $* (1 / (k n * k (Suc n))) + |a| * (1 / (k n * k (Suc n))) ^2$ by (intro add-mono mult-right-mono) (auto intro!: mult-nonneg-nonneg k-nonneg) finally show  $|f(R n)| < \dots$ by (simp add: mult-right-mono add-mono divide-left-mono) qed

have h-eq-conv-k:  $h \ i = R \ i * k \ i$  for iusing conv-denom-pos[of e i] unfolding R-def **by** (*subst conv-num-denom*) (*auto simp: h-def k-def*) have  $A \ n = k \ (n - 1) \ \widehat{2} * f \ (R \ (n - 1))$  for nby (simp add: algebra-simps A-def n' k-def power2-eq-square h-eq-conv-k f-def) have A-bound:  $|A i| \leq A'$  if i > 0 for i proof have  $k \ i > 0$ by simp hence  $k \ i \ge 1$ by linarith have  $A \ i = k \ (i - 1) \ \widehat{2} * f \ (R \ (i - 1))$ by (simp add: algebra-simps A-def k-def power2-eq-square h-eq-conv-k f-def) also have  $|...| = k(i-1) \hat{2} * |f(R(i-1))|$ **by** (*simp add: abs-mult f-def*) also have ...  $\leq k (i - 1) \hat{z} * ((2 * |a| * |x| + |b|) * (1 / (k (i - 1) * k$ (Suc (i - 1))) + $|a| * (1 / (k (i - 1) * k (Suc (i - 1)))) ^2)$ by (intro mult-left-mono f-abs-bound) auto also have ... =  $k(i - 1) / ki * (2 * |a| * |x| + |b|) + |a| / ki^2$  using  $\langle i > 0 \rangle$ **by** (*simp add: power2-eq-square field-simps*) also have  $\ldots \leq 1 * (2 * |a| * |x| + |b|) + |a| / 1$  using  $\langle i > 0 \rangle \langle k | i \geq 1 \rangle$ **by** (*intro add-mono divide-left-mono mult-right-mono*) (auto intro!: k-leI one-le-power simp: of-nat-ge-1-iff) **also have** ... = 2 \* |a| \* |x| + |a| + |b| by simp finally show ?thesis unfolding A'-def by linarith qed have C n = A (n - 1) by (simp add: A-def C-def n') hence C-bound:  $|C n| \leq A'$  using A-bound[of n - 1] n by simp have B n = k (n - 1) \* k (n - 2) \*(f(R(n-1)) + f(R(n-2)) - a \* (R(n-1) - R(n-2))2)by (simp add: B-def h-eq-conv-k algebra-simps power2-eq-square f-def) also have |...| = k (n - 1) \* k (n - 2) \*|f(R(n-1)) + f(R(n-2)) - a \* (R(n-1) - R(n-2))| $2)) ^{2}$ **by** (*simp add: abs-mult k-nonneg*) also have ...  $\leq k (n - 1) * k (n - 2) *$ (((2 \* |a| \* |x| + |b|) \* (1 / (k (n - 1) \* k (Suc (n - 1)))) + $|a| * (1 / (k (n - 1) * k (Suc (n - 1)))) ^2) +$ ((2 \* |a| \* |x| + |b|) \* (1 / (k (n - 2) \* k (Suc (n - 2))))) + $|a| * (1 / (k (n - 2) * k (Suc (n - 2)))) ^2) +$  $|a| * |R (Suc (n - 2)) - R (n - 2)| ^2)$  (is  $- \le - * (?S1 + 2)$ (S2 + (S3))

**by** (*intro mult-left-mono order.trans*[*OF abs-triangle-ineq4*] *order.trans*[*OF abs-triangle-ineq*]

add-mono f-abs-bound order.refl)

(insert n, auto simp: abs-mult Suc-diff-Suc numeral-2-eq-2 k-nonneg)

also have |R (Suc (n - 2)) - R (n - 2)| = 1 / (k (n - 2) \* k (Suc (n - 2)))

**unfolding** *R*-def *k*-def **by** (rule abs-diff-successive-convs)

also have of-int  $(k (n - 1) * k (n - 2)) * (?S1 + ?S2 + |a| * ... ^2) = (k (n - 2) / k n + 1) * (2 * |a| * |x| + |b|) + |a| * (k (n - 2) / (k (n - 1) * k n ^2) + 2 / (k (n - 1) * k (n - 1)))$ 

2)))

(is - ?S) using n by (simp add: field-simps power2-eq-square numeral-2-eq-2 Suc-diff-Suc)

also {

have A: 2 \* real-of-int (k (n - 2)) < of-int (k n)using conv-denom-plus2-ratio-ge[of e n - 2] n by (simp add: numeral-2-eq-2 Suc-diff-Suc k-def) have fib (Suc 2)  $\leq k$  2 unfolding k-def by (intro conv-denom-lower-bound) also have  $\ldots \leq k \ n$  by (intro k-leI n) finally have  $k \ n \ge 2$  by (simp add: numeral-3-eq-3) hence B: of-int  $(k (n - 2)) * 2 \ 2 \le (of-int (k (n - 1)) * (of-int (k n))^2)$ :: real)by (intro mult-mono power-mono) (auto intro: k-leI k-nonneg) have C:  $1 * 1 \leq real-of-int (k (n - 1)) * of-int (k (n - 2))$  using k-ge-1 by (intro mult-mono) (auto simp: Suc-le-eq of-nat-ge-1-iff k-nonneg) note A B C} hence  $?S \leq (1 / 2 + 1) * (2 * |a| * |x| + |b|) + |a| * (1 / 4 + 2)$ by (intro add-mono mult-right-mono mult-left-mono) (auto simp: field-simps) also have ... = (3 / 2) \* (2 \* |a| \* |x| + |b|) + 9 / 4 \* |a| by simp finally have *B*-bound:  $|B n| \leq B'$  unfolding *B'*-def by linarith **from** A-bound[of n] B-bound C-bound nshow  $(A \ n, B \ n, C \ n) \in \{-A'..A'\} \times \{-B'..B'\} \times \{-A'..A'\}$  by auto  $\mathbf{qed}$ have A-nz: A  $n \neq 0$  if n > 1 for n using that **proof** (*induction n rule*: *dec-induct*) case base

show ?case proof assume  $A \ 1 = 0$ hence real-of-int  $(A \ 1) = 0$  by simp also have real-of-int  $(A \ 1) =$ real-of-int a \* of-int (cfrac-nth  $e \ 0) \ 2 +$ real-of-int b \* cfrac-nth  $e \ 0 + real$ -of-int cby (simp add: A-def h-def k-def)

finally have  $root': \ldots = 0$ .

have cfrac-nth  $e \ \theta \in \mathbb{Q}$  by auto also from root' and  $\langle a \neq 0 \rangle$  have ?this  $\leftrightarrow$  is-square (nat  $(b^2 - 4 * a * a)$ c))by (intro quadratic-equation-solution-rat-iff) auto also from *root* and  $\langle a \neq 0 \rangle$  have  $\ldots \longleftrightarrow x \in \mathbb{Q}$ by (intro quadratic-equation-solution-rat-iff [symmetric]) auto finally show False using  $\langle x \notin \mathbb{Q} \rangle$  by contradiction qed next case (step m) hence nz: C (Suc m)  $\neq 0$  by (simp add: C-def A-def) show A (Suc m)  $\neq 0$ proof assume [simp]: A (Suc m) = 0have X (Suc m) > 0 unfolding X-def by (intro cfrac-remainder-pos) auto with X-root[of Suc m] step.hyps nz have X (Suc m) = -C (Suc m) / B (Suc m) **by** (*auto simp: divide-simps mult-ac*) also have  $\ldots \in \mathbb{Q}$  by *auto* finally show False by simp  $\mathbf{qed}$ qed have finite  $(\{-A'..A'\} \times \{-B'..B'\} \times \{-A'..A'\})$  by auto from this and bounds have finite  $((\lambda n. (A n, B n, C n)) ` \{2..\})$ **by** (*blast intro: finite-subset*) **moreover have** infinite ( $\{2..\}$  :: nat set) by (simp add: infinite-Ici) ultimately have  $\exists k_1 \in \{2..\}$ . infinite  $\{n \in \{2..\}$ .  $(A \ n, B \ n, C \ n) = (A \ k_1, B \ n, C \ n)$ k1, Ck1)**by** (*intro pigeonhole-infinite*) then obtain k0 where  $k0: k0 \ge 2$  infinite  $\{n \in \{2..\}\}$ .  $(A \ n, B \ n, C \ n) = (A \ n, B)$  $k\theta, B k\theta, C k\theta$ by auto from infinite-countable-subset [OF this(2)] obtain  $g :: nat \Rightarrow$  where g: inj g range  $g \subseteq \{n \in \{2..\}, (A \ n, B \ n, C \ n) = (A \ k0, B \ k0, C \ k0)\}$ **by** blast hence q-qe-2: q k > 2 for k by autofrom g have [simp]: A (g k) = A k 0 B (g k) = B k 0 C (g k) = C k 0 for k by auto from g(1) have  $[simp]: g k1 = g k2 \leftrightarrow k1 = k2$  for k1 k2 by (auto simp: *inj-def*) define z where  $z = (A \ k\theta, B \ k\theta, C \ k\theta)$ let  $?h = \lambda k. (A (g k), B (g k), C (g k))$ from g have g': distinct [g 1, g 2, g 3]?h 0 = z?h 1 = z?h 2 = z**by** (*auto simp*: *z*-*def*) have fin: finite {x :: real. A  $k0 * x \hat{2} + B k0 * x + C k0 = 0$ } using A-nz[of k0 ] k0(1)

**by** (subst finite-quadratic-equation-solutions-reals) auto **from** X-root[of  $g \ 0$ ] X-root[of  $g \ 1$ ] X-root[of  $g \ 2$ ] g-ge-2 g have  $(X \circ g)$  '  $\{0, 1, 2\} \subseteq \{x. A \ k0 \ *x \ 2 \ +B \ k0 \ *x \ +C \ k0 = 0\}$ by *auto* hence card  $((X \circ g) ` \{0, 1, 2\}) \leq card \dots$ by (intro card-mono fin) auto also have  $\ldots \leq 2$ by (rule card-quadratic-equation-solutions-reals-le-2) **also have** ... < card  $\{0, 1, 2 :: nat\}$  by simp finally have  $\neg inj$ -on  $(X \circ g) \{0, 1, 2\}$ **by** (*rule pigeonhole*) then obtain m1 m2 where  $m12: m1 \in \{0, 1, 2\} m2 \in \{0, 1, 2\} X (g m1) = X (g m2) m1 \neq m2$ unfolding *inj-on-def* o-def by blast define n and l where n = min (g m1) (g m2) and l = nat | int (g m1) - gm2with m12 q' have l: l > 0 X (n + l) = X nby (auto simp: min-def nat-diff-distrib split: if-splits) from l have cfrac-lim (cfrac-drop (n + l) e) = cfrac-lim (cfrac-drop n e) **by** (*simp add: X-def cfrac-remainder-def*) hence cfrac-drop (n + l) e = cfrac-drop n e**by** (*simp add: cfrac-lim-eq-iff*) **hence** cfrac-nth (cfrac-drop (n + l) e) = cfrac-nth (cfrac-drop n e)**by** (*simp only*:) hence period: cfrac-nth e(n + l + k) = cfrac-nth e(n + k) for k by (simp add: fun-eq-iff add-ac) have period: cfrac-nth e(k + l) = cfrac-nth ek if  $k \ge n$  for k using period [of k - n] that by (simp add: add-ac) have period: cfrac-nth e(k + m \* l) = cfrac-nth e k if  $k \ge n$  for k musing that **proof** (*induction* m) case (Suc m) have cfrac-nth e(k + Suc m \* l) = cfrac-nth e(k + m \* l + l)**by** (*simp add: algebra-simps*) also have  $\ldots = c frac \cdot n th \ e \ (k + m * l)$ using Suc.prems by (intro period) auto also have  $\ldots = cfrac$ -nth e kusing Suc.prems by (intro Suc.IH) auto finally show ?case . qed simp-all

from this and l and that [of l n] show ?thesis by (simp add: X-def) qed

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

**proof** safe **assume** \*: quadratic-irrational xwith assms have \*\*: quadratic-irrational (cfrac-lim (cfrac-of-real x)) by auto obtain N l where Nl: l > 0 $\bigwedge n \ m. \ N \le n \Longrightarrow cfrac-nth \ (cfrac-of-real x) \ (n + m * l) = cfrac-nth \ (n + m * l) \ (n + m * l) = cfrac-nth \ (n + m * l) \ (n + m * l) = cfrac-nth \ (n + m * l) \ (n + m * l) \ (n + m * l) = cfrac-nth \ (n + m * l) \ ($ x) ncfrac-remainder (cfrac-of-real x) (N + l) = cfrac-remainder (cfrac-of-real x) Ncfrac-length (cfrac-of-real x) =  $\infty$ using quadratic-irrational-imp-periodic-cfrac [OF \*\*] by metis **show**  $\exists N \ l. \ l > 0 \land (\forall n \ge N. \ cfrac-nth \ (cfrac-of-real x) \ (n + l) = cfrac-nth$ (cfrac-of-real x) n)by (rule exI[of - N], rule exI[of - l]) (insert Nl(1) Nl(2)[of - 1], auto)  $\mathbf{next}$ fix N l assume  $l > 0 \forall n > N$ . cfrac-nth (cfrac-of-real x) (n + l) = cfrac-nth (cfrac-of-real x) nhence quadratic-irrational (cfrac-lim (cfrac-of-real x)) using assms by (intro periodic-cfrac-imp-quadratic-irrational [of - l N]) auto with assms show quadratic-irrational x by simp qed The following result can e.g. be used to show that a number is *not* a

**lemma** quadratic-irrational-cfrac-nth-range-finite: assumes quadratic-irrational (cfrac-lim e) **shows** finite (range (cfrac-nth e)) proof – from quadratic-irrational-imp-periodic-cfrac[OF assms] obtain l N where period:  $l > 0 \land m n$ .  $n \ge N \Longrightarrow cfrac-nth \ e \ (n + m * l) = cfrac-nth \ e$ nby *metis* have cfrac-nth  $e \ k \in cfrac$ -nth  $e' \{..< N+l\}$  for k **proof** (cases k < N + l) case False define n m where  $n = N + (k - N) \mod l$  and  $m = (k - N) \dim l$ have cfrac-nth  $e \ n \in cfrac$ -nth  $e' \{..< N+l\}$ using  $\langle l > 0 \rangle$  by (intro imageI) (auto simp: n-def) also have cfrac-nth e = cfrac-nth e (n + m \* l)**by** (subst period) (auto simp: n-def) also have n + m \* l = kusing False by (simp add: n-def m-def) finally show ?thesis . ged auto hence range (cfrac-nth e)  $\subseteq$  cfrac-nth e ' {..<N+l} by blast thus ?thesis by (rule finite-subset) auto

 $\mathbf{qed}$ 

quadratic irrational.

# 3 The continued fraction expansion of e

 $\mathbf{end}$ 

```
theory E-CFrac
imports
 HOL-Analysis. Analysis
  Continued-Fractions
  Quadratic-Irrationals
begin
lemma fact-real-at-top: filterlim (fact :: nat \Rightarrow real) at-top at-top
proof (rule filterlim-at-top-mono)
 have real n \leq real (fact n) for n
   unfolding of-nat-le-iff by (rule fact-ge-self)
 thus eventually (\lambda n. real n \leq fact n) at-top by simp
qed (fact filterlim-real-sequentially)
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
 unfolding filterlim-at-top
proof
 fix C :: nat
 from assms(1) have eventually (\lambda x. f x \ge C * m) F
   by (auto simp: filterlim-at-top)
  thus eventually (\lambda x. f x \text{ div } m \ge C) F
 proof eventually-elim
   case (elim x)
   hence (C * m) div m \leq f x div m
     by (intro div-le-mono)
   thus ?case using \langle m > 0 \rangle by simp
 qed
qed
```

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*  $n \dim 3$ ) *else* 1)

**lemma** cfrac-nth-e: cfrac-nth e-cfrac  $n = (if \ n = 0 \ then \ 2 \ else \ if \ n \ mod \ 3 = 2 \ then \ 2 * (Suc \ n \ div \ 3) \ else \ 1)$ **unfolding** e-cfrac-def **by** (subst cfrac-nth-cfrac) (auto simp: is-cfrac-def)

**lemma** cfrac-length-e [simp]: cfrac-length e-cfrac =  $\infty$ by (simp add: e-cfrac-def)

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

### context

**fixes** A B C ::  $nat \Rightarrow real$  and  $p q :: nat \Rightarrow int$  and  $a :: nat \Rightarrow int$ defines  $A \equiv (\lambda n. integral \{0..1\} (\lambda x. exp \ x * x \ n * (x - 1) \ n \ fact \ n))$ and  $B \equiv (\lambda n. integral \{0..1\} (\lambda x. exp \ x * x \cap Suc \ n * (x - 1) \cap n \ fact \ n))$ and  $C \equiv (\lambda n. integral \{0...1\} (\lambda x. exp \ x * x \ n * (x - 1) \ Suc \ n \ fact \ n))$ and  $p \equiv (\lambda n. if n \le 1 then 1 else conv-num e-cfrac (n - 2))$ and  $q \equiv (\lambda n. if n = 0 then 1 else if n = 1 then 0 else conv-denom e-cfrac (n)$ -2))and  $a \equiv (\lambda n. \text{ if } n \mod 3 = 2 \text{ then } 2 * (Suc n \dim 3) \text{ else } 1)$ begin lemma assumes  $n \geq 2$ 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 have *n*-minus-3: n - 3 = n - Suc (Suc (Suc 0))

by (simp add: numeral-3-eq-3)

consider  $n = 2 \mid n = 3 \mid n \ge 4$ 

using assms by force

hence  $?th1 \land ?th2$ 

by cases (auto simp: p-def q-def cfrac-nth-e a-def conv-num-rec conv-denom-rec n-minus-3)

```
thus ?th1 ?th2 by blast+
qed
```

### lemma

assumes  $n \ge 1$ shows *p*-rec0: p(3 \* n) = p(3 \* n - 1) + p(3 \* n - 2)and *q*-rec $\theta$ : q(3 \* n) = q(3 \* n - 1) + q(3 \* n - 2)proof – define n' where n' = n - 1from assms have  $(3 * n' + 1) \mod 3 \neq 2$ by presburger also have (3 \* n' + 1) = 3 \* n - 2using assms by (simp add: n'-def) finally show p(3 \* n) = p(3 \* n - 1) + p(3 \* n - 2)q (3 \* n) = q (3 \* n - 1) + q (3 \* n - 2)using assms by (subst p-rec q-rec; simp add: a-def)+ qed

### lemma

assumes  $n \geq 1$ 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 define n' where n' = n - 1from assms have  $(3 * n' + 2) \mod 3 = 2$ by presburger

also have (3 \* n' + 2) = 3 \* n - 1using assms by (simp add: n'-def) finally show p(3 \* n + 1) = 2 \* int n \* p(3 \* n) + p(3 \* n - 1)q (3 \* n + 1) = 2 \* int n \* q (3 \* n) + q (3 \* n - 1)using assms by (subst p-rec q-rec; simp add: a-def)+  $\mathbf{qed}$ **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)**by** (subst p-rec q-rec; simp add: a-def nat-mult-distrib nat-add-distrib)+ lemma A-0: A  $0 = exp \ 1 - 1$  and B-0: B 0 = 1 and C-0: C  $0 = 2 - exp \ 1$ proof **have**  $(exp \ has-integral \ (exp \ 1 - exp \ 0)) \ \{0..1::real\}$ by (*intro fundamental-theorem-of-calculus*) (auto intro!: derivative-eq-intros simp flip: has-real-derivative-iff-has-vector-derivative) **thus**  $A \ 0 = exp \ 1 - 1$  by (simp add: A-def has-integral-iff) have  $((\lambda x. exp \ x \ * \ x) \ has-integral \ (exp \ 1 \ * \ (1 \ - \ 1) \ - \ exp \ 0 \ * \ (0 \ - \ 1)))$  $\{0..1::real\}$ **by** (*intro fundamental-theorem-of-calculus*) (auto intro!: derivative-eq-intros simp flip: has-real-derivative-iff-has-vector-derivative simp: algebra-simps) thus  $B \ 0 = 1$  by (simp add: B-def has-integral-iff) have  $((\lambda x. exp \ x * (x - 1)) has-integral (exp \ 1 * (1 - 2) - exp \ 0 * (0 - 2)))$  $\{0..1::real\}$ **by** (*intro fundamental-theorem-of-calculus*) (auto intro!: derivative-eq-intros simp flip: has-real-derivative-iff-has-vector-derivative simp: algebra-simps) thus  $C \ \theta = 2 - exp \ 1$  by (simp add: C-def has-integral-iff) qed **lemma** A-bound: norm  $(A \ n) \leq exp \ 1 \ / fact \ n$ proof have norm  $(exp \ t * t \ \hat{n} * (t - 1) \ \hat{n} \ / \ fact \ n) \le exp \ 1 * 1 \ \hat{n} * 1 \ \hat{n} \ / \ fact$ if  $t \in \{0...1\}$  for t :: real using that unfolding norm-mult norm-divide norm-power norm-fact by (intro mult-mono divide-right-mono power-mono) auto hence norm  $(A \ n) \leq exp \ 1 \ / fact \ n * (1 - 0)$ 

**unfolding** *A*-def **by** (*intro integral-bound*) (*auto intro*!: *continuous-intros*) **thus** ?*thesis* **by** *simp* 

qed

lemma B-bound: norm  $(B \ n) \le exp \ 1 \ / \ fact \ n$ proof – have norm  $(exp \ t * t \ Suc \ n * (t - 1) \ n \ / \ fact \ n) \le exp \ 1 * 1 \ Suc \ n * 1$  n / fact n

if  $t \in \{0..1\}$  for t :: real using that unfolding norm-mult norm-divide norm-power norm-fact

by (intro mult-mono divide-right-mono power-mono) auto hence norm  $(B n) \leq exp \ 1 \ / \ fact \ n * (1 - 0)$ unfolding B-def by (intro integral-bound) (auto introl: continuous-intros) thus ?thesis by simp qed lemma C-bound: norm  $(C n) \leq exp \ 1 \ / \ fact \ n$ proof have norm (exp  $t * t \ n * (t - 1) \ Suc \ n \ / \ fact \ n) \leq exp \ 1 * 1 \ n * 1 \ Suc \ n \ / \ fact \ n$ 

if  $t \in \{0..1\}$  for t :: real using that unfolding norm-mult norm-divide norm-power norm-fact

by (intro mult-mono divide-right-mono power-mono) auto hence norm  $(C n) \leq exp \ 1 \ / \ fact \ n * (1 - 0)$ 

**unfolding** *C-def* **by** (*intro integral-bound*) (*auto intro*!: *continuous-intros*) **thus** *?thesis* **by** *simp* 

### qed

lemma A-Suc: A (Suc n) = -B n - C nproof let  $?g = \lambda x$ .  $x \cap Suc \ n * (x - 1) \cap Suc \ n * exp \ x \ / fact \ (Suc \ n)$ have pos: fact n + real n \* fact n > 0 by (intro add-pos-nonneg) auto have A(Suc n) + B n + C n =integral  $\{0..1\}$  ( $\lambda x. exp \ x * x \ Suc \ n * (x - 1) \ Suc \ n \ / fact \ (Suc \ n) +$  $exp \ x * x \ Suc \ n * (x - 1) \ n \ fact \ n + exp \ x * x \ n * (x - 1)$ Suc n / fact n) unfolding A-def B-def C-def **apply** (subst integral-add [symmetric]) subgoal by (auto introl: integrable-continuous-real continuous-intros) subgoal by (auto introl: integrable-continuous-real continuous-intros) **apply** (*subst integral-add* [*symmetric*]) **apply** (*auto intro*!: *integrable-continuous-real continuous-intros*) done also have ... = integral  $\{0..1\}$  ( $\lambda x$ . exp x / fact (Suc n) \*  $(x \land Suc \ n \ast (x - 1) \land Suc \ n + Suc \ n \ast x \land Suc \ n \ast (x - 1) \land n$ +Suc  $n * x \land n * (x - 1) \land Suc n)$ (is - = integral - ?f)**apply** (simp add: divide-simps) **apply** (*simp add: field-simps*)? done also have  $(?f has-integral (?g 1 - ?g 0)) \{0...1\}$ **apply** (*intro fundamental-theorem-of-calculus*)

subgoal
```
by simp
   unfolding has-real-derivative-iff-has-vector-derivative [symmetric]
   apply (rule derivative-eq-intros refl | simp)+
   apply (simp add: algebra-simps)?
   done
 hence integral \{0..1\} ?f = 0
   by (simp add: has-integral-iff)
 finally show ?thesis by simp
qed
lemma B-Suc: B (Suc n) = -2 * Suc n * A (Suc n) + C n
proof -
 let ?g = \lambda x. x \cap Suc \ n * (x - 1) \cap (n+2) * exp \ x \ / fact \ (Suc \ n)
 have pos: fact n + real \ n * fact \ n > 0 by (intro add-pos-nonneg) auto
 have B(Suc n) + 2 * Suc n * A(Suc n) - C n =
        integral \{0..1\} (\lambda x. exp \ x * x^{(n+2)} * (x - 1)^{(n+1)} / fact (Suc \ n) +
2 * Suc n *
           exp \ x * x \ Suc \ n * (x - 1) \ Suc \ n \ fact \ (Suc \ n) - exp \ x * x \ n *
(x - 1) \cap Suc \ n \ / \ fact \ n)
   unfolding A-def B-def C-def integral-mult-right [symmetric]
   apply (subst integral-add [symmetric])
   subgoal
     by (auto introl: integrable-continuous-real continuous-intros)
   subgoal
     by (auto introl: integrable-continuous-real continuous-intros)
   apply (subst integral-diff [symmetric])
    apply (auto introl: integrable-continuous-real continuous-intros simp: mult-ac)
   done
 also have ... = integral \{0..1\} (\lambda x. exp x / fact (Suc n) *
                (x^{(n+2)} * (x-1)^{(n+1)} + 2 * Suc n * x^{Suc n} * (x-1)^{(n+1)}
Suc n -
                 Suc n * x \cap n * (x - 1) \cap Suc n)
   (is - = integral - ?f)
   apply (simp add: divide-simps)
   apply (simp add: field-simps)?
   done
 also have (?f has-integral (?g 1 - ?g 0)) \{0...1\}
   apply (intro fundamental-theorem-of-calculus)
    apply (simp; fail)
   unfolding has-real-derivative-iff-has-vector-derivative [symmetric]
   apply (rule derivative-eq-intros refl | simp)+
   apply (simp add: algebra-simps)?
   done
 hence integral \{0..1\}?f = 0
   by (simp add: has-integral-iff)
 finally show ?thesis by (simp add: algebra-simps)
ged
lemma C-Suc: C n = B n - A n
```

#### unfolding A-def B-def C-def

**by** (*subst integral-diff* [*symmetric*])

(auto intro!: integrable-continuous-real continuous-intros simp: field-simps)

**lemma** unfold-add-numeral: c \* n + numeral b = Suc (c \* n + pred-numeral b)by simp

```
lemma ABC:
 A \ n = q \ (3 \ * \ n) \ * \ exp \ 1 \ - \ p \ (3 \ * \ n) \ \land
  B n = p (Suc (3 * n)) - q (Suc (3 * n)) * exp 1 \land
  C n = p (Suc (Suc (3 * n))) - q (Suc (Suc (3 * n))) * exp 1
proof (induction n)
 case \theta
 thus ?case by (simp add: A-0 B-0 C-0 a-def p-def q-def cfrac-nth-e)
next
 case (Suc n)
 note [simp] =
  conjunct1 [OF Suc.IH] conjunct1 [OF conjunct2 [OF Suc.IH]] conjunct2 [OF con-
junct2[OF Suc.IH]]
 have [simp]: 3 + m = Suc (Suc (Suc m)) for m by simp
 have A': A (Suc n) = of-int (q (3 * Suc n)) * exp 1 - of-int (p (3 * Suc n))
   unfolding A-Suc
   by (subst p-rec0 q-rec0, simp)+ (auto simp: algebra-simps)
 have B': B(Suc n) = of-int (p(3 * Suc n + 1)) - of-int (q(3 * Suc n + 1))
* exp 1
   unfolding B-Suc
   by (subst p-rec1 q-rec1 p-rec0 q-rec0, simp)+ (auto simp: algebra-simps A-Suc)
 have C': C (Suc n) = of-int (p (3*Suc n+2)) - of-int (q (3*Suc n+2)) * exp
1
   unfolding A-Suc B-Suc C-Suc using p-rec2[of n] q-rec2[of n]
   by ((subst p-rec2 q-rec2)+, (subst p-rec0 q-rec0 p-rec1 q-rec1, simp)+)
     (auto simp: algebra-simps A-Suc B-Suc)
 from A' B' C' show ?case by simp
qed
lemma q-pos: q \ n > 0 if n \neq 1
 using that by (auto simp: q-def)
lemma conv-diff-exp-bound: norm (exp 1 - p n / q n) \leq exp 1 / fact (n div 3)
proof (cases n = 1)
 case False
 define n' where n' = n \ div \ 3
 consider n \mod 3 = 0 \mid n \mod 3 = 1 \mid n \mod 3 = 2
   by force
 hence diff [unfolded n'-def]: q \ n * exp \ 1 - p \ n =
   (if n \mod 3 = 0 then A n' else if n \mod 3 = 1 then -B n' else -C n')
 proof cases
   assume n \mod 3 = 0
```

hence 3 \* n' = n unfolding n'-def by presburger with ABC[of n'] show ?thesis by auto  $\mathbf{next}$ assume  $*: n \mod 3 = 1$ hence Suc (3 \* n') = n unfolding n'-def by presburger with \* ABC[of n'] show ?thesis by auto  $\mathbf{next}$ assume  $*: n \mod 3 = 2$ hence Suc (Suc (3 \* n')) = n unfolding n'-def by presburger with \* ABC[of n'] show ?thesis by auto qed **note** [[*linarith-split-limit* = 0]] have norm  $((q \ n * exp \ 1 - p \ n) / q \ n) \le exp \ 1 / fact (n \ div \ 3) / 1$  unfolding diff norm-divide using A-bound of n div 3 B-bound of n div 3 C-bound of n div 3 q-pos OF  $\langle n \neq 1 \rangle$ **by** (*subst frac-le*) (*auto simp: of-nat-ge-1-iff*) also have (q n \* exp 1 - p n) / q n = exp 1 - p n / q nusing q-pos $[OF \langle n \neq 1 \rangle]$  by (simp add: divide-simps) finally show ?thesis by simp **qed** (*auto simp*: *p*-*def q*-*def*) **theorem** *e-cfrac*: cfrac-lim *e-cfrac* = exp 1 proof have num: conv-num e-cfrac n = p (n + 2)and denom: conv-denom e-cfrac n = q (n + 2) for n **by** (*simp-all add*: *p-def q-def*) have  $(\lambda n. exp \ 1 - p \ n / q \ n) \longrightarrow 0$ **proof** (*rule Lim-null-comparison*) show eventually  $(\lambda n. norm (exp \ 1 - p \ n / q \ n) \le exp \ 1 / fact (n \ div \ 3))$ at-top using conv-diff-exp-bound by (intro always-eventually) auto show  $(\lambda n. exp \ 1 \ / fact \ (n \ div \ 3) :: real) \longrightarrow 0$ by (rule real-tendsto-divide-at-top tendsto-const filterlim-div-nat-at-top filterlim-ident filterlim-compose[OF fact-real-at-top])+ auto qed **moreover have** eventually  $(\lambda n. exp \ 1 - p \ n \ / q \ n = exp \ 1 - conv \ e-cfrac \ (n \ n \ n))$ (-2)) at-top using eventually-ge-at-top[of 2]  ${\bf proof}\ eventually\mbox{-}elim$ case  $(elim \ n)$ with num[of n - 2] denom[of n - 2] wf show ?case by (simp add: eval-nat-numeral Suc-diff-Suc conv-num-denom) qed ultimately have  $(\lambda n. exp \ 1 - conv \ e\text{-}cfrac \ (n - 2)) \longrightarrow 0$ using Lim-transform-eventually by fast hence  $(\lambda n. exp \ 1 - (exp \ 1 - conv \ e-cfrac \ (Suc \ (Suc \ n) - 2))) \longrightarrow exp \ 1 - exp \ 1$ 

 $\begin{array}{l} \mathbf{by} \ (subst\ filterlim-sequentially-Suc)+\ (intro\ tendsto-diff\ tendsto-const)\\ \mathbf{hence}\ conv\ e-cfrac \longrightarrow exp\ 1\ \mathbf{by}\ simp\\ \mathbf{moreover\ have}\ conv\ e-cfrac \longrightarrow cfrac-lim\ e-cfrac\\ \mathbf{by}\ (intro\ LIMSEQ-cfrac-lim\ wf)\ auto\\ \mathbf{ultimately\ have}\ exp\ 1\ =\ cfrac-lim\ e-cfrac\\ \mathbf{by}\ (rule\ LIMSEQ-unique)\\ \mathbf{thus}\ ?thesis\ ..\\ \mathbf{qed}\end{array}$ 

```
corollary e-cfrac-altdef: e-cfrac = cfrac-of-real (exp 1)
by (metis e-cfrac cfrac-infinite-iff cfrac-length-e cfrac-of-real-cfrac-lim-irrational)
```

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) \notin \mathbb{Q}
by (metis cfrac-length-e e-cfrac cfrac-infinite-iff)
```

```
corollary exp1-not-quadratic-irrational: ¬quadratic-irrational (exp 1 :: real)
proof -
 have range (\lambda n. \ 2 * (int \ n + 1)) \subseteq range (cfrac-nth \ e-cfrac)
 proof safe
   fix n :: nat
   have cfrac-nth e-cfrac (3*n+2) \in range (cfrac-nth e-cfrac)
    by blast
   also have (3 * n + 2) \mod 3 = 2
    by presburger
   hence cfrac-nth e-cfrac (3*n+2) = 2*(int n + 1)
     by (simp add: cfrac-nth-e)
   finally show 2 * (int n + 1) \in range (cfrac-nth e-cfrac).
 qed
 moreover have infinite (range (\lambda n. 2 * (int n + 1)))
   by (subst finite-image-iff) (auto intro!: injI)
 ultimately have infinite (range (cfrac-nth e-cfrac))
   using finite-subset by blast
 thus ?thesis using quadratic-irrational-cfrac-nth-range-finite[of e-cfrac]
   by (auto simp: e-cfrac)
qed
```

end end

0

# 4 Continued fraction expansions for square roots of naturals

theory Sqrt-Nat-Cfrac imports Quadratic-Irrationals HOL–Library.While-Combinator HOL–Library.IArray

### $\mathbf{begin}$

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

**lemma** butlast-nth [simp]:  $n < length xs - 1 \implies$  butlast xs ! n = xs ! nby (induction xs arbitrary: n) (auto simp: nth-Cons split: nat.splits)

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

 $\begin{array}{l} \textbf{definition } sqrt-nat-period-length :: nat \Rightarrow nat \textbf{ where} \\ sqrt-nat-period-length \ D = \\ (if is-square \ D \ then \ 0 \\ else \ (LEAST \ l. \ l > 0 \ \land \ (\forall \ n. \ cfrac-nth \ (cfrac-of-real \ (sqrt \ D)) \ (Suc \ n + l) = \\ cfrac-nth \ (cfrac-of-real \ (sqrt \ D)) \ (Suc \ n)))) \end{array}$ 

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 ::  $nat \Rightarrow nat \times nat \times nat$  list where sqrt-cfrac-info D = (sqrt-nat-period-length D, floor-sqrt D,  $map (\lambda n. nat (cfrac$ -nth (cfrac-of-real (sqrt D)) (Suc n))) [0..< sqrt-nat-period-length D])

**lemma** sqrt-nat-period-length-square [simp]: is-square  $D \Longrightarrow$  sqrt-nat-period-length D = 0

by (auto simp: sqrt-nat-period-length-def)

**definition** sqrt- $cfrac :: nat \Rightarrow cfrac$ where sqrt-cfrac D = cfrac-of-real (sqrt (real D))

**context fixes** D D' :: nat **defines**  $D' \equiv nat \lfloor sqrt D \rfloor$ **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*  $\Rightarrow$  *bool* **where** *red-assoc* =  $(\lambda(p, q))$ .

 $\begin{array}{l} q > 0 \ \land \ q \ dvd \ (D - p^2) \ \land \ (sqrt \ D + p) \ / \ q > 1 \ \land \ (-sqrt \ D + p) \ / \ q \in \{-1 < .. < 0\}) \end{array}$ 

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  $\times$  nat  $\Rightarrow$  real where surd-to-real =  $(\lambda(p, q). (sqrt D + p) / q)$ 

**definition** surd-to-real-cnj :: nat  $\times$  nat  $\Rightarrow$  real where surd-to-real-cnj = ( $\lambda(p, q)$ . (-sqrt D + p) / q)

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

**definition** sqrt-remainder-step :: nat  $\times$  nat  $\Rightarrow$  nat  $\times$  nat where sqrt-remainder-step =  $(\lambda(p, q). let X = (p + D') div q; p' = X * q - p in (p', (D - p'^2) div q))$ 

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  $\Rightarrow$  nat  $\times$  nat where sqrt-remainder-surd =  $(\lambda n. (sqrt-remainder-step \frown n) (D', D - D'^2))$ 

 $\operatorname{context}$ 

fixes sqrt-cfrac-nth ::  $nat \Rightarrow nat$  and lassumes nonsquare:  $\neg is$ -square Ddefines sqrt-cfrac- $nth \equiv (\lambda n. case sqrt$ -remainder- $surd n of (p, q) \Rightarrow (D' + p)$ div q)defines  $l \equiv sqrt$ -nat-period-length Dbegin

lemma D'-pos: D' > 0unfolding D'-def using nonsquare of-nat-ge-1-iff by force

lemma D'-sqr-less-D:  $D'^2 < D$ proof – have  $D' \leq sqrt D$  by (auto simp: D'-def) hence real  $D' \uparrow 2 \leq sqrt D \uparrow 2$  by (intro power-mono) auto also have ... = D by simp finally have  $D'^2 \leq D$  by simp moreover from nonsquare have  $D \neq D'^2$  by auto ultimately show ?thesis by simp qed

```
\begin{array}{l} \textbf{lemma red-assoc-imp-irrat:}\\ \textbf{assumes red-assoc pq}\\ \textbf{shows surd-to-real pq} \notin \mathbb{Q}\\ \textbf{proof}\\ \textbf{assume rat: surd-to-real pq} \in \mathbb{Q}\\ \textbf{with assms rat show False using irrat-sqrt-nonsquare}[OF nonsquare]\\ \textbf{by (auto simp: field-simps red-assoc-def surd-to-real-def divide-in-Rats-iff2 add-in-Rats-iff1)}\\ \textbf{qed} \end{array}
```

**lemma** *surd-to-real-cnj-irrat*: **assumes** *red-assoc pq* 

surd-to-real-cnj  $pq \notin \mathbb{Q}$ shows proof assume rat: surd-to-real-cnj  $pq \in \mathbb{Q}$ with assms rat show False using irrat-sqrt-nonsquare[OF nonsquare] by (auto simp: field-simps red-assoc-def surd-to-real-cnj-def divide-in-Rats-iff2 diff-in-Rats-iff1) qed **lemma** surd-to-real-nonneg [intro]: surd-to-real  $pq \geq 0$ by (auto simp: surd-to-real-def case-prod-unfold divide-simps introl: divide-nonneg-nonneg) **lemma** surd-to-real-pos [intro]: red-assoc  $pq \implies$  surd-to-real pq > 0by (auto simp: surd-to-real-def case-prod-unfold divide-simps red-assoc-def *intro*!: *divide-nonneg-nonneg*) **lemma** surd-to-real-nz [simp]: red-assoc  $pq \implies$  surd-to-real  $pq \neq 0$ by (auto simp: surd-to-real-def case-prod-unfold divide-simps red-assoc-def intro!: divide-nonneg-nonneg) **lemma** surd-to-real-cnj-nz [simp]: red-assoc  $pq \implies$  surd-to-real-cnj  $pq \neq 0$ using surd-to-real-cnj-irrat[of pq] by auto **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-to-real pq' = 1 / frac (surd-to-real pq) surd-to-real-cnj pq' = 1 / (surd-to-real-cnj pq - X)  $X > 0 X * snd pq \leq 2 * D' X = nat | surd-to-real pq |$  $X = nat \mid -1 \mid surd-to-real-cnj pq' \mid$ proof **obtain** p q where [simp]: pq = (p, q) by (cases pq) obtain p' q' where [simp]: pq' = (p', q') by (cases pq') define  $\alpha$  where  $\alpha = (sqrt D + p) / q$ define  $\alpha'$  where  $\alpha' = 1 / frac \alpha$  $(int \ p)^2) \ div \ q)$ from assms(1) have  $\alpha > 0$  q > 0by (auto simp:  $\alpha$ -def red-assoc-def) from assms(1) nonsquare have  $\alpha \notin \mathbb{Q}$ by (auto simp:  $\alpha$ -def red-assoc-def divide-in-Rats-iff2 add-in-Rats-iff2 irrat-sqrt-nonsquare) hence  $\alpha'$ -pos: frac  $\alpha > 0$  using Ints-subset-Rats by auto from  $\langle pq' = (p', q') \rangle$  have p'-def: p' = X \* q - p and q'-def:  $q' = (D - p'^2)$ div qunfolding pq'-def sqrt-remainder-step-def X-def by (auto simp: Let-def add-ac) have D' + p = |sqrt D + p|by (auto simp: D'-def)

also have ... div int q = |(sqrt D + p) / q|**by** (subst floor-divide-real-eq-div [symmetric]) auto finally have X-altdef: X = nat |(sqrt D + p) / q|**unfolding** X-def zdiv-int [symmetric] by auto have nz: sqrt (real D) +  $(X * q - real p) \neq 0$ proof assume sqrt (real D) + (X \* q - real p) = 0hence sqrt (real D) = real p - X \* q by (simp add: algebra-simps) also have  $\ldots \in \mathbb{Q}$  by *auto* finally show False using irrat-sqrt-nonsquare nonsquare by blast qed from assms(1) have real  $(p \ 2) \leq sqrt \ D \ 2$ unfolding of-nat-power by (intro power-mono) (auto simp: red-assoc-def field-simps) also have  $sart D \ 2 = D$  by simpfinally have  $p^2 \le D$  by (subst (asm) of-nat-le-iff)have frac  $\alpha = \alpha - X$ by (simp add: X-altdef frac-def  $\alpha$ -def) also have  $\ldots = (sqrt D - (X * q - int p)) / q$ using  $\langle q > 0 \rangle$  by (simp add: field-simps  $\alpha$ -def) finally have 1 / frac  $\alpha = q$  / (sqrt D - (X \* q - int p)) by simp also have  $\ldots = q * (sqrt D + (X * q - int p)) /$ ((sqrt D - (X \* q - int p)) \* (sqrt D + (X \* q - int p))) (is -= (A / B)using nz by (subst mult-divide-mult-cancel-right) auto also have  $?B = real-of-int (D - int p \ 2 + 2 * X * p * q - int X \ 2 * q \)$ 2)**by** (*auto simp: algebra-simps power2-eq-square*) also have  $q \ dvd \ (D - p \ \widehat{\ } 2)$  using assms(1) by (auto simp: red-assoc-def) with  $\langle p^2 \leq D \rangle$  have int q dvd (int D - int  $p \uparrow 2$ ) **by** (*metis of-nat-diff of-nat-dvd-iff of-nat-power*) hence  $D - int p \ 2 + 2 * X * p * q - int X \ 2 * q \ 2 = q * ((D - (X * q)))$  $-int p)^2$  div q) **by** (*auto simp: power2-eq-square algebra-simps*) also have  $?A / \ldots = (sqrt D + (X * q - int p)) / ((D - (X * q - int p)^2) div$ q)**unfolding** *of-int-mult of-int-of-nat-eq* by (rule mult-divide-mult-cancel-left) (insert  $\langle q > 0 \rangle$ , auto) finally have  $\alpha': \alpha' = \dots$  by (simp add:  $\alpha'$ -def) have dvd: q dvd  $(D - (X * q - int p)^2)$ using  $assms(1) \langle int q dvd (int D - int p \hat{2}) \rangle$ **by** (*auto simp: power2-eq-square algebra-simps*) have  $X \leq (sqrt D + p) / q$  unfolding X-altdef by simp

moreover have  $X \neq (sqrt D + p) / q$  uniforming X-acceptible similar moreover have  $X \neq (sqrt D + p) / q$ 

#### proof

assume X = (sqrt D + p) / qhence sqrt D = q \* X - real p using  $\langle q > 0 \rangle$  by (auto simp: field-simps) also have  $\ldots \in \mathbb{Q}$  by *auto* finally show False using irrat-sqrt-nonsquare[OF nonsquare] by simp qed ultimately have X < (sqrt D + p) / q by simp hence \*: (X \* q - int p) < sqrt D**using**  $\langle q > 0 \rangle$  **by** (simp add: field-simps) moreover have pos: real-of-int (int  $D - (int X * int q - int p)^2) > 0$ **proof** (cases  $X * q \ge p$ ) case True hence real  $p \leq real X * real q$  unfolding of-nat-mult [symmetric] of-nat-le-iff hence real-of-int  $((X * q - int p) \hat{2}) < sqrt D \hat{2}$  using \* **unfolding** of-int-power by (intro power-strict-mono) auto also have  $\ldots = D$  by simpfinally show ?thesis by simp  $\mathbf{next}$ case False hence less: real X \* real q < real punfolding of-nat-mult [symmetric] of-nat-less-iff by auto have  $(real X * real q - real p)^2 = (real p - real X * real q)^2$ **by** (*simp add: power2-eq-square algebra-simps*) also have  $\ldots \leq real \ p \ 2$  using less by (intro power-mono) auto also have  $\ldots < sqrt D \uparrow 2$ using  $\langle q > 0 \rangle$  assms(1) unfolding of-int-power by (intro power-strict-mono) (auto simp: red-assoc-def field-simps) also have  $\ldots = D$  by simpfinally show ?thesis by simp qed hence pos': int  $D - (int X * int q - int p)^2 > 0$ **by** (*subst* (*asm*) *of-int-0-less-iff*) from pos have real-of-int ((int  $D - (int X * int q - int p)^2)$  div q) > 0 using  $\langle q > 0 \rangle$  dvd by (subst real-of-int-div) (auto introl: divide-pos-pos) ultimately have cnj-neg:  $cnj-\alpha' < 0$  unfolding  $cnj-\alpha'$ -def using dvd unfolding of-int-0-less-iff by (intro divide-neg-pos) auto have (p - sqrt D) / q < 0using assms(1) by (auto simp: red-assoc-def X-altdef le-nat-iff) also have  $X \ge 1$ using assms(1) by (auto simp: red-assoc-def X-altdef le-nat-iff) hence  $0 \leq real X - 1$  by simp finally have q < sqrt D + int q \* X - pusing  $\langle q > 0 \rangle$  by (simp add: field-simps) hence q \* (sqrt D - (int q \* X - p)) < (sqrt D + (int q \* X - p)) \* (sqrt D-(int q \* X - p))using \* by (intro mult-strict-right-mono) (auto simp: red-assoc-def X-altdef *field-simps*) also have  $\ldots = D - (int \ q * X - p) \widehat{\phantom{a}} 2$ **by** (*simp add: power2-eq-square algebra-simps*) finally have  $cnj-\alpha' > -1$ using dvd pos  $\langle q > 0 \rangle$  by (simp add: real-of-int-div field-simps cnj- $\alpha'$ -def) from *cnj*-neg and this have cnj- $\alpha' \in \{-1 < .. < 0\}$  by *auto* have  $\alpha' > 1$  using  $\langle frac \ \alpha > 0 \rangle$ by (auto simp:  $\alpha'$ -def field-simps frac-lt-1) have 0 = 1 + (-1 :: real)by simp also have  $1 + -1 < \alpha' + cnj - \alpha'$ using  $\langle cnj \cdot \alpha' \rangle -1 \rangle$  and  $\langle \alpha' \rangle 1 \rangle$  by (intro add-strict-mono) also have  $\alpha' + cnj \cdot \alpha' = 2 * (real X * q - real p) / ((int D - (int X * q - int x)))$  $(p)^2$ ) div int q) by (simp add:  $\alpha'$  cnj- $\alpha'$ -def add-divide-distrib [symmetric]) finally have real X \* q - real p > 0 using pos dvd  $\langle q > 0 \rangle$ by (subst (asm) zero-less-divide-iff, subst (asm) (1 2 3) real-of-int-div) (auto simp: field-simps) hence real (X \* q) > real p unfolding of-nat-mult by simp hence *p*-less-Xq: p < X \* q by (simp only: of-nat-less-iff) from pos' and p-less-Xq have int  $D > int ((X * q - p)^2)$ **by** (subst of-nat-power) (auto simp: of-nat-diff) hence  $pos'': D > (X * q - p)^2$  unfolding of-nat-less-iff. from dvd have int q dvd int  $(D - (X * q - p)^2)$ using *p*-less-Xq pos'' by (subst of-nat-diff) (auto simp: of-nat-diff) with dvd have dvd': q dvd  $(D - (X * q - p)^2)$ by simp have  $\alpha'$ -altdef:  $\alpha' = (sqrt D + p') / q'$ using dvd dvd' pos'' p-less-Xq  $\alpha'$ by (simp add: real-of-int-div p'-def q'-def real-of-nat-div mult-ac of-nat-diff) have cnj- $\alpha'$ -altdef: cnj- $\alpha' = (-sqrt D + p') / q'$ using dvd dvd' pos'' p-less-Xq unfolding cnj- $\alpha'$ -def by (simp add: real-of-int-div p'-def q'-def real-of-nat-div mult-ac of-nat-diff) from dvd' have dvd'':  $q' dvd (D - p'^2)$ by (auto simp: mult-ac p'-def q'-def) have real  $((D - p'^2) div q) > 0$  unfolding p'-def by (subst real-of-nat-div[OF dvd'], rule divide-pos-pos) (insert  $\langle q > 0 \rangle$  pos'', auto) hence q' > 0 unfolding q'-def of-nat-0-less-iff. show red-assoc pq' using  $\langle \alpha' > 1 \rangle$  and  $\langle cnj \cdot \alpha' \in \cdot \rangle$  and dvd'' and  $\langle q' > 0 \rangle$ **by** (auto simp: red-assoc-def  $\alpha'$ -altdef cnj- $\alpha'$ -altdef)

from assms(1) have real p < sqrt D

by (auto simp add: field-simps red-assoc-def) hence  $p \leq D'$  unfolding D'-def by linarith with \* have real (X \* q) < sqrt (real D) + D'by simp thus  $X * snd pq \leq 2 * D'$  unfolding D'-def  $\langle pq = (p, q) \rangle$  snd-conv by linarith have  $(sqrt D + p') / q' = \alpha'$ by (rule  $\alpha'$ -altdef [symmetric]) also have  $\alpha' = 1 / frac ((sqrt D + p) / q)$ by (simp add:  $\alpha'$ -def  $\alpha$ -def) finally show surd-to-real pq' = 1 / frac (surd-to-real pq) by (simp add: surd-to-real-def) from  $\langle X \geq 1 \rangle$  show X > 0 by simp from X-altdef show X = nat | surd-to-real pq | by (simp add: surd-to-real-def) have sqrt (real D) < real p + 1 \* real qusing assms(1) by (auto simp: red-assoc-def field-simps) also have  $\ldots \leq real \ p + real \ X * real \ q$ using  $\langle X > 0 \rangle$  by (intro add-left-mono mult-right-mono) (auto simp: of-nat-ge-1-iff) finally have sqrt (real D) < .... have real p < sqrt Dusing assms(1) by (auto simp add: field-simps red-assoc-def) also have  $\ldots \leq sqrt D + q * X$ by linarith finally have less: real p < sqrt D + X \* q by (simp add: algebra-simps) moreover have D + p \* p' + X \* q \* sqrt D = q \* q' + p \* sqrt D + p' \* sqrtD + X \* p' \* qusing dvd' pos'' p-less- $Xq \langle q > 0 \rangle$  unfolding p'-def q'-def of-nat-mult of-nat-add **by** (*simp add: power2-eq-square field-simps of-nat-diff real-of-nat-div*) **ultimately show** \*: surd-to-real-cnj pq' = 1 / (surd-to-real-cnj pq - X) using  $\langle q > 0 \rangle \langle q' > 0 \rangle$  by (auto simp: surd-to-real-cnj-def field-simps) have \*\*:  $a = nat \lfloor y \rfloor$  if  $x \ge 0$  x < 1 real a + x = y for a :: nat and x y :: realusing that by linarith from assms(1) have surd-to-real-cnj: surd-to-real-cnj  $(p, q) \in \{-1 < ... < 0\}$ **by** (*auto simp: surd-to-real-cnj-def red-assoc-def*) have surd-to-real-cnj (p, q) < Xusing assms(1) less by (auto simp: surd-to-real-cnj-def field-simps red-assoc-def) hence real X = surd-to-real-cnj (p, q) - 1 / surd-to-real-cnj (p', q') using \* using surd-to-real-cnj-irrat assms(1) (red-assoc pq') by (auto simp: field-simps) thus X = nat |-1| / surd-to-real-cnj pq'| using surd-to-real-cnj by (intro \*\*[of -surd-to-real-cnj (p, q)]) auto qed lemma red-assoc-denom-2D: assumes red-assoc (p, q)defines  $X \equiv (D' + p) div q$ 

defines  $X \equiv (D' + p) di$ assumes X > D'shows q = 1 proof – have  $X * q \leq 2 * D' X > 0$ using red-assoc-step(4,5) [OF assms(1)] by (simp-all add: X-def) note this(1)also have 2 \* D' < 2 \* Xby (intro mult-strict-left-mono assms) auto finally have q < 2 using  $\langle X > 0 \rangle$  by simp moreover from assms(1) have q > 0 by (auto simp: red-assoc-def) ultimately show ?thesis by simp qed **lemma** red-assoc-denom-1: assumes red-assoc (p, 1)shows p = D'proof from assms have sqrt D > p sqrt D < real p + 1**by** (*auto simp: red-assoc-def*) thus p = D' unfolding D'-def by *linarith* qed lemma red-assoc-begin: red-assoc  $(D', D - D'^2)$ surd-to-real  $(D', D - D'^2) = 1 / frac (sqrt D)$ surd-to-real-cnj  $(D', D - D'^2) = -1 / (sqrt D + D')$ proof – have pos:  $D > \theta D' > \theta$ using D'-def D'-pos is-nth-power-0-iff by force+ have sqrt  $D \neq D'$ using *irrat-sqrt-nonsquare*[OF nonsquare] by *auto* moreover have sqrt D > 0 by simp hence  $D' \leq sqrt D$  unfolding D'-def by linarith ultimately have less: D' < sqrt D by simp have sqrt  $D \neq D' + 1$ using *irrat-sqrt-nonsquare*[OF nonsquare] by *auto* moreover have sqrt D > 0 by simp hence  $D' \ge sqrt D - 1$  unfolding D'-def by linarith ultimately have gt: D' > sqrt D - 1 by simpfrom less have real  $D' \ 2 < sqrt D \ 2$  by (intro power-strict-mono) auto also have  $\ldots = D$  by simpfinally have less':  $D^{\prime 2} < D$  unfolding of-nat-power [symmetric] of-nat-less-iff

**moreover have** real D' \* (real D' - 1) < sqrt D \* (sqrt D - 1) **using** less pos **by** (intro mult-strict-mono diff-strict-right-mono) (auto simp: of-nat-ge-1-iff) hence  $D'^2 + sqrt D < D' + D$ 

**by** (simp add: field-simps power2-eq-square)

**moreover have** (sqrt D - 1) \* sqrt D < real D' \* (real D' + 1)using pos gt by (intro mult-strict-mono) auto

hence  $D < sqrt D + D'^2 + D'$  by (simp add: power2-eq-square field-simps)

ultimately show red-assoc  $(D', D - D'^2)$ 

**by** (*auto simp: red-assoc-def field-simps of-nat-diff less*)

have frac: frac (sqrt D) = sqrt D - D' unfolding frac-def D'-def by auto

**show** surd-to-real  $(D', D - D'^2) = 1$  / frac (sqrt D) **unfolding** surd-to-real-def **using** less less' pos **by** (subst frac) (auto simp: of-nat-diff power2-eq-square field-simps)

have surd-to-real-cnj  $(D', D - D'^2) = -((sqrt D - D') / (D - D'^2))$ using less less' pos by (auto simp: surd-to-real-cnj-def field-simps) also have real  $(D - D'^2) = (sqrt D - D') * (sqrt D + D')$ using less' by (simp add: power2-eq-square algebra-simps of-nat-diff) also have  $(sqrt D - D') / \ldots = 1 / (sqrt D + D')$ using less by (subst nonzero-divide-mult-cancel-left) auto finally show surd-to-real-cnj  $(D', D - D'^2) = -1 / (sqrt D + D')$  by simp qed

**lemma** cfrac-remainder-surd-to-real:

**assumes** red-assoc pq

**shows** cfrac-remainder (cfrac-of-real (surd-to-real pq)) n =surd-to-real ((sqrt-remainder-step  $\frown n$ ) pq) using assms(1)**proof** (*induction n arbitrary: pq*) case  $\theta$ hence cfrac-lim (cfrac-of-real (surd-to-real pq)) = surd-to-real pq**by** (*intro cfrac-lim-of-real red-assoc-imp-irrat* 0) thus ?case using  $\theta$ by auto  $\mathbf{next}$ case (Suc n) **obtain**  $p \ q$  where [simp]: pq = (p, q) by (cases pq) **have** surd-to-real ((sqrt-remainder-step  $\frown$  Suc n) pq) = surd-to-real ((sqrt-remainder-step  $\frown$  n) (sqrt-remainder-step (p, q))) **by** (subst funpow-Suc-right) auto also have  $\ldots = cfrac$ -remainder (cfrac-of-real (surd-to-real (sqrt-remainder-step (p, q)))) nusing red-assoc-step(1) [of (p, q)] Suc.prems by (intro Suc.IH [symmetric]) (auto simp: sqrt-remainder-step-def Let-def add-ac) also have surd-to-real (sqrt-remainder-step (p, q)) = 1 / frac (surd-to-real (p, q)) q))using red-assoc-step(2)[of (p, q)] Suc.prems

by (auto simp: sqrt-remainder-step-def Let-def add-ac surd-to-real-def)

also have  $cfrac-of-real \ldots = cfrac-tl (cfrac-of-real (surd-to-real (p, q)))$ 

 ${\bf using} \ Suc.prems \ Ints-subset-Rats \ red-assoc-imp-irrat \ {\bf by} \ (subst \ cfrac-tl-of-real) \\ auto$ 

also have cfrac-remainder ... n = cfrac-remainder (cfrac-of-real (surd-to-real (p, q))) (Suc n)

**by** (*simp add: cfrac-drop-Suc-right cfrac-remainder-def*)

finally show ?case by simp

 $\mathbf{qed}$ 

**lemma** red-assoc-step' [intro]: red-assoc  $pq \implies$  red-assoc (sqrt-remainder-step pq) using red-assoc-step(1)[of pq]

by (simp add: sqrt-remainder-step-def case-prod-unfold add-ac Let-def)

```
lemma red-assoc-steps [intro]: red-assoc pq \implies red-assoc ((sqrt-remainder-step \frown n) pq)
```

```
by (induction n) auto
```

```
lemma floor-sqrt-less-sqrt: D' < sqrt D
proof -
```

```
have D' \leq sqrt D unfolding D'-def by auto
moreover have sqrt D \neq D'
using irrat-sqrt-nonsquare[OF nonsquare] by auto
ultimately show ?thesis by auto
```

```
qed
```

```
lemma red-assoc-bounds:
 assumes red-assoc pq
 shows pq \in (SIGMA \ p:\{0 < ... D'\}. \{Suc \ D' - p... D' + p\})
proof -
 obtain p q where [simp]: pq = (p, q) by (cases pq)
 from assms have *: p < sqrt D
   by (auto simp: red-assoc-def field-simps)
 hence p: p \leq D' unfolding D'-def by linarith
 from assms have p > 0 by (auto introl: Nat.gr0I simp: red-assoc-def)
 have q > sqrt D - p q < sqrt D + p
   using assms by (auto simp: red-assoc-def field-simps)
 hence q \ge D' + 1 - p \ q \le D' + p
   unfolding D'-def by linarith+
 with p \langle p > 0 \rangle show ?thesis by simp
\mathbf{qed}
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' \leftrightarrow pq = pq'
proof
 assume eq: surd-to-real-cnj pq = surd-to-real-cnj pq'
 from assms have pos: snd pq > 0 snd pq' > 0 by (auto simp: red-assoc-def)
 have snd pq = snd pq'
```

**proof** (rule ccontr) assume snd  $pq \neq snd pq'$ with eq have sqrt D = (real (fst pq' \* snd pq) - fst pq \* snd pq') / (real (sndpq) - snd pq') using pos by (auto simp: field-simps surd-to-real-cnj-def case-prod-unfold) also have  $\ldots \in \mathbb{Q}$  by *auto* finally show False using irrat-sqrt-nonsquare [OF nonsquare] by auto qed **moreover from** this eq pos have  $fst \ pq = fst \ pq'$ **by** (*auto simp: surd-to-real-cnj-def case-prod-unfold*) **ultimately show** pq = pq' by (simp add: prod-eq-iff) qed auto **lemma** red-assoc-sqrt-remainder-surd [intro]: red-assoc (sqrt-remainder-surd n) **by** (*auto simp: sqrt-remainder-surd-def intro*!: *red-assoc-begin*) **lemma** *surd-to-real-sqrt-remainder-surd*: surd-to-real (sqrt-remainder-surd n) = cfrac-remainder (cfrac-of-real (sqrt D))  $(Suc \ n)$ **proof** (*induction* n) case  $\theta$ from nonsquare have D > 0 by (auto intro!: Nat.gr0I) with red-assoc-begin show ?case using nonsquare irrat-sqrt-nonsquare [OF nonsquare] using Ints-subset-Rats cfrac-drop-Suc-right cfrac-remainder-def cfrac-tl-of-real sqrt-remainder-surd-def by fastforce next case (Suc n) **have** surd-to-real (sqrt-remainder-surd (Suc n)) = surd-to-real (sqrt-remainder-step (sqrt-remainder-surd n))**by** (simp add: sqrt-remainder-surd-def) also have  $\ldots = 1 / frac (surd-to-real (sqrt-remainder-surd n))$ using red-assoc-step[OF red-assoc-sqrt-remainder-surd[of n]] by simp also have surd-to-real (sqrt-remainder-surd n) = cfrac-remainder (cfrac-of-real (sqrt D)) (Suc n) (is - = ?X) by (rule Suc.IH) **also have** |cfrac-remainder (cfrac-of-real (sqrt (real D))) (Suc n)| =cfrac-nth (cfrac-of-real (sqrt (real D))) (Suc n) using irrat-sqrt-nonsquare[OF nonsquare] by (intro floor-cfrac-remainder) auto hence 1 / frac ?X = cfrac-remainder (cfrac-of-real (sqrt D)) (Suc (Suc n)) using *irrat-sqrt-nonsquare*[OF nonsquare] **by** (subst cfrac-remainder-Suc[of Suc n]) (simp-all add: frac-def cfrac-length-of-real-irrational) finally show ?case . qed **lemma** sqrt-cfrac: sqrt-cfrac-nth n = cfrac-nth (cfrac-of-real (sqrt D)) (Suc n)proof –

have cfrac-nth (cfrac-of-real (sqrt D)) (Suc n) =

 $\lfloor cfrac-remainder \ (cfrac-of-real \ (sqrt \ D)) \ (Suc \ n) \rfloor \\ \textbf{using } irrat-sqrt-nonsquare[OF \ nonsquare] \ \textbf{by } (subst \ floor-cfrac-remainder) \ auto \\ \textbf{also have } cfrac-remainder \ (cfrac-of-real \ (sqrt \ D)) \ (Suc \ n) = surd-to-real \ (sqrt-remainder-surd \ n) \\ \textbf{by } (rule \ surd-to-real-sqrt-remainder-surd \ [symmetric]) \\ \textbf{also have } nat \ \lfloor surd-to-real \ (sqrt-remainder-surd \ n) \rfloor = sqrt-cfrac-nth \ n \\ \textbf{unfolding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-nth-def \ \textbf{using } red-assoc-step(6)[OF \ red-assoc-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-sqrt-remainder-surd[of \ n] \\ \textbf{bischeding } sqrt-cfrac-sqrt-remainder-sqrt$ 

n]]

by (simp add: case-prod-unfold)
finally show ?thesis
by (simp add: nat-eq-iff)
qed

```
lemma sqrt-cfrac-pos: sqrt-cfrac-nth k > 0

using red-assoc-step(4)[OF red-assoc-sqrt-remainder-surd[of k]]

by (simp add: sqrt-cfrac-nth-def case-prod-unfold)
```

**lemma** snd-sqrt-remainder-surd-pos: snd (sqrt-remainder-surd n) > 0 using red-assoc-sqrt-remainder-surd[of n] by (auto simp: red-assoc-def)

#### lemma

**shows** *period-nonempty*: l > 0and period-length-le-aux:  $l \leq D' * (D' + 1)$ and sqrt-remainder-surd-periodic:  $\bigwedge n$ . sqrt-remainder-surd n = sqrt-remainder-surd  $(n \mod l)$ and sqrt-cfrac-periodic:  $\Lambda n$ . sqrt-cfrac-nth n =sqrt-cfrac-nth  $(n \mod l)$ and sqrt-remainder-surd-smallest-period:  $\wedge n. n \in \{0 < ... < l\} \implies$  sqrt-remainder-surd  $n \neq$  sqrt-remainder-surd 0 and snd-sqrt-remainder-surd-gt-1:  $\bigwedge n. n < l - 1 \implies snd (sqrt-remainder-surd)$ n) > 1and *sqrt-cfrac-le*:  $\bigwedge n. \ n < l - 1 \implies sqrt-cfrac-nth \ n \leq D'$ 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:  $\bigwedge n. n < l - 1 \implies \text{sqrt-cfrac-nth} (l - n - 2) =$ sqrt-cfrac-nth nand *sqrt-cfrac-smallest-period*:  $\bigwedge l'. l' > 0 \Longrightarrow (\bigwedge k. \ sqrt-cfrac-nth \ (k + l') = sqrt-cfrac-nth \ k) \Longrightarrow l' \ge l$ proof **note** [simp] = sqrt-remainder-surd-def define f where f = sqrt-remainder-surd have \*[intro]: red-assoc (f n) for n **unfolding** *f-def* **by** (*rule red-assoc-sqrt-remainder-surd*) define S where  $S = (SIGMA \ p:\{0 < ... D'\}, \{Suc \ D' - p... D' + p\})$ have [intro]: finite S by (simp add: S-def) have card  $S = (\sum p=1..D'. \ 2 * p)$  unfolding S-def **by** (*subst card-SigmaI*) (*auto intro*!: *sum.cong*) also have ... = D' \* (D' + 1)

by (induction D') (auto simp: power2-eq-square) finally have [simp]: card S = D' \* (D' + 1). have  $D' * (D' + 1) + 1 = card \{...D' * (D' + 1)\}$  by simp define k1 where  $k1 = (LEAST \ k1. \ k1 \le D' * (D' + 1) \land (\exists k2. \ k2 \le D' * (D' + 1) \land k1 \ne D' )$  $k2 \wedge f k1 = f k2)$ define k2 where  $k2 = (LEAST \ k2. \ k2 \le D' * (D' + 1) \land k1 \ne k2 \land f \ k1 = f \ k2)$ have  $f \in \{..D' * (D' + 1)\} \subseteq S$  unfolding S-def using red-assoc-bounds[OF \*] by blast hence card  $(f ` \{..D' * (D' + 1)\}) \leq card S$ by (intro card-mono) auto also have card S = D' \* (D' + 1) by simp also have ... < card {..D' \* (D' + 1)} by simp finally have  $\neg inj$ -on  $f \{..D' * (D' + 1)\}$ **by** (*rule piqeonhole*) hence  $\exists k1. k1 \leq D' * (D' + 1) \land (\exists k2. k2 \leq D' * (D' + 1) \land k1 \neq k2 \land f k1$ = f k2by (auto simp: inj-on-def) **from** LeastI-ex[OF this, folded k1-def] have  $k1 \leq D' * (D' + 1) \exists k2 \leq D' * (D' + 1)$ .  $k1 \neq k2 \land fk1 = fk2$  by auto **moreover from** LeastI-ex[OF this(2), folded k2-def]have  $k^{2} \leq D' * (D' + 1) k^{2} \neq k^{2} f k^{2} = f k^{2}$  by *auto* moreover have  $k1 \leq k2$ **proof** (*rule ccontr*) assume  $\neg (k1 < k2)$ hence  $k2 \leq D' * (D' + 1) \land (\exists k2', k2' \leq D' * (D' + 1) \land k2 \neq k2' \land f k2$ = f k 2'using  $\langle k1 \leq D' * (D' + 1) \rangle$  and  $\langle k1 \neq k2 \rangle$  and  $\langle fk1 = fk2 \rangle$  by *auto* hence  $k1 \leq k2$  unfolding k1-def by (rule Least-le) with  $\langle \neg (k1 \leq k2) \rangle$  show False by simp qed ultimately have  $k12: k1 < k2 \ k2 \le D' * (D' + 1) \ f \ k1 = f \ k2$  by *auto* have [simp]: k1 = 0**proof** (cases k1) case (Suc k1') define k2' where k2' = k2 - 1have Suc':  $k2 = Suc \ k2'$  using k12 by (simp add: k2'-def) have nz: surd-to-real-cnj (sqrt-remainder-step  $(f k1')) \neq 0$ surd-to-real-cnj (sqrt-remainder-step  $(f \ k2')) \neq 0$ using surd-to-real-cnj-nz[OF \* [of k2]] surd-to-real-cnj-nz[OF \* [of k1]] **by** (*simp-all add: f-def Suc Suc'*) define a where a = (D' + fst (f k1)) div snd (f k1)define a' where a' = (D' + fst (f k1')) div snd (f k1')

define a'' where a'' = (D' + fst (f k2')) div snd (f k2')

have  $a' = nat \lfloor -1 / surd-to-real-cnj (sqrt-remainder-step (f k1')) \rfloor$ using red-assoc-step[OF \* [of k1']] by (simp add: a'-def) **also have** sqrt-remainder-step (f k1') = f k1by (simp add: Suc f-def) also have f k1 = f k2 by fact also have f k2 = sqrt-remainder-step (f k2') by  $(simp \ add: \ Suc' \ f$ -def) also have nat |-1| surd-to-real-cnj (sqrt-remainder-step (f k2'))| = a'' using red-assoc-step[OF \*[of k2']] by (simp add: a''-def) finally have  $a' \cdot a''$ : a' = a''. have surd-to-real-cnj (f k2')  $\neq a''$ using surd-to-real-cnj-irrat[OF \* [of k2']] by auto hence surd-to-real-cnj (f k2') = 1 / surd-to-real-cnj (sqrt-remainder-step (f k2')) + a''using red-assoc-step(3)[OF \*[of k2'], folded a''-def] nz **by** (*simp add: field-simps*) also have  $\ldots = 1 / surd-to-real-cnj (sqrt-remainder-step (f k1')) + a'$ using k12 by (simp add: a'-a'' k12 Suc Suc' f-def) also have nz': surd-to-real-cnj  $(f k1') \neq a'$ using surd-to-real-cnj-irrat[OF \* [of k1']] by auto hence 1 / surd-to-real-cnj (sqrt-remainder-step (f k1')) + a' = surd-to-real-cnj(f k1')using red-assoc-step(3)[OF \*[of k1'], folded a'-def] nz nz' **by** (*simp add: field-simps*) finally have f k1' = f k2'**by** (subst (asm) surd-to-real-cnj-eq-iff) auto with k12 have  $k1' \leq D' * (D' + 1) \land (\exists k2 \leq D' * (D' + 1)) \land k1' \neq k2 \land fk1'$ = f k2**by** (auto simp: Suc Suc' intro!: exI[of - k2']) hence  $k1 \leq k1'$  unfolding k1-def by (rule Least-le) thus k1 = 0 by (simp add: Suc) **ged** auto have smallest-period:  $f k \neq f 0$  if  $k \in \{0 < .. < k2\}$  for k proof assume  $f k = f \theta$ hence  $k \leq D' * (D' + 1) \land k1 \neq k \land f k1 = f k$ using k12 that by auto hence  $k^2 \leq k$  unfolding  $k^2$ -def by (rule Least-le) with that show False by auto  $\mathbf{qed}$ have snd-f-gt-1: snd (f k) > 1 if  $k < k^2 - 1$  for k proof – have snd  $(f k) \neq 1$ proof assume snd (f k) = 1hence f k = (D', 1) using red-assoc-denom-1 [of fst (f k)] \*[of k] by (cases f k) auto

hence sqrt-remainder-step  $(fk) = (D', D - D'^2)$  by (auto simp: sqrt-remainder-step-def) hence f (Suc k) = f 0 by (simp add: f-def) moreover have f (Suc k)  $\neq f 0$ using that by (intro smallest-period) auto ultimately show False by contradiction qed **moreover have** snd (f k) > 0 using \*[of k] by (auto simp: red-assoc-def) ultimately show ?thesis by simp qed have sqrt-cfrac-le: sqrt-cfrac-nth  $k \leq D'$  if  $k < k^2 - 1$  for k proof define p and q where p = fst (f k) and q = snd (f k)have  $q \geq 2$  using snd-f-gt-1 [of k] that by (auto simp: q-def) also have sqrt-cfrac-nth k \* q < D' \* 2using red-assoc-step(5)[OF \* [of k]] by (simp add: sqrt-cfrac-nth-def p-def q-def case-prod-unfold f-def) finally show ?thesis by simp qed have *last*: f(k2 - 1) = (D', 1)proof define p and q where p = fst (f (k2 - 1)) and q = snd (f (k2 - 1))have pq: f(k2 - 1) = (p, q) by (simp add: p-def q-def) have sqrt-remainder-step (f(k2 - 1)) = f(Suc(k2 - 1))by (simp add: f-def) also from k12 have Suc (k2 - 1) = k2 by simpalso have f k2 = f 0using k12 by simp also have  $f \theta = (D', D - D'^2)$  by (simp add: f-def) finally have eq: sqrt-remainder-step  $(f(k2 - 1)) = (D', D - D'^2)$ . hence  $(D - D'^2)$  div  $q = D - D'^2$  unfolding sqrt-remainder-step-def Let-def pqby auto moreover have q > 0 using \*[of k2 - 1]**by** (*auto simp*: *red-assoc-def q-def*) ultimately have q = 1 using D'-sqr-less-D **by** (subst (asm) div-eq-dividend-iff) auto hence p = D'using red-assoc-denom-1 [of p] \*[of k2 - 1] unfolding pq by auto with  $\langle q = 1 \rangle$  show f(k2 - 1) = (D', 1) unfolding pq by simp qed have period: sqrt-remainder-surd n = sqrt-remainder-surd  $(n \mod k2)$  for n

unfolding sqrt-remainder-surd-def using k12

**by** (metis  $\langle k1 = 0 \rangle$  f-def funpow-mod-eq funpow-0 sqrt-remainder-surd-def) **have** period': sqrt-cfrac-nth k =sqrt-cfrac-nth (k mod k2) **for** k **using** period[of k] **by** (simp add: sqrt-cfrac-nth-def)

have k2-le:  $l \ge k2$  if  $l > 0 \ Ak$ . sqrt-cfrac-nth (k + l) = sqrt-cfrac-nth k for l **proof** (*rule ccontr*) assume  $*: \neg (l \ge k2)$ hence sqrt-cfrac-nth  $(k2 - Suc \ l) = sqrt-cfrac-nth \ (k2 - 1)$ using that(2)[of k2 - Suc l] by simpalso have  $\ldots = 2 * D'$ using last by (simp add: sqrt-cfrac-nth-def f-def) finally have 2 \* D' = sqrt-cfrac-nth  $(k2 - Suc \ l)$ .. also have  $\ldots \leq D'$  using k12 that \***by** (*intro sqrt-cfrac-le diff-less-mono2*) *auto* finally show False using D'-pos by simp qed have  $l = (LEAST l. \ 0 < l \land (\forall n. int (sqrt-cfrac-nth (n + l))) = int (sqrt-cfrac-nth)$ n)))using *nonsquare* unfolding *sqrt-cfrac-def* **by** (*simp add: l-def sqrt-nat-period-length-def sqrt-cfrac*) **hence** *l*-altdef:  $l = (LEAST \ l. \ 0 < l \land (\forall n. sqrt-cfrac-nth \ (n + l) = sqrt-cfrac-nth \ (n + l))$ n))by simp have [simp]:  $D \neq 0$  using nonsquare by (auto introl: Nat.gr0I) have  $\exists l. l > 0 \land (\forall k. sqrt-cfrac-nth (k + l) = sqrt-cfrac-nth k)$ **proof** (*rule exI*, *safe*) fix k show sqrt-cfrac-nth (k + k2) = sqrt-cfrac-nth k using period'[of k] period'[of k + k2] k12 by simp  $\mathbf{qed}$  (insert k12, auto) **from** LeastI-ex[OF this, folded l-altdef] have  $l: l > 0 \ Ak$ . sqrt-cfrac-nth (k + l) = sqrt-cfrac-nth k **by** (*simp-all add: sqrt-cfrac*) have  $l \leq k2$  unfolding *l*-altdef by (rule Least-le) (subst (1 2) period', insert k12, auto) moreover have  $k^2 \leq l$  using  $k^2$ -le l by blast ultimately have [simp]: l = k2 by *auto* define x' where  $x' = (\lambda k. -1 / surd-to-real-cnj (f k))$ { fix k :: nathave nz: surd-to-real-cnj  $(f k) \neq 0$  surd-to-real-cnj  $(f (Suc k)) \neq 0$ using surd-to-real-cnj-nz[OF \*, of k] surd-to-real-cnj-nz[OF \*, of Suc k] by (simp-all add: f-def) have surd-to-real-cnj  $(f k) \neq sqrt$ -cfrac-nth k using surd-to-real-cnj-irrat[OF \* [of k]] by auto hence  $x'(Suc \ k) = sqrt$ -cfrac-nth k + 1 / x' kusing red-assoc-step(3)[OF \*[of k]] nz

by (simp add: field-simps sqrt-cfrac-nth-def case-prod-unfold f-def x'-def)

} note x'-Suc = this

have x'-nz:  $x' k \neq 0$  for k using surd-to-real-cnj-nz[OF \*[of k]] by (auto simp: x'-def) have  $x' \cdot \theta$ :  $x' \theta$  = real D' + sqrt Dusing red-assoc-begin by (simp add: x'-def f-def) define c' where  $c' = c frac (\lambda n. sqrt-c frac-nth (l - Suc n))$ define c'' where  $c'' = cfrac (\lambda n. if n = 0 then 2 * D' else sqrt-cfrac-nth (n -$ 1)) have nth-c' [simp]:  $cfrac-nth \ c' \ n = sqrt-cfrac-nth \ (l - Suc \ n)$  for n**unfolding** c'-def by (subst cfrac-nth-cfrac) (auto simp: is-cfrac-def intro!: *sqrt-cfrac-pos*) have nth-c'' [simp]: cfrac-nth c'' n = (if n = 0 then 2 \* D' else sqrt-cfrac-nth (n)(-1) for n**unfolding** c''-def **by** (subst cfrac-nth-cfrac) (auto simp: is-cfrac-def intro!: *sqrt-cfrac-pos*) have conv' c' n (x' (l - n)) = x' l if  $n \leq l$  for nusing that **proof** (*induction* n) case (Suc n) have x' l = conv' c' n (x' (l - n))using Suc.prems by (intro Suc.IH [symmetric]) auto also have l - n = Suc (l - Suc n)using Suc.prems by simp also have  $x' \dots = c frac \cdot n + 1 / x' (l - Suc n)$ by (subst x'-Suc) simp also have  $conv' c' n \ldots = conv' c' (Suc n) (x' (l - Suc n))$ **by** (*simp add: conv'-Suc-right*) finally show ?case .. **qed** simp-all from this[of l] have  $conv' - x' - \theta$ :  $conv' c' l (x' \theta) = x' \theta$ using k12 by (simp add: x'-def) have cfrac-nth (cfrac-of-real  $(x' \ 0)$ )  $n = cfrac-nth \ c'' n$  for n **proof** (cases n) case  $\theta$ thus ?thesis by (simp add: x'-0 D'-def)  $\mathbf{next}$ case (Suc n') have sqrt  $D \notin \mathbb{Z}$ using red-assoc-begin(1) red-assoc-begin(2) by auto **hence** cfrac-nth (cfrac-of-real (real D' + sqrt (real D))) (Suc n') = cfrac-nth (cfrac-of-real (sqrt (real D))) (Suc n')by (simp add: cfrac-tl-of-real frac-add-of-nat Ints-add-left-cancel flip: cfrac-nth-tl) thus ?thesis using x'-nz[of 0] by (simp add: x'-0 sqrt-cfrac Suc)  $\mathbf{qed}$ 

show sqrt-cfrac-nth (l - n - 2) = sqrt-cfrac-nth n if n < l - 1 for n proof have D > 1 using nonsquare by (cases D) (auto introl: Nat.gr0I) hence D' + sqrt D > 0 + 1 using D'-pos by (intro add-strict-mono) auto hence  $x' \theta > 1$  by (auto simp:  $x' - \theta$ ) hence cfrac-nth c'(Suc n) = cfrac-nth (cfrac-of-real (conv' c' l (x' 0))) (Sucn)using  $\langle n < l - 1 \rangle$  using cfrac-of-real-conv' by auto also have  $\ldots = cfrac - nth (cfrac - of - real (x' 0)) (Suc n)$ by (subst conv'-x'- $\theta$ ) auto also have  $\ldots = cfrac \cdot nth \ c'' (Suc \ n)$  by fact finally show sqrt-cfrac-nth (l - n - 2) = sqrt-cfrac-nth nby simp qed show l > 0  $l \le D' * (D' + 1)$  using k12 by simp-all **show** sqrt-remainder-surd n =sqrt-remainder-surd  $(n \mod l)$ sqrt-cfrac-nth n = sqrt-cfrac-nth (n mod l) for nusing period[of n] period'[of n] by simp-allshow sqrt-remainder-surd  $n \neq$  sqrt-remainder-surd 0 if  $n \in \{0 < .. < l\}$  for n using smallest-period [of n] that by (auto simp: f-def) show snd (sqrt-remainder-surd n) > 1 if n < l - 1 for nusing that snd-f-gt-1[of n] by (simp add: f-def) show f(l - 1) = (D', 1) and sqrt-cfrac-nth (l - 1) = 2 \* D'using last by (simp-all add: sqrt-cfrac-nth-def f-def) show sqrt-cfrac-nth  $k \leq D'$  if k < l - 1 for k using sqrt-cfrac-le[of k] that by simpshow  $l' \ge l$  if  $l' > 0 \ Ak$ . sqrt-cfrac-nth (k + l') = sqrt-cfrac-nth k for l'using k2-le[of l'] that by auto qed

**theorem** cfrac-sqrt-periodic: cfrac-nth (cfrac-of-real (sqrt D)) (Suc n) = cfrac-nth (cfrac-of-real (sqrt D)) (Suc (n mod l))**using** sqrt-cfrac-periodic[of n] **by** (metis sqrt-cfrac)

**theorem** cfrac-sqrt-le:  $n \in \{0 < ... < l\} \implies$  cfrac-nth (cfrac-of-real (sqrt D))  $n \le D'$ using sqrt-cfrac-le[of n - 1] by (metis Suc-less-eq Suc-pred add.right-neutral greaterThanLessThan-iff of-nat-mono period-nonempty plus-1-eq-Suc sqrt-cfrac)

**theorem** cfrac-sqrt-last: cfrac-nth (cfrac-of-real (sqrt D)) l = 2 \* D'using sqrt-cfrac-last by (metis One-nat-def Suc-pred period-nonempty sqrt-cfrac)

**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 -
 have cfrac-nth (cfrac-of-real (sqrt D)) (l - n) = sqrt-cfrac-nth (l - n - 1)
   using assms by (subst sqrt-cfrac) (auto simp: Suc-diff-Suc)
 also have \ldots = sqrt-cfrac-nth (n - 1)
   using assms by (subst sqrt-cfrac-palindrome [symmetric]) auto
 also have \ldots = cfrac - nth (cfrac - of - real (sqrt D)) n
   using assms by (subst sqrt-cfrac) auto
 finally show ?thesis .
qed
lemma sqrt-cfrac-info-palindrome:
 assumes sqrt-cfrac-info D = (a, b, cs)
 shows rev (butlast cs) = butlast cs
proof (rule List.nth-equalityI; safe?)
 fix i assume i < length (rev (butlast cs))
 with period-nonempty have Suc i < length \ cs \ by \ simp
 thus rev (butlast cs) ! i = butlast cs ! i
   using assms cfrac-sqrt-palindrome[of Suc i] period-nonempty unfolding l-def
   by (auto simp: sqrt-cfrac-info-def rev-nth algebra-simps Suc-diff-Suc simp del:
cfrac.simps)
qed simp-all
lemma sqrt-cfrac-info-last:
 assumes sqrt-cfrac-info D = (a, b, cs)
 shows last cs = 2 * floor-sqrt D
proof -
 from assms show ?thesis using period-nonempty cfrac-sqrt-last
   by (auto simp: sqrt-cfrac-info-def last-map l-def D'-def Discrete-sqrt-altdef)
```

```
qed
```

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**  $b \equiv (\lambda(-, pq). snd pq \neq 1)$ 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 define P where  $P = (\lambda(as, pq))$ . let n = length as in  $n < l \land pq = sqrt$ -remainder-surd  $n \land as = rev$  (map sqrt-cfrac-nth [0..< n]))define  $\mu$  :: nat list  $\times$  (nat  $\times$  nat)  $\Rightarrow$  nat where  $\mu = (\lambda(as, -), l - length as)$ have [simp]: P initial using period-nonempty **by** (*auto simp: initial-def P-def sqrt-remainder-surd-def*) have step':  $P(step' s) \land Suc(length(fst s)) < l \text{ if } P s b s \text{ for } s$ **proof** (cases s)

case (fields as p q) define n where n = length as from that fields sqrt-remainder-surd-last have Suc  $n \leq l$ **by** (*auto simp*: *b-def P-def Let-def n-def* [*symmetric*]) **moreover from** that fields sqrt-remainder-surd-last have Suc  $n \neq l$ **by** (*auto simp*: *b-def P-def Let-def n-def* [*symmetric*]) ultimately have Suc n < l by auto with that fields sqrt-remainder-surd-last show P (step' s)  $\wedge$  Suc (length (fst s)) < lby (simp add: b-def P-def Let-def n-def step'-def sqrt-cfrac-nth-def sqrt-remainder-surd-def case-prod-unfold) aed **have** [simp]: length (fst (step' s)) = Suc (length (fst s)) for s **by** (*simp add: step'-def case-prod-unfold*) have  $\exists x. while option b step' initial = Some x$ **proof** (*rule measure-while-option-Some*) fix s assume \*: P s b sfrom step'[OF \*] show  $P(step' s) \land \mu(step' s) < \mu s$ by (auto simp: b-def  $\mu$ -def case-prod-unfold intro!: diff-less-mono2) qed auto then obtain x where x: while-option b step' initial = Some x ... have P x by (rule while-option-rule[OF - x]) (insert step', auto) have  $\neg b x$  using while-option-stop[OF x] by auto **obtain** as  $p \ q$  where [simp]: x = (as, (p, q)) by (cases x)define n where n = length as have [simp]: q = 1 using  $\langle \neg b \rangle x by (auto simp: b-def)$ have [simp]: p = D' using  $\langle P x \rangle$ using red-assoc-denom-1 [of p] by (auto simp: P-def Let-def) have n < l sqrt-remainder-surd (length as) =  $(D', Suc \ 0)$ and as:  $as = rev (map \ sqrt-cfrac-nth \ [0..< n]) \ using \langle P \ x \rangle$ by (auto simp: P-def Let-def n-def) hence  $\neg (n < l - 1)$ using snd-sqrt-remainder-surd-gt-1 [of n] by (intro notI) auto with  $\langle n < l \rangle$  have [simp]: n = l - 1 by auto **show** ?thesis **by** (simp add: as x) qed **lemma** while-option-sqrt-cfrac-info: **defines** step'  $\equiv (\lambda(as, pq))$ . ((D' + fst pq) div snd pq # as, sqrt-remainder-step)pq))defines  $b \equiv (\lambda(-, pq))$ . 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, -)  $\Rightarrow$  (Suc (length as), D', rev ((2 \* D') # as)))

proof -

have nat (cfrac-nth (cfrac-of-real (sqrt (real D))) (Suc k)) = sqrt-cfrac-nth k for

k by (metis nat-int sqrt-cfrac) thus ?thesis unfolding assms while-option-sqrt-cfrac using period-nonempty sqrt-cfrac-last by (cases l) (auto simp: sqrt-cfrac-info-def D'-def l-def Discrete-sqrt-altdef) qed end end end lemma sqrt-nat-period-length-le: sqrt-nat-period-length  $D \le nat \lfloor sqrt D \rfloor * (nat \lfloor sqrt D \rfloor + 1)$ by (cases is-square D) (use period-length-le-aux[of D] in auto) lemma sqrt-nat-period-length-0-iff [simp]: sqrt-nat-period-length  $D = 0 \longleftrightarrow$  is-square D

using period-nonempty[of D] by (cases is-square D) auto lemma sqrt-nat-period-length-pos-iff [simp]: sqrt-nat-period-length  $D > 0 \leftrightarrow \neg is$ -square D

```
using period-nonempty[of D] by (cases is-square D) auto
```

```
lemma sqrt-cfrac-info-code [code]:
```

```
sqrt-cfrac-info D =
    (let \ D' = floor\text{-}sqrt \ D
     in if D'^2 = D then (0, D', [])
         else
          case while-option
                (\lambda(-, pq)). snd pq \neq 1
                (\lambda(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, -) \Rightarrow (Suc (length as), D', rev ((2 * D') # as)))
proof –
 define D' where D' = floor-sqrt D
 show ?thesis
 proof (cases is-square D)
   case True
   hence D' \cap 2 = D by (auto simp: D'-def elim!: is-nth-powerE)
   thus ?thesis using True
     by (simp add: D'-def Let-def sqrt-cfrac-info-def sqrt-nat-period-length-def)
  \mathbf{next}
   case False
   hence D' \widehat{\phantom{a}} 2 \neq D by (subst eq-commute) auto
   thus ?thesis using while-option-sqrt-cfrac-info[OF False]
     by (simp add: sqrt-cfrac-info-def D'-def Let-def
               case-prod-unfold Discrete-sqrt-altdef add-ac sqrt-remainder-step-def)
 qed
qed
```

**lemma** sqrt-nat-period-length-code [code]: sqrt-nat-period-length D = fst (sqrt-cfrac-info D) **by** (simp add: sqrt-cfrac-info-def)

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  $D = (case \ sqrt-cfrac-info \ D \ of \ (a, b, c) \Rightarrow (a, b, IArray \ c))$ 

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

**by** (*simp add: sqrt-cfrac-info-array-def sqrt-cfrac-info-def*)

**lemma** snd-sqrt-cfrac-info-array [simp]: fst (snd (sqrt-cfrac-info-array D)) = floor-sqrt D

**by** (*simp add: sqrt-cfrac-info-array-def sqrt-cfrac-info-def*)

**definition**  $cfrac-sqrt-nth :: nat \times nat \times nat iarray \Rightarrow nat \Rightarrow nat where$  $<math>cfrac-sqrt-nth info \ n =$   $(case info \ of \ (l, \ a0, \ as) \Rightarrow if \ n = 0 \ then \ a0 \ else \ as !! \ ((n - 1) \ mod \ l))$ **lemma** cfrac-sqrt-nth:

```
assumes \neg 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** (cases n) case (Suc n') define l where l = sqrt-nat-period-length Dfrom period-nonempty[OF assms] have l > 0 by (simp add: l-def) have cfrac-nth (cfrac-of-real (sqrt D)) (Suc n') = cfrac-nth (cfrac-of-real (sqrt D)) (Suc (n' mod l)) unfolding *l-def* using cfrac-sqrt-periodic [OF assms, of n'] by simp also have  $\ldots = map(\lambda n. nat(cfrac-nth(cfrac-of-real(sqrt D))(Suc n)))[0...<l]$  $! (n' \mod l)$ using  $\langle l > 0 \rangle$  by (subst nth-map) auto finally show ?thesis using Suc by (simp add: sqrt-cfrac-info-array-def sqrt-cfrac-info-def l-def cfrac-sqrt-nth-def) qed (simp-all add: sqrt-cfrac-info-def sqrt-cfrac-info-array-def Discrete-sqrt-altdef cfrac-sqrt-nth-def)

**by** (*auto elim*!: *is-nth-powerE*) thus ?thesis using True by (auto simp: Let-def sqrt-cfrac-info-array-def sqrt-cfrac-info-def sqrt-cfrac-def)  $\mathbf{next}$ case False have cfrac-sqrt-nth (sqrt-cfrac-info-array D) n > 0 if n > 0 for n proof – have int (cfrac-sqrt-nth (sqrt-cfrac-info-array D) n) > 0 using False that by (subst cfrac-sqrt-nth [symmetric]) auto thus ?thesis by simp qed moreover have sqrt  $D \notin \mathbb{Q}$ using False irrat-sqrt-nonsquare by blast ultimately have sqrt-cfrac D = cfrac (cfrac-sqrt-nth (sqrt-cfrac-info-array D)) using cfrac-sqrt-nth[OF False] **by** (*intro cfrac-eqI*) (*auto simp: sqrt-cfrac-def is-cfrac-def*) thus ?thesis using False by (simp add: Let-def sqrt-cfrac-info-array-def sqrt-cfrac-info-def) qed

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 ( $\lambda n. |conv (sqrt-cfrac 129) n \ 2 - 129|$ ) [0..<20]

 $\mathbf{end}$ 

# 5 Lifting solutions of Pell's Equation

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

### 5.1 Auxiliary material

**definition** square-squarefree-part-nat :: nat  $\Rightarrow$  nat  $\times$  nat where square-squarefree-part-nat n = (square-part n, squarefree-part n)

**lemma** prime-factorization-squarefree-part:

assumes  $x \neq 0$ **shows** prime-factorization (squarefree-part x) = mset-set  $\{p \in prime-factors x. odd (multiplicity p x)\}$  (is ?lhs = ?rhs) **proof** (*rule multiset-eqI*) fix p show count ? lhs p = count ? rhs p**proof** (cases prime p) case False thus ?thesis by (auto simp: count-prime-factorization) next case True have finite (prime-factors x) by simp hence finite  $\{p, p \ dvd \ x \land prime \ p\}$  using assms by (subst (asm) prime-factors-dvd) (auto simp: conj-commute) hence finite  $\{p, p \ dvd \ x \land prime \ p \land odd \ (multiplicity \ p \ x)\}$ **by** (rule finite-subset [rotated]) auto **moreover have** odd  $(n :: nat) \leftrightarrow n \mod 2 = Suc \ 0$  for n by presburger ultimately show ?thesis using assms **by** (cases  $p \ dvd \ x$ ; cases even (multiplicity  $p \ x$ )) (auto simp: count-prime-factorization prime-multiplicity-squarefree-part *in-prime-factors-iff not-dvd-imp-multiplicity-0*) qed  $\mathbf{qed}$ **lemma** squarefree-part-nat: squarefree-part  $(n :: nat) = (\prod \{ p \in prime-factors n. odd (multiplicity p n) \})$ **proof** (cases n = 0) case False **hence**  $(\prod \{p \in prime-factors n. odd (multiplicity p n)\}) =$  $prod-mset \ (prime-factorization \ (squarefree-part \ n))$ by (subst prime-factorization-squarefree-part) (auto simp: prod-unfold-prod-mset) also have  $\ldots = squarefree-part n$ by (intro prod-mset-prime-factorization-nat Nat.gr0I) auto finally show ?thesis .. qed auto **lemma** prime-factorization-square-part: assumes  $x \neq 0$ **shows** prime-factorization (square-part x) =  $(\sum p \in prime-factors x. replicate-mset (multiplicity p x div 2) p)$  (is ?lhs = ?rhs**proof** (*rule multiset-eqI*) fix p show count ? lhs p = count ? rhs p**proof** (cases prime  $p \land p \ dvd \ x$ ) case False thus ?thesis by (auto simp: count-prime-factorization count-sum prime-multiplicity-square-part not-dvd-imp-multiplicity-0) next case True thus ?thesis using assms

```
by (cases p \ dvd \ x)
        (auto simp: count-prime-factorization prime-multiplicity-squarefree-part
                  in-prime-factors-iff count-sum prime-multiplicity-square-part)
 qed
ged
lemma prod-mset-sum: prod-mset (sum f A) = (\prod x \in A. prod-mset (f x))
 by (induction A rule: infinite-finite-induct) auto
lemma square-part-nat:
 assumes n > \theta
  shows square-part (n :: nat) = (\prod p \in prime-factors n. p \land (multiplicity p n))
div 2))
proof -
 have (\prod p \in prime-factors n. p \land (multiplicity p n div 2)) =
        prod-mset (prime-factorization (square-part n)) using assms
    by (subst prime-factorization-square-part) (auto simp: prod-unfold-prod-mset
prod-mset-sum)
 also have \ldots = square-part \ n \ using \ assms
   by (intro prod-mset-prime-factorization-nat Nat.gr0I) auto
 finally show ?thesis ..
\mathbf{qed}
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 ((\prod p \in set\text{-mset } ps. p \cap (count ps p div 2))),
             \prod (Set.filter (\lambda p. odd (count ps p)) (set-mset ps))))
  by (cases n = \theta)
   (auto simp: Let-def square-squarefree-part-nat-def squarefree-part-nat Set.filter-def
               count-prime-factorization square-part-nat intro!: prod.cong)
lemma square-part-nat-code [code-unfold]:
  square-part (n :: nat) = (if n = 0 then 0)
    else let ps = prime-factorization n in (\prod p \in set-mset ps. p \uparrow (count ps p div
2)))
  using square-squarefree-part-nat-code[of n]
 by (simp add: square-squarefree-part-nat-def Let-def split: if-splits)
lemma squarefree-part-nat-code [code-unfold]:
  squarefree-part (n :: nat) = (if n = 0 then 1)
     else let ps = prime-factorization n in (\prod (Set.filter (\lambda p. odd (count ps p)))
(set-mset ps))))
 using square-squarefree-part-nat-code[of n]
 by (simp add: square-squarefree-part-nat-def Let-def split: if-splits)
```

**lemma** *is-nth-power-mult-nth-powerD*: assumes *is-nth-power n*  $(a * b \ n) b > 0 n > 0$ 

 $is-nth-power \ n \ (a::nat)$ shows proof from assms obtain k where k:  $k \uparrow n = a * b \uparrow n$ **by** (*auto elim: is-nth-powerE*) with assms(2,3) have b dvd k **by** (*metis dvd-triv-right pow-divides-pow-iff*) then obtain l where k = b \* l**by** *auto* with k have  $a = l \cap n$  using assms(2)**by** (*simp add: power-mult-distrib*) thus ?thesis by auto qed **lemma** (in *pell*) *fund-sol-eq-fstI*: **assumes** nontriv-solution (x, y)assumes  $\bigwedge x' y'$ . nontriv-solution  $(x', y') \Longrightarrow x \le x'$ **shows** fund-sol = (x, y)proof have x = fst fund-sol**using** fund-sol-is-nontriv-solution assms(1) fund-sol-minimal''[of (x, y)] by (auto introl: antisym assms(2)[of fst fund-sol snd fund-sol]) moreover from this have y = snd fund-sol **using** assms(1) solutions-linorder-strict [of x y fst fund-sol snd fund-sol] fund-sol-is-nontriv-solutionby (auto simp: nontriv-solution-imp-solution prod-eq-iff) ultimately show ?thesis by simp qed **lemma** (in *pell*) *fund-sol-eqI-fst'*: assumes nontriv-solution xy assumes  $\bigwedge x' y'$ . nontriv-solution  $(x', y') \Longrightarrow fst xy \le x'$ shows fund-sol = xyusing fund-sol-eq-fstI[of fst xy snd xy] assms by simp **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 – have y = snd fund-sol using fund-sol-is-nontriv-solution assms(1) fund-sol-minimal''[of (x, y)] **by** (*auto intro*!: *antisym assms*(2)[*of fst fund-sol snd fund-sol*]) **moreover from** this have x = fst fund-sol **using** assms(1) solutions-linorder-strict [of x y fst fund-sol snd fund-sol]  ${\it fund-sol-is-nontriv-solution}$ by (auto simp: nontriv-solution-imp-solution prod-eq-iff) ultimately show ?thesis by simp

 $\mathbf{qed}$ 

**lemma** (in pell) fund-sol-eqI-snd': **assumes** nontriv-solution xy **assumes**  $\bigwedge x' y'$ . nontriv-solution  $(x', y') \Longrightarrow$  snd  $xy \le y'$  **shows** fund-sol = xy**using** fund-sol-eq-sndI[of fst xy snd xy] assms by simp

## 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' :: natassumes nz: a > 0defines  $D' \equiv D * a^2$ begin

```
lemma nonsquare-D': \neg is-square D'
```

**using** nonsquare-D is-nth-power-mult-nth-powerD[of 2 D a] nz by (auto simp: D'-def)

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

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

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

```
sublocale lift: pell D'
by unfold-locales (fact nonsquare-D')
```

```
lemma lift-solution-iff: lift.solution xy \leftrightarrow solution (lower-solution xy)

unfolding solution-def lift.solution-def

by (auto simp: lower-solution-def D'-def case-prod-unfold power-mult-distrib)
```

lemma lift-solution:
 assumes solution xy liftable-solution xy
 shows lift.solution (lift-solution xy)
 using assms unfolding solution-def lift.solution-def
 by (auto simp: liftable-solution-def lift-solution-def D'-def case-prod-unfold power-mult-distrib
 elim!: dvdE)

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

**lemma** *lift-fund-sol*:

assumes  $\bigwedge n$ .  $\theta < n \implies n < m \implies \neg$  liftable-solution (nth-solution n) assumes liftable-solution (nth-solution m) m > 0**shows** lift.fund-sol = lift-solution (nth-solution m)**proof** (*rule lift.fund-sol-eqI-fst'*) **from** assms **have** nontriv-solution (nth-solution m) by (intro nth-solution-sound') hence lift-solution (nth-solution  $m) \neq (1, 0)$  using  $nz \ assms(2)$ by (auto simp: lift-solution-def case-prod-unfold nontriv-solution-def liftable-solution-def) with assms show lift.nontriv-solution (lift-solution (nth-solution m)) **by** (*auto simp: lift.nontriv-solution-altdef intro: lift-solution*) next fix x' y' :: nat**assume** \*: *lift.nontriv-solution* (x', y')hence nz':  $x' \neq 1$  using nonsquare-D' **by** (*auto simp: lift.nontriv-solution-altdef lift.solution-def*) from \* have solution (lower-solution (x', y')) **by** (*simp add: lift-solution-iff lift.nontriv-solution-altdef*) **hence** lower-solution  $(x', y') \in$  range nth-solution by (rule nth-solution-complete) then obtain n where n: nth-solution n = lower-solution (x', y') by auto with nz' have n > 0 by (auto introl: Nat. gr0I simp: nth-solution-def lower-solution-def) with *n* have liftable-solution (*nth-solution n*) **by** (*auto simp: liftable-solution-def lower-solution-def*) with (n > 0) and assms(1)[of n] have  $n \ge m$  by (cases  $n \ge m$ ) auto hence fst (*nth-solution* m)  $\leq fst$  (*nth-solution* n) using strict-mono-less-eq[OF strict-mono-nth-solution(1)] by simp thus fst (lift-solution (nth-solution m))  $\leq x'$ by (simp add: lift-solution-def lower-solution-def n case-prod-unfold) qed

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 -
```

```
from assms obtain y where [simp]: x = y \hat{2}
   by (auto simp: is-nth-power-def)
 have unit-factor x * normalize x = squarefree-part x * square-part x ^2
   by (subst squarefree-decompose [symmetric]) auto
 also have \ldots = squarefree-part x * normalize x
   by (simp add: square-part-even-power normalize-power)
 finally show ?thesis using assms
   by (subst (asm) mult-cancel-right) auto
qed
lemma squarefree-part-1-imp-square:
 assumes squarefree-part x = 1
 shows is-square x
proof -
 have is-square (square-part x \uparrow 2)
   by auto
 also have square-part x \uparrow 2 = squarefree-part x * square-part x \uparrow 2
   using assms by simp
 also have \ldots = x
   by (rule squarefree-decompose [symmetric])
 finally show ?thesis .
```

```
\mathbf{qed}
```

```
definition find-fund-sol-fast where

find-fund-sol-fast D =

(let (a, D') = square-squarefree-part-nat D

in

if D' = 0 \lor D' = 1 then (0, 0)

else if a = 1 then pell.fund-sol D

else map-prod id (\lambda y. y \text{ div } a)

(shd (sdrop-while (\lambda(-, y). y = 0 \lor \neg a \text{ dvd } y) (pell-solutions D'))))
```

```
lemma find-fund-sol-fast: find-fund-sol D = find-fund-sol-fast D

proof (cases is-square D \lor square-part D = 1)

case True

thus ?thesis

using squarefree-part-1-imp-square[of D]

by (cases D = 0)

(auto simp: find-fund-sol-correct find-fund-sol-fast-def

square-squarefree-part-nat-def square-test-correct unit-factor-nat-def)

next

case False

define D' a where D' = squarefree-part D and a = square-part D

have D > 0

using False by (intro Nat.gr0I) auto

have a > 0

using \langle D > 0 \rangle by (intro Nat.gr0I) (auto simp: a-def)
```

unfolding D'-def by (metis False is-nth-power-mult is-nth-power-nth-power squarefree-decompose) ultimately interpret *lift*: *pell-lift* D' a D using False  $\langle D > 0 \rangle$ by unfold-locales (auto simp: D'-def a-def squarefree-decompose [symmetric]) define i where  $i = (LEAST \ i. \ case \ lift.nth-solution \ i \ of \ (-, \ y) \Rightarrow y > 0 \ \land a$ dvd yhave ex:  $\exists i. case lift.nth-solution i of (-, y) \Rightarrow y > 0 \land a dvd y$ proof **define** sol **where** sol = lift.lift.fund-sol have is-sol: lift.solution (lift.lower-solution sol) unfolding sol-def using lift.lift.fund-sol-is-nontriv-solution lift.lift-solution-iff by blast then obtain *j* where *j*: *lift.lower-solution* sol = lift.nth-solution *j* using *lift.solution-iff-nth-solution* by *blast* have snd (lift.lower-solution sol) > 0**proof** (*rule Nat.gr0I*) **assume** \*: snd (lift.lower-solution sol) = 0 have lift.solution (fst (lift.lower-solution sol), snd (lift.lower-solution sol)) using *is-sol* by *simp* hence fst (lift.lower-solution sol) = 1 **by** (*subst* (*asm*) \*) *simp* with \* have lift.lower-solution sol = (1, 0)by (cases lift.lower-solution sol) auto hence  $fst \ sol = 1$ unfolding lift.lower-solution-def by (auto simp: lift.lower-solution-def *case-prod-unfold*) thus False unfolding sol-def using *lift.lift.fund-sol-is-nontriv-solution*  $\langle D > 0 \rangle$ **by** (*auto simp: lift.lift.nontriv-solution-def*) qed moreover have a dvd snd (lift.lower-solution sol) **by** (*auto simp: lift.lower-solution-def case-prod-unfold*) ultimately show *?thesis* using *j* by (*auto simp*: *case-prod-unfold*) qed define sol where sol = lift.nth-solution i have sol: snd sol > 0 a dvd snd solusing LeastI-ex[OF ex] by (simp-all add: sol-def i-def case-prod-unfold) have  $i > \theta$ using sol by (intro Nat.gr0I) (auto simp: sol-def lift.nth-solution-def) have find-fund-sol-fast  $D = map-prod \ id \ (\lambda y. \ y \ div \ a)$ (shd (sdrop-while ( $\lambda(-, y)$ ).  $y = 0 \vee \neg a \, dvd \, y$ ) (pell-solutions D'))) unfolding D'-def a-def find-fund-sol-fast-def using False squarefree-part-1-imp-square[of D

**by** (*auto simp: square-squarefree-part-nat-def*) also have sdrop-while  $(\lambda(-, y), y = 0 \lor \neg a \ dvd \ y)$  (pell-solutions D') =sdrop-while (Not  $\circ$  ( $\lambda$ (-, y).  $y > 0 \land a \, dvd \, y$ )) (pell-solutions D') **by** (*simp add: o-def case-prod-unfold*) also have  $\ldots = sdrop \ i \ (pell-solutions \ D')$ using ex by (subst sdrop-while-sdrop-LEAST) (simp-all add: lift.snth-pell-solutions i-def) also have  $shd \ldots = sol$ **by** (*simp add: lift.snth-pell-solutions sol-def*) finally have eq: find-fund-sol-fast  $D = map-prod \ id \ (\lambda y. \ y \ div \ a) \ sol$ . **have** *lift.lift.fund-sol* = *lift.lift-solution sol* unfolding sol-def proof (rule lift.lift-fund-sol) show i > 0 by fact **show** *lift.liftable-solution* (*lift.nth-solution i*) using sol by (simp add: sol-def lift.liftable-solution-def case-prod-unfold)  $\mathbf{next}$ fix j :: nat assume j: j > 0 j < i**show**  $\neg$ *lift.liftable-solution* (*lift.nth-solution j*) proof **assume** *liftable: lift.liftable-solution* (*lift.nth-solution j*) have snd (lift.nth-solution j) > 0using (j > 0) by (metis gr0I lift.nontriv-solution-altdef lift.nth-solution-sound) *lift.solution-0-snd-nat-iff prod.collapse*) hence case lift.nth-solution j of  $(-, y) \Rightarrow y > 0 \land a \, dvd \, y$ using  $\langle j > 0 \rangle$  liftable by (auto simp: lift.liftable-solution-def) hence  $i \leq j$ unfolding *i*-def by (rule Least-le) thus False using  $\langle j < i \rangle$  by simp qed  $\mathbf{qed}$ also have  $\ldots = find$ -fund-sol-fast D by (simp add: eq lift.lift-solution-def case-prod-unfold map-prod-def) finally show *?thesis* using  $\langle D > 0 \rangle$  False by (simp add: find-fund-sol-correct) qed

 $\mathbf{end}$ 

# 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 = \theta \land b = \theta
proof -
 have a = \theta
 proof (rule ccontr)
   assume a \neq 0
   with assms(1) have p = b / a by (auto simp: field-simps)
   also have \ldots \in \mathbb{Q} by auto
   finally show False using assms(2) by contradiction
 qed
 with assms show ?thesis by simp
qed
```

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 by unfold-locales fact+

**lemma** cfrac-length-infinite [simp]: cfrac-length  $c = \infty$  **proof** – **have** sqrt  $D \notin \mathbb{Q}$  **using** nonsquare **by** (simp add: irrat-sqrt-nonsquare) **thus** ?thesis **by** (simp add: c-def) **qed** 

```
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 – define D' where D' = floor-sqrt D have  $h \ 0 \ 2 - D * k \ 0 \ 2 = int \ (D' \ 2) - int D$ by (simp-all add: h-def k-def c-def Discrete-sqrt-altdef D'-def) also {
have int  $(D' \uparrow 2) - int D \leq 0$ using floor-sqrt-power2-le[of D] by (simp add: D'-def) moreover have  $D \neq D' \uparrow 2$  using nonsquare by auto ultimately have int  $(D' \uparrow 2) - int D < 0$  by linarith finally show  $h \ 0 \ \hat{2} - D * k \ 0 \ \hat{2} < 0$ .  $\mathbf{next}$ assume  $m > \theta$ define n where n = m - 1define  $\alpha$  where  $\alpha = cfrac$ -remainder cdefine  $\alpha'$  where  $\alpha' = sqrt$ -remainder-surd D have m:  $m = Suc \ n \text{ using } \langle m > 0 \rangle$  by (simp add: n-def) from *nonsquare* have D > 1by (cases D) (auto introl: Nat.gr0I) **from** *nonsquare* have *irrat*: *sqrt*  $D \notin \mathbf{Q}$ using *irrat-sqrt-nonsquare* by *blast* have [simp]: cfrac-lim c = sqrt Dusing irrat  $\langle D > 1 \rangle$  by (simp add: c-def) have  $\alpha$ -pos:  $\alpha \ n > 0$  for n**unfolding**  $\alpha$ -def using wf  $\langle D > 1 \rangle$  cfrac-remainder-pos[of c n] by (cases n = 0) auto have  $\alpha': \alpha' n = (P n, Q n)$  for n by (simp add:  $\alpha'$ -def P-def Q-def) have Q-pos: Q n > 0 for nusing snd-sqrt-remainder-surd-pos[OF nonsquare] by (simp add: Q-def) have k-pos: k n > 0 for n**by** (*auto simp: k-def intro*!: *conv-denom-pos*) have k-nonneg:  $k \ n \ge 0$  for n **by** (*auto simp: k-def intro*!: *conv-denom-nonneg*) let ?A = (sqrt D + P (n + 1)) \* h (n + 1) + Q (n + 1) \* h nlet ?B = (sqrt D + P (n + 1)) \* k (n + 1) + Q (n + 1) \* k nhave ?B > 0 using k-pos Q-pos k-nonneg by (intro add-nonneg-pos mult-nonneg-nonneg add-nonneg-nonneg) auto have sqrt D = conv' c (Suc (Suc n)) ( $\alpha$  (Suc (Suc n))) **unfolding**  $\alpha$ -def by (subst conv'-cfrac-remainder) auto also have ... =  $(\alpha (n + 2) * h (n + 1) + h n) / (\alpha (n + 2) * k (n + 1) + k)$ n)using wf  $\alpha$ -pos by (subst conv'-num-denom) (simp-all add: h-def k-def) also have  $\alpha$   $(n + 2) = surd-to-real D (\alpha' (Suc n))$ using surd-to-real-sqrt-remainder-surd[OF nonsquare, of Suc n] by (simp add:  $\alpha'$ -def  $\alpha$ -def c-def) also have  $\ldots = (sqrt \ D + P \ (Suc \ n)) \ / \ Q \ (Suc \ n) \ (is \ - = \ ?\alpha)$ by (simp add:  $\alpha'$  surd-to-real-def) also have  $?\alpha * h (n + 1) + h n =$ 1 / Q(n + 1) \* ((sqrt D + P(n + 1)) \* h(n + 1) + Q(n + 1) \*h nusing Q-pos by (simp add: field-simps) also have  $?\alpha * k (n + 1) + k n =$ 

1 / Q (n + 1) \* ((sqrt D + P (n + 1)) \* k (n + 1) + Q (n + 1) \*

(is - = ?f k) using Q-pos by (simp add: field-simps) also have ?f h / ?f k = ((sqrt D + P (n + 1)) \* h (n + 1) + Q (n + 1) \* h n) /

((sqrt D + P (n + 1)) \* k (n + 1) + Q (n + 1) \* k n)

(is - = ?A / ?B) using Q-pos by (intro mult-divide-mult-cancel-left) auto finally have sqrt D \* ?B = ?A

using  $\langle B \rangle = 0$  by (simp add: divide-simps)

moreover have sqrt D \* sqrt D = D by simp

ultimately have sqrt D \* (P(n+1) \* k(n+1) + Q(n+1) \* kn - h(n+1)) =

P(n + 1) \* h(n + 1) + Q(n + 1) \* hn - k(n + 1) \* D

**unfolding** *of-int-add of-int-mult of-int-diff of-int-of-nat-eq of-nat-mult of-nat-add* **by** *Groebner-Basis.algebra* 

**from** *irrational-times-int-eq-intD*[OF this] *irrat* 

have 1: h(Suc n) = P(Suc n) \* k(Suc n) + Q(Suc n) \* k nand 2: D \* k(Suc n) = P(Suc n) \* h(Suc n) + Q(Suc n) \* h nby (simp-all del: of-nat-add of-nat-mult)

have h (Suc n) \* h (Suc n) - D \* k (Suc n) \* k (Suc n) = Q (Suc n) \* (k n \* h (Suc n) - k (Suc n) \* h n)by (subst 1, subst 2) (simp add: algebra-simps) also have  $k n * h (Suc n) - k (Suc n) * h n = (-1) ^n$ unfolding h-def k-def by (rule conv-num-denom-prod-diff) finally have  $h (Suc n) ^2 - D * k (Suc n) ^2 = (-1) ^n * Q (Suc n)$ by (simp add: power2-eq-square algebra-simps) thus  $h m ^2 - D * k m ^2 = (-1) ^Suc m * Q m$ 

thus h m = 2 - D \* k m = 2 = (-1) Suc m \* Qby (simp add: m)

#### qed

k n

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 $(int x, int y) \in range (\lambda n. (conv-num c n, conv-denom c n))$ shows proof have  $\exists n. enat n \leq cfrac-length c \land (int x, int y) = (conv-num c n, conv-denom)$ c n**proof** (*rule frac-is-convergentI*) have  $gcd(x^2)(y^2) = 1$  unfolding assms(1)using gcd-add-mult[of  $y^2 D 1$ ] by (simp add: gcd.commute) thus coprime (int x) (int y) **by** (*simp add: coprime-iff-gcd-eq-1*)  $\mathbf{next}$ from assms have D > 1using nonsquare by (cases D) (auto introl: Nat.gr0I) hence pos: x + y \* sqrt D > 0 using assms by (intro add-nonneg-pos) auto

from assms have real  $(x^2) = real (Suc (D * y^2))$ **by** (*simp only: of-nat-eq-iff*) hence  $1 = real x \hat{z} - D * real y \hat{z}$ **unfolding** *of-nat-power* by *simp* also have  $\dots = (x - y * sqrt D) * (x + y * sqrt D)$ **by** (*simp add: field-simps power2-eq-square*) finally have \*: x - y \* sqrt D = 1 / (x + y \* sqrt D)using pos by (simp add: field-simps) from pos have  $\theta < 1 / (x + y * sqrt D)$ by (intro divide-pos-pos) auto also have  $\ldots = x - y * sqrt D$  by (rule \* [symmetric])finally have less: y \* sqrt D < x by simp have sqrt D - x / y = -((x - y \* sqrt D) / y)using  $\langle y > 0 \rangle$  by (simp add: field-simps) also have  $|\ldots| = (x - y * sqrt D) / y$ using less by simp also have (x - y \* sqrt D) / y = 1 / (y \* (x + y \* sqrt D))using  $\langle y > 0 \rangle$  by (subst \*) auto also have  $\ldots \leq 1 / (y * (y * sqrt D + y * sqrt D))$ using  $\langle y > 0 \rangle \langle D > 1 \rangle$  pos less by (intro divide-left-mono mult-left-mono add-right-mono mult-pos-pos) auto **also have** ... =  $1 / (2 * y^2 * sqrt D)$ **by** (*simp add: power2-eq-square*) also have  $\ldots < 1 / (real (2 * y^2) * 1)$  using  $\langle y > 0 \rangle \langle D > 1 \rangle$ by (intro divide-strict-left-mono mult-strict-left-mono mult-pos-pos) auto finally show  $|cfrac-lim c - int x / int y| < 1 / (2 * int y ^2)$ **unfolding** *c-def* **using** *irrat-sqrt-nonsquare* [of D]  $\langle \neg is$ -square  $D \rangle$  by simp **qed** (insert assms irrat-sqrt-nonsquare[of D], auto simp: c-def) thus ?thesis by auto

#### $\mathbf{qed}$

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:

 $\begin{array}{l} \textit{nontriv-solution } (x, y) \longleftrightarrow \\ (\exists m > 0. \ \textit{int } x = h \ (l \ast m - 1) \land \textit{int } y = k \ (l \ast m - 1) \land \textit{even } (l \ast m)) \\ \textbf{proof } \textit{safe} \\ \textbf{fx } m \ \textbf{assume } xy: x = h \ (l \ast m - 1) \ y = k \ (l \ast m - 1) \\ \textbf{and } lm: \ \textit{even } (l \ast m) \ \textbf{and } m: \ m > 0 \\ \textbf{have } l: \ l > 0 \ \textbf{using } \textit{period-nonempty}[OF \ \textit{nonsquare}] \ \textbf{by } (\textit{auto } \textit{simp: } l\text{-}def) \\ \textbf{from } lm \ \textbf{have } l \ast m \neq 1 \ \textbf{by } (\textit{intro notI}) \ \textit{auto} \\ \textbf{with } l \ m \ \textbf{have } lm': \ l \ast m > 1 \ \textbf{by } (\textit{cases } l \ast m) \ \textit{auto} \end{array}$ 

have  $(h \ (l * m - 1))^2 - D * (k \ (l * m - 1))^2 =$ 

 $(-1) \cap Suc \ (l * m - 1) * int \ (Q \ (l * m - 1)))$ using lm' by (intro conv-num-denom-pell) auto also have  $(-1) \cap Suc \ (l * m - 1) = (1 :: int)$ using lm l m by (subst neg-one-even-power) auto **also have** Q(l \* m - 1) = Q((l \* m - 1) mod l)unfolding Q-def l-def o-def by (subst sqrt-remainder-surd-periodic[OF nonsquare]) simp also { have l \* m - 1 = (m - 1) \* l + (l - 1)using  $m \ l \ lm'$  by (cases m) (auto simp: mult-ac) also have ...  $mod \ l = (l - 1) \ mod \ l$ by simp also have  $\ldots = l - 1$ using l by (intro mod-less) auto also have  $Q \ldots = 1$ using sqrt-remainder-surd-last [OF nonsquare] by (simp add: Q-def l-def) finally have  $Q((l * m - 1) \mod l) = 1$ . finally have  $h(l * m - 1) \hat{2} = D * k(l * m - 1) \hat{2} + 1$ **unfolding** of-nat-Suc by (simp add: algebra-simps) hence  $h(l * m - 1) \hat{2} = D * k(l * m - 1) \hat{2} + 1$ **by** (*simp only: of-nat-eq-iff*) moreover have k (l \* m - 1) > 0**unfolding** *k*-*def* **by** (*intro conv*-*denom*-*pos*) **ultimately have** nontriv-solution (int x, int y) using xy by (simp add: nontriv-solution-def) **thus** nontriv-solution (x, y)by simp  $\mathbf{next}$ **assume** nontriv-solution (x, y)hence asm:  $x \uparrow 2 = Suc (D * y \uparrow 2) y > 0$ by (auto simp: nontriv-solution-def abs-square-eq-1 introl: Nat.gr0I) from asm have asm': int  $x \uparrow 2 = int D * int y \uparrow 2 + 1$ by (metis add.commute of-nat-Suc of-nat-mult of-nat-power-eq-of-nat-cancel-iff) have l: l > 0 using period-nonempty[OF nonsquare] by (auto simp: l-def) from *pell-solution-is-conv*[OF asm] obtain m where xy: h m = x k m = y by (auto simp: c-def h-def k-def) have  $m: m > \theta$ using asm' conv-num-denom-pell(1) xy by (intro Nat.gr0I) auto have  $1 = h m \hat{2} - D * k m \hat{2}$ using *asm'* xy by *simp* also have  $\ldots = (-1) \cap Suc \ m * int \ (Q \ m)$ using conv-num-denom-pell(2)[OF m]. finally have  $*: (-1) \cap Suc \ m * int \ (Q \ m) = 1$ . from \* have m': odd  $m \land Q m = 1$ **by** (cases even m) auto

define n where n = Suc m div l

have  $l \, dvd \, Suc \, m$ **proof** (rule ccontr) assume \*:  $\neg(l \ dvd \ Suc \ m)$ have  $Q m = Q (m \mod l)$ unfolding Q-def l-def o-def by (subst sqrt-remainder-surd-periodic[OF nonsquare]) simp also { have  $m \mod l < l \text{ using } \langle l > 0 \rangle$  by simpmoreover have Suc  $(m \mod l) \neq l$  using  $* l \langle m > 0 \rangle$ using mod-Suc[of m l] by auto ultimately have  $m \mod l < l - 1$  by simphence  $Q \ (m \ mod \ l) > 1$  unfolding Q-def o-def l-def **by** (rule snd-sqrt-remainder-surd-gt-1[OF nonsquare]) } finally show False using m' by simp qed hence *m*-eq: Suc m = n \* l m = n \* l - 1by (simp-all add: n-def) hence n > 0 by (auto introl: Nat.gr0I) **thus**  $\exists n > 0$ . int  $x = h (l * n - 1) \land$  int  $y = k (l * n - 1) \land$  even (l \* n)using xy m-eq m' by (intro exI[of - n]) (auto simp: mult-ac) qed

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-sol = (nat (h n), nat (k n))**proof** (*rule fund-sol-eq-sndI*) have [simp]:  $h \ n \ge 0 \ k \ n \ge 0$  for n**by** (*auto simp: h-def k-def c-def intro*!: *conv-num-nonneg*) **show** nontriv-solution (nat  $(h \ n)$ , nat  $(k \ n)$ ) by (subst nontriv-solution-iff-conv-num-denom, rule exI[of - if even l then 1 else 2])(simp-all add: n-def mult-ac) next fix x y :: nat assume nontriv-solution (x, y)then obtain m where m: m > 0 x = h (l \* m - 1) y = k (l \* m - 1) even (l\* mby (subst (asm) nontriv-solution-iff-conv-num-denom) auto have l: l > 0 using period-nonempty[OF nonsquare] by (auto simp: l-def) from  $m \ l$  have  $Suc \ n \leq l * m$  by (auto simp: n-def) hence n < l \* m - 1 by simp hence  $k n \leq k (l * m - 1)$ **unfolding** *k*-*def c*-*def* **using** *irrat-sqrt-nonsquare*[*OF nonsquare*] by (intro conv-denom-leI) auto with m show nat  $(k \ n) \leq y$  by simp qed

#### 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 infoin if l = 0 then (0, 0) else letc = 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 have \*: is-cfrac (cfrac-sqrt-nth (sqrt-cfrac-info-array D)) if  $\neg$ is-square D using that cfrac-sqrt-nth[of D] unfolding is-cfrac-def by (metis cfrac-nth-nonzero neq0-conv of-nat-0 of-nat-0-less-iff) have \*\*:  $cfrac(\lambda x. int(cfrac-sqrt-nth(sqrt-cfrac-info-array D)x)) = cfrac-of-real$ (sqrt D)if  $\neg is$ -square D using that cfrac-sqrt-nth[of D] \* by (intro cfrac-eqI) auto **show** ?thesis using \* \*\* by (auto simp: square-test-correct find-fund-sol-correct conv-num-fun-eq conv-denom-fun-eq Let-def cfrac-sqrt-nth fund-sol-conv-num-denom conv-num-nonneg) qed **lemma** find-nth-solution-square [simp]: is-square  $D \implies$  find-nth-solution D = n $(\theta, \theta)$ by (simp add: find-nth-solution-def) **lemma** fst-find-fund-sol-eq-0-iff [simp]: fst (find-fund-sol D) =  $0 \iff is$ -square D **proof** (cases is-square D) case False then interpret *pell D* by *unfold-locales* from False have find-fund-sol D = fund-sol by (simp add: find-fund-sol-correct) moreover from fund-sol-is-nontriv-solution have fst fund-sol > 0**by** (*auto simp: nontriv-solution-def intro*!: *Nat.gr0I*) ultimately show *?thesis* using *False* 

by (simp add: find-fund-sol-def square-test-correct split: if-splits)

 ${\bf qed} \ (auto \ simp: \ find-fund-sol-def \ square-test-correct)$ 

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

in if fst xy = 0 then (0, 0) else efficient-pell-power D xy n) **proof** (cases is-square D) case False then interpret *pell* D by *unfold-locales* from fund-sol-is-nontriv-solution have fst fund-sol > 0**by** (*auto simp: nontriv-solution-def intro*!: *Nat.gr0I*) thus ?thesis using False by (simp add: find-nth-solution-correct Let-def nth-solution-def pell-power-def pell-mul-commutes[of - fund-sol] find-fund-sol-correct)  $\mathbf{qed} \ auto$ **lemma** *nth-solution-code* [*code*]: pell.nth-solution D n =(let info = sqrt-cfrac-info-array D;l = fst infoin if l = 0 then Code.abort (STR "nth-solution is undefined for perfect square parameter.")  $(\lambda$ -. pell.nth-solution D n) else letc = cfrac-sqrt-nth info; m = if even l then l - 1 else 2 \* l - 1; $fund-sol = (nat (conv-num-fun \ c \ m), nat (conv-denom-fun \ c \ m))$ inefficient-pell-power D fund-sol n) **proof** (cases is-square D) case False then interpret *pell* by *unfold-locales* **have** \*: *is-cfrac* (*cfrac-sqrt-nth* (*sqrt-cfrac-info-array* D)) using False cfrac-sqrt-nth[of D] unfolding is-cfrac-def by (metis cfrac-nth-nonzero neq0-conv of-nat-0 of-nat-0-less-iff) have \*\*:  $cfrac(\lambda x. int(cfrac-sqrt-nth(sqrt-cfrac-info-array D)x)) = cfrac-of-real$ (sqrt D)using False cfrac-sqrt-nth[of D] \* by (intro cfrac-eqI) auto from False \* \*\* show ?thesis by (auto simp: Let-def cfrac-sqrt-nth fund-sol-conv-num-denom nth-solution-def pell-power-def pell-mul-commutes[of - (-, -)] *conv-num-fun-eq conv-denom-fun-eq conv-num-nonneq*) qed auto **lemma** fund-sol-code [code]: pell.fund-sol D = (let info = sqrt-cfrac-info-array D;l = fst infoin if l = 0 then Code.abort (STR "fund-sol is undefined for perfect square parameter.")  $(\lambda$ -. 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** (cases is-square D) case False then interpret *pell* by *unfold-locales* **have** \*: *is-cfrac* (*cfrac-sqrt-nth* (*sqrt-cfrac-info-array* D)) using False cfrac-sqrt-nth[of D] unfolding is-cfrac-def by (metis cfrac-nth-nonzero neq0-conv of-nat-0 of-nat-0-less-iff) **have** \*\*:  $cfrac (\lambda x. int (cfrac-sqrt-nth (sqrt-cfrac-info-array D) x)) = cfrac-of-real$ (sqrt D)using False cfrac-sqrt-nth[of D] \* by (intro cfrac-eqI) auto from False \* \*\* show ?thesis by (auto simp: Let-def cfrac-sqrt-nth fund-sol-conv-num-denom nth-solution-def pell-power-def pell-mul-commutes[of - (-, -)] *conv-num-fun-eq conv-denom-fun-eq conv-num-nonneg*) qed auto

 $\mathbf{end}$ 

# 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

 ${\bf code-lazy-type}\ stream$ 

**lemma** lnth-code [code]:  $lnth xs \ 0 = (if \ lnull \ xs \ then \ undefined \ (0 :: nat) \ else \ lhd \ xs)$   $lnth \ xs \ (Suc \ n) = (if \ lnull \ xs \ then \ undefined \ (Suc \ n) \ else \ lnth \ (ltl \ xs) \ n)$ **by** (auto simp:  $lnth.simps \ split: \ llist.splits)$ 

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) = fold  $(\lambda(x, y) x')$  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  $\circ$  find-fund-sol) [0..<1001]

The same for D < 100000:

value [code] arg-max-list (fst  $\circ$  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- $aux :: nat \Rightarrow nat \Rightarrow nat$  where log10-aux acc n =  $(if n \ge 10000000000 then log10$ -aux (acc + 10) (n div 1000000000)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

 $\mathbf{end}$ 

### 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  $\Rightarrow$  int  $\times$  int where float-to-rat  $f = (if exponent f \ge 0 then$ 

(mantissa  $f * 2 \cap nat$  (exponent f), 1) else (mantissa  $f, 2 \cap nat$  (-exponent f))))

**lemma** float-to-rat: fst (float-to-rat f) / snd (float-to-rat f) = real-of-float f by (auto simp: float-to-rat-def mantissa-exponent powr-int)

**lemma** snd-float-to-rat-pos [simp]: snd (float-to-rat f) > 0 by (simp add: float-to-rat-def)

**function** cfrac-from-approx :: int  $\times$  int  $\Rightarrow$  int  $\times$  int  $\Rightarrow$  int list where cfrac-from-approx (nl, dl) (nu, du) = (if  $nl = 0 \lor nu = 0 \lor dl = 0 \lor du = 0$  then [] else let l = nl div dl; u = nu div du in if  $l \neq u$  then [] else  $l \notin$  (let m = nl mod dl in if m = 0 then [] else cfrac-from-approx (du, nu mod du) (dl, m)))

by *auto* termination proof (relation measure ( $\lambda((nl, dl), (nu, du))$ ). nat (abs dl + abs du)), goal-cases) case (2 nl dl nu du)hence  $|nl \mod dl| + |nu \mod du| < |dl| + |du|$ **by** (*intro* add-strict-mono) (*auto* simp: abs-mod-less) thus ?case using 2 by simp qed auto **lemmas**  $[simp \ del] = cfrac-from-approx.simps$ **lemma** cfrac-from-approx-correct: **assumes**  $x \in \{fst \ l \ / \ snd \ l..fst \ u \ / \ snd \ u\}$  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 ! iusing assms **proof** (*induction l u arbitrary: i x rule: cfrac-from-approx.induct*) case  $(1 \ nl \ dl \ nu \ du \ i \ x)$ **from** 1.prems have \*: nl div dl = nu div du nl  $\neq 0$  nu  $\neq 0$  dl > 0 du > 0  $\mathbf{by} \ (auto \ simp: \ cfrac-from-approx.simps \ Let-def \ split: \ if-splits)$ have  $\lfloor nl / dl \rfloor \leq \lfloor x \rfloor \lfloor x \rfloor \leq \lfloor nu / du \rfloor$ using 1.prems(1) by (intro floor-mono; simp)+**hence**  $nl \ div \ dl \leq \lfloor x \rfloor \lfloor x \rfloor \leq nu \ div \ du$ **by** (*simp-all add: floor-divide-of-int-eq*) with \* have  $|x| = nu \ div \ du$ by *linarith* show ?case **proof** (cases i) case  $\theta$ with 0 and  $\langle |x| = \rightarrow$  show ?thesis using 1.prems **by** (*auto simp: Let-def cfrac-from-approx.simps*) next case [simp]: (Suc i') from 1.prems \* have  $nl \mod dl \neq 0$ by (subst (asm) cfrac-from-approx.simps) (auto split: if-splits) have frac-eq: frac x = x - nu div duusing  $\langle |x| = \rightarrow$  by (simp add: frac-def) have frac  $x \ge nl / dl - nl div dl$ **using** \* 1.prems **by** (simp add: frac-eq) also have nl / dl - nl div dl = (nl - dl \* (nl div dl)) / dl**using** \* **by** (*simp add: field-simps*) also have  $nl - dl * (nl \, div \, dl) = nl \, mod \, dl$ **by** (subst minus-div-mult-eq-mod [symmetric]) auto finally have frac  $x \ge (nl \mod dl) / dl$ . have  $nl \mod dl \ge 0$ 

**using** \* **by** (*intro pos-mod-sign*) *auto* with  $\langle nl \mod dl \neq 0 \rangle$  have  $nl \mod dl > 0$ by *linarith* hence  $0 < (nl \mod dl) / dl$ **using** \* **by** (*intro divide-pos-pos*) *auto* also have  $\ldots \leq frac x$ by fact finally have frac x > 0. have frac  $x \leq nu / du - nu div du$ using \* 1.prems by (simp add: frac-eq) also have  $\ldots = (nu - du * (nu \ div \ du)) / du$ **using** \* **by** (*simp add: field-simps*) also have  $nu - du * (nu \ div \ du) = nu \ mod \ du$ by (subst minus-div-mult-eq-mod [symmetric]) auto finally have frac  $x \leq real-of-int (nu \mod du) / real-of-int du$ . have  $\theta < frac x$ by fact also have  $\ldots \leq (nu \mod du) \ / \ du$ by fact finally have  $nu \mod du > 0$ **using** \* **by** (*auto simp: field-simps*) have cfrac-nth (cfrac-of-real x) i = cfrac-nth (cfrac-tl (cfrac-of-real x)) i'by simp also have cfrac-tl (cfrac-of-real x) = cfrac-of-real (1 / frac x) using  $\langle frac \ x > 0 \rangle$  by (intro cfrac-tl-of-real) auto also have cfrac-nth (cfrac-of-real (1 / frac x))) i' =cfrac-from-approx (du, nu mod du) (dl, nl mod dl) ! i'**proof** (rule 1.IH[OF - refl refl - refl]) show  $\neg$   $(nl = 0 \lor nu = 0 \lor dl = 0 \lor du = 0) \neg nl div dl \neq nu div du$ using 1.prems by (auto split: if-splits simp: Let-def cfrac-from-approx.simps)  $\mathbf{next}$ show i' < length (cfrac-from-approx (du, nu mod du) (dl, nl mod dl)) using 1.prems by (subst (asm) cfrac-from-approx.simps) (auto split: if-splits simp: Let-def)  $\mathbf{next}$ have 1 / frac  $x \leq dl$  / (nl mod dl) using  $\langle frac \ x > 0 \rangle$  and  $\langle nl \ mod \ dl > 0 \rangle$  and  $\langle frac \ x \ge (nl \ mod \ dl) \ / \ dl \rangle$ and \* **by** (*auto simp: field-simps*) moreover have 1 / frac  $x \ge du$  / (nu mod du) using  $\langle frac \ x > 0 \rangle$  and  $\langle nu \ mod \ du > 0 \rangle$  and  $\langle frac \ x \leq (nu \ mod \ du) \ / \ du \rangle$ and \* by (auto simp: field-simps) ultimately show 1 / frac  $x \in \{\text{real-of-int (fst (du, nu mod du))} / \text{real-of-int (snd (du, nu mod du))} \}$ mod du))..

real-of-int (fst (dl, nl mod dl)) / real-of-int (snd (dl, nl mod  $dl))\}$ by simp show snd  $(du, nu \mod du) > 0$  snd  $(dl, nl \mod dl) > 0$  and  $nl \mod dl \neq 0$ using  $\langle nu \mod du > 0 \rangle$  and  $\langle nl \mod dl > 0 \rangle$  by simp-all qed also have cfrac-from-approx (du, nu mod du) (dl, nl mod dl) ! i' =cfrac-from-approx (nl, dl) (nu, du) ! iusing 1.prems \* (nl mod dl  $\neq 0$ ) by (subst (2) cfrac-from-approx.simps) autofinally show ?thesis . qed qed **definition** *cfrac-from-approx'* :: *float*  $\Rightarrow$  *float*  $\Rightarrow$  *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  $x \in \{real \text{-} of \text{-} float \ l..real \text{-} of \text{-} float \ u\}$ assumes i < length (cfrac-from-approx' l u) **shows** cfrac-nth (cfrac-of-real x) i = cfrac-from-approx' l u ! iusing assms unfolding cfrac-from-approx'-def by (intro cfrac-from-approx-correct) (auto simp: float-to-rat cfrac-from-approx'-def) **definition** approx-cfrac ::  $nat \Rightarrow floatarith \Rightarrow int list$  where  $approx-cfrac \ prec \ e =$ (case approx' prec e [] ofNone  $\Rightarrow$  [] | Some  $ivl \Rightarrow cfrac-from-approx'$  (lower ivl) (upper ivl)) **ML-file** *(approximation-cfrac.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  $(\lambda n. (conv-num-fun ((!) c) n, conv-denom-fun ((!) c) n)) [0..< length$ 

```
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)
```

# References

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

 $\mathbf{end}$