

Continued Fractions

Manuel Eberl

May 14, 2024

Abstract

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

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

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

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

Contents

1	Continued Fractions	3
1.1	Auxiliary results	3
1.2	Bounds on alternating decreasing sums	7
1.3	Non-canonical continued fractions	47
1.4	Approximation properties	62
1.5	Efficient code for convergents	79
1.6	Computing the continued fraction expansion of a rational number	80
2	Quadratic Irrationals	83
2.1	Basic results on rationality of square roots	84
2.2	Definition of quadratic irrationals	86
2.3	Real solutions of quadratic equations	89
2.4	Periodic continued fractions and quadratic irrationals	91
3	The continued fraction expansion of e	104
4	Continued fraction expansions for square roots of naturals	112
5	Lifting solutions of Pell's Equation	135
5.1	Auxiliary material	135
5.2	The lifting mechanism	139
5.3	Accelerated computation of the fundamental solution for non- squarefree inputs	140
6	The Connection between the continued fraction expansion of square roots and Pell's equation	143
7	Tests for Continued Fractions of Square Roots and Pell's Equation	152
7.1	Fundamental solutions of Pell's equation	153
7.2	Tests for other operations	153
8	Computing continued fraction expansions through interval arithmetic	153

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 lfinite :: 'a llist  $\Rightarrow$  bool where
  lfinite xs  $\Longrightarrow$  lfinite (LCons x xs)

```

```

lemma llength-llist-of-stream [simp]: llength (llist-of-stream xs) =  $\infty$ 
by (simp add: not-lfinite-llength)

```

```

lemma lfinite-conv-llength: lfinite xs  $\longleftrightarrow$  llength xs =  $\infty$ 

```

proof

```

  assume lfinite xs
  thus llength xs =  $\infty$ 
  proof (coinduction arbitrary: xs rule: enat-coinduct2)
    fix xs :: 'a llist
    assume llength xs  $\neq$  0 lfinite xs
    thus ( $\exists$  xs'::'a llist. epred (llength xs) = llength xs'  $\wedge$  epred  $\infty$  =  $\infty$   $\wedge$  lfinite
xs')  $\vee$ 
      epred (llength xs) = epred  $\infty$ 
    by (intro disjI1 exI[of - ltl xs]) (auto simp: lfinite.simps[of xs])
  next
    fix xs :: 'a llist assume lfinite xs thus (llength xs = 0)  $\longleftrightarrow$  ( $\infty$  = (0::enat))
    by (subst (asm) lfinite.simps) auto
  qed
next
  assume llength xs =  $\infty$ 
  thus lfinite xs
  proof (coinduction arbitrary: xs)
    case lfinite
    thus  $\exists$  xsa x.
      xs = LCons x xsa  $\wedge$ 
      ( $\exists$  xs. xsa = xs  $\wedge$  llength xs =  $\infty$ )  $\vee$ 
      lfinite 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 0
  thus ?case
  by (cases xs) (auto simp: lnth-default-def simp flip: zero-enat-def)
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
qed

```

```

lemma enat-le-iff:
  enat n ≤ m ⟷ m = ∞ ∨ (∃ m'. m = enat m' ∧ n ≤ m')
  by (cases m) auto

```

```

lemma enat-less-iff:
  enat n < m ⟷ m = ∞ ∨ (∃ m'. m = enat m' ∧ n < m')
  by (cases m) auto

```

```

lemma real-of-int-divide-in-Ints-iff:
  real-of-int a / real-of-int b ∈ ℤ ⟷ b dvd a ∨ b = 0
proof safe
  assume real-of-int a / real-of-int b ∈ ℤ b ≠ 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
qed auto

```

```

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

```

lemma *frac-add-of-int*: $\text{frac } (\text{of-int } y + x) = \text{frac } x$
unfolding *frac-def* **by** *simp*

lemma *frac-fraction*: $\text{frac } (\text{real-of-int } a / \text{real-of-int } b) = (a \bmod b) / b$
proof –
have $\text{frac } (a / b) = \text{frac } ((a \bmod b + b * (a \text{ div } b)) / b)$
by (*subst mod-mult-div-eq*) *auto*
also have $(a \bmod b + b * (a \text{ div } b)) / b = \text{of-int } (a \text{ div } b) + a \bmod b / b$
unfolding *of-int-add* **by** (*subst add-divide-distrib*) *auto*
also have $\text{frac } \dots = \text{frac } (a \bmod b / b)$
by (*rule frac-add-of-int*)
also have $\dots = a \bmod b / b$
by (*simp add: floor-divide-of-int-eq frac-def*)
finally show *?thesis* .
qed

lemma *Suc-fib-ge*: $\text{Suc } (\text{fib } n) \geq n$
proof (*induction n rule: fib.induct*)
case ($\exists n$)
show *?case*
proof (*cases n < 2*)
case *True*
thus *?thesis* **by** (*cases n*) *auto*
next
case *False*
hence $\text{Suc } (\text{Suc } (\text{Suc } n)) \leq \text{Suc } n + n$ **by** *simp*
also have $\dots \leq \text{Suc } (\text{fib } (\text{Suc } n)) + \text{Suc } (\text{fib } n)$
by (*intro add-mono 3*)
also have $\dots = \text{Suc } (\text{Suc } (\text{fib } (\text{Suc } (\text{Suc } n))))$
by *simp*
finally show *?thesis* **by** (*simp only: Suc-le-eq*)
qed
qed *auto*

lemma *fib-ge*: $\text{fib } n \geq n - 1$
using *Suc-fib-ge[of n]* **by** *simp*

lemma *frac-diff-of-nat-right* [*simp*]: $\text{frac } (x - \text{of-nat } y) = \text{frac } x$
using *floor-diff-of-int[of x int y]* **by** (*simp add: frac-def*)

lemma *of-nat-ge-1-iff*: $\text{of-nat } n \geq (1 :: 'a :: \text{linordered-semidom}) \longleftrightarrow n > 0$
using *of-nat-le-iff[of 1 n]* **unfolding** *of-nat-1* **by** *auto*

lemma *not-frac-less-0*: $\neg \text{frac } x < 0$
by (*simp add: frac-def not-less*)

lemma *frac-le-1*: $\text{frac } x \leq 1$
unfolding *frac-def* **by** *linarith*

lemma *divide-in-Rats-iff1*:
 $(x::real) \in \mathbb{Q} \implies x \neq 0 \implies x / y \in \mathbb{Q} \longleftrightarrow y \in \mathbb{Q}$
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*:
 $(y::real) \in \mathbb{Q} \implies y \neq 0 \implies x / y \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
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*: $x \in \mathbb{Q} \implies x + y \in \mathbb{Q} \longleftrightarrow y \in \mathbb{Q}$
 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*: $x \in \mathbb{Q} \implies x - y \in \mathbb{Q} \longleftrightarrow y \in \mathbb{Q}$
 using *Rats-diff*[of $x \ x - y$] by *auto*

lemma *diff-in-Rats-iff2*: $y \in \mathbb{Q} \implies x - y \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
 using *Rats-add*[of $x - y \ y$] by *auto*

lemma *frac-in-Rats-iff* [*simp*]: $\text{frac } x \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
 by (simp add: *frac-def diff-in-Rats-iff2*)

lemma *filterlim-sequentially-shift*:
 $\text{filterlim } (\lambda n. f (n + m)) \ F \text{ sequentially} \longleftrightarrow \text{filterlim } f \ F \text{ sequentially}$
proof (*induction m*)
 case (*Suc m*)
 have $\text{filterlim } (\lambda n. f (n + \text{Suc } m)) \ F \text{ at-top} \longleftrightarrow$
 $\text{filterlim } (\lambda n. f (\text{Suc } n + m)) \ F \text{ at-top}$ by *simp*
 also have $\dots \longleftrightarrow \text{filterlim } (\lambda n. f (n + m)) \ F \text{ at-top}$
 by (rule *filterlim-sequentially-Suc*)
 also have $\dots \longleftrightarrow \text{filterlim } f \ F \text{ at-top}$
 by (rule *Suc.IH*)
 finally show ?case .
qed *simp-all*

1.2 Bounds on alternating decreasing sums

lemma *alternating-decreasing-sum-bounds*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$

assumes $m \leq n \wedge k. k \in \{m..n\} \implies f\ k \geq 0$

$\wedge k. k \in \{m..<n\} \implies f\ (\text{Suc}\ k) \leq f\ k$

defines $S \equiv (\lambda m. (\sum_{k=m..n}. (-1)^\wedge k * f\ k))$

shows *if even m then S m ∈ {0..f m} else S m ∈ {-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*]: $S\ m' = (-1)^\wedge m' * f\ m' + S\ (\text{Suc}\ m')$

using *step.hyps unfolding S-def*

by (*subst sum.atLeast-Suc-atMost simp-all*)

from *step.hyps* **have** *nonneg: f m' ≥ 0*

by (*intro assms*) *auto*

from *step.hyps* **have** *mono: f (Suc m') ≤ f m'*

by (*intro assms*) *auto*

show *?case*

proof (*cases even m'*)

case *True*

hence $0 \leq f\ (\text{Suc}\ m') + S\ (\text{Suc}\ m')$

using *step.IH* **by** *simp*

also note *mono*

finally show *?thesis* **using** *True step.IH* **by** *auto*

next

case *False*

with *step.IH* **have** $S\ (\text{Suc}\ m') \leq f\ (\text{Suc}\ m')$

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 $f :: \text{nat} \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$

assumes $m < n \wedge k. k \in \{m..n-1\} \implies f\ k \geq 0$

$\wedge k. k \in \{m..<n-1\} \implies f\ (\text{Suc}\ k) \leq f\ k$

defines $S \equiv (\lambda m. (\sum_{k=m..<n}. (-1)^\wedge k * f\ k))$

shows *if even m then S m ∈ {0..f m} else S m ∈ {-f m..0}*

proof (*cases n*)

case *0*

thus *?thesis* **using** *assms* **by** *auto*

next

case (*Suc n'*)

hence *if even m then* $(\sum_{k=m..n-1}. (-1)^\wedge k * f\ k) \in \{0..f\ m\}$

else $(\sum_{k=m..n-1}. (-1)^\wedge k * f\ k) \in \{-f\ m..0\}$

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

also have $(\sum_{k=m..n-1}. (-1) \wedge k * f k) = S m$
 unfolding S -def by (intro sum.cong) (auto simp: Suc)
 finally show ?thesis .
 qed

lemma *alternating-decreasing-sum-upper-bound*:
 fixes $f :: nat \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$
 assumes $m \leq n \wedge k. k \in \{m..n\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n\} \implies f (Suc k) \leq f k$
 shows $(\sum_{k=m..n}. (-1) \wedge k * f k) \leq f m$
 using *alternating-decreasing-sum-bounds*[of $m n f$, $OF \text{ assms}$] *assms*(1)
 by (auto split: if-splits intro: order.trans[$OF - \text{assms}(2)$])

lemma *alternating-decreasing-sum-upper-bound'*:
 fixes $f :: nat \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$
 assumes $m < n \wedge k. k \in \{m..n-1\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n-1\} \implies f (Suc k) \leq f k$
 shows $(\sum_{k=m..<n}. (-1) \wedge k * f k) \leq f m$
 using *alternating-decreasing-sum-bounds'*[of $m n f$, $OF \text{ assms}$] *assms*(1)
 by (auto split: if-splits intro: order.trans[$OF - \text{assms}(2)$])

lemma *abs-alternating-decreasing-sum-upper-bound*:
 fixes $f :: nat \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$
 assumes $m \leq n \wedge k. k \in \{m..n\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n\} \implies f (Suc k) \leq f k$
 shows $|(\sum_{k=m..n}. (-1) \wedge k * f k)| \leq f m$ (is abs ? $S \leq -$)
 using *alternating-decreasing-sum-bounds*[of $m n f$, $OF \text{ assms}$]
 by (auto split: if-splits simp: minus-le-iff)

lemma *abs-alternating-decreasing-sum-upper-bound'*:
 fixes $f :: nat \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$
 assumes $m < n \wedge k. k \in \{m..n-1\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n-1\} \implies f (Suc k) \leq f k$
 shows $|(\sum_{k=m..<n}. (-1) \wedge k * f k)| \leq f m$
 using *alternating-decreasing-sum-bounds'*[of $m n f$, $OF \text{ assms}$]
 by (auto split: if-splits simp: minus-le-iff)

lemma *abs-alternating-decreasing-sum-lower-bound*:
 fixes $f :: nat \Rightarrow 'a :: \{\text{linordered-ring}, \text{ring-1}\}$
 assumes $m < n \wedge k. k \in \{m..n\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n\} \implies f (Suc k) \leq f k$
 shows $|(\sum_{k=m..n}. (-1) \wedge k * f k)| \geq f m - f (Suc m)$
proof –
 have $(\sum_{k=m..n}. (-1) \wedge k * f k) = (\sum_{k \in \text{insert } m \{m<..n\}}. (-1) \wedge k * f k)$
 using *assms* by (intro sum.cong) auto
 also have $\dots = (-1) \wedge m * f m + (\sum_{k \in \{m<..n\}}. (-1) \wedge k * f k)$
 by auto
 also have $(\sum_{k \in \{m<..n\}}. (-1) \wedge k * f k) = (\sum_{k \in \{m..<n\}}. (-1) \wedge Suc k * f (Suc k))$
 (Suc k))

by (intro sum.reindex-bij-witness[of - Suc $\lambda i. i - 1$]) auto
 also have $(-1)^{\wedge m} * f m + \dots = (-1)^{\wedge m} * f m - (\sum k \in \{m..<n\}. (-1)^{\wedge k} * f (Suc k))$
 by (simp add: sum-negf)
 also have $|\dots| \geq |(-1)^{\wedge m} * f m| - |(\sum k \in \{m..<n\}. (-1)^{\wedge k} * f (Suc k))|$
 by (rule abs-triangle-ineq2)
 also have $|(-1)^{\wedge m} * f m| = f m$
 using assms by (cases even m) auto
 finally have $f m - |\sum k = m..<n. (-1)^{\wedge k} * f (Suc k)|$
 $\leq |\sum k = m..n. (-1)^{\wedge k} * f k|$.
 moreover have $f m - |(\sum k \in \{m..<n\}. (-1)^{\wedge k} * f (Suc k))| \geq f m - f (Suc m)$
 using assms by (intro diff-mono abs-alternating-decreasing-sum-upper-bound') auto
 ultimately 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 $m+1 < n \wedge k. k \in \{m..n\} \implies f k \geq 0$
 $\wedge k. k \in \{m..<n\} \implies f (Suc k) \leq f k$
 shows $|(\sum k=m..<n. (-1)^{\wedge k} * f k)| \geq f m - f (Suc m)$
proof (cases n)
 case 0
 thus ?thesis using assms by auto
next
 case (Suc n')
 hence $|(\sum k=m..n-1. (-1)^{\wedge k} * f k)| \geq f m - f (Suc m)$
 using assms by (intro abs-alternating-decreasing-sum-lower-bound) auto
 also have $(\sum k=m..n-1. (-1)^{\wedge k} * f k) = (\sum k=m..<n. (-1)^{\wedge k} * f k)$
 by (intro sum.cong) (auto simp: Suc)
 finally show ?thesis .
 qed

lemma *alternating-decreasing-suminf-bounds*:
 assumes $\wedge k. f k \geq (0 :: real) \wedge k. f (Suc k) \leq f k$
 $f \longrightarrow 0$
 shows $(\sum k. (-1)^{\wedge k} * f k) \in \{f 0 - f 1..f 0\}$
proof -
 have summable $(\lambda k. (-1)^{\wedge k} * f k)$
 by (intro summable-Leibniz' assms)
 hence $\lim: (\lambda n. \sum k \leq n. (-1)^{\wedge k} * f k) \longrightarrow (\sum k. (-1)^{\wedge k} * f k)$
 by (auto dest: summable-LIMSEQ')
 have bounds: $(\sum k=0..n. (-1)^{\wedge k} * f k) \in \{f 0 - f 1..f 0\}$
 if $n > 0$ 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 $(\sum k. (-1)^\wedge k * f k) \geq f 0 - f 1$
 by (rule *tendsto-lowerbound*) (insert *bounds*, *auto*)
 moreover from *lim* have $(\sum k. (-1)^\wedge k * f k) \leq f 0$
 by (rule *tendsto-upperbound*) (use *bounds* in *auto*)
 ultimately show *?thesis* by *simp*
 qed

lemma

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

proof –

have *summable*: *summable* $(\lambda k. (-1)^\wedge k * f (k + m))$
 using *assms* by (intro *summable-Leibniz'*) (auto simp: *filterlim-sequentially-shift*)
 thus *summable* $(\lambda k. (-1)^\wedge (k + m) * f (k + m))$
 by (subst *add.commute*) (auto simp: *power-add mult.assoc intro: summable-mult*)
 have $S = (\sum k. (-1)^\wedge m * ((-1)^\wedge k * f (k + m)))$
 by (simp add: *S-def power-add mult-ac*)
 also have $\dots = (-1)^\wedge m * (\sum k. (-1)^\wedge k * f (k + m))$
 using *summable* by (rule *suminf-mult*)
 finally have $S = (-1)^\wedge m * (\sum k. (-1)^\wedge k * f (k + m))$.
 moreover have $(\sum k. (-1)^\wedge k * f (k + m)) \in$
 $\{f (0 + m) - f (1 + m) .. f (0 + m)\}$
 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*)

qed

lemma

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

proof –

define *S'* where $S' = (\sum k. (-1)^\wedge (k + \text{Suc } (\text{Suc } m)) * f (k + \text{Suc } (\text{Suc } m)))$

```

have ( $\lambda k. (-1) \wedge (k + m) * f (k + m)$ ) 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':  $S' = S - (-1) \wedge m * (f m - f (Suc m))$ 
  by (simp-all add: sums-iff S'-def algebra-simps lessThan-nat-numeral)
  have if even (Suc (Suc m)) then S' ∈ {f (Suc (Suc m)) - f (Suc (Suc (Suc m)))..f (Suc (Suc m))}
    else S' ∈ {-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 ( $\lambda x. \text{case } x \text{ of } CFrac \ a \ b \Rightarrow (a, b)$ ) ( $\lambda(x,y). CFrac \ x \ y$ ) 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
   $\lambda xs. (shd \ xs, llist-of-stream (smap (\lambda x. \text{nat } (x - 1)) (stl \ xs)))$  .

```

```

instantiation cfrac :: zero
begin
definition zero-cfrac where  $0 = \text{cfrac-of-int } 0$ 
instance ..
end

```

```

instantiation cfrac :: one
begin
definition one-cfrac where  $1 = \text{cfrac-of-int } 1$ 
instance ..
end

```

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

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

lemma *cfrac-tl-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* $\hat{\sim}$ *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**

$\lambda(-, bs) \Rightarrow \text{length } bs .$

lemma *cfrac-length-code* [code]: *cfrac-length* (*CFrac* *a* *bs*) = *length* *bs*

by (simp add: *cfrac-length-def*)

lemma *cfrac-length-tl* [simp]: *cfrac-length* (*cfrac-tl* *c*) = *cfrac-length* *c* - 1

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

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

by (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*) = ∞

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 *length* *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  $\longleftrightarrow$  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  $\geq$  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  $\geq$  1
  by transfer auto

lemma cfrac-nth-not-less-1[simp, intro]: n > 0  $\implies$   $\neg$ cfrac-nth c n < 1
  by 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 = n
  by transfer auto

lemma cfrac-nth-gt0-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 n
  using 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**
 $\lambda f. (f\ 0, \text{inf-llist } (\lambda n. \text{nat } (f\ (\text{Suc } n) - 1)))$.

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

lemma $cfrac\text{-nth-cfrac}$ [simp]:
assumes $is-cfrac\ f$
shows $cfrac\text{-nth } (cfrac\ f)\ n = f\ n$
using $assms$ **unfolding** $is-cfrac\text{-def}$ **by** $transfer\ auto$

lemma $llength\text{-eq-infity-lnth}$: $llength\ b = \infty \implies \text{inf-llist } (\text{lnth } b) = b$
by ($\text{simp add: } llength\text{-eq-infity-conv-lfinite}$)

lemma $cfrac\text{-cfrac-nth}$ [simp]: $cfrac\text{-length } c = \infty \implies cfrac\ (cfrac\text{-nth } c) = c$
by $transfer\ (auto\ simp: llength\text{-eq-infity-lnth})$

lemma $cfrac\text{-length-cfrac}$ [simp]: $cfrac\text{-length } (cfrac\ f) = \infty$
by $transfer\ auto$

lift-definition $cfrac\text{-of-list} :: int\ list \Rightarrow cfrac$ **is**
 $\lambda xs. \text{if } xs = [] \text{ then } (0, \text{LNil}) \text{ else } (\text{hd } xs, \text{llist-of } (\text{map } (\lambda n. \text{nat } n - 1) (\text{tl } xs)))$.

lemma $cfrac\text{-length-of-list}$ [simp]: $cfrac\text{-length } (cfrac\text{-of-list } xs) = \text{length } xs - 1$
by $transfer\ (auto\ simp: \text{zero-enat-def})$

lemma $cfrac\text{-of-list-Nil}$ [simp]: $cfrac\text{-of-list } [] = 0$
unfolding zero-cfrac-def **by** $transfer\ auto$

lemma $cfrac\text{-nth-of-list}$ [simp]:
assumes $n < \text{length } xs$ **and** $\forall i \in \{0 < .. < \text{length } xs\}. xs\ !\ i > 0$
shows $cfrac\text{-nth } (cfrac\text{-of-list } xs)\ n = xs\ !\ n$
using $assms$
proof ($transfer, \text{goal-cases}$)
case ($1\ n\ xs$)
show $?case$
proof ($\text{cases } n$)
case ($\text{Suc } n'$)
with 1 **have** $xs\ !\ n > 0$
using 1 **by** $auto$
hence $\text{int } (\text{nat } (\text{tl } xs\ !\ n') - \text{Suc } 0) + 1 = xs\ !\ \text{Suc } n'$
using $1(1)\ \text{Suc}$ **by** ($auto\ simp: \text{nth-tl of-nat-diff}$)
thus $?thesis$
using $\text{Suc } 1(1)$ **by** ($auto\ simp: \text{hd-conv-nth zero-enat-def}$)
qed ($\text{use } 1$ **in** $\langle auto\ simp: \text{hd-conv-nth} \rangle$)
qed

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

cfrac-of-real-aux $x =$
 (if $x \in \{0 < \dots < 1\}$ 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 $x > 0 \wedge x < 1$ 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]: $x \notin \{0 < \dots < 1\} \implies cfrac-of-real-aux \ x = LNil$
 by (subst *cfrac-of-real-aux.code*) auto

lemma *cfrac-of-real-aux-0* [simp]: *cfrac-of-real-aux* $0 = LNil$
 by (subst *cfrac-of-real-aux.code*) auto

lemma *cfrac-of-real-aux-eq-LNil-iff* [simp]: *cfrac-of-real-aux* $x = LNil \iff x \notin \{0 < \dots < 1\}$
 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

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

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*: $m = epred \ n \iff m = 0 \wedge n = 0 \vee n = eSuc \ m$
 by (cases m ; cases n) (auto simp: *enat-0-iff enat-eSuc-iff infinity-eq-eSuc-iff*)

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

lemma *epred-less*: $n > 0 \implies n \neq \infty \implies epred \ n < n$
 by (cases n) (auto simp: *enat-0-iff*)

lemma *cfrac-nth-of-real-0* [simp]:
cfrac-nth (*cfrac-of-real* *x*) 0 = $\lfloor x \rfloor$
by *transfer auto*

lemma *frac-eq-0* [simp]: $x \in \mathbb{Z} \implies \text{frac } x = 0$
by *simp*

lemma *cfrac-tl-of-real*:
assumes $x \notin \mathbb{Z}$
shows $\text{cfrac-tl } (\text{cfrac-of-real } x) = \text{cfrac-of-real } (1 / \text{frac } x)$
using *assms*
proof (*transfer, goal-cases*)
case (1 *x*)
hence $\text{int } (\text{nat } \lfloor 1 / \text{frac } x \rfloor - \text{Suc } 0) + 1 = \lfloor 1 / \text{frac } x \rfloor$
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 $\text{cfrac-nth } (\text{cfrac-of-real } x) (\text{Suc } n) = \text{cfrac-nth } (\text{cfrac-of-real } (1 / \text{frac } x)) n$
proof –
have $\text{cfrac-nth } (\text{cfrac-of-real } x) (\text{Suc } n) =$
 $\text{cfrac-nth } (\text{cfrac-tl } (\text{cfrac-of-real } x)) n$
by *simp*
also have $\text{cfrac-tl } (\text{cfrac-of-real } x) = \text{cfrac-of-real } (1 / \text{frac } x)$
by (*simp add: cfrac-tl-of-real assms*)
finally show ?*thesis* .
qed

fun *conv* :: $\text{cfrac} \Rightarrow \text{nat} \Rightarrow \text{real}$ **where**
 $\text{conv } c \ 0 = \text{real-of-int } (\text{cfrac-nth } c \ 0)$
 $\mid \text{conv } c \ (\text{Suc } n) = \text{real-of-int } (\text{cfrac-nth } c \ 0) + 1 / \text{conv } (\text{cfrac-tl } c) \ n$

The numerator and denominator of a convergent:

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

fun *conv-denom* :: $\text{cfrac} \Rightarrow \text{nat} \Rightarrow \text{int}$ **where**
 $\text{conv-denom } c \ 0 = 1$
 $\mid \text{conv-denom } c \ (\text{Suc } 0) = \text{cfrac-nth } c \ 1$
 $\mid \text{conv-denom } c \ (\text{Suc } (\text{Suc } n)) = \text{cfrac-nth } c \ (\text{Suc } (\text{Suc } n)) * \text{conv-denom } c \ (\text{Suc } n)$
 $+ \text{conv-denom } c \ n$

lemma *conv-num-rec*:

$n \geq 2 \implies \text{conv-num } c \ n = \text{cfrac-nth } c \ n * \text{conv-num } c \ (n - 1) + \text{conv-num } c \ (n - 2)$
by (cases *n*; cases *n* - 1) *auto*

lemma *conv-denom-rec*:

$n \geq 2 \implies \text{conv-denom } c \ n = \text{cfrac-nth } c \ n * \text{conv-denom } c \ (n - 1) + \text{conv-denom } c \ (n - 2)$
by (cases *n*; cases *n* - 1) *auto*

fun *conv'* :: *cfrac* \Rightarrow *nat* \Rightarrow *real* \Rightarrow *real* **where**

conv' *c* 0 *z* = *z*
| *conv'* *c* (*Suc* *n*) *z* = *conv'* *c* *n* (*real-of-int* (*cfrac-nth* *c* *n*) + 1 / *z*)

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

definition *conv-num-int* :: *cfrac* \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 $n \geq 0$
shows $\text{conv-num-int } c \ n = \text{cfrac-nth } c \ (\text{nat } n) * \text{conv-num-int } c \ (n - 1) + \text{conv-num-int } c \ (n - 2)$
proof (cases $n \geq 2$)
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** $\langle \text{auto } \text{simp: conv-num-int-def} \rangle$)

lemma *conv-denom-int-rec*:

assumes $n \geq 0$
shows $\text{conv-denom-int } c \ n = \text{cfrac-nth } c \ (\text{nat } n) * \text{conv-denom-int } c \ (n - 1) + \text{conv-denom-int } c \ (n - 2)$
proof -
consider $n = 0 \mid n = 1 \mid n \geq 2$
using *assms* **by** *force*
thus ?thesis
proof *cases*
assume $n \geq 2$

```

define  $n'$  where  $n' = \text{nat } (n - 2)$ 
have  $n$ :  $n = \text{int } (\text{Suc } (\text{Suc } n'))$ 
  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 <auto simp: conv-denom-int-def>)
qed

```

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

```

definition cfrac-lim :: cfrac  $\Rightarrow$  real where
  cfrac-lim  $c =$ 
    (case cfrac-length c of  $\infty \Rightarrow \text{lim } (\text{conv } c) \mid \text{enat } l \Rightarrow \text{conv } c \ l$ )

```

```

lemma cfrac-lim-code [code]:
  cfrac-lim  $c =$ 
    (case cfrac-length c of  $\text{enat } l \Rightarrow \text{conv } c \ l$ 
       $\mid - \Rightarrow \text{Code.abort } (\text{STR "Cannot compute infinite continued fraction"}) (\lambda \cdot$ 
cfrac-lim c))
  by (simp add: cfrac-lim-def split: enat.splits)

```

```

definition cfrac-remainder where cfrac-remainder  $c \ n = \text{cfrac-lim } (\text{cfrac-drop } n \ c)$ 

```

```

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

```

```

lemma conv'-Suc-left:
  assumes  $z > 0$ 
  shows conv'  $c \ (\text{Suc } n) \ z =$ 
    real-of-int (cfrac-nth  $c \ 0$ ) +  $1 / \text{conv}' (\text{cfrac-tl } c) \ n \ z$ 
  using assms
proof (induction n arbitrary: z)
  case ( $\text{Suc } n \ z$ )
  have conv'  $c \ (\text{Suc } (\text{Suc } n)) \ z =$ 
    conv'  $c \ (\text{Suc } n) \ (\text{real-of-int } (\text{cfrac-nth } c \ (\text{Suc } n)) + 1 / z)$ 
  by simp
  also have  $\dots = \text{cfrac-nth } c \ 0 + 1 / \text{conv}' (\text{cfrac-tl } c) (\text{Suc } n) \ z$ 
  using Suc.prems by (subst Suc.IH) (auto intro!: add-nonneg-pos cfrac-nth-nonneg)
  finally show ?case .
qed simp-all

```

```

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

```

```

lemma conv'-left-induct:
  assumes  $\bigwedge c. P \ c \ 0 \ z \ \bigwedge c \ n. P \ (\text{cfrac-tl } c) \ n \ z \Longrightarrow P \ c \ (\text{Suc } n) \ z$ 
  shows  $P \ c \ n \ z$ 
  using assms by (rule conv.induct)

```

```

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

```

assumes $a = \infty \vee b < \infty \vee c < \infty$
shows $a < c - (b :: \text{enat}) \iff a + b < c$
using *assms* **by** (*cases* *a*; *cases* *b*; *cases* *c*) *auto*

lemma *conv-eq-conv'*: $\text{conv } c \ n = \text{conv}' \ c \ n \ (\text{cfrac-nth } c \ n)$
proof (*cases* $n = 0$)
 case *False*
 hence $\text{cfrac-nth } c \ n > 0$ **by** (*auto* *intro!*: *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 $\text{cfrac-nth } c \ 0 > 0$
shows $\text{conv-num } c \ n > 0$
using *assms* **by** (*induction* n *rule*: *fib.induct*) (*auto* *simp*: *intro!*: *add-pos-nonneg*)

lemma *conv-num-nonneg*: $\text{cfrac-nth } c \ 0 \geq 0 \implies \text{conv-num } c \ n \geq 0$
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*:
 $\text{cfrac-nth } c \ 0 \geq 0 \implies n > 0 \implies \text{conv-num } c \ n > 0$
by (*induction* $c \ n$ *rule*: *conv-num.induct*)
 (*auto* *intro!*: *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*]: $\text{conv-denom } c \ n > 0$
by (*induction* $c \ n$ *rule*: *conv-num.induct*)
 (*auto* *intro!*: *add-nonneg-pos* *mult-nonneg-nonneg* *cfrac-nth-nonneg*
 simp: *enat-le-iff*)

lemma *conv-denom-not-nonpos* [*simp*]: $\neg \text{conv-denom } c \ n \leq 0$
using *conv-denom-pos*[*of* $c \ n$] **by** *linarith*

lemma *conv-denom-not-neg* [*simp*]: $\neg \text{conv-denom } c \ n < 0$
using *conv-denom-pos*[*of* $c \ n$] **by** *linarith*

lemma *conv-denom-nonzero* [*simp*]: $\text{conv-denom } c \ n \neq 0$
using *conv-denom-pos*[*of* $c \ n$] **by** *linarith*

lemma *conv-denom-nonneg* [*simp*, *intro*]: $\text{conv-denom } c \ n \geq 0$
using *conv-denom-pos*[*of* $c \ n$] **by** *linarith*

lemma *conv-num-int-neg1* [*simp*]: $\text{conv-num-int } c \ (-1) = 1$
by (*simp* *add*: *conv-num-int-def*)

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

lemma *conv-num-int-of-nat* [simp]: $\text{conv-num-int } c \ (\text{int } n) = \text{conv-num } c \ n$
by (simp add: conv-num-int-def)

lemma *conv-num-int-nonneg* [simp]: $n \geq 0 \implies \text{conv-num-int } c \ n = \text{conv-num } c \ n$
by (simp add: conv-num-int-def)

lemma *conv-denom-int-neg2* [simp]: $\text{conv-denom-int } c \ (-2) = 1$
by (simp add: conv-denom-int-def)

lemma *conv-denom-int-neg* [simp]: $n < 0 \implies n \neq -2 \implies \text{conv-denom-int } c \ n = 0$
by (simp add: conv-denom-int-def)

lemma *conv-denom-int-of-nat* [simp]: $\text{conv-denom-int } c \ (\text{int } n) = \text{conv-denom } c \ n$
by (simp add: conv-denom-int-def)

lemma *conv-denom-int-nonneg* [simp]: $n \geq 0 \implies \text{conv-denom-int } c \ n = \text{conv-denom } c \ n$
by (simp add: conv-denom-int-def)

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

lemma *conv'-gt-1*:
assumes *cfrac-nth* $c \ 0 > 0 \ x > 1$
shows $\text{conv}' \ c \ n \ x > 1$
using *assms*
proof (induction *n* arbitrary: $c \ x$)
case (Suc $n \ c \ x$)
from *Suc.prem*s **have** *pos*: $\text{cfrac-nth } c \ n > 0$ **using** *cfrac-nth-pos*[of $n \ c$]
by (cases $n = 0$) (auto simp: enat-le-iff)
have $1 < 1 + 1 / x$
using *Suc.prem*s **by** simp
also have $\dots \leq \text{cfrac-nth } c \ n + 1 / x$ **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.prem*s **in** (auto simp: enat-le-iff))
qed auto

lemma *enat-eq-iff*: $a = \text{enat } b \iff (\exists a'. a = \text{enat } a' \wedge a' = b)$
by (cases *a*) auto

lemma *eq-enat-iff*: $\text{enat } a = b \iff (\exists b'. b = \text{enat } b' \wedge a = b')$
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' m
  using 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 > 1
    by (rule conv'-gt-1;
      use Suc.prem 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.prem 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 gt have floor (cfrac-nth c 0 + 1 / conv' (cfrac-tl c) n x) = cfrac-nth
c 0
    by (intro floor-unique) auto
  also from gt have floor (cfrac-nth c' 0 + 1 / conv' (cfrac-tl c') n x) = cfrac-nth
c' 0
    by (intro floor-unique) auto
  finally have [simp]: cfrac-nth c 0 = cfrac-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 x
    by simp
  hence cfrac-nth (cfrac-tl c) m' = cfrac-nth (cfrac-tl c') m'
    using Suc.prem
  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 k
  defines h ≡ conv-num c and k ≡ conv-denom c
begin

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

```

```

proof (induction n arbitrary: z)
  case 0
  hence  $1 + z * \text{cfrac-nth } c \ 1 > 0$ 
  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)
next
  case (Suc n)
  have [simp]:  $h \ (Suc \ (Suc \ n)) = \text{cfrac-nth } c \ (n+2) * h \ (n+1) + h \ n$ 
  by (simp add: h-def)
  have [simp]:  $k \ (Suc \ (Suc \ n)) = \text{cfrac-nth } c \ (n+2) * k \ (n+1) + k \ n$ 
  by (simp add: k-def)
  define  $z'$  where  $z' = \text{cfrac-nth } c \ (n+2) + 1 / z$ 
  from  $\langle z > 0 \rangle$  have  $z' > 0$ 
  by (auto simp: z'-def intro!: add-nonneg-pos cfrac-nth-nonneg)

  have  $z * \text{real-of-int } (h \ (Suc \ (Suc \ n))) + \text{real-of-int } (h \ (Suc \ n)) =$ 
     $z * (z' * h \ (Suc \ n) + h \ n)$ 
  using  $\langle z > 0 \rangle$  by (simp add: algebra-simps z'-def)
  also have  $\dots = z * (\text{conv}' \ c \ (Suc \ (Suc \ n)) \ z' * (z' * k \ (Suc \ n) + k \ n))$ 
  using  $\langle z' > 0 \rangle$  by (subst Suc.IH [symmetric]) auto
  also have  $\dots = \text{conv}' \ c \ (Suc \ (Suc \ (Suc \ n))) \ z *$ 
     $(z * k \ (Suc \ (Suc \ n)) + k \ (Suc \ n))$ 
  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  $z > 0$ 
  shows  $\text{conv}' \ c \ (Suc \ (Suc \ n)) \ z =$ 
     $(z * h \ (Suc \ n) + h \ n) / (z * k \ (Suc \ n) + k \ n)$ 
proof -
  have  $z * \text{real-of-int } (k \ (Suc \ n)) + \text{real-of-int } (k \ n) > 0$ 
  using 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:  $\text{conv } c \ n = h \ n / k \ n$ 
proof -
  consider  $n = 0 \mid n = Suc \ 0 \mid m$  where  $n = Suc \ (Suc \ m)$ 
  using not0-imply-Suc by blast
  thus ?thesis
  proof cases
  assume  $n = Suc \ 0$ 
  thus ?thesis
  by (auto simp: h-def k-def field-simps max-def conv-Suc)
next

```

```

fix m assume [simp]: n = Suc (Suc m)
have conv c n = conv' c (Suc (Suc m)) (cfraction c (Suc (Suc m)))
  by (subst conv-eq-conv') simp-all
also have ... = 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 ≥ 2
  shows conv' c n z = (z * h (n - 1) + h (n - 2)) / (z * k (n - 1) + k (n - 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 > 0
  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 | n = 1 | n ≥ 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)
    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: cfraction c 0 ≥ 0 ⟹ conv c n ≥ 0
  by (subst conv-num-denom)
  (auto intro!: divide-nonneg-nonneg conv-num-nonneg simp: h-def k-def)

lemma conv-pos:
  assumes cfraction 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 ... > 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*:
 $k\ n * h\ (Suc\ n) - k\ (Suc\ n) * h\ n = (-1) \wedge n$
by (*induction c n rule: conv-num.induct*)
(auto simp: k-def h-def algebra-simps)

lemma *conv-num-denom-prod-diff'*:
 $k\ (Suc\ n) * h\ n - k\ n * h\ (Suc\ n) = (-1) \wedge Suc\ n$
by (*induction c n rule: conv-num.induct*)
(auto simp: k-def h-def algebra-simps)

lemma
fixes $n :: int$
assumes $n \geq -2$
shows *conv-num-denom-int-prod-diff*:
 $conv-denom-int\ c\ n * conv-num-int\ c\ (n + 1) -$
 $conv-denom-int\ c\ (n + 1) * conv-num-int\ c\ n = (-1) \wedge (nat\ (n + 2))$
(is ?th1)
and *conv-num-denom-int-prod-diff'*:
 $conv-denom-int\ c\ (n + 1) * conv-num-int\ c\ n -$
 $conv-denom-int\ c\ n * conv-num-int\ c\ (n + 1) = (-1) \wedge (nat\ (n + 3))$
(is ?th2)
proof -
from *assms* **consider** $n = -2 \mid n = -1 \mid n \geq 0$ **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 $\dots = 1$
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  $\text{coprime } (\text{conv-num-int } c \ n) (\text{conv-denom-int } c \ n)$ 
  proof -
    from assms consider  $n = -2 \mid n = -1 \mid n \geq 0$  by force
    thus ?thesis by cases (insert coprime-conv-num-denom[of nat n], auto simp: h-def k-def)
  qed

```

```

lemma mono-conv-num:
  assumes  $\text{cfrac-nth } c \ 0 \geq 0$ 
  shows mono h
proof (rule incseq-SucI)
  show  $h \ n \leq h \ (\text{Suc } n)$  for  $n$ 
  proof (cases  $n$ )
    case 0
    have  $1 * \text{cfrac-nth } c \ 0 + 1 \leq \text{cfrac-nth } c \ (\text{Suc } 0) * \text{cfrac-nth } c \ 0 + 1$ 
      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)
  next
    case ( $\text{Suc } m$ )
    have  $1 * h \ (\text{Suc } m) + 0 \leq \text{cfrac-nth } c \ (\text{Suc } (\text{Suc } m)) * h \ (\text{Suc } m) + h \ m$ 
      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  $k \ n \leq k \ (\text{Suc } n)$  for  $n$ 
  proof (cases  $n$ )
    case 0
    thus ?thesis by (simp add: le-Suc-eq Suc-le-eq k-def)
  next
    case ( $\text{Suc } m$ )
    have  $1 * k \ (\text{Suc } m) + 0 \leq \text{cfrac-nth } c \ (\text{Suc } (\text{Suc } m)) * k \ (\text{Suc } m) + k \ m$ 
      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:  $\text{cfrac-nth } c \ 0 \geq 0 \implies m \leq n \implies h \ m \leq h \ n$ 
  using mono-conv-num by (auto simp: mono-def)

```

```

lemma conv-denom-leI:  $m \leq n \implies k \ m \leq k \ n$ 
  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  $\dots \leq \text{cfrac-nth } c\ n * k\ n' + 0$ 
      using assms by (intro add-mono mult-mono) (auto simp: Suc-le-eq k-def)
    also have  $\dots < \text{cfrac-nth } c\ n * k\ n' + k\ n''$  unfolding k-def
      by (intro add-strict-left-mono conv-denom-pos assms)
    also have  $\dots = k\ n$  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  $\text{cfrac-nth } c\ 0 \geq 0$ 
  shows  $h\ n \geq \text{fib } n$  unfolding h-def
  using assms
proof (induction  $c\ n$  rule: conv-denom.induct)
  case ( $\exists\ c\ n$ )
  hence  $\text{conv-num } c\ (\text{Suc } (\text{Suc } n)) \geq 1 * \text{int } (\text{fib } (\text{Suc } n)) + \text{int } (\text{fib } n)$ 
    using 3.prem unfolding conv-num.simps
    by (intro add-mono mult-mono 3.IH) auto
  thus ?case by simp
qed auto

lemma conv-denom-lower-bound:  $k\ n \geq \text{fib } (\text{Suc } n)$ 
  unfolding k-def
proof (induction  $c\ n$  rule: conv-denom.induct)
  case ( $\exists\ c\ n$ )
  hence  $\text{conv-denom } c\ (\text{Suc } (\text{Suc } n)) \geq 1 * \text{int } (\text{fib } (\text{Suc } (\text{Suc } n))) + \text{int } (\text{fib } (\text{Suc } n))$ 
    using 3.prem 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:  $\text{conv } c\ (\text{Suc } n) - \text{conv } c\ n = (-1) ^ n / (k\ n * k\ (\text{Suc } n))$ 
proof -
  have pos:  $k\ n > 0$   $k\ (\text{Suc } n) > 0$  unfolding k-def
    by (intro conv-denom-pos)+
  have  $\text{conv } c\ (\text{Suc } n) - \text{conv } c\ n =$ 
     $(k\ n * h\ (\text{Suc } n) - k\ (\text{Suc } n) * h\ n) / (k\ n * k\ (\text{Suc } n))$ 
    using pos by (subst (1 2) conv-num-denom) (simp add: conv-num-denom
    field-simps)

```

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

lemma conv-telescope:

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

lemma fib-at-top: filterlim fib at-top at-top

proof (rule filterlim-at-top-mono)
 show eventually $(\lambda n. fib\ n \geq n - 1)$ at-top
 by (intro always-eventually fib-ge allI)
 show filterlim $(\lambda n::nat. n - 1)$ at-top at-top
 by (subst filterlim-sequentially-Suc [symmetric])
 (simp-all add: filterlim-ident)
 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:
 $summable\ (\lambda i. (-1) \wedge i / (k\ i * k\ (Suc\ i)))$ (is ?th1)
 and cfrac-remainder-bounds:
 $|(\sum_{i=m..<n}. (-1) \wedge i / (k\ (i + m) * k\ (Suc\ i + m)))| \in$
 $\{1/(k\ m * (k\ m + k\ (Suc\ m))) <..< 1 / (k\ m * k\ (Suc\ m))\}$ (is ?th2)

proof -

have [simp]: $k\ n > 0\ k\ n \geq 0 \neg k\ n = 0$ for n
 by (auto simp: k-def)
 have k-rec: $k\ (Suc\ (Suc\ n)) = cfrac-nth\ c\ (Suc\ (Suc\ n)) * k\ (Suc\ n) + k\ n$ for n
 by (simp add: k-def)
 have [simp]: $a + b = 0 \longleftrightarrow a = 0 \wedge b = 0$ if $a \geq 0\ b \geq 0$ for $a\ b :: real$
 using that by linarith

```

define  $g$  where  $g = (\lambda i. \text{inverse } (\text{real-of-int } (k \ i * k \ (\text{Suc } i))))$ 

{
  fix  $m :: \text{nat}$ 
  have  $\text{filterlim } (\lambda n. k \ n) \text{ at-top at-top}$  and  $\text{filterlim } (\lambda n. k \ (\text{Suc } n)) \text{ at-top at-top}$ 
    by  $(\text{force simp: filterlim-sequentially-Suc intro: conv-denom-at-top}) +$ 
  hence  $\text{lim: } g \longrightarrow 0$ 
    unfolding  $g\text{-def of-int-mult}$ 
    by  $(\text{intro tendsto-inverse-0-at-top filterlim-at-top-mult-at-top}$ 
       $\text{filterlim-compose[OF filterlim-real-of-int-at-top]})$ 
  from  $\text{lim have } A: \text{summable } (\lambda n. (-1) ^ (n + m) * g \ (n + m))$  unfolding
 $g\text{-def}$ 
    by  $(\text{intro summable-alternating-decreasing})$ 
     $(\text{auto intro!: conv-denom-leI mult-nonneg-nonneg})$ 

  have  $1 / (k \ m * (\text{real-of-int } (k \ (\text{Suc } m)) + k \ m / 1)) \leq$ 
     $1 / (k \ m * (k \ (\text{Suc } m) + k \ m / \text{cfrac-nth } c \ (m+2)))$ 
  by  $(\text{intro divide-left-mono mult-left-mono add-left-mono mult-pos-pos add-pos-pos}$ 
 $\text{divide-pos-pos})$ 
     $(\text{auto simp: of-nat-ge-1-iff})$ 
  also have  $\dots = g \ m - g \ (\text{Suc } m)$ 
    by  $(\text{simp add: } g\text{-def } k\text{-rec field-simps add-pos-pos})$ 
  finally have  $\text{le: } 1 / (k \ m * (\text{real-of-int } (k \ (\text{Suc } m)) + k \ m / 1)) \leq g \ m - g$ 
 $(\text{Suc } m)$  by  $\text{simp}$ 
  have  $*$ :  $|(\sum i. (-1) ^ (i + m) * g \ (i + m))| \in \{g \ m - g \ (\text{Suc } m) <..< g \ m\}$ 
    using  $\text{lim unfolding } g\text{-def}$ 
    by  $(\text{intro abs-alternating-decreasing-suminf-strict}) (\text{auto intro!: conv-denom-lessI})$ 
  also from  $\text{le have } \dots \subseteq \{1 / (k \ m * (k \ (\text{Suc } m) + k \ m)) <..< g \ m\}$ 
    by  $(\text{subst greaterThanLessThan-subseteq-greaterThanLessThan}) \text{ auto}$ 
  finally have  $B: |\sum i. (-1) ^ (i + m) * g \ (i + m)| \in \dots$ 
  note  $A \ B$ 
} note  $AB = \text{this}$ 

from  $AB(1)[\text{of } 0]$  show  $?th1$  by  $(\text{simp add: field-simps } g\text{-def})$ 
from  $AB(2)[\text{of } m]$  show  $?th2$  by  $(\text{simp add: } g\text{-def divide-inverse add-ac})$ 
qed

lemma  $\text{convergent-conv: convergent } (\text{conv } c)$ 
proof –
  have  $\text{convergent } (\lambda n. \text{conv } c \ 0 + (\sum i < n. (-1) ^ i / (k \ i * k \ (\text{Suc } i))))$ 
    using  $\text{summable-conv-telescope}$ 
    by  $(\text{intro convergent-add convergent-const})$ 
     $(\text{simp-all add: summable-iff-convergent})$ 
  also have  $\dots = \text{conv } c$ 
    by  $(\text{rule ext, subst } (2) \text{ conv-telescope [of } 0, \text{ symmetric}]) (\text{simp-all add: atLeast0LessThan})$ 
  finally show  $?thesis$  .
qed

lemma  $\text{LIMSEQ-cfrac-lim: cfrac-length } c = \infty \implies \text{conv } c \longrightarrow \text{cfrac-lim } c$ 

```

```

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

lemma cfrac-lim-nonneg:
  assumes cfrac-nth c 0  $\geq$  0
  shows cfrac-lim c  $\geq$  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-nonneg)
qed

lemma sums-cfrac-lim-minus-conv:
  assumes cfrac-length c =  $\infty$ 
  shows  $(\lambda i. (-1)^\wedge (i + m) / (k (i + m) * k (Suc i + m))) \text{ sums } (cfrac-lim c - conv c m)$ 
proof -
  have  $(\lambda n. conv c (n + m) - conv c m) \longrightarrow cfrac-lim c - conv c m$ 
    by (auto intro!: tendsto-diff LIMSEQ-cfrac-lim simp: filterlim-sequentially-shift assms)
  also have  $(\lambda n. conv c (n + m) - conv c m) =$ 
     $(\lambda n. (\sum i=0 + m..<n + m. (-1)^\wedge i / (k i * k (Suc i))))$ 
    by (subst conv-telescope [of m, symmetric]) simp-all
  also have  $\dots = (\lambda n. (\sum i<n. (-1)^\wedge (i + m) / (k (i + m) * k (Suc i + m))))$ 
    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  $\leq$  cfrac-length c
  shows  $|cfrac-lim c - conv c m| \leq 1 / (k m * k (Suc m))$ 
proof (cases cfrac-length c)
  case infinity
  have  $cfrac-lim c - conv c m = (\sum i. (-1)^\wedge (i + m) / (k (i + m) * k (Suc i + m)))$ 
    by (using sums-cfrac-lim-minus-conv infinity by (simp add: sums-iff))
  also note cfrac-remainder-bounds[of m]
  finally show ?thesis by simp
next
  case [simp]: (enat l)
  show ?thesis
  proof (cases l = m)
    case True

```

```

    thus ?thesis by (auto simp: cfrac-lim-def k-def)
  next
    case False
    let ?S = ( $\sum_{i=m..<l} (-1)^i * (1 / \text{real-of-int } (k \ i * k \ (\text{Suc } i)))$ )
    have [simp]:  $k \ n \geq 0 \ k \ n > 0$  for  $n$ 
      by (simp-all add: k-def)
    hence  $\text{cfrac-lim } c - \text{conv } c \ m = \text{conv } c \ l - \text{conv } c \ m$ 
      by (simp add: cfrac-lim-def)
    also have  $\dots = ?S$ 
      using assms by (subst conv-telescope [symmetric, of m]) auto
    finally have  $\text{cfrac-lim } c - \text{conv } c \ m = ?S$  .
    moreover have  $|?S| \leq 1 / \text{real-of-int } (k \ m * k \ (\text{Suc } m))$ 
      unfolding of-int-mult using assms False
      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  $m < \text{cfrac-length } c$ 
  shows  $|\text{cfrac-lim } c - \text{conv } c \ m| \geq 1 / (k \ m * (k \ m + k \ (\text{Suc } m)))$ 
proof (cases cfrac-length c)
  case infinity
  have  $\text{cfrac-lim } c - \text{conv } c \ m = (\sum i. (-1)^{(i+m)} / (k \ (i+m) * k \ (\text{Suc } i + m)))$ 
    using sums-cfrac-lim-minus-conv infinity by (simp add: sums-iff)
  also note cfrac-remainder-bounds[of m]
  finally show ?thesis by simp
next
  case [simp]: (enat l)
  let ?S = ( $\sum_{i=m..<l} (-1)^i * (1 / \text{real-of-int } (k \ i * k \ (\text{Suc } i)))$ )
  have [simp]:  $k \ n \geq 0 \ k \ n > 0$  for  $n$ 
    by (simp-all add: k-def)
  hence  $\text{cfrac-lim } c - \text{conv } c \ m = \text{conv } c \ l - \text{conv } c \ m$ 
    by (simp add: cfrac-lim-def)
  also have  $\dots = ?S$ 
    using assms by (subst conv-telescope [symmetric, of m]) (auto simp: split:
      enat.splits)
  finally have  $\text{cfrac-lim } c - \text{conv } c \ m = ?S$  .

  moreover have  $|?S| \geq 1 / (k \ m * (k \ m + k \ (\text{Suc } m)))$ 
proof (cases  $m < \text{cfrac-length } c - 1$ )
  case False
  hence [simp]:  $m = l - 1$  and  $l > 0$  using assms
    by (auto simp: not-less)
  have  $1 / (k \ m * (k \ m + k \ (\text{Suc } m))) \leq 1 / (k \ m * k \ (\text{Suc } m))$ 
    unfolding of-int-mult

```

```

    by (intro divide-left-mono mult-mono mult-pos-pos) (auto intro!: add-pos-pos)
  also from  $\langle l > 0 \rangle$  have  $\{m..<l\} = \{m\}$  by auto
  hence  $1 / (k\ m * k\ (Suc\ m)) = |\mathcal{S}|$ 
    by simp
  finally show ?thesis .
next
case True
with assms have less:  $m < l - 1$ 
  by auto
have  $k\ m + k\ (Suc\ m) > 0$ 
  by (intro add-pos-pos) (auto simp: k-def)
hence  $1 / (k\ m * (k\ m + k\ (Suc\ m))) \leq 1 / (k\ m * k\ (Suc\ m)) - 1 / (k\ (Suc\ m) * k\ (Suc\ (Suc\ m)))$ 
  by (simp add: divide-simps) (auto simp: k-def algebra-simps)
also have  $\dots \leq |\mathcal{S}|$ 
  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 .
qed
ultimately show ?thesis by simp
qed

```

```

lemma cfrac-lim-minus-conv-bounds:
  assumes  $m < cfrac-length\ c$ 
  shows  $|cfrac-lim\ c - conv\ c\ m| \in \{1 / (k\ m * (k\ m + k\ (Suc\ m)))..1 / (k\ m * k\ (Suc\ m))\}$ 
  using cfrac-lim-minus-conv-lower-bound[of m] cfrac-lim-minus-conv-upper-bound[of m] assms
  by auto
end

```

```

lemma conv-pos':
  assumes  $n > 0$  cfrac-nth  $c\ 0 \geq 0$ 
  shows  $conv\ c\ n > 0$ 
  using assms by (cases n) (auto simp: conv-Suc intro!: add-nonneg-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  $0 < z1$   $z1 \leq z2$ 
  shows conv'-even-mono:  $even\ n \implies conv'\ c\ n\ z1 \leq conv'\ c\ n\ z2$ 
    and conv'-odd-mono:  $odd\ n \implies conv'\ c\ n\ z1 \geq conv'\ c\ n\ z2$ 
proof -
  let ?P =  $(\lambda n\ (f::nat \Rightarrow real \Rightarrow real))$ .

```

```

      if even n then f n z1 ≤ f n z2 else f n z1 ≥ f n z2)
have ?P n (conv' c) using assms
proof (induction n arbitrary: z1 z2)
  case (Suc n)
  note z12 = Suc.prem
  consider n = 0 | even n n > 0 | odd n by force
  thus ?case
  proof cases
    assume n = 0
    thus ?thesis using Suc by (simp add: conv'-Suc-right field-simps)
  next
    assume n: even n n > 0
    with Suc.IH have IH: conv' c n z1 ≤ conv' c n z2
    if 0 < z1 z1 ≤ z2 for z1 z2 using that by auto
    show ?thesis using Suc.prem n z12
    by (auto simp: conv'-Suc-right field-simps intro!: IH add-pos-nonneg
mult-nonneg-nonneg)
  next
    assume n: odd n
    hence [simp]: n > 0 by (auto intro!: Nat.gr0I)
    from n and Suc.IH have IH: conv' c n z1 ≥ conv' c n z2
    if 0 < z1 z1 ≤ z2 for z1 z2 using that by auto
    show ?thesis using Suc.prem n
    by (auto simp: conv'-Suc-right field-simps
intro!: IH add-pos-nonneg mult-nonneg-nonneg)
  qed
qed auto
thus even n ⇒ conv' c n z1 ≤ conv' c n z2
odd n ⇒ conv' c n z1 ≥ conv' c n z2 by auto
qed

lemma
  shows conv-even-mono: even n ⇒ n ≤ m ⇒ conv c n ≤ conv c m
  and conv-odd-mono: odd n ⇒ n ≤ m ⇒ conv c n ≥ conv c m
proof -
  assume even n
  have A: conv c n ≤ conv c (Suc (Suc n)) if even n for n
  proof (cases n = 0)
    case 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)

  have B: conv c n ≤ conv c (Suc n) if even n for n
  proof (cases n = 0)
    case 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  $\text{conv } c \ n \leq \text{conv } c \ m$  if  $n \leq m$  for  $m$ 
  using that
proof (induction  $m$  rule: less-induct)
  case (less m)
  from  $\langle n \leq m \rangle$  consider  $m = n \mid \text{even } m \ m > n \mid \text{odd } m \ m > n$ 
    by force
  thus ?case
  proof cases
    assume  $m$ : even m m > n
    with  $\langle \text{even } n \rangle$  have  $m'$ :  $m - 2 \geq n$  by presburger
    with  $m$  have  $\text{conv } c \ n \leq \text{conv } c \ (m - 2)$ 
      by (intro less.IH) auto
    also have  $\dots \leq \text{conv } c \ (\text{Suc } (\text{Suc } (m - 2)))$ 
      using  $m \ m'$  by (intro A) auto
    also have  $\text{Suc } (\text{Suc } (m - 2)) = m$ 
      using  $m$  by presburger
    finally show ?thesis .
  next
    assume  $m$ : odd m m > n
    hence  $\text{conv } c \ n \leq \text{conv } c \ (m - 1)$ 
      by (intro less.IH) auto
    also have  $\dots \leq \text{conv } c \ (\text{Suc } (m - 1))$ 
      using  $m$  by (intro B) auto
    also have  $\text{Suc } (m - 1) = m$ 
      using  $m$  by simp
    finally show ?thesis .
  qed simp-all
qed
next
assume odd n
have  $A$ :  $\text{conv } c \ n \geq \text{conv } c \ (\text{Suc } (\text{Suc } n))$  if odd n for  $n$ 
  using that
  by (auto simp add: conv-eq-conv' conv'-Suc-right odd-pos intro!: conv'-odd-mono)
have  $B$ :  $\text{conv } c \ n \geq \text{conv } c \ (\text{Suc } n)$  if odd n for  $n$  using that
  by (auto simp add: conv-eq-conv' conv'-Suc-right odd-pos intro!: conv'-odd-mono)

show  $\text{conv } c \ n \geq \text{conv } c \ m$  if  $n \leq m$  for  $m$ 
  using that
proof (induction  $m$  rule: less-induct)
  case (less m)
  from  $\langle n \leq m \rangle$  consider  $m = n \mid \text{even } m \ m > n \mid \text{odd } m \ m > n$ 
    by force
  thus ?case
  proof cases
    assume  $m$ : odd m m > n
    with  $\langle \text{odd } n \rangle$  have  $m'$ :  $m - 2 \geq n \ m \geq 2$  by presburger+
    from  $m$  and  $\langle \text{odd } n \rangle$  have  $m = \text{Suc } (\text{Suc } (m - 2))$  by presburger
    also have  $\text{conv } c \ \dots \leq \text{conv } c \ (m - 2)$ 

```

```

    using m m' by (intro A) auto
  also have ... ≤ conv c n
    using m m' by (intro less.IH) auto
  finally show ?thesis .
next
  assume m: even m m > n
  from m have m = Suc (m - 1) by presburger
  also have conv c ... ≤ conv c (m - 1)
    using m by (intro B) auto
  also have ... ≤ conv c n
    using m by (intro less.IH) auto
  finally show ?thesis .
qed simp-all
qed
qed

lemma
  assumes m ≤ cfrac-length c
  shows conv-le-cfrac-lim: even m ⇒ conv c m ≤ cfrac-lim c
    and conv-ge-cfrac-lim: odd m ⇒ conv c m ≥ cfrac-lim c
proof -
  have if even m then conv c m ≤ cfrac-lim c else conv c m ≥ cfrac-lim c
  proof (cases cfrac-length c)
    case [simp]: infinity
    show ?thesis
    proof (cases even m)
      case True
      have eventually (λi. conv c m ≤ 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 ≤ cfrac-lim c
      by (intro tendsto-lowerbound[OF LIMSEQ-cfrac-lim]) auto
      thus ?thesis using True by simp
    next
      case False
      have eventually (λi. conv c m ≥ 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 ≥ cfrac-lim c
      by (intro tendsto-upperbound[OF LIMSEQ-cfrac-lim]) auto
      thus ?thesis using False by simp
    qed
  next
    case [simp]: (enat l)
    show ?thesis
    using conv-even-mono[of m l c] conv-odd-mono[of m l c] asms
    by (auto simp: cfrac-lim-def)
  qed
thus even m ⇒ conv c m ≤ cfrac-lim c and odd m ⇒ conv c m ≥ cfrac-lim c

```

by auto
qed

lemma *cfrac-lim-ge-first*: $cfrac\text{-}lim\ c \geq cfrac\text{-}nth\ c\ 0$
using *conv-le-cfrac-lim*[*of 0 c*] **by** (auto simp: *less-eq-enat-def split: enat.splits*)

lemma *cfrac-lim-pos*: $cfrac\text{-}nth\ c\ 0 > 0 \implies cfrac\text{-}lim\ c > 0$
by (rule *less-le-trans*[*OF - cfrac-lim-ge-first*]) auto

lemma *conv'-eq-iff*:
assumes $0 \leq z1 \vee 0 \leq z2$
shows $conv'\ c\ n\ z1 = conv'\ c\ n\ z2 \longleftrightarrow z1 = z2$
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*)
next
case *False*
have $conv'\ c\ n\ (real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n) + 1 / z1) =$
 $conv'\ c\ n\ (real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n) + 1 / z2)$ **using** *Suc.prem*s
by (*simp add: conv'-Suc-right*)
hence $real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n) + 1 / z1 = real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n) + 1 /$
 $z2$
by (rule *Suc.IH*)
(*insert Suc.prem*s *False*, auto *intro!*: *add-nonneg-pos add-nonneg-nonneg*)
with *Suc.prem*s **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 = 0*)
case *True*
thus ?thesis **using** *assms* **by** (auto simp: *conv-Suc*)
next
case *False*
hence $conv'\ c\ n\ (real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n)) \neq$
 $conv'\ c\ n\ (real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ n) + 1 / real\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ (Suc$
 $n)))$

```

    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
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 ...  $\leq$  conv c m using assms <m  $\neq$  n + 1>
  by (intro conv-even-mono) auto
finally show ?thesis .
qed
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  $\geq$  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 ⟨n > 0⟩ have conv c n ≠ 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 ≥ 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)) ≥ 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 ... ≤ 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 ≤ conv c (n + 2)
    by (intro conv-ge-cfrac-lim) auto
  also from assms have ... < 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 ≠ 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 ≥ 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  $\wedge$  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  $\longleftrightarrow$  cfrac-length c = 0

lemma cfrac-is-int-code [code]: cfrac-is-int (CFrac n xs)  $\longleftrightarrow$  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  $\longleftrightarrow$  ( $\exists$  n. c = cfrac-of-int n)
proof -
  have c = cfrac-of-int (cfrac-nth c 0) if cfrac-is-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 0 < cfrac-nth (cfrac-tl c) 0
      by simp
    also have ...  $\leq$  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)
qed

```

```

lemma cfrac-lim-tl:
  assumes  $\neg$ cfrac-is-int c
  shows  $\text{cfrac-lim } (\text{cfrac-tl } c) = 1 / (\text{cfrac-lim } c - \text{cfrac-nth } c \ 0)$ 
  using cfrac-lim-reduce[OF assms] by simp

```

```

lemma cfrac-remainder-Suc':
  assumes  $n < \text{cfrac-length } c$ 
  shows  $\text{cfrac-remainder } c \ (\text{Suc } n) * (\text{cfrac-remainder } c \ n - \text{cfrac-nth } c \ n) = 1$ 
proof -
  have  $0 < \text{real-of-int } (\text{cfrac-nth } c \ (\text{Suc } n))$  by simp
  also have  $\text{cfrac-nth } c \ (\text{Suc } n) \leq \text{cfrac-remainder } c \ (\text{Suc } n)$ 
    using cfrac-lim-ge-first[of cfrac-drop (Suc n) c]
    by (simp add: cfrac-remainder-def)
  finally have  $\dots > 0$  .

  have  $\text{cfrac-remainder } c \ (\text{Suc } n) = \text{cfrac-lim } (\text{cfrac-tl } (\text{cfrac-drop } n \ c))$ 
    by (simp add: o-def cfrac-remainder-def cfrac-drop-Suc-left)
  also have  $\dots = 1 / (\text{cfrac-remainder } c \ n - \text{cfrac-nth } c \ n)$  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  $\langle \text{cfrac-remainder } c \ (\text{Suc } n) > 0 \rangle$ 
    by (auto simp add: cfrac-remainder-def field-simps)
qed

```

```

lemma cfrac-remainder-Suc:
  assumes  $n < \text{cfrac-length } c$ 
  shows  $\text{cfrac-remainder } c \ (\text{Suc } n) = 1 / (\text{cfrac-remainder } c \ n - \text{cfrac-nth } c \ n)$ 
proof -
  have  $\text{cfrac-remainder } c \ (\text{Suc } n) = \text{cfrac-lim } (\text{cfrac-tl } (\text{cfrac-drop } n \ c))$ 
    by (simp add: o-def cfrac-remainder-def cfrac-drop-Suc-left)
  also have  $\dots = 1 / (\text{cfrac-remainder } c \ n - \text{cfrac-nth } c \ n)$  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 .
qed

```

```

lemma cfrac-remainder-0 [simp]:  $\text{cfrac-remainder } c \ 0 = \text{cfrac-lim } c$ 
  by (simp add: cfrac-remainder-def)

```

```

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

lemma cfrac-lim-eq-num-denom-remainder-aux:
  assumes  $\text{Suc } ( \text{Suc } n ) \leq \text{cfrac-length } c$ 
  shows  $\text{cfrac-lim } c * (k ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + k n) = h ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + h n$ 
  using assms
proof (induction n)
  case 0
  have  $\text{cfrac-lim } c \neq \text{cfrac-nth } c\ 0$ 
    using conv-neq-cfrac-lim[of 0 c] 0 by (auto simp: enat-le-iff)
  moreover have  $\text{cfrac-nth } c\ 1 * ( \text{cfrac-lim } c - \text{cfrac-nth } c\ 0 ) \neq 1$ 
    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 ( $\text{Suc } n$ )
  have less:  $\text{enat } ( \text{Suc } ( \text{Suc } n ) ) < \text{cfrac-length } c$ 
    using Suc.premis by (cases cfrac-length c) auto
  have *:  $x ( \text{Suc } ( \text{Suc } n ) ) \neq \text{real-of-int } ( \text{cfrac-nth } c\ ( \text{Suc } ( \text{Suc } n ) ) )$ 
    using conv-neq-cfrac-lim[of 0 cfrac-drop (n+2) c] Suc.premis
    by (cases cfrac-length c) (auto simp: x-def cfrac-remainder-def)
  hence  $\text{cfrac-lim } c * (k ( \text{Suc } ( \text{Suc } n ) ) * x ( \text{Suc } ( \text{Suc } ( \text{Suc } n ) ) ) + k ( \text{Suc } n ) ) =$ 
     $( \text{cfrac-lim } c * (k ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + k n) ) / ( x ( \text{Suc } ( \text{Suc } n ) ) -$ 
     $\text{cfrac-nth } c\ ( \text{Suc } ( \text{Suc } n ) ) )$ 
    unfolding x-def k-def h-def using less
    by (subst cfrac-remainder-Suc) (auto simp: field-simps)
  also have  $\text{cfrac-lim } c * (k ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + k n) =$ 
     $h ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + h n$  using less
    by (intro Suc.IH) auto
  also have  $(h ( \text{Suc } n ) * x ( \text{Suc } ( \text{Suc } n ) ) + h n) / ( x ( \text{Suc } ( \text{Suc } n ) ) - \text{cfrac-nth } c$ 
     $( \text{Suc } ( \text{Suc } n ) ) ) =$ 
     $h ( \text{Suc } ( \text{Suc } n ) ) * x ( \text{Suc } ( \text{Suc } ( \text{Suc } n ) ) ) + h ( \text{Suc } n )$  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:  $\text{cfrac-nth } c\ n \geq 0 \implies \text{cfrac-remainder } c\ n \geq 0$ 
  unfolding cfrac-remainder-def by (rule cfrac-lim-nonneg) auto

lemma cfrac-remainder-pos:  $\text{cfrac-nth } c\ n > 0 \implies \text{cfrac-remainder } c\ n > 0$ 
  unfolding cfrac-remainder-def by (rule cfrac-lim-pos) auto

```


lemma *cfrac-lim-eq-num-denom-remainder*:
assumes $\text{Suc } (\text{Suc } n) < \text{cfrac-length } c$
shows $\text{cfrac-lim } c = (h (\text{Suc } n) * x (\text{Suc } (\text{Suc } n)) + h n) / (k (\text{Suc } n) * x (\text{Suc } (\text{Suc } n)) + k n)$
proof –
have $k (\text{Suc } n) * x (\text{Suc } (\text{Suc } n)) + k n > 0$
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 $|\text{conv } c (\text{Suc } n) - \text{conv } c n| = 1 / (k n * k (\text{Suc } n))$
proof –
have [*simp*]: $k n \neq 0$ **for** $n :: \text{nat}$
unfolding *k-def* **using** *conv-denom-pos*[*of c n*] **by** *auto*
have $\text{conv } c (\text{Suc } n) - \text{conv } c n = h (\text{Suc } n) / k (\text{Suc } n) - h n / k n$
by (*simp add: conv-num-denom k-def h-def*)
also have $\dots = (k n * h (\text{Suc } n) - k (\text{Suc } n) * h n) / (k n * k (\text{Suc } n))$
by (*simp add: field-simps*)
also have $k n * h (\text{Suc } n) - k (\text{Suc } n) * h n = (-1) ^ n$
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*: $k (\text{Suc } (\text{Suc } n)) \geq 2 * k n$
proof –
have $1 * k n + k n \leq \text{cfrac-nth } c (\text{Suc } (\text{Suc } n)) * k (\text{Suc } n) + k n$
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 $n < \text{cfrac-length } c$
shows $\text{conv}' c n (\text{cfrac-remainder } c n) = \text{cfrac-lim } c$
using *assms*
proof (*induction n arbitrary: c*)
case (*Suc n c*)
have $\text{conv}' c (\text{Suc } n) (\text{cfrac-remainder } c (\text{Suc } n)) =$
 $\text{cfrac-nth } c 0 + 1 / \text{conv}' (\text{cfrac-tl } c) n (\text{cfrac-remainder } c (\text{Suc } n))$
using *Suc.prem*s
by (*subst conv'-Suc-left*) (*auto intro!: cfrac-remainder-pos*)
also have $\text{cfrac-remainder } c (\text{Suc } n) = \text{cfrac-remainder } (\text{cfrac-tl } c) n$
by (*simp add: cfrac-remainder-def cfrac-drop-Suc-right*)

also have $\text{conv}' (\text{cfrac-tl } c) \ n \ \dots = \text{cfrac-lim } (\text{cfrac-tl } c)$
using *Suc.prem*s **by** (*subst Suc.IH*) (*auto simp: cfrac-remainder-def enat-less-iff*)
also have $\text{cfrac-nth } c \ 0 + 1 / \dots = \text{cfrac-lim } c$
using *Suc.prem*s **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*

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

lemma *linfinite-cfrac-of-real-aux*:
 $x \notin \mathbb{Q} \implies x \in \{0 < \cdot < 1\} \implies \text{linfinite } (\text{cfrac-of-real-aux } x)$
proof (*coinduction arbitrary: x*)
case (*linfinite x*)
hence $1 / x \notin \mathbb{Q}$ **using** *Rats-divide*[*of 1 1 / x*] **by** *auto*
thus *?case* **using** *linfinite Ints-subset-Rats*
by (*intro disjI1 exI*[*of - nat ⌊1/x⌋ - 1*] *exI*[*of - cfrac-of-real-aux (frac (1/x))*]
exI[*of - frac (1/x)*] *conjI*)
(auto simp: cfrac-of-real-aux.code[*of x*] *frac-lt-1*)
qed

lemma *cfrac-length-of-real-irrational*:
assumes $x \notin \mathbb{Q}$
shows $\text{cfrac-length } (\text{cfrac-of-real } x) = \infty$
proof (*insert assms, transfer, clarify*)
fix $x :: \text{real}$ **assume** $x \notin \mathbb{Q}$
thus $\text{llength } (\text{cfrac-of-real-aux } (\text{frac } x)) = \infty$
using *linfinite-cfrac-of-real-aux*[*of frac x*] *Ints-subset-Rats*
by (*auto simp: linfinite-conv-llength frac-lt-1*)
qed

lemma *cfrac-length-of-real-reduce*:
assumes $x \notin \mathbb{Z}$
shows $\text{cfrac-length } (\text{cfrac-of-real } x) = \text{eSuc } (\text{cfrac-length } (\text{cfrac-of-real } (1 / \text{frac } x)))$
using *assms*
by (*transfer, subst cfrac-of-real-aux.code*) (*auto simp: frac-lt-1*)

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

lemma *conv-cfrac-of-real-le-ge*:
assumes $n \leq \text{cfrac-length } (\text{cfrac-of-real } x)$
shows *if even n then* $\text{conv } (\text{cfrac-of-real } x) \ n \leq x$ *else* $\text{conv } (\text{cfrac-of-real } x) \ n \geq x$
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.prem 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.prem 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  $x \leq \text{real-of-int } \lfloor x \rfloor + \text{frac } x$ 
    by (simp add: frac-def)
    also have  $\text{frac } x \leq 1 / \text{conv } (\text{cfrac-of-real } ?x') n$ 
    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)
  next
    case False
    have  $\text{real-of-int } \lfloor x \rfloor + 1 / \text{conv } (\text{cfrac-of-real } ?x') n \leq \text{real-of-int } \lfloor x \rfloor + \text{frac } x$ 
    using IH False remainder-pos frac-gt-0-iff[of x] by (simp add: field-simps)
    also have ... = x
    by (simp add: frac-def)
    finally show ?thesis using False
    by (auto simp: conv-Suc cfrac-tl-of-real)
qed
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 0
        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.prem 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)
  qed
  thus ?thesis by (simp add: enat cfrac-lim-def)
next
case [simp]: infinity
have lim: conv (cfrac-of-real x) ⟶ cfrac-lim (cfrac-of-real x)
  by (simp add: LIMSEQ-cfrac-lim)
have cfrac-lim (cfrac-of-real x) ≤ x
proof (rule tendsto-upperbound)
  show (λn. conv (cfrac-of-real x) (n * 2)) ⟶ cfrac-lim (cfrac-of-real x)
    by (intro filterlim-compose[OF lim] mult-nat-right-at-top) auto
  show eventually (λn. conv (cfrac-of-real x) (n * 2) ≤ x) at-top
    using conv-cfrac-of-real-le-ge[of n * 2 x for n] by (intro always-eventually)
auto
qed auto
moreover have cfrac-lim (cfrac-of-real x) ≥ x
proof (rule tendsto-lowerbound)
  show (λn. conv (cfrac-of-real x) (Suc (n * 2))) ⟶ cfrac-lim (cfrac-of-real
x)
    by (intro filterlim-compose[OF lim] filterlim-compose[OF filterlim-Suc]
        mult-nat-right-at-top) auto
  show eventually (λn. conv (cfrac-of-real x) (Suc (n * 2)) ≥ x) at-top
    using conv-cfrac-of-real-le-ge[of Suc (n * 2) x for n] by (intro always-eventually)
auto
qed auto
ultimately show ?thesis by (rule antisym)
qed

lemma Ints-add-left-cancel:  $x \in \mathbb{Z} \implies x + y \in \mathbb{Z} \longleftrightarrow y \in \mathbb{Z}$ 
  using Ints-diff[of x + y x] by auto

lemma Ints-add-right-cancel:  $y \in \mathbb{Z} \implies x + y \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$ 
  using Ints-diff[of x + y y] by auto

lemma cfrac-of-real-conv':
  fixes m n :: nat
  assumes  $x > 1$   $m < n$ 
  shows cfrac-nth (cfrac-of-real (conv' c n x)) m = cfrac-nth c m
  using assms
proof (induction n arbitrary: c m)
  case (Suc n c m)
  from Suc.prem have gt-1:  $1 < \text{conv}' (cfrac\text{-tl } c) n x$ 
    by (intro conv'-gt-1) (auto simp: enat-le-iff intro: cfrac-nth-pos)
  show ?case
  proof (cases m)
    case 0
    thus ?thesis using gt-1 Suc.prem
      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 ∈ {0<..<1}
  by auto
have 1 / conv' (cfrac-tl c) n x ∉ ℤ
proof
  assume 1 / conv' (cfrac-tl c) n x ∈ ℤ
  then obtain k :: int where k: 1 / conv' (cfrac-tl c) n x = of-int k
    by (elim Ints-cases)
  have real-of-int k ∈ {0<..<1}
    using gt-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 ∉ 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) n x))
(Suc m')
  using ⟨x > 1⟩ by (subst conv'-Suc-left) (auto simp: Suc)
also have ... = cfrac-nth (cfrac-of-real (1 / frac (1 / conv' (cfrac-tl c) n x)))
m'
  using ⟨x > 1⟩ 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 ∈ {0<..<1} using gt-1
  by (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.prem 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 = ∞
  shows cfrac-lim c ∉ ℚ
proof
  assume cfrac-lim c ∈ ℚ
  then obtain a :: int and b :: nat where ab: b > 0 cfrac-lim c = a / b
    by (auto simp: Rats-eq-int-div-nat)
  define h and k where h = conv-num c and k = conv-denom c

  have filterlim (λm. 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) ≥ 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 $|cfrac\text{-lim } c - conv\ c\ m| = |(a * k\ m - b * h\ m) / (k\ m * b)|$
by (*subst **) (*auto simp: ab h-def k-def conv-num-denom*)
also have $\dots = |a * k\ m - b * h\ m| / (k\ m * b)$
by (*simp add: k-def*)
finally have $eq: |cfrac\text{-lim } c - conv\ c\ m| = of\text{-int } |a * k\ m - b * h\ m| / of\text{-int } (k\ m * b)$.

have $|cfrac\text{-lim } c - conv\ c\ m| * (k\ m * b) \neq 0$
using *conv-neq-cfrac-lim*[*of m c*] $\langle b > 0 \rangle$ **by** (*auto simp: k-def*)
also have $|cfrac\text{-lim } c - conv\ c\ m| * (k\ m * b) = of\text{-int } |a * k\ m - b * h\ m|$
using $\langle b > 0 \rangle$ **by** (*subst eq*) (*auto simp: k-def*)
finally have $|a * k\ m - b * h\ m| \geq 1$ **by** *linarith*
hence $real\text{-of-int } |a * k\ m - b * h\ m| \geq 1$ **by** *linarith*
hence $1 / of\text{-int } (k\ m * b) \leq of\text{-int } |a * k\ m - b * h\ m| / real\text{-of-int } (k\ m * b)$
using $\langle b > 0 \rangle$ **by** (*intro divide-right-mono*) (*auto simp: k-def*)
also have $\dots = |cfrac\text{-lim } c - conv\ c\ m|$
by (*rule eq [symmetric]*)
also have $\dots \leq 1 / real\text{-of-int } (conv\text{-denom } c\ m * conv\text{-denom } c\ (Suc\ m))$
by (*intro cfrac-lim-minus-conv-upper-bound*) *auto*
also have $\dots = 1 / (real\text{-of-int } (k\ m) * real\text{-of-int } (k\ (Suc\ m)))$
by (*simp add: k-def*)
also have $\dots < 1 / (real\text{-of-int } (k\ m) * real\ b)$
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*: $cfrac\text{-length } c = \infty \longleftrightarrow cfrac\text{-lim } c \notin \mathbb{Q}$
using *cfrac-lim-irrational*[*of c*] *cfrac-lim-rational*[*of c*] **by** *auto*

lemma *cfrac-lim-rational-iff*: $cfrac\text{-lim } c \in \mathbb{Q} \longleftrightarrow cfrac\text{-length } c \neq \infty$
using *cfrac-lim-irrational*[*of c*] *cfrac-lim-rational*[*of c*] **by** *auto*

lemma *cfrac-of-real-infinite-iff* [*simp*]: $cfrac\text{-length } (cfrac\text{-of-real } x) = \infty \longleftrightarrow x \notin \mathbb{Q}$
by (*simp add: cfrac-infinite-iff*)

lemma *cfrac-remainder-rational-iff* [*simp*]:
 $cfrac\text{-remainder } c\ n \in \mathbb{Q} \longleftrightarrow cfrac\text{-length } c < \infty$
proof –
have $cfrac\text{-remainder } c\ n \in \mathbb{Q} \longleftrightarrow cfrac\text{-lim } (cfrac\text{-drop } n\ c) \in \mathbb{Q}$
by (*simp add: cfrac-remainder-def*)
also have $\dots \longleftrightarrow cfrac\text{-length } c \neq \infty$
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**

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

lemma *cfrac-nth-cons*:

assumes *cfrac-nth* $x \ 0 \geq 1$

shows *cfrac-nth* (*cfrac-cons* $a \ x$) $n = (\text{if } n = 0 \text{ then } a \text{ else } \text{cfrac-nth } x \ (n - 1))$

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* $x \ 0 \geq 1$

shows *cfrac-length* (*cfrac-cons* $a \ x$) = *eSuc* (*cfrac-length* x)

using *assms* **by** *transfer* *auto*

lemma *cfrac-tl-cons* [*simp*]:

assumes *cfrac-nth* $x \ 0 \geq 1$

shows *cfrac-tl* (*cfrac-cons* $a \ x$) = x

using *assms* **by** *transfer* *auto*

lemma *cfrac-cons-tl*:

assumes $\neg \text{cfrac-is-int } x$

shows *cfrac-cons* (*cfrac-nth* $x \ 0$) (*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**
 $\lambda(x, xs). \neg \text{lfinit}e\ xs \vee \text{lnull}\ xs \vee \text{llast}\ xs \neq 0$.

lemma *cfrac-canonical* [code]:
cfrac-canonical (*CFrac* *x xs*) $\longleftrightarrow \text{lnull}\ xs \vee \text{llast}\ xs \neq 0 \vee \neg \text{lfinit}e\ xs$
by (*auto simp add: cfrac-canonical-def*)

lemma *cfrac-canonical-iff*:
cfrac-canonical *c* \longleftrightarrow
cfrac-length *c* $\in \{0, \infty\} \vee \text{cfrac-nth}\ c\ (\text{the-enat}\ (\text{cfrac-length}\ c)) \neq 1$
proof (*transfer, clarify, goal-cases*)
case (*1 x xs*)
show ?*case*
by (*cases llength xs*)
(auto simp: llast-def enat-0 lfinit-conv-llength-enat split: nat.splits)
qed

lemma *llast-cfrac-of-real-aux-nonzero*:
assumes *lfinit* (*cfrac-of-real-aux* *x*) $\neg \text{lnull}\ (\text{cfrac-of-real-aux}\ x)$
shows *llast* (*cfrac-of-real-aux* *x*) $\neq 0$
using *assms*
proof (*induction cfrac-of-real-aux x arbitrary: x rule: lfinit-induct*)
case (*LCons* *x*)
from *LCons.prem*s **have** *x* $\in \{0 <..<1\}$
by (*subst (asm) cfrac-of-real-aux.code*) (*auto split: if-splits*)
show ?*case*
proof (*cases 1 / 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]*)
next
case *True*
then obtain *n* **where** *n*: *1 / x* = *of-int* *n*
by (*elim Ints-cases*)
have *1 / x* > 1 **using** $\langle x \in \cdot \rangle$ **by** *auto*
with *n* **have** *n* > 1 **by** *simp*
from *n* **have** *x* = *1 / of-int* *n*
using $\langle n > 1 \rangle \langle x \in \cdot \rangle$ **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*)
qed
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 $LNil$)

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 = $LNil$

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

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

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-til-of-real-alt*:

assumes $x \notin \mathbb{Z}$

shows *cfrac-til* (*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 $1 / frac\ x \geq 2$

by *simp*

hence $int\ (nat\ \lfloor 1 / frac\ x \rfloor - 2) + 2 = \lfloor 1 / frac\ x \rfloor$

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
 qed

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

assumes $x \notin \mathbb{Z}$
 shows $\text{cfrac-nth } (\text{cfrac-of-real-alt } x) (\text{Suc } n) = \text{cfrac-nth } (\text{cfrac-of-real-alt } (1 / \text{frac } x)) n$
proof –
 have $\text{cfrac-nth } (\text{cfrac-of-real-alt } x) (\text{Suc } n) =$
 $\text{cfrac-nth } (\text{cfrac-tl } (\text{cfrac-of-real-alt } x)) n$
 by simp
 also have $\text{cfrac-tl } (\text{cfrac-of-real-alt } x) = \text{cfrac-of-real-alt } (1 / \text{frac } x)$
 by (simp add: cfrac-tl-of-real-alt assms)
 finally show ?thesis .
 qed

lemma cfrac-nth-gt0-of-real-int [simp]:

$m > 0 \implies \text{cfrac-nth } (\text{cfrac-of-real } (\text{of-int } n)) m = 1$
 by transfer (auto simp: lnth-LCons eSuc-def enat-0-iff split: nat.splits)

lemma cfrac-nth-0-of-real-alt-int [simp]:

$\text{cfrac-nth } (\text{cfrac-of-real-alt } (\text{of-int } n)) 0 = n - 1$
 by transfer auto

lemma cfrac-nth-gt0-of-real-alt-int [simp]:

$m > 0 \implies \text{cfrac-nth } (\text{cfrac-of-real-alt } (\text{of-int } n)) m = 1$
 by transfer (auto simp: lnth-LCons eSuc-def split: nat.splits)

lemma llength-cfrac-of-real-alt-aux:

assumes $x \in \{0 < .. < 1\}$
 shows $\text{llength } (\text{cfrac-of-real-alt-aux } x) = \text{eSuc } (\text{llength } (\text{cfrac-of-real-aux } x))$
 using assms
proof (coinduction arbitrary: x rule: enat-coinduct)
 case (Eq-enat x)
 show ?case
proof (cases 1 / x ∈ \mathbb{Z})
 case False
 with Eq-enat have $\text{frac } (1 / x) \in \{0 < .. < 1\}$
 by (auto intro: frac-lt-1)
 hence $\exists x'. \text{llength } (\text{cfrac-of-real-alt-aux } (\text{frac } (1 / x))) =$
 $\text{llength } (\text{cfrac-of-real-alt-aux } x') \wedge$
 $\text{llength } (\text{cfrac-of-real-aux } (\text{frac } (1 / x))) = \text{llength } (\text{cfrac-of-real-aux } x')$
 \wedge
 $0 < x' \wedge x' < 1$
 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])
 qed (use Eq-enat in <auto simp: cfrac-of-real-alt-aux.code[of x] cfrac-of-real-aux.code[of

$x] \rangle$
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 $x \in \{0 < \cdot < 1\}$ *llength (cfrac-of-real-aux x) = ∞*

shows *cfrac-of-real-alt-aux x = cfrac-of-real-aux x*

using *assms*

proof (*coinduction arbitrary: x*)

case (*Eq-llist x*)

hence $\exists x'. \text{cfrac-of-real-aux (frac (1 / x))} =$

cfrac-of-real-aux x' \wedge

cfrac-of-real-alt-aux (frac (1 / x)) =

cfrac-of-real-alt-aux x' $\wedge 0 < x' \wedge x' < 1 \wedge \text{llength (cfrac-of-real-aux x')} =$

∞

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) = ∞*

using *cfrac-length-of-real-irrational by blast*

with *assms show ?thesis*

by *transfer*

(use Ints-subset-Rats in

$\langle \text{auto intro! : cfrac-of-real-alt-aux-eq-regular simp: frac-lt-1 llength-cfrac-of-real-alt-aux} \rangle$)

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 $n :: \text{nat}$ **and** $x :: \text{real}$

defines $c \equiv \text{cfrac-of-real } x$

defines $c' \equiv \text{cfrac-of-real-alt } x$

defines $l \equiv \text{cfrac-length } c$

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  $x \notin \mathbb{Q} \mid x \in \mathbb{Z} \mid n = 0 \mid x \in \mathbb{Q} - \mathbb{Z} \mid n'$  where  $n = \text{Suc } n' \mid x \in \mathbb{Q} - \mathbb{Z}$ 
  by (cases n) auto
  thus ?case
proof cases
  assume  $x \notin \mathbb{Q}$ 
  thus ?thesis
  by (auto simp: cfrac-length-of-real-irrational)
next
  assume  $x \in \mathbb{Z}$ 
  thus ?thesis
  by (auto simp: Ints-def one-enat-def zero-enat-def)
next
  assume *:  $n = 0 \mid x \in \mathbb{Q} - \mathbb{Z}$ 
  have  $\text{enat } 0 \neq \text{cfrac-length } (\text{cfrac-of-real } x) + 1$ 
  using zero-enat-def by auto
  moreover have  $\text{enat } 0 \neq \text{cfrac-length } (\text{cfrac-of-real } x)$ 
  using * cfrac-length-of-real-reduce zero-enat-def by auto
  ultimately show ?thesis using *
  by (auto simp: cfrac-nth-of-real-alt-0)
next
  fix n' assume *:  $n = \text{Suc } n' \mid x \in \mathbb{Q} - \mathbb{Z}$ 
  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)

qed
qed

```

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

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

lemma *conv-cong*:
 assumes $(\bigwedge k. k \leq n \implies \text{cfrac-nth } c \ k = \text{cfrac-nth } c' \ k) \mid n = n'$
 shows $\text{conv } c \ n = \text{conv } c' \ n'$
 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  $n \leq \text{cfrac-length } (\text{cfrac-of-real } x)$ 
  shows  $\text{conv}' (\text{cfrac-of-real-alt } x) \ n \ y = \text{conv}' (\text{cfrac-of-real } x) \ n \ y$ 
proof (cases cfrac-length (cfrac-of-real x))
  case infinity
  thus ?thesis by auto
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]:  $\text{cfrac-lim } (\text{cfrac-of-real-alt } x) = x$ 
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 0
  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)
next
  case (Suc l x)
  hence *:  $\neg \text{cfrac-is-int } (\text{cfrac-of-real-alt } x) \ x \notin \mathbb{Z}$ 
  by (auto simp: cfrac-is-int-def cfrac-length-of-real-alt Ints-def zero-enat-def eSuc-def)
  hence  $\text{cfrac-lim } (\text{cfrac-of-real-alt } x) =$ 
     $\text{of-int } \lfloor x \rfloor + 1 / \text{cfrac-lim } (\text{cfrac-tl } (\text{cfrac-of-real-alt } x))$ 
  by (subst cfrac-lim-reduce) (auto simp: cfrac-nth-of-real-alt-0)
  also have  $\text{cfrac-length } (\text{cfrac-of-real } (1 / \text{frac } x)) = l$ 
  using Suc.prem * by (metis cfrac-length-of-real-reduce eSuc-enat eSuc-inject)
  hence  $1 / \text{cfrac-lim } (\text{cfrac-tl } (\text{cfrac-of-real-alt } x)) = \text{frac } x$ 
  by (subst cfrac-tl-of-real-alt[OF *(2)], subst Suc) (use Suc.prem * in auto)
  also have  $\text{real-of-int } \lfloor x \rfloor + \text{frac } x = x$ 
  by (simp add: frac-def)
  finally show ?case .
qed
qed

lemma cfrac-eqI:
  assumes  $\text{cfrac-length } c = \text{cfrac-length } c'$  and  $\bigwedge n. \text{cfrac-nth } c \ n = \text{cfrac-nth } c' \ n$ 
  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
next
  case (2 a xs b ys)
  define f where f = (λxs n. if enat (Suc n) ≤ llength xs then int (lnth xs n) +
1 else 1)
  have ∀ n. f xs n = f ys n
    using 2(2)[of Suc n 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 ∨ lnull ys)
      case False
      from False have *: enat (Suc 0) ≤ 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 ys
        using 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 n
        using 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
    next
      case True
      thus ?thesis
        using Eq-llist(1) by auto
    qed
  qed
qed

lemma cfrac-eq-0I:
  assumes cfrac-lim c = 0 cfrac-nth c 0 ≥ 0
  shows c = 0
  proof -
    have *: cfrac-is-int c
    proof (rule ccontr)
      assume *: ¬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 0 < 0
        using assms by simp
      thus False
        using assms by simp
    qed
  from * assms have cfrac-nth c 0 = 0

```

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-1I:

assumes $\text{cfrac-lim } c = 1$ $\text{cfrac-nth } c \ 0 \neq 0$

shows $c = 1$

proof -

have *: $\text{cfrac-is-int } c$

proof (rule ccontr)

assume *: $\neg \text{cfrac-is-int } c$

from * have $\text{conv } c \ 0 < \text{cfrac-lim } c$

by (intro conv-less-cfrac-lim) (auto simp: cfrac-is-int-def simp flip: zero-enat-def)

hence $\text{cfrac-nth } c \ 0 < 0$

using assms by simp

have $\text{cfrac-lim } c = \text{real-of-int } (\text{cfrac-nth } c \ 0) + 1 / \text{cfrac-lim } (\text{cfrac-tl } c)$

using * by (subst cfrac-lim-reduce) auto

also have $\text{real-of-int } (\text{cfrac-nth } c \ 0) < 0$

using $\langle \text{cfrac-nth } c \ 0 < 0 \rangle$ by simp

also have $1 / \text{cfrac-lim } (\text{cfrac-tl } c) \leq 1$

proof -

have $1 \leq \text{cfrac-nth } (\text{cfrac-tl } c) \ 0$

by auto

also have $\dots \leq \text{cfrac-lim } (\text{cfrac-tl } c)$

by (rule cfrac-lim-ge-first)

finally show ?thesis by simp

qed

finally show False

using assms by simp

qed

from * assms have $\text{cfrac-nth } c \ 0 = 1$

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

qed

lemma cfrac-coinduct [coinduct type: cfrac]:

assumes $R \ c1 \ c2$

assumes IH: $\bigwedge c1 \ c2. R \ c1 \ c2 \implies$

$\text{cfrac-is-int } c1 = \text{cfrac-is-int } c2 \wedge$

$\text{cfrac-nth } c1 \ 0 = \text{cfrac-nth } c2 \ 0 \wedge$

$(\neg \text{cfrac-is-int } c1 \longrightarrow \neg \text{cfrac-is-int } c2 \longrightarrow R \ (\text{cfrac-tl } c1) \ (\text{cfrac-tl } c2))$

shows $c1 = c2$

proof (rule cfrac-eqI)

show $\text{cfrac-nth } c1 \ n = \text{cfrac-nth } c2 \ n$ for n

using assms(1)

```

proof (induction n arbitrary: c1 c2)
  case 0
  from IH[OF this] show ?case
  by auto
next
  case (Suc n)
  thus ?case
  using IH by (metis cfrac-is-int-iff cfrac-nth-0-of-int cfrac-nth-tl)
qed
next
show cfrac-length c1 = cfrac-length c2
  using assms(1)
proof (coinduction arbitrary: c1 c2 rule: enat-coinduct)
  case (Eq-enat c1 c2)
  show ?case
  proof (cases cfrac-is-int c1)
    case True
    thus ?thesis
    using IH[OF Eq-enat(1)] by (auto simp: cfrac-is-int-def)
  next
  case False
  with IH[OF Eq-enat(1)] have **:  $\neg$ cfrac-is-int c1  $\wedge$  (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:
   $\text{cfrac-nth } c \ 0 = \lfloor \text{cfrac-lim } c \rfloor \vee \text{cfrac-nth } c \ 0 = \lfloor \text{cfrac-lim } c \rfloor - 1 \wedge \text{cfrac-tl } c = 1$ 
proof (cases cfrac-is-int c)
  case True
  hence cfrac-nth c 0 =  $\lfloor \text{cfrac-lim } c \rfloor$ 
  by (auto simp: cfrac-lim-def cfrac-is-int-def)
  thus ?thesis by blast
next
  case False
  note not-int = this
  have bounds:  $1 / \text{cfrac-lim } (\text{cfrac-tl } c) \geq 0 \wedge 1 / \text{cfrac-lim } (\text{cfrac-tl } c) \leq 1$ 
  proof -
    have  $1 \leq \text{cfrac-nth } (\text{cfrac-tl } c) \ 0$ 
    by simp
    also have  $\dots \leq \text{cfrac-lim } (\text{cfrac-tl } c)$ 

```



```

    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  $\lfloor \text{cfrac-lim } c \rfloor = \text{cfrac-nth } c \ 0 + \lfloor 1 / \text{cfrac-lim } (\text{cfrac-tl } c) \rfloor$ 
    using not-int by (subst cfrac-lim-reduce) auto
  also have  $1 / \text{cfrac-lim } (\text{cfrac-tl } c) \geq 0 \wedge 1 / \text{cfrac-lim } (\text{cfrac-tl } c) < 1$ 
    using bounds False by (auto simp: divide-simps)
  hence  $\lfloor 1 / \text{cfrac-lim } (\text{cfrac-tl } c) \rfloor = 0$ 
    by linarith
  finally show ?thesis by simp
next
  case True
  have  $\text{cfrac-nth } c \ 0 = \lfloor \text{cfrac-lim } c \rfloor - 1$ 
    using not-int True by (subst cfrac-lim-reduce) auto
  moreover have  $\text{cfrac-tl } c = 1$ 
    using True by (intro cfrac-eq-1I) auto
  ultimately show ?thesis by blast
qed
qed

lemma cfrac-length-1 [simp]:  $\text{cfrac-length } 1 = 0$ 
  unfolding one-cfrac-def by simp

lemma cfrac-nth-1 [simp]:  $\text{cfrac-nth } 1 \ m = 1$ 
  unfolding one-cfrac-def by transfer (auto simp: enat-0-iff)

lemma cfrac-lim-1 [simp]:  $\text{cfrac-lim } 1 = 1$ 
  by (auto simp: cfrac-lim-def)

lemma cfrac-nth-0-not-int:
  assumes  $\text{cfrac-lim } c \notin \mathbb{Z}$ 
  shows  $\text{cfrac-nth } c \ 0 = \lfloor \text{cfrac-lim } c \rfloor$ 
proof -
  have  $\text{cfrac-tl } c \neq 1$ 
  proof
    assume eq:  $\text{cfrac-tl } c = 1$ 
    have  $\neg \text{cfrac-is-int } c$ 
      using assms by (auto simp: cfrac-lim-def cfrac-is-int-def)
    hence  $\text{cfrac-lim } c = \text{of-int } \lfloor \text{cfrac-nth } c \ 0 \rfloor + 1$ 
      using eq by (subst cfrac-lim-reduce) auto
    hence  $\text{cfrac-lim } c \in \mathbb{Z}$ 
      by auto
    with assms show False by auto
  qed

```

```

qed
with cfrac-nth-0-cases[of c] show ?thesis by auto
qed

lemma cfrac-of-real-cfrac-lim-irrational:
  assumes cfrac-lim c  $\notin$   $\mathbb{Q}$ 
  shows cfrac-of-real (cfrac-lim c) = c
proof (rule cfrac-eqI)
  from assms show cfrac-length (cfrac-of-real (cfrac-lim c)) = cfrac-length c
    using cfrac-lim-rational-iff by auto
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
next
case (Suc n c)
from Suc.premis 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.premis Ints-subset-Rats by (subst cfrac-tl-of-real) auto
  also have 1 / frac (cfrac-lim c) = cfrac-lim (cfrac-tl c)
    using Suc.premis 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.premis by (subst Suc.IH) (auto simp: cfrac-lim-rational-iff)
  finally show ?case .
qed
qed

lemma cfrac-eqI-first:
  assumes  $\neg$ cfrac-is-int c  $\neg$ cfrac-is-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]*: $cfrac\text{-}tl\ (cfrac\text{-}of\text{-}real\text{-}alt\ (of\text{-}int\ n)) = 1$
unfolding *one-cfrac-def* **by** *transfer auto*

lemma *cfrac-is-intI*:
assumes *cfrac-nth c 0* $\geq \lfloor cfrac\text{-}lim\ c \rfloor$ **and** *cfrac-lim c* $\in \mathbb{Z}$
shows *cfrac-is-int c*
proof (*rule ccontr*)
assume *: $\neg cfrac\text{-}is\text{-}int\ c$
from * **have** *conv c 0* $< cfrac\text{-}lim\ c$
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* $\geq \lfloor cfrac\text{-}lim\ c \rfloor$ **and** *cfrac-lim c* $\in \mathbb{Z}$
shows $c = cfrac\text{-}of\text{-}int\ \lfloor cfrac\text{-}lim\ c \rfloor$
proof –
from *assms* **have** *int: cfrac-is-int c*
by (*intro cfrac-is-intI*) *auto*
have [*simp*]: $cfrac\text{-}lim\ c = cfrac\text{-}nth\ c\ 0$
using *int* **by** (*simp add: cfrac-lim-def cfrac-is-int-def*)
from *int* **have** $c = cfrac\text{-}of\text{-}int\ (cfrac\text{-}nth\ c\ 0)$
unfolding *cfrac-is-int-def* **by** *transfer auto*
also from *assms* **have** $cfrac\text{-}nth\ c\ 0 = \lfloor cfrac\text{-}lim\ c \rfloor$
using *int* **by** *auto*
finally show *?thesis* .
qed

lemma *cfrac-lim-of-int [simp]*: $cfrac\text{-}lim\ (cfrac\text{-}of\text{-}int\ n) = of\text{-}int\ n$
by (*simp add: cfrac-lim-def*)

lemma *cfrac-of-real-of-int [simp]*: $cfrac\text{-}of\text{-}real\ (of\text{-}int\ n) = cfrac\text{-}of\text{-}int\ n$
by *transfer auto*

lemma *cfrac-of-real-of-nat [simp]*: $cfrac\text{-}of\text{-}real\ (of\text{-}nat\ n) = cfrac\text{-}of\text{-}int\ (int\ n)$
by *transfer auto*

lemma *cfrac-int-cases*:
assumes $cfrac\text{-}lim\ c = of\text{-}int\ n$
shows $c = cfrac\text{-}of\text{-}int\ n \vee c = cfrac\text{-}of\text{-}real\text{-}alt\ (of\text{-}int\ n)$
proof –
from *cfrac-nth-0-cases*[*of c*] **show** *?thesis*
proof (*rule disj-forward*)
assume *eq: cfrac-nth c 0* $= \lfloor cfrac\text{-}lim\ c \rfloor$
have $c = cfrac\text{-}of\text{-}int\ \lfloor cfrac\text{-}lim\ c \rfloor$
using *assms eq* **by** (*intro cfrac-eq-of-intI*) *auto*

```

    with assms eq show  $c = \text{cfrac-of-int } n$ 
      by simp
  next
    assume *:  $\text{cfrac-nth } c \ 0 = \lfloor \text{cfrac-lim } c \rfloor - 1 \wedge \text{cfrac-tl } c = 1$ 
    have  $\neg \text{cfrac-is-int } c$ 
      using * by (auto simp: cfrac-is-int-def cfrac-lim-def)
    hence  $\text{cfrac-length } c = \text{eSuc } (\text{cfrac-length } (\text{cfrac-tl } c))$ 
      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  $\text{cfrac-tl } c = 1$ 
      using * by auto
    finally have  $\text{cfrac-length } c = 1$ 
      by (simp add: eSuc-def one-enat-def)
    show  $c = \text{cfrac-of-real-alt } (\text{of-int } n)$ 
      by (rule cfrac-eqI-first)
        (use  $\neg \text{cfrac-is-int } c$  * assms in  $\langle \text{auto simp: cfrac-not-is-int-of-real-alt} \rangle$ )
  qed
qed

lemma cfrac-cases:
   $c \in \{\text{cfrac-of-real } (\text{cfrac-lim } c), \text{cfrac-of-real-alt } (\text{cfrac-lim } c)\}$ 
proof (cases cfrac-length c)
  case infinity
  hence  $\text{cfrac-lim } c \notin \mathbb{Q}$ 
    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 (0 c)
    hence  $c = \text{cfrac-of-real } (\text{cfrac-nth } c \ 0)$ 
      by transfer (auto simp flip: zero-enat-def)
    with 0 show ?case by (auto simp: cfrac-lim-def)
  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)
    next
      case [simp]: False
      have  $\neg \text{cfrac-is-int } c$ 
        using Suc.premis 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 ∈ ℤ
      using ⟨¬cfrac-is-int c⟩ by (subst cfrac-lim-reduce) auto
    thus ?thesis
      by (auto simp: cfrac-int-cases)
  next
    assume eq: cfrac-nth c 0 = ⌊cfrac-lim c⌋
    have cfrac-tl c = cfrac-of-real (cfrac-lim (cfrac-tl c)) ∨
      cfrac-tl c = cfrac-of-real-alt (cfrac-lim (cfrac-tl c))
      using Suc.IH[of cfrac-tl c] Suc.prem by auto
    hence c = cfrac-of-real (cfrac-lim c) ∨
      c = cfrac-of-real-alt (cfrac-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 ⟨¬cfrac-is-int c⟩ 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)) 0 = ⌊cfrac-lim c⌋
        using Suc.prem by (subst cfrac-nth-of-real-alt-0) auto
      show c = cfrac-of-real-alt (cfrac-lim c)
        by (rule cfrac-eqI-first)
        (use ⟨¬cfrac-is-int c⟩ eq eq' eq'' in
        ⟨auto simp: cfrac-not-is-int-of-real-alt cfrac-tl-of-real-alt cfrac-lim-tl
        frac-def⟩)
    qed
  qed
  thus ?thesis by simp
  qed
  qed
  qed
  qed
  qed

```

lemma *cfrac-lim-eq-iff*:

```

  assumes cfrac-length c = ∞ ∨ cfrac-length c' = ∞
  shows cfrac-lim c = cfrac-lim c' ⟷ 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 = ∞

```

shows $\lfloor \text{cfrac-remainder } c \ n \rfloor = \text{cfrac-nth } c \ n$
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: sgn-if*)

lemma *conv-ge-one*: $\text{cfrac-nth } c \ 0 > 0 \implies \text{conv } c \ n \geq 1$

by (*rule order.trans[OF - conv-ge-first]*) *auto*

context

fixes $c \ h \ k$

defines $h \equiv \text{conv-num } c$ **and** $k \equiv \text{conv-denom } c$

begin

lemma *abs-diff-le-abs-add*:

fixes $x \ y :: \text{real}$
assumes $x \geq 0 \wedge y \geq 0 \vee x \leq 0 \wedge y \leq 0$
shows $|x - y| \leq |x + y|$
using *assms* **by** *linarith*

lemma *abs-diff-less-abs-add*:

fixes $x \ y :: \text{real}$
assumes $x > 0 \wedge y > 0 \vee x < 0 \wedge y < 0$
shows $|x - y| < |x + y|$
using *assms* **by** *linarith*

lemma *abs-diff-le-imp-same-sign*:

assumes $|x - y| \leq d \ d < |y|$
shows $\text{sgn } x = \text{sgn } (y :: \text{real})$
using *assms* **by** (*auto simp: sgn-if*)

lemma *conv-nonpos*:

assumes $\text{cfrac-nth } c \ 0 < 0$
shows $\text{conv } c \ n \leq 0$

proof (*cases n*)

case 0

thus *?thesis* **using** *assms* **by** *auto*

next

case [*simp*]: (*Suc n'*)

have $\text{conv } c \ n = \text{real-of-int } (\text{cfrac-nth } c \ 0) + 1 / \text{conv } (\text{cfrac-tl } c) \ n'$

by (*simp add: conv-Suc*)

also have $\dots \leq -1 + 1 / 1$

```

    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 0 < 0
  shows cfrac-lim c ≤ 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⟩)
next
  case (enat l)
  thus ?thesis by (auto simp: cfrac-lim-def conv-nonpos assms)
qed

lemma conv-num-nonpos:
  assumes cfrac-nth c 0 < 0
  shows h n ≤ 0
proof (induction n rule: fib.induct)
  case 2
  have cfrac-nth c (Suc 0) * cfrac-nth c 0 ≤ 1 * cfrac-nth c 0
    using assms by (intro mult-right-mono-neg) auto
  also have ... + 1 ≤ 0 using assms by auto
  finally show ?case by (auto simp: h-def)
next
  case (3 n)
  have cfrac-nth c (Suc (Suc n)) * h (Suc n) ≤ 0
    using 3 by (simp add: mult-nonneg-nonpos)
  also have ... + h n ≤ 0
    using 3 by simp
  finally show ?case
    by (auto simp: h-def)
qed (use assms in ⟨auto simp: h-def⟩)

lemma conv-best-approximation-aux:
  cfrac-lim c ≥ 0 ∧ h n ≥ 0 ∨ cfrac-lim c ≤ 0 ∧ h n ≤ 0
proof (cases cfrac-nth c 0 ≥ 0)
  case True
  from True have 0 ≤ conv c 0
    by simp
  also have ... ≤ cfrac-lim c
    by (rule conv-le-cfrac-lim) (auto simp: enat-0)
  finally have cfrac-lim c ≥ 0 .
  moreover from True have h n ≥ 0
    unfolding h-def by (intro conv-num-nonneg)
  ultimately show ?thesis by (simp add: sgn-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 ≤ cfrac-length c
  assumes 0 < b and b ≤ k n and coprime a b and n > 0
  assumes (a, b) ≠ (h n, k n)
  assumes ¬(cfrac-length c = 1 ∧ n = 0)
  assumes Suc n ≠ cfrac-length c ∨ cfrac-canonical c
  defines x ≡ cfrac-lim c
  shows |k n * x - h n| < |b * x - a|
proof (cases |a| = |h n| ∧ 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 ∨ 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)
    next
      assume [simp]: c = cfrac-of-real-alt 0
      hence n = 0 ∨ 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)
    qed
  thus ?thesis ..
next
  case False
  have h n ≠ 0
    using True assms(6) h-def by auto
  hence x > 0 ∧ h n > 0 ∨ x < 0 ∧ h n < 0
    using ⟨x ≠ 0⟩ 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 ⟨k n > 0⟩
  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 ... = 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 ≠ 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 * b) = (of-int (k n * a) :: real)
    by (simp only: of-int-mult)
    hence h n * b = k n * a
    by linarith
    hence h n = a ∧ 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 0 < |b * x - a|
  by simp
  finally show ?thesis .
next
case False

define s where s = (-1) ^ n * (a * k n - b * h n)
define r where r = (-1) ^ n * (b * h (Suc n) - a * k (Suc n))
have k n ≤ k (Suc n)
  unfolding k-def by (intro conv-denom-leI) auto

have r * h n + s * h (Suc n) =
  (-1) ^ 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 ... = 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 $r * k \ n + s * k \ (Suc \ n) =$
 $(-1) \wedge Suc \ n * b * (k \ (Suc \ n) * h \ n - k \ n * h \ (Suc \ n))$
 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 $r = 0$
 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 $*: abs \ (h \ (Suc \ n)) = abs \ a \wedge k \ (Suc \ n) = b$
 unfolding 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 $s = 0$
 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 co-
 prime-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 $\dots \in \{0 <..< k \ (Suc \ n)\}$ using assms $\langle k \ n < k \ (Suc \ n) \rangle$ by auto
 finally have $*: r * k \ n + s * k \ (Suc \ n) \in \dots$.

have opposite-signs1: $r > 0 \wedge s < 0 \vee r < 0 \wedge s > 0$
 proof (cases $r \geq 0$; cases $s \geq 0$)
 assume $r \geq 0 \ s \geq 0$
 hence $0 * (k \ n) + 1 * (k \ (Suc \ n)) \leq r * k \ n + s * k \ (Suc \ n)$
 using $\langle s \neq 0 \rangle$ by (intro add-mono mult-mono) (auto simp: k-def)
 with * show ?thesis by auto
 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  $r \neq 1$ 
proof
  assume [simp]:  $r = 1$ 
  have  $b = r * k\ n + s * k\ (Suc\ n)$ 
    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  $\dots \leq 0$ 
    unfolding k-def by (auto intro!: conv-denom-leI)
  finally show False using  $\langle b > 0 \rangle$  by simp
qed

have  $enat\ n \leq cfrac\text{-}length\ c\ enat\ (Suc\ n) \leq cfrac\text{-}length\ c$ 
  using assms False by (cases cfrac-length c; simp)+
hence  $conv\ c\ n \geq x \wedge conv\ c\ (Suc\ n) \leq x \vee conv\ c\ n \leq x \wedge conv\ c\ (Suc\ n) \geq x$ 
  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-signs2:  $k\ n * x - h\ n \geq 0 \wedge k\ (Suc\ n) * x - h\ (Suc\ n) \leq 0 \vee$ 
   $k\ n * x - h\ n \leq 0 \wedge k\ (Suc\ n) * x - h\ (Suc\ n) \geq 0$ 
  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:
   $r * (k\ n * x - h\ n) \geq 0 \wedge s * (k\ (Suc\ n) * x - h\ (Suc\ n)) \geq 0 \vee$ 
   $r * (k\ n * x - h\ n) \leq 0 \wedge s * (k\ (Suc\ n) * x - h\ (Suc\ n)) \leq 0$ 
  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]:  $r = -1$ 
      have  $r * k\ n + s * k\ (Suc\ n) = b$ 
        by 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)
    end
  end

```

finally have $s < 2$
 by (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 simp

have $b = (\text{cfrac-nth } c \text{ (Suc } n) - 1) * k \ n + k \ (n - 1)$
 using eq2 $\langle n > 0 \rangle$ by (cases n) (auto simp: k-def algebra-simps)
 also have $\text{cfrac-nth } c \text{ (Suc } n) > 1$
 proof -
 have cfrac-canonical c
 using assms True by auto
 hence $\text{cfrac-nth } c \text{ (Suc } n) \neq 1$
 using True[symmetric] by (auto simp: cfrac-canonical-iff enat-0-iff)
 moreover have $\text{cfrac-nth } c \text{ (Suc } n) > 0$
 by auto
 ultimately show $\text{cfrac-nth } c \text{ (Suc } n) > 1$
 by linarith
 qed
 hence $(\text{cfrac-nth } c \text{ (Suc } n) - 1) * k \ n + k \ (n - 1) \geq 1 * k \ n + k \ (n - 1)$
 by (intro add-mono mult-right-mono) (auto simp: k-def)
 finally have $b > k \ n$
 using 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 $|r| > 1$
 by auto

from $\langle s \neq 0 \rangle$ have $k \ n * x \neq h \ n$
 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 $\dots = |r| * |k \ n * x - h \ n| + 0$ by simp
 also have $\dots \leq |r * (k \ n * x - h \ n)| + |s * (k \text{ (Suc } n) * x - h \text{ (Suc } n))|$
 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 $\dots = |r * (k \ n * x - h \ n) + s * (k \text{ (Suc } n) * x - h \text{ (Suc } n))|$
 using same-signs by auto
 also have $\dots = |(r * k \ n + s * k \text{ (Suc } n)) * x - (r * h \ n + s * h \text{ (Suc } n))|$
 by (simp add: algebra-simps)
 also have $\dots = |b * x - a|$
 unfolding eq1 eq2 by simp
 finally show ?thesis by simp

next
 case False
 from assms have $\text{Suc } n < \text{cfrac-length } c$
 using False $\langle \text{Suc } n \leq \text{cfrac-length } c \rangle$ by force
 have $1 * |k \ n * x - h \ n| \leq |r| * |k \ n * x - h \ n|$
 using $\langle r \neq 0 \rangle$ by (intro mult-right-mono) auto

```

    also have ... = |r| * |k n * x - h n| + 0 by simp
    also have x ≠ h (Suc n) / k (Suc n)
      using conv-neq-cfrac-lim[of Suc n c] ⟨Suc n < cfrac-length c⟩
      by (auto simp: conv-num-denom h-def k-def x-def)
    hence |s * (k (Suc n) * x - h (Suc n))| > 0
      using ⟨s ≠ 0⟩ by (auto simp: field-simps k-def)
    also have |r| * |k n * x - h n| + ... ≤
      |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-nonneg-nonneg)
  auto
    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 ... = |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 and x :: real
  assumes n ≤ cfrac-length c
  assumes 0 < b and b < k (Suc n) and coprime a b
  defines x ≡ cfrac-lim c
  shows |k n * x - h n| ≤ |b * x - a|
proof (cases |a| = |h n| ∧ b = k n)
  case True
    note * = this
    show ?thesis
    proof (cases sgn a = sgn (h n))
      case True
        with * have [simp]: a = h n
          by (auto simp: abs-if split: if-splits)
        thus ?thesis using * by auto
      next
        case False
          with True have [simp]: a = -h n
            by (auto simp: abs-if split: if-splits)
          have |real-of-int (k n) * x - real-of-int (h n)| ≤ |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 ... = h n / k n
    by (auto simp: h-def k-def conv-num-denom)
    finally show ?thesis by (auto simp: k-def)
  next
    case False

    define s where s = (-1) ^ n * (a * k n - b * h n)
    define r where r = (-1) ^ n * (b * h (Suc n) - a * k (Suc n))

    have r * h n + s * h (Suc n) =
      (-1) ^ 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 ... = 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 ... = 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 ≠ 0
    proof
      assume r = 0
      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 ≠ 0
    proof
      assume s = 0
      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 ∧ |a| = |h n| unfolding abs-mult h-def k-def using co-
      prime-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 $\dots \in \{0 < .. < k \ (Suc \ n)\}$ **using** *assms* **by** *auto*
 finally have $\ast: r * k \ n + s * k \ (Suc \ n) \in \dots$.

have *opposite-signs1*: $r > 0 \wedge s < 0 \vee r < 0 \wedge s > 0$
proof (*cases* $r \geq 0$; *cases* $s \geq 0$)
 assume $r \geq 0 \ s \geq 0$
 hence $0 * (k \ n) + 1 * (k \ (Suc \ n)) \leq r * k \ n + s * k \ (Suc \ n)$
 using $\langle s \neq 0 \rangle$ **by** (*intro add-mono mult-mono*) (*auto simp: k-def*)
 with \ast **show** *?thesis* **by** *auto*
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 \ast **show** *?thesis* **by** *auto*
qed (*insert* $\langle r \neq 0 \rangle \ \langle s \neq 0 \rangle$, *auto*)

have $enat \ n \leq cfrac\text{-length} \ c \ enat \ (Suc \ n) \leq cfrac\text{-length} \ c$
 using *assms* *False* **by** (*cases cfrac-length c*; *simp*)
 hence $conv \ c \ n \geq x \wedge conv \ c \ (Suc \ n) \leq x \vee conv \ c \ n \leq x \wedge conv \ c \ (Suc \ n) \geq x$
 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-signs2*: $k \ n * x - h \ n \geq 0 \wedge k \ (Suc \ n) * x - h \ (Suc \ n) \leq 0 \vee$
 $k \ n * x - h \ n \leq 0 \wedge k \ (Suc \ n) * x - h \ (Suc \ n) \geq 0$
 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*:
 $r * (k \ n * x - h \ n) \geq 0 \wedge s * (k \ (Suc \ n) * x - h \ (Suc \ n)) \geq 0 \vee$
 $r * (k \ n * x - h \ n) \leq 0 \wedge s * (k \ (Suc \ n) * x - h \ (Suc \ n)) \leq 0$
by (*auto intro: mult-nonpos-nonneg mult-nonneg-nonpos mult-nonneg-nonneg*
mult-nonpos-nonpos)

have $1 * |k \ n * x - h \ n| \leq |r| * |k \ n * x - h \ n|$
 using $\langle r \neq 0 \rangle$ **by** (*intro mult-right-mono*) *auto*
 also have $\dots = |r| * |k \ n * x - h \ n| + 0$ **by** *simp*
 also have $\dots \leq |r * (k \ n * x - h \ n)| + |s * (k \ (Suc \ n) * x - h \ (Suc \ n))|$
 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 $\dots = |r * (k \ n * x - h \ n) + s * (k \ (Suc \ n) * x - h \ (Suc \ n))|$
 using *same-signs* **by** *auto*
 also have $\dots = |(r * k \ n + s * k \ (Suc \ n)) * x - (r * h \ n + s * h \ (Suc \ n))|$
by (*simp add: algebra-simps*)
 also have $\dots = |b * x - a|$
 unfolding *eq1 eq2* **by** *simp*
 finally **show** *?thesis* **by** *simp*
qed

qed

lemma *cfrac-canonical-reduce*:

cfrac-canonical $c \longleftrightarrow$

$cfrac-is-int\ c \vee \neg cfrac-is-int\ c \wedge cfrac-tl\ c \neq 1 \wedge cfrac-canonical\ (cfrac-tl\ c)$

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 $cfrac-canonical\ c \vee cfrac-length\ c \neq 1$

shows $cfrac-nth\ c\ 0 = \lfloor cfrac-lim\ c \rfloor$

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*

next

case *False*

hence $cfrac-length\ c > 1$

using $\langle \neg cfrac-is-int\ c \rangle$

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 $0 < cfrac-length\ c - 1$

proof (*cases cfrac-length c*)

case [*simp*]: (*enat l*)

have $cfrac-length\ c - 1 = enat\ (l - 1)$

by *auto*

also have $\dots > enat\ 0$

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 $\dots = cfrac-length\ (cfrac-tl\ c)$

by *simp*

also have $cfrac-tl\ c = 1$

by *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 $a\ b :: \text{nat}$ **and** $x :: \text{real}$
assumes $n \leq \text{cfrac-length } c$ $0 < b$ $b < k\ (Suc\ n)$ *coprime* $a\ b$
shows $|k\ n * \text{cfrac-lim } c - h\ n| \leq |b * \text{cfrac-lim } c - a|$
using *conv-best-approximation-ex-weak*[*OF* *assms*(1), *of* $b\ a$] *assms* **by** *auto*

lemma *abs-mult-nonneg-left*:
assumes $x \geq (0 :: 'a :: \{\text{ordered-ab-group-add-abs, idom-abs-sgn}\})$
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 $a\ b :: \text{int}$ **and** $x :: \text{real}$
assumes $n \leq \text{cfrac-length } c$
assumes $0 < b$ **and** $b < k\ n$ **and** *coprime* $a\ b$
defines $x \equiv \text{cfrac-lim } c$
shows $|x - \text{conv } c\ n| \leq |x - a / b|$
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 $|x - \text{conv } c\ n| = |k\ n * x - h\ n| / k\ n$
using *conv-denom-pos*[*of* $c\ n$] *assms*(1)
by (*auto simp: conv-num-denom field-simps k-def h-def*)
also **have** $\dots \leq |b * x - a| / k\ n$ **unfolding** *x-def* **using** *assms* $*$
by (*intro divide-right-mono conv-best-approximation-ex-weak*) *auto*
also **from** *assms* **have** $\dots \leq |b * x - a| / b$
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-ex1I*)
from *conv-denom-at-top*[*of* c] *assms* **have** $*$: $\exists n. k\ n \geq y + 1$
by (*auto simp: k-def filterlim-at-top eventually-at-top-linorder*)

```

define n where n = (LEAST n. k n ≥ 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) ≤ y
proof (rule ccontr)
  assume ¬k (n - 1) ≤ y
  hence k (n - 1) ≥ y + 1 by auto
  hence n - 1 ≥ n unfolding n-def by (rule Least-le)
  with ⟨n > 0⟩ show False by simp
qed
with n and ⟨n > 0⟩ have y ∈ {k (n - 1)..<k (Suc (n - 1))} by auto
thus ∃ n. y ∈ {k n..

```

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 ≡ cfrac-lim c
  assumes b > 0 and coprime a b and |x - a / b| < 1 / (2 * b2)
  shows ∃ n. enat n ≤ cfrac-length c ∧ (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 0 < x x < 1/2
    using assms by auto
  hence x ∉ 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 0, k 0)
    by (auto simp: h-def k-def)
  thus ?thesis by (intro exI[of - 0]) (auto simp flip: zero-enat-def)
next
case less
hence  $x < 0 \wedge x > -1/2$ 
  using assms by auto
hence  $x \notin \mathbb{Z}$ 
  by (auto simp: Ints-def)
hence not-int:  $\neg \text{cfrac-is-int } c$ 
  by (auto simp: cfrac-is-int-def x-def cfrac-lim-def)
have cfrac-nth c 0 =  $\lfloor x \rfloor$ 
  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 ... = -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 ...  $\in \{1 < .. < 2\}$ 
  by (auto simp: divide-simps)
finally have *: cfrac-lim (cfrac-tl c)  $\in \{1 < .. < 2\}$  .
have cfrac-nth (cfrac-tl c) 0 =  $\lfloor \text{cfrac-lim } (cfrac-tl c) \rfloor$ 
  using * by (subst cfrac-nth-0-not-int) (auto simp: Ints-def)
also have ... = 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)
next
case equal
show ?thesis
  using cfrac-nth-0-cases[of c]
proof
  assume cfrac-nth c 0 =  $\lfloor \text{cfrac-lim } c \rfloor$ 
  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 =  $\lfloor \text{cfrac-lim } c \rfloor - 1 \wedge \text{cfrac-tl } c = 1$ 
  have [simp]: cfrac-nth c 0 = -1
    using * equal by (auto simp: x-def)

```

```

from * have ¬cfrac-is-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 ⟨¬cfrac-is-int c⟩ have cfrac-length c ≥ 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 ≠ 0 by auto

have x ≠ 0
proof
  assume [simp]: x = 0
  hence |a| / b < 1 / (2 * b ^ 2)
    using assms by simp
  hence |a| < 1 / (2 * b)
    using assms by (simp add: field-simps power2-eq-square)
  also have ... ≤ 1 / 2
    using assms by (intro divide-left-mono) auto
  finally have a = 0 by auto
  with ⟨a ≠ 0⟩ show False by simp
qed

show ?thesis
proof (rule ccontr)
  assume no-convergent: ¬∃ n. enat n ≤ cfrac-length c ∧ (a, b) = (h n, k n)
  from assms have ∃! r. b ∈ {k r..<k (Suc r)}
    by (intro conv-denom-partition) auto
  then obtain r where r: b ∈ {k r..<k (Suc r)} by auto
  have k r > 0
    using conv-denom-pos[of c r] assms by (auto simp: k-def)

  show False
proof (cases enat r ≤ cfrac-length c)
  case False
  then obtain l where l: cfrac-length c = enat l
    by (cases cfrac-length c) auto
  have k l ≤ k r
    using False l unfolding k-def by (intro conv-denom-leI) auto

```

also have $\dots \leq b$
 using r by *simp*
 finally have $b \geq k \, l$.

have $x = \text{conv } c \, l$
 by (*auto simp: x-def cfrac-lim-def l*)
 hence $x\text{-eq}: x = h \, l / k \, l$
 by (*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* $\langle k \, l > 0 \rangle$ 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 $\langle b > 0 \rangle \langle k \, l > 0 \rangle$ by (*subst abs-mult*) *auto*
 also have $\dots = \text{of-int } |b * h \, l - a * k \, l|$
 using $\langle b > 0 \rangle \langle k \, l > 0 \rangle$ by (*simp add: algebra-simps*)
 also have $k \, l / (2 * b) < 1$
 using $\langle b \geq k \, l \rangle \langle b > 0 \rangle$ by *auto*
 finally have $a * k \, l = b * h \, l$
 by *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 *coprime a b* 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*

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 $|k \, r * x - h \, r| \leq |b * x - a|$
 using *assms r True* unfolding *x-def* by (*intro conv-best-approximation-ex-weak*)
auto

also have $\dots = b * |x - a / b|$
 using $\langle b > 0 \rangle$ by (*simp add: field-simps*)
 also have $\dots < b * (1 / (2 * b^2))$
 using $\langle b > 0 \rangle$ by (*intro mult-strict-left-mono assms*) *auto*
 finally have *less*: $|x - \text{conv } c \, r| < 1 / (2 * b * k \, r)$
 using $\langle k \, r > 0 \rangle$ and $\langle b > 0 \rangle$ and *assms*
 by (*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 $\dots = 1 / (2 * b) * (1 / b)$
 by (*simp add: power2-eq-square*)
 also have $\dots \leq 1 / (2 * b) * (|a| / b)$

```

    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)
  auto
  also have ... = |a / b| using assms by simp
  finally have sgn x = sgn (a / b)
    by (auto simp: sgn-if split: if-splits)
  hence sgn x = sgn a using assms by (auto simp: sgn-of-int)
  hence a ≥ 0 ∧ x ≥ 0 ∨ a ≤ 0 ∧ x ≤ 0
    by (auto simp: sgn-if split: if-splits)
  moreover have h r ≥ 0 ∧ x ≥ 0 ∨ h r ≤ 0 ∧ x ≤ 0
    using conv-best-approximation-aux[of r] by (auto simp: h-def x-def)
  ultimately have signs: h r ≥ 0 ∧ a ≥ 0 ∨ h r ≤ 0 ∧ a ≤ 0
    using ⟨x ≠ 0⟩ by auto

  with no-convergent assms assms True have |h r| ≠ |a| ∨ b ≠ k r
    by (auto simp: h-def k-def)

  hence |h r| * |b| ≠ |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 ≠ |a| * k r using assms by (simp add: k-def)
  hence k r * a - h r * b ≠ 0
    using signs by (auto simp: algebra-simps)
  hence |k r * a - h r * b| ≥ 1 by presburger
  hence real-of-int 1 / (k r * b) ≤ |k r * a - h r * b| / (k r * b)
    using assms
    by (intro divide-right-mono, subst of-int-le-iff) (auto simp: k-def)
  also have ... = |(real-of-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 r
    using assms ⟨k r > 0⟩ 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 ... ≤ |x - conv c r| + |x - a / b|
    by (rule abs-triangle-ineq4)
  also have ... < 1 / (2 * b * k r) + 1 / (2 * b2)
    by (intro add-strict-mono assms less)
  finally have k r > b
    using ⟨b > 0⟩ and ⟨k r > 0⟩ 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 ⇒ int) ⇒ int × int × nat ⇒ nat ⇒ 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 (λ(-, (-, -, n), N). Suc N - n)) auto

lemmas [simp del] = conv-gen.simps

```

```

lemma conv-gen-aux-simps [simp]:
  n > N ⇒ conv-gen c (a, b, n) N = b
  n ≤ N ⇒ 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 ≤ N ⇒ conv-num c n = b * cfrac-nth c n + a ⇒
    conv-num c (Suc n) = conv-num c n * cfrac-nth c (Suc n) + b ⇒
    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) ≤ N)
    case True
    thus ?thesis
    by (subst 1) (insert 1.prem, auto)
  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 ≤ N ⇒ conv-denom c n = b * cfrac-nth c n + a ⇒
    conv-denom c (Suc n) = conv-denom c n * cfrac-nth c (Suc n) + b ⇒
    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) ≤ N)
    case True
    thus ?thesis
    by (subst 1) (insert 1.prem, auto)
  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*) *n*

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

proof –

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*

by (*simp-all add: conv-num-fun-def conv-num-code conv-denom-fun-def conv-denom-code*)
qed

1.6 Computing the continued fraction expansion of a rational number

function *cfrac-list-of-rat* :: *int* × *int* ⇒ *int* list **where**

cfrac-list-of-rat (*a*, *b*) =

(if *b* = 0 then [0]

else *a* div *b* # (if *a* mod *b* = 0 then [] else *cfrac-list-of-rat* (*b*, *a* mod *b*)))

by *auto*

termination

by (*relation measure* ($\lambda(a,b). \text{nat } (\text{abs } b)$)) (*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 ∧ (∀ *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  $c' = \text{cfrac-of-real } (b / (a \bmod b))$ 
define  $xs'$  where  $xs' = (\text{if } a \bmod b = 0 \text{ then } [] \text{ else } \text{cfrac-list-of-rat } (b, a \bmod b))$ 
define  $xs$  where  $xs = a \text{ div } b \# xs'$ 

have  $[simp]: \text{cfrac-nth } c \ 0 = a \text{ div } b$ 
by  $(\text{auto simp: c-def floor-divide-of-int-eq})$ 

obtain  $l$  where  $l: \text{cfrac-length } c = \text{enat } l$ 
by  $(\text{cases cfrac-length } c) (\text{auto simp: c-def})$ 

have  $\text{length } xs = l + 1 \wedge (\forall i < \text{length } xs. xs ! i = \text{cfrac-nth } c \ i)$ 
proof  $(\text{cases } b \text{ dvd } a)$ 
  case True
    thus ?thesis using  $l$ 
    by  $(\text{auto simp: Let-def xs-def xs'-def c-def of-int-divide-in-Ints one-enat-def enat-0-iff})$ 
  next
    case False
    have  $l \neq 0$ 
    using  $l \text{ False cfrac-of-real-length-eq-0-iff}[of \ a \ / \ b] \langle b \neq 0 \rangle$ 
    by  $(\text{auto simp: c-def zero-enat-def real-of-int-divide-in-Ints-iff intro!: Nat.gr0I})$ 
    have  $c': c' = \text{cfrac-tl } c$ 
    using  $\langle b \neq 0 \rangle$  unfolding  $c'\text{-def } c\text{-def}$ 
    by  $(\text{subst cfrac-tl-of-real}) (\text{auto simp: real-of-int-divide-in-Ints-iff frac-fraction})$ 
    from  $1$  have  $\text{enat } (\text{length } xs') = \text{cfrac-length } c' + 1$ 
    and  $xs': \forall i < \text{length } xs'. xs' ! i = \text{cfrac-nth } c' \ i$ 
    using  $\langle b \neq 0 \rangle \langle \neg b \text{ dvd } a \rangle$  by  $(\text{auto simp: Let-def xs'-def } c'\text{-def})$ 

    have  $\text{enat } (\text{length } xs') = \text{cfrac-length } c' + 1$ 
    by fact
    also have  $\dots = \text{enat } l - 1 + 1$ 
    using  $c' \ l$  by simp
    also have  $\dots = \text{enat } (l - 1 + 1)$ 
    by  $(\text{metis enat-diff-one one-enat-def plus-enat-simps}(1))$ 
    also have  $l - 1 + 1 = l$ 
    using  $\langle l \neq 0 \rangle$  by simp
    finally have  $[simp]: \text{length } xs' = l$ 
    by simp

    from  $xs'$  show ?thesis
    by  $(\text{auto simp: xs-def nth-Cons } c' \text{ split: nat.splits})$ 
  qed
thus ?thesis using  $l \text{ False}$ 
by  $(\text{subst cfrac-list-of-rat.simps}) (\text{simp-all add: xs-def xs'-def c-def one-enat-def})$ 
qed
qed

lemma conv-num-cong:

```

assumes $(\bigwedge k. k \leq n \implies \text{cfrac-nth } c \ k = \text{cfrac-nth } c' \ k) \ n = n'$
shows $\text{conv-num } c \ n = \text{conv-num } c' \ n$
proof –
have $\text{conv-num } c \ n = \text{conv-num } c' \ n$
using $\text{assms}(1)$
by $(\text{induction } n \text{ arbitrary: rule: conv-num.induct}) \text{ simp-all}$
thus $?thesis$ **using** $\text{assms}(2)$
by simp
qed

lemma conv-denom-cong :
assumes $(\bigwedge k. k \leq n \implies \text{cfrac-nth } c \ k = \text{cfrac-nth } c' \ k) \ n = n'$
shows $\text{conv-denom } c \ n = \text{conv-denom } c' \ n'$
proof –
have $\text{conv-denom } c \ n = \text{conv-denom } c' \ n$
using $\text{assms}(1)$
by $(\text{induction } n \text{ arbitrary: rule: conv-denom.induct}) \text{ simp-all}$
thus $?thesis$ **using** $\text{assms}(2)$
by simp
qed

lemma cfrac-lim-diff-le :
assumes $\forall k \leq \text{Suc } n. \text{cfrac-nth } c1 \ k = \text{cfrac-nth } c2 \ k$
assumes $n \leq \text{cfrac-length } c1 \ n \leq \text{cfrac-length } c2$
shows $|\text{cfrac-lim } c1 - \text{cfrac-lim } c2| \leq 2 / (\text{conv-denom } c1 \ n * \text{conv-denom } c1 \ (\text{Suc } n))$
proof –
define d **where** $d = (\lambda k. \text{conv-denom } c1 \ k)$
have $|\text{cfrac-lim } c1 - \text{cfrac-lim } c2| \leq |\text{cfrac-lim } c1 - \text{conv } c1 \ n| + |\text{cfrac-lim } c2 - \text{conv } c1 \ n|$
by linarith
also have $|\text{cfrac-lim } c1 - \text{conv } c1 \ n| \leq 1 / (d \ n * d \ (\text{Suc } n))$
unfolding $d\text{-def}$ **using** assms
by $(\text{intro } \text{cfrac-lim-minus-conv-upper-bound}) \text{ auto}$
also have $\text{conv } c1 \ n = \text{conv } c2 \ n$
using assms **by** $(\text{intro } \text{conv-cong}) \text{ auto}$
also have $|\text{cfrac-lim } c2 - \text{conv } c2 \ n| \leq 1 / (\text{conv-denom } c2 \ n * \text{conv-denom } c2 \ (\text{Suc } n))$
using assms **unfolding** $d\text{-def}$ **by** $(\text{intro } \text{cfrac-lim-minus-conv-upper-bound}) \text{ auto}$
also have $\text{conv-denom } c2 \ n = d \ n$
unfolding $d\text{-def}$ **using** assms **by** $(\text{intro } \text{conv-denom-cong}) \text{ auto}$
also have $\text{conv-denom } c2 \ (\text{Suc } n) = d \ (\text{Suc } n)$
unfolding $d\text{-def}$ **using** assms **by** $(\text{intro } \text{conv-denom-cong}) \text{ auto}$
also have $1 / (d \ n * d \ (\text{Suc } n)) + 1 / (d \ n * d \ (\text{Suc } n)) = 2 / (d \ n * d \ (\text{Suc } n))$
by simp
finally show $?thesis$
by $(\text{simp add: } d\text{-def})$
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  $\forall k \leq Suc\ n. cfrac\text{-}nth\ c1\ k = cfrac\text{-}nth\ c2\ k$ 
  assumes  $n \leq cfrac\text{-}length\ c1 \wedge n \leq cfrac\text{-}length\ c2$ 
  shows  $|cfrac\text{-}lim\ c1 - cfrac\text{-}lim\ c2| \leq 2 / (fib\ (n+1) * fib\ (n+2))$ 
proof -
  have  $|cfrac\text{-}lim\ c1 - cfrac\text{-}lim\ c2| \leq 2 / (conv\text{-}denom\ c1\ n * conv\text{-}denom\ c1\ (Suc\ n))$ 
  by (rule cfrac-lim-diff-le) (use assms in auto)
  also have  $\dots \leq 2 / (int\ (fib\ (Suc\ n)) * int\ (fib\ (Suc\ (Suc\ n))))$ 
  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-neg-0-nat simp del: fib.simps)
  also have  $\dots = 2 / (fib\ (n+1) * fib\ (n+2))$ 
  by simp
  finally show ?thesis .
qed

end

```

2 Quadratic Irrationals

```

theory Quadratic-Irrationals
imports
  Continued-Fractions
  HOL-Computational-Algebra.Computational-Algebra
  HOL-Library.Discrete
  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
next
  case False
  have  $0 < length\ xs$ 
  using assms by simp
  also have  $\dots \leq n$ 
  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]: $\text{inverse } (x :: \text{real}) \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
 by (auto simp: inverse-eq-divide divide-in-Rats-iff1)

lemma *nonneg-sqrt-nat-or-irrat*:

assumes $x^2 = \text{real } a$ and $x \geq 0$

shows $x \in \mathbb{N} \vee x \notin \mathbb{Q}$

proof *safe*

assume $x \notin \mathbb{N}$ and $x \in \mathbb{Q}$

from *Rats-abs-nat-div-natE*[OF this(2)]

obtain $p \ q :: \text{nat}$ where *q-nz* [simp]: $q \neq 0$ and $\text{abs } x = p / q$ and *coprime*:

coprime $p \ q$.

with ⟨ $x \geq 0$ ⟩ have $x = p / q$

by *simp*

with *assms* have $\text{real } (q^2) * \text{real } a = \text{real } (p^2)$

by (*simp* add: *field-simps*)

also have $\text{real } (q^2) * \text{real } a = \text{real } (q^2 * a)$

by *simp*

finally have $p^2 = q^2 * a$

by (*subst* (*asm*) *of-nat-eq-iff*) *auto*

hence $q^2 \text{ dvd } p^2$

by *simp*

hence $q \text{ dvd } p$

by *simp*

with *coprime* have $q = 1$

by *auto*

with x and ⟨ $x \notin \mathbb{N}$ ⟩ show *False*

by *simp*

qed

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

corollary *sqrt-nat-or-irrat*:

assumes $x^2 = \text{real } a$

```

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

```

```

corollary sqrt-nat-or-irrat':
   $\text{sqrt } (\text{real } a) \in \mathbb{N} \vee \text{sqrt } (\text{real } a) \notin \mathbb{Q}$ 
  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:
   $\text{sqrt } (\text{real } n) \in \mathbb{N} \longleftrightarrow \text{is-square } n$ 
proof
  assume  $\text{sqrt } (\text{real } n) \in \mathbb{N}$ 
  then obtain k where  $\text{sqrt } (\text{real } n) = \text{real } k$  by (auto elim!: Nats-cases)
  hence  $\text{sqrt } (\text{real } n) ^ 2 = \text{real } (k ^ 2)$  by (simp only: of-nat-power)
  also have  $\text{sqrt } (\text{real } n) ^ 2 = \text{real } n$  by simp
  finally have  $n = k ^ 2$  by (simp only: of-nat-eq-iff)
  thus is-square n by blast
qed (auto elim!: is-nth-powerE)

```

```

corollary irrat-sqrt-nonsquare:  $\neg \text{is-square } n \implies \text{sqrt } (\text{real } n) \notin \mathbb{Q}$ 
  using sqrt-nat-or-irrat'[of n] by (auto simp: sqrt-nat-iff-is-square)

```

```

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

```

```

lemma Discrete-sqrt-altdef:  $\text{Discrete.sqrt } n = \text{nat } \lfloor \text{sqrt } n \rfloor$ 
proof –
  have  $\text{real } (\text{Discrete.sqrt } n ^ 2) \leq \text{sqrt } n ^ 2$ 
    by simp
  hence  $\text{Discrete.sqrt } n \leq \text{sqrt } n$ 
    unfolding of-nat-power by (rule power2-le-imp-le) auto
  moreover have  $\text{real } (\text{Suc } (\text{Discrete.sqrt } n) ^ 2) > \text{real } n$ 
    unfolding of-nat-less-iff by (rule Suc-sqrt-power2-gt)

```

```

hence real (Discrete.sqrt n + 1) ^ 2 > sqrt n ^ 2
  unfolding of-nat-power by simp
hence real (Discrete.sqrt n + 1) > sqrt n
  by (rule power2-less-imp-less) auto
hence Discrete.sqrt n + 1 > sqrt n by simp
ultimately show ?thesis by linarith
qed

```

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**
 $x \notin \mathbb{Q} \implies \text{real-of-int } a * x^2 + \text{real-of-int } b * x + \text{real-of-int } c = 0 \implies$
 $a \neq 0 \vee b \neq 0 \vee c \neq 0 \implies \text{quadratic-irrational } x$

lemma *quadratic-irrational-sqrt* [intro]:
assumes $\neg \text{is-square } n$
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-uminus-iff* [simp]:
 $\text{quadratic-irrational } (-x) \longleftrightarrow \text{quadratic-irrational } x$
using *quadratic-irrational-uminus*[of x] *quadratic-irrational-uminus*[of $-x$] **by** auto

lemma *quadratic-irrational-plus-int* [intro]:
assumes *quadratic-irrational* x
shows *quadratic-irrational* ($x + \text{of-int } n$)
using *assms*
proof *induction*
case (1 x a b c)
define x' **where** $x' = x + \text{of-int } n$
define a' b' c' **where**
 $a' = a$ **and** $b' = b - 2 * \text{of-int } n * a$ **and**
 $c' = a * \text{of-int } n^2 - b * \text{of-int } n + c$
from 1 **have** $0 = a * (x' - \text{of-int } n)^2 + b * (x' - \text{of-int } n) + c$
by (simp add: x' -def)

also have $\dots = a' * x' \wedge 2 + b' * x' + c'$
 by (simp add: algebra-simps a'-def b'-def c'-def power2-eq-square)
 finally have $\dots = 0$..
 moreover have $x' \notin \mathbb{Q}$
 using 1 by (auto simp: x'-def add-in-Rats-iff2)
 moreover have $a' \neq 0 \vee b' \neq 0 \vee c' \neq 0$
 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-int n) \longleftrightarrow quadratic-irrational x
 using 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-int n) \longleftrightarrow 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-nat n) \longleftrightarrow 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-nat n) \longleftrightarrow 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) \longleftrightarrow 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) \longleftrightarrow 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) \longleftrightarrow 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) \longleftrightarrow quadratic-irrational x
 using quadratic-irrational-plus-int-iff[of x -numeral n]

```

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 ≠ 0 by auto
  have 0 = (real-of-int a * x2 + real-of-int b * x + real-of-int c) / x2
    by (subst 1) simp
  also have ... = real-of-int c * (inverse x)2 + real-of-int b * inverse x +
    real-of-int a
    using ⟨x ≠ 0⟩ by (simp add: field-simps power2-eq-square)
  finally have ... = 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) ⟷ 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) ⟷ quadratic-irrational (cfrac-lim c)
proof (cases cfrac-length c = ∞)
  case False
  thus ?thesis
    by (auto simp: quadratic-irrational.simps)
next
  case [simp]: True
  show ?thesis
  proof (induction n)
    case (Suc n)
    from Suc.premis 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 ... ⟷ quadratic-irrational (cfrac-remainder c
n)
    by simp
    also have ... ⟷ quadratic-irrational (cfrac-lim c)
    by (rule Suc.IH)
    finally show ?case .
  qed auto
qed

```


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  $a\ b\ c :: \text{real}$ 
defines  $f \equiv (\lambda x. a * x^2 + b * x + c)$ 
defines  $\text{discr} \equiv (b^2 - 4 * a * c)$ 
shows  $\{x. f\ x = 0\} =$ 
    (if  $a = 0$  then
      (if  $b = 0$  then if  $c = 0$  then UNIV else  $\{\}$  else  $\{-c/b\}$ )
    else if  $\text{discr} \geq 0$  then  $\{(-b + \text{sqrt } \text{discr}) / (2 * a), (-b - \text{sqrt } \text{discr}) /$ 
     $(2 * a)\}$ 
    else  $\{\}$ ) (is ?th1)

proof (cases  $a = 0$ )
  case [simp]: True
  show ?th1
  proof (cases  $b = 0$ )
    case [simp]: True
    hence  $\{x. f\ x = 0\} =$  (if  $c = 0$  then UNIV else  $\{\}$ )
    by (auto simp: f-def)
    thus ?th1 by simp
  next
  case False
  hence  $\{x. f\ x = 0\} = \{-c / b\}$  by (auto simp: f-def field-simps)
  thus ?th1 using False by simp
  qed
next
  case [simp]: False
  show ?th1
  proof (cases  $\text{discr} \geq 0$ )
    case True
    {
      fix  $x :: \text{real}$ 
      have  $f\ x = a * (x - (-b + \text{sqrt } \text{discr}) / (2 * a)) * (x - (-b - \text{sqrt } \text{discr}) /$ 
       $(2 * a))$ 
      using True by (simp add: f-def field-simps discr-def power2-eq-square)
      also have  $\dots = 0 \longleftrightarrow x \in \{(-b + \text{sqrt } \text{discr}) / (2 * a), (-b - \text{sqrt } \text{discr}) /$ 
       $(2 * a)\}$ 
      by simp
      finally have  $f\ x = 0 \longleftrightarrow \dots$ 
    }
    hence  $\{x. f\ x = 0\} = \{(-b + \text{sqrt } \text{discr}) / (2 * a), (-b - \text{sqrt } \text{discr}) / (2 * a)\}$ 
    by blast
    thus ?th1 using True by simp
  next
  case False
  {

```

```

fix x :: real
assume x: f x = 0
have 0 ≤ (x + b / (2 * a)) ^ 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)) ^ 2 - b ^ 2 / (4 * a ^ 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 ≤ (b^2 / (4 * a^2) - c / a) * (4 * a^2)
  by (intro mult-nonneg-nonneg) auto
also have ... = b^2 - 4 * a * c by (simp add: field-simps power2-eq-square)
also have ... < 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 ≡ (b^2 - 4 * a * c)
shows finite {x. a * x ^ 2 + b * x + c = 0} ↔ a ≠ 0 ∨ b ≠ 0 ∨ c ≠ 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 ≡ (b^2 - 4 * a * c)
shows card {x. a * x ^ 2 + b * x + c = 0} =
  (if a = 0 then
    (if b = 0 then 0 else 1)
  else if discr ≥ 0 then if discr = 0 then 1 else 2 else 0) (is ?th1)
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} ≤ 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 ≡ (λx::real. a * x ^ 2 + b * x + c)
defines discr ≡ nat (b ^ 2 - 4 * a * c)
assumes a ≠ 0 f x = 0
shows x ∈ ℚ ↔ is-square discr
proof -
  define discr' where discr' ≡ 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  $\text{discr}' \geq 0$  unfolding discr'-def f-def of-nat-diff
  by (subst (asm) quadratic-equation-reals) (auto simp: discr-def split: if-splits)
  hence  $\ast: \text{sqrt}(\text{discr}') = \text{sqrt}(\text{real } \text{discr})$  unfolding of-int-0-le-iff discr-def
discr'-def
  by (simp add: algebra-simps nat-diff-distrib)
from  $\langle x \in \{x. f\ x = 0\} \rangle$  have  $x = (-b + \text{sqrt } \text{discr}) / (2 \ast a) \vee x = (-b - \text{sqrt}$ 
discr) / (2 * a)
  using  $\langle a \neq 0 \rangle \ast$  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

```

2.4 Periodic continued fractions and quadratic irrationals

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

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

theorem *periodic-cfrac-imp-quadratic-irrational:*

assumes [*simp*]: *cfrac-length* $c = \infty$

and *period*: $l > 0 \wedge k. k \geq N \implies \text{cfrac-nth } c\ (k + l) = \text{cfrac-nth } c\ k$

shows *quadratic-irrational* (*cfrac-lim* c)

proof –

define h' **and** k' **where** $h' = \text{conv-num-int } (\text{cfrac-drop } N\ c)$

and $k' = \text{conv-denom-int } (\text{cfrac-drop } N\ c)$

define x' **where** $x' = \text{cfrac-remainder } c\ N$

have *c-pos*: $\text{cfrac-nth } c\ n > 0$ **if** $n \geq N$ **for** n

proof –

from *assms*(1,2) **have** $\text{cfrac-nth } c\ (n + l) > 0$ **by** *auto*

with *assms*(3)[*OF that*] **show** *?thesis* **by** *simp*

qed

have $k'\text{-pos}$: $k'\ n > 0$ **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 $k'\text{-nonneg}$: $k'\ n \geq 0$ **if** $n \geq -2$ **for** n

using *that* **by** (*auto simp: k'-def conv-denom-int-def intro!: conv-denom-pos*)

have $\text{cfrac-nth } c\ (n + (N + l)) = \text{cfrac-nth } c\ (n + N)$ **for** n

using *period*(2)[*of* $n + N$] **by** (*simp add: add-ac*)

have $\text{cfrac-drop } (N + l)\ c = \text{cfrac-drop } N\ c$

by (*rule cfrac-eqI*) (*use period*(2)[*of* $n + N$ **for** n] **in** $\langle \text{auto simp: algebra-simps} \rangle$)

hence $x'\text{-altdef}$: $x' = \text{cfrac-remainder } c\ (N + l)$

by (*simp add: x'-def cfrac-remainder-def*)

have $x'\text{-pos}$: $x' > 0$ **unfolding** $x'\text{-def}$

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)) > 0
  using x'-pos <l > 0>
  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 +
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' + 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' ^ 2 + B * x' + C = 0
  by (simp add: algebra-simps power2-eq-square A-def B-def C-def)
moreover have x' ∉ ℚ unfolding x'-def
  by auto
moreover have A > 0 using <l > 0> by (auto simp: A-def intro!: k'-pos)
ultimately have quadratic-irrational x' using <x' ∉ ℚ>
  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 ⇒ cfrac **is**
 $\lambda xs. \text{if } xs = [] \text{ then } (0, LNil) \text{ else } (int (hd\ xs), llist\ of\ stream\ (cycle\ (map\ (\lambda n. n - 1)\ (tl\ xs\ @\ [hd\ xs])))) .$

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

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

lemma cfrac-length-pproduct-cfrac [simp]:
 $xs \neq [] \implies \text{cfrac-length } (pproduct\ cfrac\ xs) = \infty$
by transfer auto

lemma cfrac-nth-pproduct-cfrac:
assumes xs ≠ [] **and** 0 ∉ 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 (λn. n - 1) xs) @ [hd (map (λn. n - 1) xs)]) !! n') +
1 =
      int (stl (cycle (map (λn. n - 1) xs)) !! n') + 1
    by (subst cycle.sel(2) [symmetric]) (rule refl)
  also have ... = int (cycle (map (λn. n - 1) xs) !! n) + 1
    by (simp add: Suc del: cycle.sel)
  also have ... = int (xs ! (n mod length xs) - 1) + 1
    by (simp add: snth-cycle ⟨xs ≠ []⟩)
  also have xs ! (n mod length xs) ∈ set xs
    using ⟨xs ≠ []⟩ by (auto simp: set-conv-nth)
  with 1 have xs ! (n mod length xs) > 0
    by (intro Nat.gr0I) 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* ⇒ *int* × *int* × *int* **where**

```

pperiodic-cfrac-info xs =
  (let l = length xs;
    h = conv-num-fun (λn. xs ! n);
    k = conv-denom-fun (λn. 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 ∀ k ∈ {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)
N
    using Suc(2,3) by (intro Suc) auto
  moreover have f n = f' n
    using bspec[OF Suc.premis, 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 $\forall k \leq n. c\ k = \text{cfrac-nth } c'\ k$

shows $\text{conv-num-fun-eq}' : \text{conv-num-fun } c\ n = \text{conv-num } c'\ n$

and $\text{conv-denom-fun-eq}' : \text{conv-denom-fun } c\ n = \text{conv-denom } c'\ n$

proof –

have $\text{conv-num } c'\ n = \text{conv-gen } (\text{cfrac-nth } c')\ (0, 1, 0)\ n$

unfolding *conv-num-code* ..

also have $\dots = \text{conv-gen } c\ (0, 1, 0)\ n$

unfolding *conv-num-fun-def* **using** *assms* **by** (*intro conv-gen-cong*) *auto*

finally show $\text{conv-num-fun } c\ n = \text{conv-num } c'\ n$

by (*simp add: conv-num-fun-def*)

next

have $\text{conv-denom } c'\ n = \text{conv-gen } (\text{cfrac-nth } c')\ (1, 0, 0)\ n$

unfolding *conv-denom-code* ..

also have $\dots = \text{conv-gen } c\ (1, 0, 0)\ n$

unfolding *conv-denom-fun-def* **using** *assms* **by** (*intro conv-gen-cong*) *auto*

finally show $\text{conv-denom-fun } c\ n = \text{conv-denom } c'\ n$

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

qed

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

by (*metis gcd.commute gcd-neg2 minus-diff-eq*)

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

by (*metis gcd-neg2 minus-diff-eq*)

lemma *periodic-cfrac-info-aux*:

fixes $D\ E\ F :: \text{int}$

assumes *pproperiodic-cfrac-info* $xs = (D, E, F)$

assumes $xs \neq []$ $0 \notin \text{set } xs$

shows $\text{cfrac-lim } (\text{pproperiodic-cfrac } xs) = (\text{sqrt } D + E) / F$

and $D > 0$ **and** $F > 0$

proof –

define c **where** $c = \text{pproperiodic-cfrac } xs$

have [*simp*]: $\text{cfrac-length } c = \infty$

using *assms* **by** (*simp add: c-def*)

define h **and** k **where** $h = \text{conv-num-int } c$ **and** $k = \text{conv-denom-int } c$

define x **where** $x = \text{cfrac-lim } c$

define l **where** $l = \text{length } xs$

define A **where** $A = (k\ (\text{int } l - 1))$

define B **where** $B = k\ (\text{int } l - 2) - h\ (\text{int } l - 1)$

define C **where** $C = -(h\ (\text{int } l - 2))$

define discr **where** $\text{discr} = B^2 - 4 * A * C$

have $l > 0$

using *assms* **by** (*simp add: l-def*)

```

have c-pos: cfrac-nth c n > 0 for n
  using assms by (auto simp: c-def cfrac-nth-pproduct-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 n
  using 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 ≥ 0 for n
  unfolding k-def by (auto simp: conv-denom-int-def)

have pos: (k (int l - 1) * x + k (int l - 2)) > 0
  using x-pos ⟨l > 0⟩
  by (intro add-pos-nonneg mult-pos-pos) (auto intro!: k-pos k-nonneg)
have cfrac-drop l c = c
  using assms by (intro cfrac-eqI) (auto simp: c-def cfrac-nth-pproduct-cfrac
l-def)

have x = conv' c l (cfrac-remainder c l)
  unfolding x-def by (rule conv'-cfrac-remainder[symmetric]) auto
also have ... = conv' c l x
  unfolding cfrac-remainder-def ⟨cfrac-drop l c = c⟩ 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 ^ 2 + B * x + C = 0
  by (simp add: algebra-simps power2-eq-square A-def B-def C-def)

have A > 0 using ⟨l > 0⟩ by (auto simp: A-def intro!: k-pos)
have discr-altdef: discr = (k (int l - 2) - h (int l - 1)) ^ 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 * 1
  using ⟨A > 0⟩ by simp
also have 0 + 4 * A * 1 ≤ discr
  unfolding discr-altdef A-def using h-pos[of int l - 2] ⟨l > 0⟩
  by (intro add-mono mult-mono order.refl k-nonneg mult-nonneg-nonneg) auto
finally have discr > 0 .

have x ∈ {x. A * x ^ 2 + B * x + C = 0}
  using quadratic by simp
hence x-cases: x = (-B - sqrt discr) / (2 * A) ∨ x = (-B + sqrt discr) / (2

```

```

* A)
  unfolding quadratic-equation-reals of-int-diff using ⟨A > 0⟩
  by (auto split: if-splits simp: discr-def)

  have B ^ 2 < discr
  unfolding discr-def by (auto intro!: mult-pos-pos k-pos h-pos ⟨l > 0⟩ simp:
A-def C-def)
  hence |B| < sqrt discr
  using ⟨discr > 0⟩ by (simp add: real-less-rsqrt)

  have x = (if x ≥ 0 then (sqrt discr - B) / (2 * A) else -(sqrt discr + B) / (2
* A))
  using x-cases
  proof
    assume x: x = (-B - sqrt discr) / (2 * A)
    have (-B - sqrt discr) / (2 * A) < 0
    using ⟨|B| < sqrt discr⟩ ⟨A > 0⟩ by (intro divide-neg-pos) auto
    also note x[symmetric]
    finally show ?thesis using x by simp
  next
    assume x: x = (-B + sqrt discr) / (2 * A)
    have (-B + sqrt discr) / (2 * A) > 0
    using ⟨|B| < sqrt discr⟩ ⟨A > 0⟩ by (intro divide-pos-pos) auto
    also note x[symmetric]
    finally show ?thesis using x by simp
  qed
  also have x ≥ 0 ⟷ floor x ≥ 0
  by auto
  also have floor x = floor (cfrac-lim c)
  by (simp add: x-def)
  also have ... = cfrac-nth c 0
  by (subst cfrac-nth-0-conv-floor) auto
  also have ... = int (hd xs)
  using assms unfolding c-def by (subst cfrac-nth-ppperiodic-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 (λn. int (xs ! n))
define k' where k' = conv-denom-fun (λn. int (xs ! n))
have num-eq: h' i = h i
  if 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-ppperiodic-cfrac)
have denom-eq: k' i = k i
  if 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-ppperiodic-cfrac)

have 1: h (int l - 1) = h' (l - 1)

```



```

  by (subst num-eq) (use <l > 0> in <auto simp: of-nat-diff>)
have 2: k (int l - 1) = k' (l - 1)
  by (subst denom-eq) (use <l > 0> in <auto simp: of-nat-diff>)
have 3: h (int l - 2) = (if l = 1 then 1 else h' (l - 2))
  using <l > 0> 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 <l > 0> 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 (B2 - 4 * A * C, B, 2 * A))
  unfolding pperiodic-cfrac-info-def Let-def using 1 2 3 4 <l > 0>
  by (auto simp: num-eq denom-eq h'-def k'-def l-def of-nat-diff)
also have ... = (B2 - 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 F
  using per-eq assms by (simp add: x-eq)
show D > 0 F > 0
  using assms per-eq <discr > 0> <A > 0> by auto
qed

```

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

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

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

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

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

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

theorem *quadratic-irrational-imp-periodic-cfrac:*

assumes *quadratic-irrational* (cfrac-lim e)

obtains *N l* **where** *l > 0* **and** $\bigwedge n m. n \geq N \implies \text{cfrac-nth } e (n + m * l) = \text{cfrac-nth } e n$

and *cfrac-remainder* e (N + l) = *cfrac-remainder* e N

and *cfrac-length* e = ∞

proof –

have [simp]: *cfrac-length* e = ∞

using *assms* **by** (auto simp: *quadratic-irrational.simps*)

```

note [intro] = assms(1)
define x where x = cfrac-lim e
from assms obtain a b c :: int where
  nontrivial:  $a \neq 0 \vee b \neq 0 \vee c \neq 0$  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 = ?f
define h and k where h = conv-num e and k = conv-denom e
define 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-nonneg:  $k\ n \geq 0$  for n
  by (auto simp: k-def)
have k-ge-1:  $k\ n \geq 1$  for n
  using k-leI[of 0 n] by (simp add: k-def)

define R where R = conv e
define A where  $A = (\lambda n. a * h\ (n - 1) ^ 2 + b * h\ (n - 1) * k\ (n - 1) + c$ 
 $* k\ (n - 1) ^ 2)$ 
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  $C = (\lambda n. a * h\ (n - 2) ^ 2 + b * h\ (n - 2) * k\ (n - 2) + c$ 
 $* k\ (n - 2) ^ 2)$ 

define A' where  $A' = \text{nat } \lfloor 2 * |a| * |x| + |a| + |b| \rfloor$ 
define B' where  $B' = \text{nat } \lfloor (3 / 2) * (2 * |a| * |x| + |b|) + 9 / 4 * |a| \rfloor$ 

have [simp]:  $X\ n \notin \mathbb{Q}$  for n unfolding X-def
  by simp
from this[of 0] have [simp]:  $x \notin \mathbb{Q}$ 
  unfolding X-def by (simp add: x-def)

have  $a \neq 0$ 
proof
  assume  $a = 0$ 
  with root and nontrivial have  $x = 0 \vee x = -c / b$ 
  by (auto simp: divide-simps add-eq-0-iff)
  hence  $x \in \mathbb{Q}$  by (auto simp del: x-def)
  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:  $A\ n * X\ n ^ 2 + B\ n * X\ n + C\ n = 0$  if  $n: n \geq 2$  for n
proof –
  define n' where  $n' = n - 2$ 
  have n':  $n = \text{Suc } (\text{Suc } n')$  using  $\langle n \geq 2 \rangle$  unfolding n'-def by simp

```

```

have *: of-int (k (n - Suc 0)) * X n + of-int (k (n - 2)) ≠ 0
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 ... ∈ ℚ by auto
  finally show False by simp
qed

let ?denom = (k (n - 1) * X n + k (n - 2))
have 0 = 0 * ?denom ^ 2 by simp
also have 0 * ?denom ^ 2 = (a * x ^ 2 + b * x + c) * ?denom ^ 2 using root
by simp
  also have ... = a * (x * ?denom) ^ 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] ⟨n ≥ 2⟩
  by (simp add: numeral-2-eq-2 Suc-diff-Suc x-def k-def h-def X-def)
  also have a * ... ^ 2 + b * ?denom * ... + c * ?denom * ?denom = A n *
X n ^ 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 ^ 2 + B n * X n + C n = 0 ..

have f-abs-bound: |f (R n)| ≤ (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) - ?f x| by (simp add: root f-def)
  also have ?f (R n) - ?f x = (R n - x) * (2 * a * x + b) + (R n - x) ^ 2
* a
  by (simp add: power2-eq-square algebra-simps)
  also have |...| ≤ |(R n - x) * (2 * a * x + b)| + |(R n - x) ^ 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 *
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| ≤ 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))) ^ 2 ≤
... * (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)| ≤ ...
  by (simp add: mult-right-mono add-mono divide-left-mono)
qed

```

```

have h-eq-conv-k: h i = R i * k i for i
  using 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) ^ 2 * f (R (n - 1)) for n
  by (simp add: algebra-simps A-def n' k-def power2-eq-square h-eq-conv-k f-def)
have A-bound: |A i| ≤ A' if i > 0 for i
proof -
  have k i > 0
  by simp
  hence k i ≥ 1
  by linarith
  have A i = k (i - 1) ^ 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) ^ 2 * |f (R (i - 1))|
  by (simp add: abs-mult f-def)
  also have ... ≤ k (i - 1) ^ 2 * ((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) / k i * (2 * |a| * |x| + |b|) + |a| / k i ^ 2 using
⟨i > 0⟩
  by (simp add: power2-eq-square field-simps)
  also have ... ≤ 1 * (2 * |a| * |x| + |b|) + |a| / 1 using ⟨i > 0⟩ ⟨k i ≥ 1⟩
  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| ≤ 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|
  by (simp add: abs-mult k-nonneg)
  also have ... ≤ 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 - ≤ - * (?S1 +
?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  $(k (n - 1) * k (n - 2)) * (?S1 + ?S2 + |a| * \dots ^ 2) =$ 
     $(k (n - 2) / k n + 1) * (2 * |a| * |x| + |b|) +$ 
     $|a| * (k (n - 2) / (k (n - 1) * k n ^ 2) + 2 / (k (n - 1) * k (n -$ 
2)))
  (is - = ?S) using n by (simp add: field-simps power2-eq-square numeral-2-eq-2
Suc-diff-Suc)
  also {
    have A:  $2 * \text{real-of-int } (k (n - 2)) \leq \text{of-int } (k n)$ 
      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  $\dots \leq k n$  by (intro k-leI n)
    finally have  $k n \geq 2$  by (simp add: numeral-3-eq-3)
    hence B:  $\text{of-int } (k (n - 2)) * 2 ^ 2 \leq (\text{of-int } (k (n - 1)) * (\text{of-int } (k n))^2 ::$ 
real)
      by (intro mult-mono power-mono) (auto intro: k-leI k-nonneg)
    have C:  $1 * 1 \leq \text{real-of-int } (k (n - 1)) * \text{of-int } (k (n - 2))$  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  $\dots = (3 / 2) * (2 * |a| * |x| + |b|) + 9 / 4 * |a|$  by simp
  finally have B-bound:  $|B n| \leq B'$  unfolding B'-def by linarith
  from A-bound[of n] B-bound C-bound n
  show  $(A n, B n, C n) \in \{-A'..A'\} \times \{-B'..B'\} \times \{-A'..A'\}$  by auto
qed

have A-nz:  $A n \neq 0$  if  $n \geq 1$  for n
  using that
proof (induction n rule: dec-induct)
  case base
  show ?case
  proof
    assume A 1 = 0
    hence  $\text{real-of-int } (A 1) = 0$  by simp
    also have  $\text{real-of-int } (A 1) =$ 
       $\text{real-of-int } a * \text{of-int } (\text{cfraction } e 0) ^ 2 +$ 
       $\text{real-of-int } b * \text{cfraction } e 0 + \text{real-of-int } c$ 
      by (simp add: A-def h-def k-def)
    finally have root':  $\dots = 0$  .

  have  $\text{cfraction } e 0 \in \mathbb{Q}$  by auto
  also from root' and  $\langle a \neq 0 \rangle$  have  $?this \longleftrightarrow \text{is-square } (\text{nat } (b^2 - 4 * a * c))$ 
    by (intro quadratic-equation-solution-rat-iff) auto

```

```

    also from root and  $\langle a \neq 0 \rangle$  have ...  $\longleftrightarrow x \in \mathbb{Q}$ 
    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]:  $A (Suc\ m) = 0$ 
    have  $X (Suc\ m) > 0$  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 ...  $\in \mathbb{Q}$  by auto
    finally show False by simp
  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  $\exists k1 \in \{2..\}. infinite \{n \in \{2..\}. (A\ n, B\ n, C\ n) = (A\ k1, B\ k1, C\ k1)\}$ 
  by (intro pigeonhole-infinite)
  then obtain k0 where k0:  $k0 \geq 2$  infinite  $\{n \in \{2..\}. (A\ n, B\ n, C\ n) = (A\ k0, B\ k0, C\ k0)\}$ 
  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 g-ge-2:  $g\ k \geq 2$  for k by auto
  from g have [simp]:  $A\ (g\ k) = A\ k0\ B\ (g\ k) = B\ k0\ C\ (g\ k) = C\ k0$  for k
  by auto

  from  $g(1)$  have [simp]:  $g\ k1 = g\ k2 \longleftrightarrow k1 = k2$  for  $k1\ k2$  by (auto simp:
inj-def)
  define z where  $z = (A\ k0, B\ k0, C\ k0)$ 
  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  $\{x :: real. A\ k0 * x^2 + B\ k0 * x + C\ k0 = 0\}$  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 $\text{card } ((X \circ g) \text{ ‘ } \{0, 1, 2\}) \leq \text{card } \dots$
 by (intro card-mono fin) auto
 also have $\dots \leq 2$
 by (rule card-quadratic-equation-solutions-reals-le-2)
 also have $\dots < \text{card } \{0, 1, 2 :: \text{nat}\}$ by simp
 finally have $\neg \text{inj-on } (X \circ g) \{0, 1, 2\}$
 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 $l = \text{nat } |\text{int } (g\ m1) - g\ m2|$
 with $m12\ g'$ have $l: l > 0\ X\ (n + l) = X\ n$
 by (auto simp: min-def nat-diff-distrib split: if-splits)

from l have $\text{cfrac-lim } (\text{cfrac-drop } (n + l)\ e) = \text{cfrac-lim } (\text{cfrac-drop } n\ e)$
 by (simp add: X-def cfrac-remainder-def)
 hence $\text{cfrac-drop } (n + l)\ e = \text{cfrac-drop } n\ e$
 by (simp add: cfrac-lim-eq-iff)
 hence $\text{cfrac-nth } (\text{cfrac-drop } (n + l)\ e) = \text{cfrac-nth } (\text{cfrac-drop } n\ e)$
 by (simp only:)
 hence period: $\text{cfrac-nth } e\ (n + l + k) = \text{cfrac-nth } e\ (n + k)$ for k
 by (simp add: fun-eq-iff add-ac)
 have period: $\text{cfrac-nth } e\ (k + l) = \text{cfrac-nth } e\ k$ if $k \geq n$ for k
 using period[of $k - n$] that by (simp add: add-ac)
 have period: $\text{cfrac-nth } e\ (k + m * l) = \text{cfrac-nth } e\ k$ if $k \geq n$ for $k\ m$
 using that
 proof (induction m)
 case (Suc m)
 have $\text{cfrac-nth } e\ (k + \text{Suc } m * l) = \text{cfrac-nth } e\ (k + m * l + l)$
 by (simp add: algebra-simps)
 also have $\dots = \text{cfrac-nth } e\ (k + m * l)$
 using Suc.prem by (intro period) auto
 also have $\dots = \text{cfrac-nth } e\ k$
 using Suc.prem 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 $x \notin \mathbb{Q}\ x \geq 0$

shows quadratic-irrational $x \longleftrightarrow$

$$(\exists N\ l.\ l > 0 \wedge (\forall n \geq N.\ \text{cfrac-nth } (\text{cfrac-of-real } x)\ (n + l) = \text{cfrac-nth } (\text{cfrac-of-real } x)\ n))$$

proof safe

assume *: quadratic-irrational x

with asms have **: quadratic-irrational $(\text{cfrac-lim } (\text{cfrac-of-real } x))$ by auto

obtain $N\ l$ where $Nl: l > 0$

```

 $\bigwedge n m. N \leq n \implies \text{cfrac-nth } (\text{cfrac-of-real } x) (n + m * l) = \text{cfrac-nth } (\text{cfrac-of-real } x) n$ 
 $\text{cfrac-remainder } (\text{cfrac-of-real } x) (N + l) = \text{cfrac-remainder } (\text{cfrac-of-real } x) N$ 
 $\text{cfrac-length } (\text{cfrac-of-real } x) = \infty$ 
using quadratic-irrational-imp-periodic-cfrac [OF **] by metis
show  $\exists N l. l > 0 \wedge (\forall n \geq N. \text{cfrac-nth } (\text{cfrac-of-real } x) (n + l) = \text{cfrac-nth } (\text{cfrac-of-real } x) n)$ 
by (rule exI[of - N], rule exI[of - l]) (insert Nl(1) Nl(2)[of - 1], auto)
next
fix N l assume  $l > 0 \forall n \geq N. \text{cfrac-nth } (\text{cfrac-of-real } x) (n + l) = \text{cfrac-nth } (\text{cfrac-of-real } x) n$ 
hence 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 quadratic irrational.

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*: $l > 0 \bigwedge m n. n \geq N \implies \text{cfrac-nth } e (n + m * l) = \text{cfrac-nth } e n$

by *metis*

have $\text{cfrac-nth } e k \in \text{cfrac-nth } e \text{ ‘ } \{..<N+l\}$ **for** *k*

proof (*cases* $k < N + l$)

case *False*

define *n m* **where** $n = N + (k - N) \bmod l$ **and** $m = (k - N) \div l$

have $\text{cfrac-nth } e n \in \text{cfrac-nth } e \text{ ‘ } \{..<N+l\}$

using $\langle l > 0 \rangle$ **by** (*intro imageI*) (*auto simp: n-def*)

also have $\text{cfrac-nth } e n = \text{cfrac-nth } e (n + m * l)$

by (*subst period*) (*auto simp: n-def*)

also have $n + m * l = k$

using *False* **by** (*simp add: n-def m-def*)

finally show *?thesis* .

qed *auto*

hence $\text{range } (\text{cfrac-nth } e) \subseteq \text{cfrac-nth } e \text{ ‘ } \{..<N+l\}$

by *blast*

thus *?thesis* **by** (*rule finite-subset*) *auto*

qed

end

3 The continued fraction expansion of *e*

theory *E-CFrac*

imports

HOL—Analysis.Analysis
Continued-Fractions
Quadratic-Irrationals
begin

lemma *fact-real-at-top*: *filterlim* (*fact* :: *nat* \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* $\geq C * m$) *F*
 by (*auto simp: filterlim-at-top*)
 thus *eventually* ($\lambda x.$ *f* *x* *div* *m* $\geq C$) *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 $[2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, \dots]$:

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

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

lemma *cfrac-length-e* [*simp*]: *cfrac-length* *e-cfrac* = ∞
 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* $^{Suc\ n}$ * (*x* - 1) 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* ≤ 1 *then* 1 *else* *conv-num* *e-cfrac* (*n* - 2))

and $q \equiv (\lambda n. \text{ if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } 0 \text{ else conv-denom e-cfrac } (n - 2))$
 and $a \equiv (\lambda n. \text{ if } n \bmod 3 = 2 \text{ then } 2 * (\text{Suc } n \text{ div } 3) \text{ else } 1)$
 begin

lemma

assumes $n \geq 2$
 shows $p\text{-rec}: p\ n = a\ (n - 2) * p\ (n - 1) + p\ (n - 2)$ (is ?th1)
 and $q\text{-rec}: q\ n = a\ (n - 2) * q\ (n - 1) + q\ (n - 2)$ (is ?th2)
 proof –
 have $n\text{-minus-3}: n - 3 = n - \text{Suc } (\text{Suc } (\text{Suc } 0))$
 by (simp add: numeral-3-eq-3)
 consider $n = 2 \mid n = 3 \mid n \geq 4$
 using assms by force
 hence ?th1 \wedge ?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 $n \geq 1$
 shows $p\text{-rec0}: p\ (3 * n) = p\ (3 * n - 1) + p\ (3 * n - 2)$
 and $q\text{-rec0}: q\ (3 * n) = q\ (3 * n - 1) + q\ (3 * n - 2)$
 proof –
 define n' where $n' = n - 1$
 from assms have $(3 * n' + 1) \bmod 3 \neq 2$
 by presburger
 also have $(3 * n' + 1) = 3 * n - 2$
 using 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 $p\text{-rec1}: p\ (3 * n + 1) = 2 * \text{int } n * p\ (3 * n) + p\ (3 * n - 1)$
 and $q\text{-rec1}: q\ (3 * n + 1) = 2 * \text{int } n * q\ (3 * n) + q\ (3 * n - 1)$
 proof –
 define n' where $n' = n - 1$
 from assms have $(3 * n' + 2) \bmod 3 = 2$
 by presburger
 also have $(3 * n' + 2) = 3 * n - 1$
 using assms by (simp add: n'-def)
 finally show $p\ (3 * n + 1) = 2 * \text{int } n * p\ (3 * n) + p\ (3 * n - 1)$
 $q\ (3 * n + 1) = 2 * \text{int } n * q\ (3 * n) + q\ (3 * n - 1)$
 using assms by (subst p-rec q-rec; simp add: a-def)+
 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 $C \ 0 = 2 - \exp 1$ **by** (*simp add: C-def has-integral-iff*)
qed

lemma *A-bound*: $\text{norm } (A \ n) \leq \exp 1 / \text{fact } n$

proof –

have $\text{norm } (\exp t * t ^ n * (t - 1) ^ n / \text{fact } n) \leq \exp 1 * 1 ^ n * 1 ^ n / \text{fact } n$
if $t \in \{0..1\}$ **for** $t :: \text{real}$ **using** *that unfolding norm-mult norm-divide norm-power norm-fact*
by (*intro mult-mono divide-right-mono power-mono*) *auto*
hence $\text{norm } (A \ n) \leq \exp 1 / \text{fact } n * (1 - 0)$
unfolding *A-def* **by** (*intro integral-bound*) (*auto intro!: continuous-intros*)
thus *?thesis* **by** *simp*
qed

lemma *B-bound*: $\text{norm } (B \ n) \leq \exp 1 / \text{fact } n$

proof –

have $\text{norm } (\exp t * t ^ \text{Suc } n * (t - 1) ^ n / \text{fact } n) \leq \exp 1 * 1 ^ \text{Suc } n * 1 ^ n / \text{fact } n$
if $t \in \{0..1\}$ **for** $t :: \text{real}$ **using** *that unfolding norm-mult norm-divide norm-power norm-fact*
by (*intro mult-mono divide-right-mono power-mono*) *auto*
hence $\text{norm } (B \ n) \leq \exp 1 / \text{fact } n * (1 - 0)$
unfolding *B-def* **by** (*intro integral-bound*) (*auto intro!: continuous-intros*)
thus *?thesis* **by** *simp*

qed

lemma *C-bound*: $\text{norm } (C\ n) \leq \exp\ 1 / \text{fact } n$

proof –

have $\text{norm } (\exp\ t * t^{\wedge} n * (t - 1)^{\wedge} \text{Suc } n / \text{fact } n) \leq \exp\ 1 * 1^{\wedge} n * 1^{\wedge} \text{Suc } n / \text{fact } n$

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

by (*intro mult-mono divide-right-mono power-mono*) *auto*

hence $\text{norm } (C\ n) \leq \exp\ 1 / \text{fact } n * (1 - 0)$

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

thus *?thesis* **by** *simp*

qed

lemma *A-Suc*: $A\ (\text{Suc } n) = -B\ n - C\ n$

proof –

let $?g = \lambda x. x^{\wedge} \text{Suc } n * (x - 1)^{\wedge} \text{Suc } n * \exp\ x / \text{fact } (\text{Suc } n)$

have *pos*: $\text{fact } n + \text{real } n * \text{fact } n > 0$ **by** (*intro add-pos-nonneg*) *auto*

have $A\ (\text{Suc } n) + B\ n + C\ n =$

$\text{integral } \{0..1\} (\lambda x. \exp\ x * x^{\wedge} \text{Suc } n * (x - 1)^{\wedge} \text{Suc } n / \text{fact } (\text{Suc } n) +$
 $\exp\ x * x^{\wedge} \text{Suc } n * (x - 1)^{\wedge} n / \text{fact } n + \exp\ x * x^{\wedge} n * (x - 1)^{\wedge}$

$\text{Suc } n / \text{fact } n)$

unfolding *A-def B-def C-def*

apply (*subst integral-add [symmetric]*)

subgoal

by (*auto intro!: integrable-continuous-real continuous-intros*)

subgoal

by (*auto intro!: integrable-continuous-real continuous-intros*)

apply (*subst integral-add [symmetric]*)

apply (*auto intro!: integrable-continuous-real continuous-intros*)

done

also have $\dots = \text{integral } \{0..1\} (\lambda x. \exp\ x / \text{fact } (\text{Suc } n) *$

$(x^{\wedge} \text{Suc } n * (x - 1)^{\wedge} \text{Suc } n + \text{Suc } n * x^{\wedge} \text{Suc } n * (x - 1)^{\wedge} n +$
 $\text{Suc } n * x^{\wedge} n * (x - 1)^{\wedge} \text{Suc } n))$

(*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 $\text{integral } \{0..1\} ?f = 0$

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 = $\lambda x. x \wedge Suc\ n * (x - 1) \wedge (n+2) * exp\ x / fact\ (Suc\ n)$
have pos: $fact\ n + real\ n * fact\ n > 0$ **by** (intro add-pos-nonneg) auto
have $B (Suc\ n) + 2 * Suc\ n * A (Suc\ n) - C\ n =$
 $integral\ \{0..1\}\ (\lambda x. exp\ x * x \wedge (n+2) * (x - 1) \wedge (n+1) / fact\ (Suc\ n) + 2$
 $* Suc\ n * exp\ x * x \wedge Suc\ n * (x - 1) \wedge Suc\ n / fact\ (Suc\ n) - exp\ x * x \wedge n * (x$
 $- 1) \wedge Suc\ n / fact\ n)$
unfolding A-def B-def C-def integral-mult-right [symmetric]
apply (subst integral-add [symmetric])
subgoal
by (auto intro!: integrable-continuous-real continuous-intros)
subgoal
by (auto intro!: integrable-continuous-real continuous-intros)
apply (subst integral-diff [symmetric])
apply (auto intro!: integrable-continuous-real continuous-intros simp: mult-ac)
done
also have $\dots = integral\ \{0..1\}\ (\lambda x. exp\ x / fact\ (Suc\ n) * (x \wedge (n+2) * (x - 1) \wedge (n+1) + 2 * Suc\ n * x \wedge Suc\ n * (x - 1) \wedge$
 $Suc\ n - Suc\ n * x \wedge n * (x - 1) \wedge Suc\ n))$
(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)

qed

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) ∧
B n = p (Suc (3 * n)) - q (Suc (3 * n)) * exp 1 ∧
C n = p (Suc (Suc (3 * n))) - q (Suc (Suc (3 * n))) * exp 1
proof (induction n)
  case 0
  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 conjunct2[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 ≠ 1
  using that by (auto simp: q-def)

lemma conv-diff-exp-bound: norm (exp 1 - p n / q n) ≤ exp 1 / fact (n div 3)
proof (cases n = 1)
  case False
  define n' where n' = n div 3
  consider n mod 3 = 0 | n mod 3 = 1 | 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
  next
    assume *: n mod 3 = 1
    hence Suc (3 * n') = n unfolding n'-def by presburger
    with * ABC[of n'] show ?thesis by auto
  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) ≤ 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 <n
  ≠ 1>]
  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 n
  using q-pos[OF <n ≠ 1>] 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 (λn. exp 1 - p n / q n) ⟶ 0
  proof (rule Lim-null-comparison)
    show eventually (λn. norm (exp 1 - p n / q n) ≤ exp 1 / fact (n div 3)) at-top
    using conv-diff-exp-bound by (intro always-eventually) auto
    show (λn. exp 1 / fact (n div 3) :: real) ⟶ 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 (λn. exp 1 - p n / q n = exp 1 - conv e-cfrac (n -
  2)) at-top
  using eventually-ge-at-top[of 2]
  proof eventually-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 (λn. exp 1 - conv e-cfrac (n - 2)) ⟶ 0
  using Lim-transform-eventually by fast
  hence (λn. exp 1 - (exp 1 - conv e-cfrac (Suc (Suc n) - 2))) ⟶ exp 1 - 0
  by (subst filterlim-sequentially-Suc)+ (intro tendsto-diff tendsto-const)
  hence conv e-cfrac ⟶ exp 1 by simp
  moreover have conv e-cfrac ⟶ cfrac-lim e-cfrac
  by (intro LIMSEQ-cfrac-lim wf) auto
  ultimately have exp 1 = cfrac-lim e-cfrac
  by (rule LIMSEQ-unique)
  thus ?thesis ..
qed

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*: $(\text{exp } 1 :: \text{real}) \notin \mathbb{Q}$
 by (metis cfrac-length-e e-cfrac cfrac-infinite-iff)

corollary *exp1-not-quadratic-irrational*: $\neg \text{quadratic-irrational } (\text{exp } 1 :: \text{real})$

proof –

have range $(\lambda n. 2 * (\text{int } n + 1)) \subseteq \text{range } (\text{cfrac-nth } e\text{-cfrac})$

proof safe

fix $n :: \text{nat}$

have $\text{cfrac-nth } e\text{-cfrac } (3*n+2) \in \text{range } (\text{cfrac-nth } e\text{-cfrac})$

by blast

also have $(3 * n + 2) \bmod 3 = 2$

by presburger

hence $\text{cfrac-nth } e\text{-cfrac } (3*n+2) = 2 * (\text{int } n + 1)$

by (simp add: cfrac-nth-e)

finally show $2 * (\text{int } n + 1) \in \text{range } (\text{cfrac-nth } e\text{-cfrac})$.

qed

moreover have infinite (range $(\lambda n. 2 * (\text{int } n + 1))$)

by (subst finite-image-iff) (auto intro!: injI)

ultimately have infinite (range $(\text{cfrac-nth } e\text{-cfrac})$)

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

4 Continued fraction expansions for square roots of naturals

theory *Sqrt-Nat-Cfrac*

imports

Quadratic-Irrationals

HOL-Library.While-Combinator

HOL-Library.IArray

begin

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

lemma *butlast-nth [simp]*: $n < \text{length } xs - 1 \implies \text{butlast } xs ! n = xs ! n$

by (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 .

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

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

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

lemma *sqrt-nat-period-length-square* [*simp*]: is-square $D \implies \text{sqrt-nat-period-length } D = 0$
by (auto simp: *sqrt-nat-period-length-def*)

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

context
fixes $D \ D' :: \text{nat}$
defines $D' \equiv \text{nat } \lfloor \text{sqrt } D \rfloor$
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 \times nat \Rightarrow bool **where**
red-assoc = ($\lambda(p, q).$
 $q > 0 \wedge q \text{ dvd } (D - p^2) \wedge (\text{sqrt } D + p) / q > 1 \wedge (-\text{sqrt } D + p) / q \in \{-1 < .. < 0\}$)

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

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

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

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**

$\text{sqrt-remainder-step} = (\lambda(p, q). \text{let } X = (p + D') \text{ div } q; p' = X * q - p \text{ in } (p', (D - p'^2) \text{ div } q))$

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

definition $\text{sqrt-remainder-surd} :: \text{nat} \Rightarrow \text{nat} \times \text{nat}$
where $\text{sqrt-remainder-surd} = (\lambda n. (\text{sqrt-remainder-step} \rightsquigarrow n) (D', D - D'^2))$

context

fixes $\text{sqrt-cfrac-nth} :: \text{nat} \Rightarrow \text{nat}$ **and** l

assumes $\text{nonsquare}: \neg \text{is-square } D$

defines $\text{sqrt-cfrac-nth} \equiv (\lambda n. \text{case } \text{sqrt-remainder-surd } n \text{ of } (p, q) \Rightarrow (D' + p) \text{ div } q)$

defines $l \equiv \text{sqrt-nat-period-length } D$

begin

lemma $D'\text{-pos}: D' > 0$

using nonsquare **by** $(\text{auto simp: } D'\text{-def of-nat-ge-1-iff intro: Nat.gr0I})$

lemma $D'\text{-sqr-less-}D: D'^2 < D$

proof –

have $D' \leq \text{sqrt } D$ **by** $(\text{auto simp: } D'\text{-def})$

hence $\text{real } D' \wedge 2 \leq \text{sqrt } D \wedge 2$ **by** $(\text{intro power-mono})$ **auto**

also have $\dots = D$ **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

lemma $\text{red-assoc-imp-irrat}$:

assumes $\text{red-assoc } pq$

shows $\text{surd-to-real } pq \notin \mathbb{Q}$

proof

assume $\text{rat: surd-to-real } pq \in \mathbb{Q}$

with assms rat **show** False **using** $\text{irrat-sqrt-nonsquare}[OF \text{ nonsquare}]$

by $(\text{auto simp: field-simps red-assoc-def surd-to-real-def divide-in-Rats-iff2 add-in-Rats-iff1})$

qed

lemma $\text{surd-to-real-cn timer-irrat}$:

assumes $\text{red-assoc } pq$

shows $\text{surd-to-real-cn timer } pq \notin \mathbb{Q}$

proof

assume $\text{rat: surd-to-real-cn timer } pq \in \mathbb{Q}$

with assms rat **show** False **using** $\text{irrat-sqrt-nonsquare}[OF \text{ nonsquare}]$

by $(\text{auto simp: field-simps red-assoc-def surd-to-real-cn timer-def divide-in-Rats-iff2 diff-in-Rats-iff1})$

qed

lemma *surd-to-real-nonneg* [intro]: *surd-to-real* *pq* ≥ 0
by (*auto simp: surd-to-real-def case-prod-unfold divide-simps intro!: divide-nonneg-nonneg*)

lemma *surd-to-real-pos* [intro]: *red-assoc* *pq* \implies *surd-to-real* *pq* > 0
by (*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-cn timer-nz* [simp]: *red-assoc* *pq* \implies *surd-to-real-cn timer* *pq* $\neq 0$
using *surd-to-real-cn timer-irrat*[of *pq*] **by** *auto*

lemma *red-assoc-step*:

assumes *red-assoc* *pq*

defines $X \equiv (D' + \text{fst } pq) \text{ div } \text{snd } pq$

defines $pq' \equiv \text{sqrt-remainder-step } pq$

shows *red-assoc* pq'

$\text{surd-to-real } pq' = 1 / \text{frac } (\text{surd-to-real } pq)$

$\text{surd-to-real-cn timer } pq' = 1 / (\text{surd-to-real-cn timer } pq - X)$

$X > 0 \ X * \text{snd } pq \leq 2 * D' \ X = \text{nat } \lfloor \text{surd-to-real } pq \rfloor$

$X = \text{nat } \lfloor -1 / \text{surd-to-real-cn timer } pq' \rfloor$

proof –

obtain *p q* **where** [simp]: *pq* = (*p*, *q*) **by** (*cases* *pq*)

obtain *p' q'* **where** [simp]: $pq' = (p', q')$ **by** (*cases* pq')

define α **where** $\alpha = (\text{sqrt } D + p) / q$

define α' **where** $\alpha' = 1 / \text{frac } \alpha$

define $\text{cn timer-}\alpha'$ **where** $\text{cn timer-}\alpha' = (-\text{sqrt } D + (X * q - \text{int } p)) / ((D - (X * q - \text{int } p)^2) \text{ div } q)$

from *assms*(1) **have** $\alpha > 0 \ q > 0$

by (*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 α' -pos: $\text{frac } \alpha > 0$ **using** *Ints-subset-Rats* **by** *auto*

from $\langle pq' = (p', q') \rangle$ **have** p' -def: $p' = X * q - p$ **and** q' -def: $q' = (D - p^2) \text{ div } q$

unfolding pq' -def *sqrt-remainder-step-def* *X-def* **by** (*auto simp: Let-def add-ac*)

have $D' + p = \lfloor \text{sqrt } D + p \rfloor$

by (*auto simp: D'-def*)

also have $\dots \text{ div } \text{int } q = \lfloor (\text{sqrt } D + p) / q \rfloor$

by (*subst floor-divide-real-eq-div [symmetric]*) *auto*

finally have *X-altdef*: $X = \text{nat } \lfloor (\text{sqrt } D + p) / q \rfloor$

unfolding *X-def* *zdiv-int* [symmetric] **by** *auto*

have *nz*: $\text{sqrt } (\text{real } D) + (X * q - \text{real } p) \neq 0$

proof

assume $\text{sqrt } (\text{real } D) + (X * q - \text{real } p) = 0$

hence $\text{sqrt } (\text{real } D) = \text{real } p - X * q$ **by** (*simp add: algebra-simps*)
 also have $\dots \in \mathbb{Q}$ **by** *auto*
 finally show *False* **using** *irrat-sqrt-nonsquare nonsquare* **by** *blast*
qed

from *assms(1)* have $\text{real } (p^2) \leq \text{sqrt } D^2$
 unfolding *of-nat-power* **by** (*intro power-mono*) (*auto simp: red-assoc-def field-simps*)
 also have $\text{sqrt } D^2 = D$ **by** *simp*
 finally have $p^2 \leq D$ **by** (*subst (asm) of-nat-le-iff*)

have $\text{frac } \alpha = \alpha - X$
by (*simp add: X-altdef frac-def α -def*)
 also have $\dots = (\text{sqrt } D - (X * q - \text{int } p)) / q$
using $\langle q > 0 \rangle$ **by** (*simp add: field-simps α -def*)
 finally have $1 / \text{frac } \alpha = q / (\text{sqrt } D - (X * q - \text{int } p))$
by *simp*
 also have $\dots = q * (\text{sqrt } D + (X * q - \text{int } p)) /$
 $((\text{sqrt } D - (X * q - \text{int } p)) * (\text{sqrt } D + (X * q - \text{int } p)))$ (*is - =*
 $?A / ?B$)
using *nz* **by** (*subst mult-divide-mult-cancel-right*) *auto*
 also have $?B = \text{real-of-int } (D - \text{int } p^2 + 2 * X * p * q - \text{int } X^2 * q^2)$
by (*auto simp: algebra-simps power2-eq-square*)
 also have $q \text{ dvd } (D - p^2)$ **using** *assms(1)* **by** (*auto simp: red-assoc-def*)
 with $\langle p^2 \leq D \rangle$ have $\text{int } q \text{ dvd } (\text{int } D - \text{int } p^2)$
by (*metis of-nat-diff of-nat-dvd-iff of-nat-power*)
 hence $D - \text{int } p^2 + 2 * X * p * q - \text{int } X^2 * q^2 = q * ((D - (X * q - \text{int } p)^2) \text{ div } q)$
by (*auto simp: power2-eq-square algebra-simps*)
 also have $?A / \dots = (\text{sqrt } D + (X * q - \text{int } p)) / ((D - (X * q - \text{int } p)^2) \text{ div } q)$
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: α' -def*)

have $\text{dvd}: q \text{ dvd } (D - (X * q - \text{int } p)^2)$
using *assms(1)* $\langle \text{int } q \text{ dvd } (\text{int } D - \text{int } p^2) \rangle$
by (*auto simp: power2-eq-square algebra-simps*)

have $X \leq (\text{sqrt } D + p) / q$ **unfolding** *X-altdef* **by** *simp*
 moreover have $X \neq (\text{sqrt } D + p) / q$
proof
 assume $X = (\text{sqrt } D + p) / q$
 hence $\text{sqrt } D = q * X - \text{real } p$ **using** $\langle q > 0 \rangle$ **by** (*auto simp: field-simps*)
 also have $\dots \in \mathbb{Q}$ **by** *auto*
 finally show *False* **using** *irrat-sqrt-nonsquare[OF nonsquare]* **by** *simp*
qed

ultimately have $X < (\text{sqrt } D + p) / q$ **by** *simp*
 hence $*(X * q - \text{int } p) < \text{sqrt } D$

```

    using ⟨q > 0⟩ 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 ≥ p)
    case True
    hence real p ≤ real X * real q unfolding of-nat-mult [symmetric] of-nat-le-iff
    .
    hence real-of-int ((X * q - int p) ^ 2) < sqrt D ^ 2 using *
      unfolding of-int-power by (intro power-strict-mono) auto
    also have ... = D by simp
    finally show ?thesis by simp
  next
  case False
  hence less: real X * real q < real p
    unfolding 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 ... ≤ real p ^ 2 using less by (intro power-mono) auto
  also have ... < sqrt D ^ 2
    using ⟨q > 0⟩ assms(1) unfolding of-int-power
    by (intro power-strict-mono) (auto simp: red-assoc-def field-simps)
  also have ... = D by simp
  finally 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 ⟨q > 0⟩ dvd by (subst real-of-int-div) (auto intro!: divide-pos-pos)
ultimately have cnj-neg: cnj-α' < 0 unfolding cnj-α'-def using dvd
  unfolding of-int-0-less-iff by (intro divide-neg-pos) auto

have (p - sqrt D) / q < 0
  using assms(1) by (auto simp: red-assoc-def X-altdef le-nat-iff)
also have X ≥ 1
  using assms(1) by (auto simp: red-assoc-def X-altdef le-nat-iff)
hence 0 ≤ real X - 1 by simp
finally have q < sqrt D + int q * X - p
  using ⟨q > 0⟩ 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 ... = D - (int q * X - p) ^ 2
  by (simp add: power2-eq-square algebra-simps)
finally have cnj-α' > -1
  using dvd pos ⟨q > 0⟩ by (simp add: real-of-int-div field-simps cnj-α'-def)

from cnj-neg and this have cnj-α' ∈ {-1 < .. < 0} by auto
have α' > 1 using ⟨frac α > 0⟩

```

by (auto simp: α' -def field-simps frac-lt-1)

have $0 = 1 + (-1 :: \text{real})$
 by simp

also have $1 + -1 < \alpha' + \text{cnj-}\alpha'$
 using $\langle \text{cnj-}\alpha' > -1 \rangle$ and $\langle \alpha' > 1 \rangle$ by (intro add-strict-mono)

also have $\alpha' + \text{cnj-}\alpha' = 2 * (\text{real } X * q - \text{real } p) / ((\text{int } D - (\text{int } X * q - \text{int } p)^2) \text{ div int } q)$
 by (simp add: α' cnj- α' -def add-divide-distrib [symmetric])

finally have $\text{real } X * q - \text{real } p > 0$ 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 $\text{real } (X * q) > \text{real } p$ unfolding of-nat-mult by simp

hence $p\text{-less-}Xq$: $p < X * q$ by (simp only: of-nat-less-iff)

from pos' and $p\text{-less-}Xq$ have $\text{int } D > \text{int } ((X * q - p)^2)$
 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 $\text{int } q \text{ dvd int } (D - (X * q - p)^2)$
 using $p\text{-less-}Xq$ pos'' by (subst of-nat-diff) (auto simp: of-nat-diff)

with dvd have dvd': $q \text{ dvd } (D - (X * q - p)^2)$
 by simp

have $\alpha'\text{-altdef}$: $\alpha' = (\text{sqrt } D + p') / q'$
 using dvd dvd' pos'' $p\text{-less-}Xq$ α'
 by (simp add: real-of-int-div p'-def q'-def real-of-nat-div mult-ac of-nat-diff)

have $\text{cnj-}\alpha'\text{-altdef}$: $\text{cnj-}\alpha' = (-\text{sqrt } D + p') / q'$
 using dvd dvd' pos'' $p\text{-less-}Xq$ unfolding cnj- α' -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'': $q' \text{ dvd } (D - p'^2)$
 by (auto simp: mult-ac p'-def q'-def)

have $\text{real } ((D - p'^2) \text{ div } q) > 0$ 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 $\langle \text{cnj-}\alpha' \in - \rangle$ and dvd'' and $\langle q' > 0 \rangle$
 by (auto simp: red-assoc-def α' -altdef cnj- α' -altdef)

from assms(1) have $\text{real } p < \text{sqrt } D$
 by (auto simp add: field-simps red-assoc-def)

hence $p \leq D'$ unfolding D'-def by linarith

with * have $\text{real } (X * q) < \text{sqrt } (\text{real } D) + D'$
 by simp

thus $X * \text{snd } pq \leq 2 * D'$ unfolding D'-def $\langle pq = (p, q) \rangle$ snd-conv by linarith

have $(\text{sqrt } D + p') / q' = \alpha'$
 by (rule α' -altdef [symmetric])

also have $\alpha' = 1 / \text{frac} ((\text{sqrt } D + p) / q)$
 by (simp add: α' -def α -def)
 finally show $\text{surd-to-real } pq' = 1 / \text{frac} (\text{surd-to-real } pq)$ by (simp add: surd-to-real-def)
 from $\langle X \geq 1 \rangle$ show $X > 0$ by simp
 from $X\text{-altdef}$ show $X = \text{nat } \lfloor \text{surd-to-real } pq \rfloor$ by (simp add: surd-to-real-def)

have $\text{sqrt } (\text{real } D) < \text{real } p + 1 * \text{real } q$
 using $\text{assms}(1)$ by (auto simp: red-assoc-def field-simps)
 also have $\dots \leq \text{real } p + \text{real } X * \text{real } q$
 using $\langle X > 0 \rangle$ by (intro add-left-mono mult-right-mono) (auto simp: of-nat-ge-1-iff)
 finally have $\text{sqrt } (\text{real } D) < \dots$

have $\text{real } p < \text{sqrt } D$
 using $\text{assms}(1)$ by (auto simp add: field-simps red-assoc-def)
 also have $\dots \leq \text{sqrt } D + q * X$
 by linarith
 finally have $\text{less: real } p < \text{sqrt } D + X * q$ by (simp add: algebra-simps)
 moreover have $D + p * p' + X * q * \text{sqrt } D = q * q' + p * \text{sqrt } D + p' * \text{sqrt } D + X * p' * q$
 using $\text{dvd' pos'' } p\text{-less-}Xq \langle q > 0 \rangle$ 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 $*: \text{surd-to-real-cnj } pq' = 1 / (\text{surd-to-real-cnj } pq - X)$
 using $\langle q > 0 \rangle \langle q' > 0 \rangle$ by (auto simp: surd-to-real-cnj-def field-simps)

have $**: a = \text{nat } \lfloor y \rfloor$ if $x \geq 0 \ x < 1$ real $a + x = y$ for $a :: \text{nat}$ and $x y :: \text{real}$
 using that by linarith
 from $\text{assms}(1)$ have $\text{surd-to-real-cnj: surd-to-real-cnj } (p, q) \in \{-1 < .. < 0\}$
 by (auto simp: surd-to-real-cnj-def red-assoc-def)
 have $\text{surd-to-real-cnj } (p, q) < X$
 using $\text{assms}(1)$ less by (auto simp: surd-to-real-cnj-def field-simps red-assoc-def)
 hence $\text{real } X = \text{surd-to-real-cnj } (p, q) - 1 / \text{surd-to-real-cnj } (p', q')$ using *
 using $\text{surd-to-real-cnj-irrat } \text{assms}(1) \langle \text{red-assoc } pq' \rangle$ by (auto simp: field-simps)
 thus $X = \text{nat } \lfloor -1 / \text{surd-to-real-cnj } pq' \rfloor$ using surd-to-real-cnj
 by (intro $**[\text{of } -\text{surd-to-real-cnj } (p, q)]$) auto

qed

lemma red-assoc-denom-2D:
 assumes red-assoc (p, q)
 defines $X \equiv (D' + p) \text{ div } q$
 assumes $X > D'$
 shows $q = 1$
 proof –
 have $X * q \leq 2 * D' \ X > 0$
 using red-assoc-step(4,5)[OF $\text{assms}(1)$] by (simp-all add: X-def)
 note this(1)
 also have $2 * D' < 2 * X$
 by (intro mult-strict-left-mono assms) auto
 finally have $q < 2$ using $\langle X > 0 \rangle$ by simp
 moreover from $\text{assms}(1)$ 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 $\sqrt{D} > p$ $\sqrt{D} < \text{real } p + 1$
 by (*auto simp: red-assoc-def*)
 thus $p = D'$ **unfolding** *D'-def*
 by *linarith*
qed

lemma *red-assoc-begin*:
 $\text{red-assoc } (D', D - D^2)$
 $\text{surd-to-real } (D', D - D^2) = 1 / \text{frac } (\sqrt{D})$
 $\text{surd-to-real-cn } (D', D - D^2) = -1 / (\sqrt{D} + D')$
proof –
 have *pos*: $D > 0$ $D' > 0$
 using *nonsquare* by (*auto simp: D'-def of-nat-ge-1-iff intro!: Nat.gr0I*)

 have $\sqrt{D} \neq D'$
 using *irr-sqrt-nonsquare*[*OF nonsquare*] by *auto*
 moreover have $\sqrt{D} \geq 0$ 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 *irr-sqrt-nonsquare*[*OF nonsquare*] by *auto*
 moreover have $\sqrt{D} \geq 0$ by *simp*
 hence $D' \geq \sqrt{D} - 1$ **unfolding** *D'-def* by *linarith*
 ultimately have *gt*: $D' > \sqrt{D} - 1$ by *simp*

from *less* have $\text{real } D'^2 < \sqrt{D}^2$ by (*intro power-strict-mono*) *auto*
 also have $\dots = D$ by *simp*
 finally have *less'*: $D^2 < D$ **unfolding** *of-nat-power [symmetric] of-nat-less-iff* .

moreover have $\text{real } D' * (\text{real } D' - 1) < \sqrt{D} * (\sqrt{D} - 1)$
 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 $(\sqrt{D} - 1) * \sqrt{D} < \text{real } D' * (\text{real } D' + 1)$
 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*: $\text{frac } (\sqrt{D}) = \sqrt{D} - D'$ **unfolding** *frac-def D'-def*


```

    by auto
  show surd-to-real ( $D', D - D^2$ ) =  $1 / \text{frac} (\text{sqrt } D)$  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 ( $D', D - D^2$ ) =  $-((\text{sqrt } D - D') / (D - D^2))$ 
    using less less' pos by (auto simp: surd-to-real-cnj-def field-simps)
  also have real ( $D - D^2$ ) =  $(\text{sqrt } D - D') * (\text{sqrt } D + D')$ 
    using less' by (simp add: power2-eq-square algebra-simps of-nat-diff)
  also have  $(\text{sqrt } D - D') / \dots = 1 / (\text{sqrt } D + D')$ 
    using less by (subst nonzero-divide-mult-cancel-left) auto
  finally show surd-to-real-cnj ( $D', D - D^2$ ) =  $-1 / (\text{sqrt } D + D')$  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 ( $((\text{sqrt-remainder-step } \sim n) \text{ } pq)$ )
  using assms(1)
proof (induction n arbitrary: pq)
  case 0
  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 0
    by auto
next
  case (Suc n)
  obtain  $p \ q$  where [simp]:  $pq = (p, q)$  by (cases pq)
  have surd-to-real ( $((\text{sqrt-remainder-step } \sim \text{Suc } n) \text{ } pq)$ ) =
    surd-to-real ( $((\text{sqrt-remainder-step } \sim n) (\text{sqrt-remainder-step } (p, q)))$ )
    by (subst funpow-Suc-right) auto
  also have  $\dots = \text{cfrac-remainder} (\text{cfrac-of-real} (\text{surd-to-real} (\text{sqrt-remainder-step}
    (p, q)))) \ n$ 
    using 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 ( $(\text{sqrt-remainder-step } (p, q))$ ) =  $1 / \text{frac} (\text{surd-to-real } (p,
    q))$ 
    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  $\dots = \text{cfrac-tl} (\text{cfrac-of-real} (\text{surd-to-real } (p, q)))$ 
    using Suc.prems Ints-subset-Rats red-assoc-imp-irrat by (subst cfrac-tl-of-real)
auto
  also have cfrac-remainder  $\dots \ n = \text{cfrac-remainder} (\text{cfrac-of-real} (\text{surd-to-real}
    (p, q))) \ (\text{Suc } n)$ 
    by (simp add: cfrac-drop-Suc-right cfrac-remainder-def)
  finally show ?case by simp
qed

```

```

lemma red-assoc-step' [intro]: red-assoc pq  $\implies$  red-assoc (sqrt-remainder-step pq)
  using red-assoc-step(I)[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  $\sim$ 
n) pq)
  by (induction n) auto

lemma floor-sqrt-less-sqrt:  $D' < \text{sqrt } D$ 
proof –
  have  $D' \leq \text{sqrt } D$  unfolding D'-def by auto
  moreover have  $\text{sqrt } D \neq D'$ 
    using irrat-sqrt-nonsquare[OF nonsquare] by auto
  ultimately show ?thesis by auto
qed

lemma red-assoc-bounds:
  assumes red-assoc pq
  shows  $pq \in (\text{SIGMA } p:\{0 <..D'\}. \{ \text{Suc } D' - p..D' + p \})$ 
proof –
  obtain p q where [simp]:  $pq = (p, q)$  by (cases pq)
  from assms have *:  $p < \text{sqrt } D$ 
    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 intro!: Nat.gr0I simp: red-assoc-def)

  have  $q > \text{sqrt } D - p$   $q < \text{sqrt } D + p$ 
    using assms by (auto simp: red-assoc-def field-simps)
  hence  $q \geq D' + 1 - p$   $q \leq D' + p$ 
    unfolding D'-def by linarith+
  with  $p \langle p > 0 \rangle$  show ?thesis by simp
qed

lemma surd-to-real-cnj-eq-iff:
  assumes red-assoc pq red-assoc pq'
  shows  $\text{surd-to-real-cnj } pq = \text{surd-to-real-cnj } pq' \longleftrightarrow pq = pq'$ 
proof
  assume eq:  $\text{surd-to-real-cnj } pq = \text{surd-to-real-cnj } pq'$ 
  from assms have pos:  $\text{snd } pq > 0$   $\text{snd } pq' > 0$  by (auto simp: red-assoc-def)
  have  $\text{snd } pq = \text{snd } pq'$ 
  proof (rule ccontr)
    assume  $\text{snd } pq \neq \text{snd } pq'$ 
    with eq have  $\text{sqrt } D = (\text{real } (\text{fst } pq' * \text{snd } pq) - \text{fst } pq * \text{snd } pq') / (\text{real } (\text{snd }
pq) - \text{snd } pq')$ 
    using pos by (auto simp: field-simps surd-to-real-cnj-def case-prod-unfold)
    also have  $\dots \in \mathbb{Q}$  by auto
    finally show False using irrat-sqrt-nonsquare[OF nonsquare] by auto
  qed
  moreover from this eq pos have  $\text{fst } pq = \text{fst } pq'$ 

```

by (auto simp: surd-to-real-cnjd-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 0
from nonsquare have $D > 0$ by (auto intro!: Nat.gr0I)
with red-*assoc-begin* show ?case using nonsquare irrat-sqrt-nonsquare[OF non-square]
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 ... = 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) =
[cfrac-remainder (cfrac-of-real (sqrt D)) (Suc n)]
using irrat-sqrt-nonsquare[OF nonsquare] by (subst floor-cfrac-remainder) auto
also have cfrac-remainder (cfrac-of-real (sqrt D)) (Suc n) = surd-to-real (sqrt-remainder-surd
 n)
by (rule surd-to-real-sqrt-remainder-surd [symmetric])
also have nat [surd-to-real (sqrt-remainder-surd n)] = sqrt-cfrac-nth n
unfolding sqrt-cfrac-nth-def using red-*assoc-step*(6)[OF red-*assoc-sqrt-remainder-surd*[of
 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 > 0*
 and *period-length-le-aux: l ≤ D' * (D' + 1)*
 and *sqrt-remainder-surd-periodic: $\bigwedge n. \text{sqrt-remainder-surd } n = \text{sqrt-remainder-surd } (n \bmod l)$*
 and *sqrt-cfrac-periodic: $\bigwedge n. \text{sqrt-cfrac-nth } n = \text{sqrt-cfrac-nth } (n \bmod l)$*
 and *sqrt-remainder-surd-smallest-period:*
 $\bigwedge n. n \in \{0 <..<l\} \implies \text{sqrt-remainder-surd } n \neq \text{sqrt-remainder-surd } 0$
 and *snd-sqrt-remainder-surd-gt-1: $\bigwedge n. n < l - 1 \implies \text{snd } (\text{sqrt-remainder-surd } n) > 1$*
 and *sqrt-cfrac-le: $\bigwedge n. n < l - 1 \implies \text{sqrt-cfrac-nth } n \leq D'$*
 and *sqrt-remainder-surd-last: $\text{sqrt-remainder-surd } (l - 1) = (D', 1)$*
 and *sqrt-cfrac-last: $\text{sqrt-cfrac-nth } (l - 1) = 2 * D'$*
 and *sqrt-cfrac-palindrome: $\bigwedge n. n < l - 1 \implies \text{sqrt-cfrac-nth } (l - n - 2) = \text{sqrt-cfrac-nth } n$*
 and *sqrt-cfrac-smallest-period:*
 $\bigwedge l'. l' > 0 \implies (\bigwedge k. \text{sqrt-cfrac-nth } (k + l') = \text{sqrt-cfrac-nth } k) \implies l' \geq l$

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 $\dots = D' * (D' + 1)$
by (*induction D'*) (*auto simp: power2-eq-square*)
 finally have [*simp*]: *card S* = $D' * (D' + 1)$.

have $D' * (D' + 1) + 1 = \text{card } \{..D' * (D' + 1)\}$ **by** *simp*
 define *k1* where
 $k1 = (\text{LEAST } k1. k1 \leq D' * (D' + 1) \wedge (\exists k2. k2 \leq D' * (D' + 1) \wedge k1 \neq k2 \wedge f k1 = f k2))$
 define *k2* where
 $k2 = (\text{LEAST } k2. k2 \leq D' * (D' + 1) \wedge k1 \neq k2 \wedge f k1 = f k2)$

have $f \cdot \{..D' * (D' + 1)\} \subseteq S$ **unfolding** $S\text{-def}$
 using $\text{red-assoc-bounds}[OF *]$ **by** blast
 hence $\text{card } (f \cdot \{..D' * (D' + 1)\}) \leq \text{card } S$
 by (intro card-mono) auto
 also have $\text{card } S = D' * (D' + 1)$ **by** simp
 also have $\dots < \text{card } \{..D' * (D' + 1)\}$ **by** simp
 finally have $\neg \text{inj-on } f \{..D' * (D' + 1)\}$
 by (rule pigeonhole)
 hence $\exists k1. k1 \leq D' * (D' + 1) \wedge (\exists k2. k2 \leq D' * (D' + 1) \wedge k1 \neq k2 \wedge f k1 = f k2)$
 by $(\text{auto simp: inj-on-def})$
 from $\text{LeastI-ex}[OF \text{ this, folded } k1\text{-def}]$
 have $k1 \leq D' * (D' + 1) \exists k2 \leq D' * (D' + 1). k1 \neq k2 \wedge f k1 = f k2$ **by** auto
 moreover from $\text{LeastI-ex}[OF \text{ this}(2), \text{folded } k2\text{-def}]$
 have $k2 \leq D' * (D' + 1) k1 \neq k2 f k1 = f k2$ **by** auto
 moreover have $k1 \leq k2$
proof (rule ccontr)
 assume $\neg(k1 \leq k2)$
 hence $k2 \leq D' * (D' + 1) \wedge (\exists k2'. k2' \leq D' * (D' + 1) \wedge k2 \neq k2' \wedge f k2 = f k2')$
 using $\langle k1 \leq D' * (D' + 1) \rangle$ **and** $\langle k1 \neq k2 \rangle$ **and** $\langle f k1 = f k2 \rangle$ **by** auto
 hence $k1 \leq k2$ **unfolding** $k1\text{-def}$ **by** (rule Least-le)
 with $\langle \neg(k1 \leq k2) \rangle$ **show** False **by** simp
qed
 ultimately have $k12: k1 < k2 k2 \leq D' * (D' + 1) f k1 = f k2$ **by** auto

have $[simp]: k1 = 0$
proof $(\text{cases } k1)$
 case $(\text{Suc } k1')$
 define $k2'$ where $k2' = k2 - 1$
 have $\text{Suc}': k2 = \text{Suc } k2'$ **using** $k12$ **by** $(\text{simp add: } k2'\text{-def})$
 have $\text{nz: surd-to-real-cn } j (\text{surd-remainder-step } (f k1')) \neq 0$
 $\text{surd-to-real-cn } j (\text{surd-remainder-step } (f k2')) \neq 0$
 using $\text{surd-to-real-cn } j \text{-nz}[OF *[of k2]] \text{ surd-to-real-cn } j \text{-nz}[OF *[of k1]]$
 by $(\text{simp-all add: } f\text{-def } \text{Suc } \text{Suc}')$

define a where $a = (D' + \text{fst } (f k1)) \text{ div } \text{snd } (f k1)$
 define a' where $a' = (D' + \text{fst } (f k1')) \text{ div } \text{snd } (f k1')$
 define a'' where $a'' = (D' + \text{fst } (f k2')) \text{ div } \text{snd } (f k2')$
 have $a' = \text{nat } \lfloor -1 / \text{surd-to-real-cn } j (\text{surd-remainder-step } (f k1')) \rfloor$
 using $\text{red-assoc-step}[OF *[of k1']]$ **by** $(\text{simp add: } a'\text{-def})$
 also have $\text{surd-remainder-step } (f k1') = f k1$
 by $(\text{simp add: } \text{Suc } f\text{-def})$
 also have $f k1 = f k2$ **by** fact
 also have $f k2 = \text{surd-remainder-step } (f k2')$ **by** $(\text{simp add: } \text{Suc}' f\text{-def})$
 also have $\text{nat } \lfloor -1 / \text{surd-to-real-cn } j (\text{surd-remainder-step } (f k2')) \rfloor = a''$
 using $\text{red-assoc-step}[OF *[of k2']]$ **by** $(\text{simp add: } a''\text{-def})$
 finally have $a' = a''$.

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* ($f\ k2'$) = $1 / \text{surd-to-real-cnj}(\text{sqrt-remainder-step}(f\ k2')) + a''$
using *red-assoc-step*(3)[$OF\ *[of\ k2']$, *folded a''-def*] *nz*
by (*simp add: field-simps*)
also have $\dots = 1 / \text{surd-to-real-cnj}(\text{sqrt-remainder-step}(f\ k1')) + a'$
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 $1 / \text{surd-to-real-cnj}(\text{sqrt-remainder-step}(f\ k1')) + a' = \text{surd-to-real-cnj}(f\ k1')$
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** $k1' \leq D' * (D' + 1) \wedge (\exists k2 \leq D' * (D' + 1). k1' \neq k2 \wedge f\ k1' = f\ k2)$
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*)
qed *auto*

have *smallest-period*: $f\ k \neq f\ 0$ **if** $k \in \{0 < .. < k2\}$ **for** k
proof
assume $f\ k = f\ 0$
hence $k \leq D' * (D' + 1) \wedge k1 \neq k \wedge f\ k1 = f\ k$
using *k12* **that** **by** *auto*
hence $k2 \leq k$ **unfolding** *k2-def* **by** (*rule Least-le*)
with that **show** *False* **by** *auto*
qed

have *snd-f-gt-1*: $\text{snd}(f\ k) > 1$ **if** $k < k2 - 1$ **for** k
proof –
have $\text{snd}(f\ k) \neq 1$
proof
assume $\text{snd}(f\ k) = 1$
hence $f\ k = (D', 1)$ **using** *red-assoc-denom-1*[*of fst (f k)*] $*[of\ k]$
by (*cases f k*) *auto*
hence *sqrt-remainder-step* ($f\ k$) = $(D', D - D'^2)$ **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 $\text{snd}(f\ k) > 0$ **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** $k < k2 - 1$ **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* $k * q \leq D' * 2$
using *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** *simp*
also have $f\ k2 = f\ 0$
using *k12* **by** *simp*
also have $f\ 0 = (D', D - D^2)$ **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*
pq
by *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* $n = \text{sqrt-remainder-surd}\ (n \bmod k2)$ **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* $k = \text{sqrt-cfrac-nth}\ (k \bmod k2)$ **for** k
using *period*[*of* k] **by** (*simp add: sqrt-cfrac-nth-def*)

have *k2-le*: $l \geq k2$ **if** $l > 0 \wedge k$. *sqrt-cfrac-nth* $(k + l) = \text{sqrt-cfrac-nth}\ k$ **for** l
proof (*rule ccontr*)
assume $*$: $\neg(l \geq k2)$
hence *sqrt-cfrac-nth* $(k2 - Suc\ l) = \text{sqrt-cfrac-nth}\ (k2 - 1)$
using *that*(2)[*of* $k2 - Suc\ l$] **by** *simp*
also have $\dots = 2 * D'$
using *last* **by** (*simp add: sqrt-cfrac-nth-def f-def*)
finally have $2 * D' = \text{sqrt-cfrac-nth}\ (k2 - Suc\ l)$..

```

    also have ... ≤ 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 ∧ (∀ 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 ∧ (∀ n. sqrt-cfrac-nth (n + l) = sqrt-cfrac-nth
n))
  by simp

have [simp]: D ≠ 0 using nonsquare by (auto intro!: Nat.gr0I)
have ∃ l. l > 0 ∧ (∀ 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
qed (insert k12, auto)
from LeastI-ex[OF this, folded l-altdef]
have l: l > 0 ∧ k. sqrt-cfrac-nth (k + l) = sqrt-cfrac-nth k
  by (simp-all add: sqrt-cfrac)

have l ≤ k2 unfolding l-altdef
  by (rule Least-le) (subst (1 2) period', insert k12, auto)
moreover have k2 ≤ l using k2-le l by blast
ultimately have [simp]: l = k2 by auto

define x' where x' = (λk. -1 / surd-to-real-cn timer (f k))
{
  fix k :: nat
  have nz: surd-to-real-cn timer (f k) ≠ 0 surd-to-real-cn timer (f (Suc k)) ≠ 0
    using surd-to-real-cn timer-nz[OF *, of k] surd-to-real-cn timer-nz[OF *, of Suc k]
    by (simp-all add: f-def)

  have surd-to-real-cn timer (f k) ≠ sqrt-cfrac-nth k
    using surd-to-real-cn timer-irrat[OF *[of k]] by auto
  hence x' (Suc k) = sqrt-cfrac-nth k + 1 / x' k
    using 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 ≠ 0 for k
  using surd-to-real-cn timer-nz[OF *[of k]] by (auto simp: x'-def)
have x'-0: x' 0 = real D' + sqrt D
  using red-assoc-begin by (simp add: x'-def f-def)

define c' where c' = cfrac (λn. sqrt-cfrac-nth (l - Suc n))
define c'' where c'' = cfrac (λn. 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 ≤ l for n
    using that
  proof (induction n)
    case (Suc n)
    have x' l = conv' c' n (x' (l - n))
      using Suc.prem by (intro Suc.IH [symmetric]) auto
    also have l - n = Suc (l - Suc n)
      using Suc.prem by simp
    also have x' ... = cfrac-nth c' n + 1 / x' (l - Suc n)
      by (subst x'-Suc) simp
    also have conv' c' n ... = 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'-0: conv' c' l (x' 0) = x' 0
    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 0
    thus ?thesis by (simp add: x'-0 D'-def)
  next
    case (Suc n')
    have sqrt D ∉ 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)
  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 intro!: Nat.gr0I)
    hence D' + sqrt D > 0 + 1 using D'-pos by (intro add-strict-mono) auto
    hence x' 0 > 1 by (auto simp: x'-0)
    hence cfrac-nth c' (Suc n) = cfrac-nth (cfrac-of-real (conv' c' l (x' 0))) (Suc
n)
      using ⟨n < l - 1⟩ using cfrac-of-real-conv' by auto

```

also have $\dots = \text{cfrac-nth } (\text{cfrac-of-real } (x' 0)) (Suc\ n)$
 by $(subst\ conv'-x'-0)\ auto$
 also have $\dots = \text{cfrac-nth } c'' (Suc\ n)$ by *fact*
 finally show $\text{sqrt-cfrac-nth } (l - n - 2) = \text{sqrt-cfrac-nth } n$
 by *simp*
 qed

show $l > 0 \wedge l \leq D' * (D' + 1)$ using *k12* by *simp-all*
 show $\text{sqrt-remainder-surd } n = \text{sqrt-remainder-surd } (n \bmod l)$
 $\text{sqrt-cfrac-nth } n = \text{sqrt-cfrac-nth } (n \bmod l)$ for n
 using *period[of n] period'[of n]* by *simp-all*
 show $\text{sqrt-remainder-surd } n \neq \text{sqrt-remainder-surd } 0$ if $n \in \{0 < .. < l\}$ for n
 using *smallest-period[of n] that* by $(auto\ simp: f-def)$
 show $\text{snd } (\text{sqrt-remainder-surd } n) > 1$ if $n < l - 1$ for n
 using *that snd-f-gt-1[of n]* by $(simp\ add: f-def)$
 show $f\ (l - 1) = (D', 1)$ and $\text{sqrt-cfrac-nth } (l - 1) = 2 * D'$
 using *last* by $(simp-all\ add: \text{sqrt-cfrac-nth-def } f-def)$
 show $\text{sqrt-cfrac-nth } k \leq D'$ if $k < l - 1$ for k
 using *sqrt-cfrac-le[of k] that* by *simp*
 show $l' \geq l$ if $l' > 0 \wedge k. \text{sqrt-cfrac-nth } (k + l') = \text{sqrt-cfrac-nth } k$ for l'
 using *k2-le[of l'] that* by *auto*
 qed

theorem *cfrac-sqrt-periodic*:
 $\text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D)) (Suc\ n) =$
 $\text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D)) (Suc\ (n \bmod l))$
 using *sqrt-cfrac-periodic[of n]* by $(metis\ \text{sqrt-cfrac})$

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

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

theorem *cfrac-sqrt-palindrome*:
 assumes $n \in \{0 < .. < l\}$
 shows $\text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D))\ (l - n) = \text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D))\ n$
 proof –
 have $\text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D))\ (l - n) = \text{sqrt-cfrac-nth } (l - n - 1)$
 using *assms* by $(subst\ \text{sqrt-cfrac})\ (auto\ simp: Suc-diff-Suc)$
 also have $\dots = \text{sqrt-cfrac-nth } (n - 1)$
 using *assms* by $(subst\ \text{sqrt-cfrac-palindrome } [symmetric])\ auto$
 also have $\dots = \text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } D))\ n$
 using *assms* by $(subst\ \text{sqrt-cfrac})\ auto$
 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  $last\ cs = 2 * Discrete.sqrt\ D$ 
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
     $P = (\lambda(as, pq). let\ n = length\ as$ 
       $in\ n < l \wedge pq = sqrt-remainder-surd\ n \wedge as = rev\ (map$ 
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  $step': P\ (step'\ s) \wedge Suc\ (length\ (fst\ s)) < l$  if  $P\ s\ b\ s$  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  $P\ (step'\ s) \wedge Suc\ (length\ (fst\ s)) < l$ 

```

```

    by (simp add: b-def P-def Let-def n-def step'-def sqrt-cfrac-nth-def
        sqrt-remainder-surd-def case-prod-unfold)
  qed
  have [simp]: length (fst (step' s)) = Suc (length (fst s)) for s
    by (simp add: step'-def case-prod-unfold)

  have  $\exists x. \text{while-option } b \text{ step' initial} = \text{Some } x$ 
  proof (rule measure-while-option-Some)
    fix s assume *:  $P \ s \ b \ s$ 
    from step'[OF *] show  $P \ (\text{step' } s) \wedge \mu \ (\text{step' } s) < \mu \ s$ 
      by (auto simp: b-def  $\mu$ -def case-prod-unfold intro!: diff-less-mono2)
  qed auto
  then obtain x where  $x: \text{while-option } b \text{ step' initial} = \text{Some } x \ ..$ 
  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  $n = \text{length } as$ 
  have [simp]:  $q = 1$  using  $\langle \neg b \ x \rangle$  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:  $as = \text{rev } (\text{map } \text{sqrt-cfrac-nth } [0..<n])$  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  $\text{step'} \equiv (\lambda(as, pq). ((D' + \text{fst } pq) \text{ div } \text{snd } pq \ \# \ as, \text{sqrt-remainder-step } pq))$ 
  defines  $b \equiv (\lambda(-, pq). \text{snd } pq \neq 1)$ 
  defines  $\text{initial} \equiv ([], (D', D - D^2))$ 
  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  $\text{nat } (\text{cfrac-nth } (\text{cfrac-of-real } (\text{sqrt } (\text{real } D)))) (\text{Suc } k) = \text{sqrt-cfrac-nth } k$  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

```

lemma *sqrt-nat-period-length-le*: *sqrt-nat-period-length* $D \leq \text{nat } \lfloor \text{sqrt } D \rfloor * (\text{nat } \lfloor \text{sqrt } D \rfloor + 1)$

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 $D = 0 \longleftrightarrow \text{is-square } D$

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

lemma *sqrt-nat-period-length-pos-iff* [*simp*]:

sqrt-nat-period-length $D > 0 \longleftrightarrow \neg \text{is-square } D$

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

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

sqrt-cfrac-info $D =$

(*let* $D' = \text{Discrete.sqrt } D$

in if $D'^2 = D$ *then* $(0, D', [])$

else

case while-option

($\lambda(-, pq). \text{snd } pq \neq 1$)

($\lambda(as, (p, q)). \text{let } X = (p + D') \text{ div } q; p' = X * q - p$
in $(X \# as, p', (D - p'^2) \text{ div } q)$)

($[], D', D - D'^2$)

of Some $(as, -) \Rightarrow (\text{Suc } (\text{length } as), D', \text{rev } ((2 * D') \# as)))$

proof –

define D' **where** $D' = \text{Discrete.sqrt } D$

show *?thesis*

proof (*cases is-square* D)

case *True*

hence $D'^2 = D$ **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*)

next

case *False*

hence $D'^2 \neq D$ **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 $D = \text{fst } (\text{sqrt-cfrac-info } D)$

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

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)) = Discrete.sqrt D*

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

definition *cfrac-sqrt-nth* :: *nat × nat × nat iarray ⇒ nat ⇒ nat* **where**

cfrac-sqrt-nth info n =

(case info of (l, a0, as) ⇒ if n = 0 then a0 else as !! ((n - 1) mod l))

lemma *cfrac-sqrt-nth*:

assumes *¬is-square D*

shows *cfrac-nth (cfrac-of-real (sqrt D)) n =*

int (cfrac-sqrt-nth (sqrt-cfrac-info-array D) n) (is ?lhs = ?rhs)

proof (*cases n*)

case (*Suc n'*)

define *l* **where** *l = sqrt-nat-period-length D*

from *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 *... = map (λn. nat (cfrac-nth (cfrac-of-real (sqrt D)) (Suc n))) [0..*l*]*
! (n' mod l)

using *⟨l > 0⟩* **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)

lemma *sqrt-cfrac-code* [code]:

sqrt-cfrac D =

(let info = sqrt-cfrac-info-array D;

(l, a0, -) = info

in if l = 0 then cfrac-of-int (int a0) else cfrac (cfrac-sqrt-nth info))

proof (*cases is-square D*)

case *True*

hence *sqrt (real D) = of-int (Discrete.sqrt D)*

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*)

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  $\text{sqrt-cfrac } D = \text{cfrac } (\text{cfrac-sqrt-nth } (\text{sqrt-cfrac-info-array } D))$ 
  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 ( $\lambda n. |\text{conv } (\text{sqrt-cfrac } 129) \ n \wedge^2 - 129|$ ) [0..<20]

```

end

5 Lifting solutions of Pell's Equation

```

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

```

5.1 Auxiliary material

```

lemma (in pell) snth-pell-solutions: snth (pell-solutions D) n = nth-solution n
  by (simp add: pell-solutions-def Let-def find-fund-sol-correct nonsquare-D nth-solution-def
    pell-power-def pell-mul-commutes[of - fund-sol])

```

```

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 ∧ prime p} using assms
  by (subst (asm) prime-factors-dvd) (auto simp: conj-commute)
hence finite {p. p dvd x ∧ prime p ∧ odd (multiplicity p x)}
  by (rule finite-subset [rotated]) auto
moreover have odd (n :: nat)  $\longleftrightarrow$  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
qed

lemma squarefree-part-nat:
  squarefree-part (n :: nat) = ( $\prod$  {p ∈ prime-factors n. odd (multiplicity p n)})
proof (cases n = 0)
case False
hence ( $\prod$  {p ∈ 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 ... = 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 ≠ 0
  shows prime-factorization (square-part x) =
    ( $\sum$  p ∈ 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 ∧ 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
hence ?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
qed

lemma prod-mset-sum: prod-mset (sum f A) = ( $\prod$  x ∈ A. prod-mset (f x))
  by (induction A rule: infinite-finite-induct) auto

```


lemma *square-part-nat*:
assumes $n > 0$
shows $\text{square-part } (n :: \text{nat}) = (\prod p \in \text{prime-factors } n. p \wedge (\text{multiplicity } p \ n \text{ div } 2))$
proof –
have $(\prod p \in \text{prime-factors } n. p \wedge (\text{multiplicity } p \ n \text{ div } 2)) =$
 $\text{prod-mset } (\text{prime-factorization } (\text{square-part } n))$ **using** *assms*
by (*subst prime-factorization-square-part*) (*auto simp: prod-unfold-prod-mset prod-mset-sum*)
also have $\dots = \text{square-part } n$ **using** *assms*
by (*intro prod-mset-prime-factorization-nat Nat.gr0I*) *auto*
finally show *?thesis* ..
qed

lemma *square-squarefree-part-nat-code* [*code*]:
 $\text{square-squarefree-part-nat } n = (\text{if } n = 0 \text{ then } (0, 1)$
 $\text{else let } ps = \text{prime-factorization } n$
 $\text{in } ((\prod p \in \text{set-mset } ps. p \wedge (\text{count } ps \ p \text{ div } 2)),$
 $\prod (\text{Set.filter } (\lambda p. \text{odd } (\text{count } ps \ p))) (\text{set-mset } ps))))$
by (*cases* $n = 0$)
(auto simp: Let-def square-squarefree-part-nat-def squarefree-part-nat Set.filter-def
 $\text{count-prime-factorization square-part-nat intro!: prod.cong})$

lemma *square-part-nat-code* [*code-unfold*]:
 $\text{square-part } (n :: \text{nat}) = (\text{if } n = 0 \text{ then } 0$
 $\text{else let } ps = \text{prime-factorization } n \text{ in } (\prod p \in \text{set-mset } ps. p \wedge (\text{count } ps \ p \text{ div } 2)))$
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*]:
 $\text{squarefree-part } (n :: \text{nat}) = (\text{if } n = 0 \text{ then } 1$
 $\text{else let } ps = \text{prime-factorization } n \text{ in } (\prod (\text{Set.filter } (\lambda p. \text{odd } (\text{count } ps \ p)))$
 $(\text{set-mset } ps))))$
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 $\text{is-nth-power } n \ (a * b \wedge n) \ b > 0 \ n > 0$
shows $\text{is-nth-power } n \ (a :: \text{nat})$
proof –
from *assms* **obtain** k **where** $k \wedge n = a * b \wedge n$
by (*auto elim: is-nth-powerE*)
with *assms*(2,3) **have** $b \text{ dvd } k$
by (*metis dvd-triv-right pow-divides-pow-iff*)
then obtain l **where** $k = b * l$
by *auto*
with k **have** $a = l \wedge n$ **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 $\bigwedge x' y'. \text{nontriv-solution } (x', y') \implies x \leq x'$
 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 intro!: 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-solution
 by (*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 $\bigwedge x' y'. \text{nontriv-solution } (x', y') \implies \text{fst } xy \leq x'$
 shows *fund-sol* = *xy*
 using *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 $\bigwedge x' y'. \text{nontriv-solution } (x', y') \implies y \leq y'$
 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*]
 fund-sol-is-nontriv-solution
 by (*auto simp: nontriv-solution-imp-solution prod-eq-iff*)
 ultimately show ?thesis by *simp*
 qed

lemma (in *pell*) *fund-sol-eqI-snd'*:
 assumes *nontriv-solution* *xy*
 assumes $\bigwedge x' y'. \text{nontriv-solution } (x', y') \implies \text{snd } xy \leq y'$
 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' :: nat
  assumes nz: a > 0
  defines D' ≡ D * a2
begin

```

```

lemma nonsquare-D': ¬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 × nat ⇒ nat × nat where
  lift-solution = (λ(x, y). (x, y div a))

```

```

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

```

```

definition liftable-solution :: nat × nat ⇒ bool where
  liftable-solution = (λ(x, y). a dvd y)

```

```

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

```

```

lemma lift-solution-iff: lift.solution xy ⟷ 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 ∧n. 0 < n ⇒ n < m ⇒ ¬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) ≠ (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' ≠ 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') ∈ 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 intro!: 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 ≥ m by (cases n ≥ m) auto
hence fst (nth-solution m) ≤ fst (nth-solution n)
  using strict-mono-less-eq[OF strict-mono-nth-solution(1)] by simp
thus fst (lift-solution (nth-solution m)) ≤ 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 ≠ 0
  shows squarefree-part x = unit-factor x
proof -
  from assms obtain y where [simp]: x = y ^ 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 ... = 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 ^ 2)*
by *auto*
also have *square-part x ^ 2 = squarefree-part x * square-part x ^ 2*
using *assms by simp*
also have *... = x*
by (*rule squarefree-decompose [symmetric]*)
finally show *?thesis .*
qed

definition *find-fund-sol-fast* **where**
find-fund-sol-fast D =
(let (a, D') = square-squarefree-part-nat D
in
if D' = 0 ∨ D' = 1 then (0, 0)
else if a = 1 then pell.fund-sol D
else map-prod id (λy. y div a)
(shd (sdrop-while (λ(-, y). y = 0 ∨ ¬a dvd y) (pell-solutions D'))))

lemma *find-fund-sol-fast: find-fund-sol D = find-fund-sol-fast D*
proof (*cases is-square D ∨ 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 *⟨D > 0⟩ by (intro Nat.gr0I) (auto simp: a-def)*
moreover have *¬is-square D'*
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 ⟨D > 0⟩*
by *unfold-locales (auto simp: D'-def a-def squarefree-decompose [symmetric])*

```

define i where i = (LEAST i. case lift.nth-solution i of ( $\lambda(-, y) \Rightarrow y > 0 \wedge a \text{ dvd } y$ )
have ex:  $\exists i$ . case lift.nth-solution i of ( $\lambda(-, y) \Rightarrow y > 0 \wedge a \text{ dvd } y$ )
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 sol
using LeastI-ex[OF ex] by (simp-all add: sol-def i-def case-prod-unfold)
have i > 0
using sol by (intro Nat.gr0I) (auto simp: sol-def lift.nth-solution-def)

have find-fund-sol-fast D = map-prod id ( $\lambda y. y \text{ div } a$ )
  (shd (sdrop-while ( $\lambda(-, y). y = 0 \vee \neg a \text{ dvd } y$ ) (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 ( $\lambda(-, y). y = 0 \vee \neg a \text{ dvd } y$ ) (pell-solutions D') =
  sdrop-while (Not  $\circ (\lambda(-, y). y > 0 \wedge a \text{ dvd } y)$ ) (pell-solutions D')
by (simp add: o-def case-prod-unfold)
also have  $\dots = \text{sdrop } i$  (pell-solutions D')
using ex by (subst sdrop-while-sdrop-LEAST) (simp-all add: lift.snth-pell-solutions)

```

```

i-def)
  also have shd ... = sol
    by (simp add: lift.snth-pell-solutions sol-def)
  finally have eq: find-fund-sol-fast D = map-prod id (λy. 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)
  next
    fix j :: nat assume j: j > 0 j < i
    show  $\neg \text{lift.liftable-solution (lift.nth-solution } j)$ 
    proof
      assume liftable: lift.liftable-solution (lift.nth-solution j)
      have snd (lift.nth-solution j) > 0
      using  $\langle j > 0 \rangle$  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  $\wedge$  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
  qed
  also have ... = 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

end

```

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

```

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

lemma irrational-times-int-eq-intD:
  assumes p * real-of-int a = real-of-int b

```

```

assumes  $p \notin \mathbb{Q}$ 
shows  $a = 0 \wedge b = 0$ 
proof –
  have  $a = 0$ 
  proof (rule ccontr)
    assume  $a \neq 0$ 
    with assms(1) have  $p = b / a$  by (auto simp: field-simps)
    also have  $\dots \in \mathbb{Q}$  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  $D :: \text{nat}$  and  $c\ h\ k\ P\ Q\ l$ 
  assumes nonsquare:  $\neg \text{is-square } D$ 
  defines  $c \equiv \text{cfrac-of-real } (\text{sqrt } D)$ 
  defines  $h \equiv \text{conv-num } c$  and  $k \equiv \text{conv-denom } c$ 
  defines  $P \equiv \text{fst} \circ \text{sqrt-remainder-surd } D$  and  $Q \equiv \text{snd} \circ \text{sqrt-remainder-surd } D$ 
  defines  $l \equiv \text{sqrt-nat-period-length } D$ 
begin

```

```

interpretation pell  $D$ 
  by unfold-locales fact+

```

```

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

```

```

lemma conv-num-denom-pell:
   $h\ 0^{\wedge} 2 - D * k\ 0^{\wedge} 2 < 0$ 
   $m > 0 \implies h\ m^{\wedge} 2 - D * k\ m^{\wedge} 2 = (-1)^{\wedge} \text{Suc } m * Q\ m$ 
proof –
  define  $D'$  where  $D' = \text{Discrete.sqrt } D$ 
  have  $h\ 0^{\wedge} 2 - D * k\ 0^{\wedge} 2 = \text{int } (D'^{\wedge} 2) - \text{int } D$ 
    by (simp-all add: h-def k-def c-def Discrete-sqrt-altdef D'-def)
  also {
    have  $\text{int } (D'^{\wedge} 2) - \text{int } D \leq 0$ 
      using Discrete.sqrt-power2-le[of D] by (simp add: D'-def)
    moreover have  $D \neq D'^{\wedge} 2$  using nonsquare by auto
    ultimately have  $\text{int } (D'^{\wedge} 2) - \text{int } D < 0$  by linarith
  }
  finally show  $h\ 0^{\wedge} 2 - D * k\ 0^{\wedge} 2 < 0$  .

```



```

next
  assume  $m > 0$ 
  define  $n$  where  $n = m - 1$ 
  define  $\alpha$  where  $\alpha = \text{cfrac-remainder } c$ 
  define  $\alpha'$  where  $\alpha' = \text{sqrt-remainder-surd } D$ 
  have  $m$ :  $m = \text{Suc } n$  using  $\langle m > 0 \rangle$  by (simp add:  $n\text{-def}$ )
  from nonsquare have  $D > 1$ 
    by (cases  $D$ ) (auto intro!:  $\text{Nat.gr0I}$ )
  from nonsquare have irrat:  $\text{sqrt } D \notin \mathbb{Q}$ 
    using irrat-sqrt-nonsquare by blast
  have [simp]:  $\text{cfrac-lim } c = \text{sqrt } D$ 
    using irrat  $\langle D > 1 \rangle$  by (simp add:  $c\text{-def}$ )
  have  $\alpha\text{-pos}$ :  $\alpha \, n > 0$  for  $n$ 
    unfolding  $\alpha\text{-def}$  using wf  $\langle D > 1 \rangle$   $\text{cfrac-remainder-pos}[of \, c \, n]$ 
    by (cases  $n = 0$ ) auto
  have  $\alpha'$ :  $\alpha' \, n = (P \, n, Q \, n)$  for  $n$  by (simp add:  $\alpha'\text{-def } P\text{-def } Q\text{-def}$ )
  have  $Q\text{-pos}$ :  $Q \, n > 0$  for  $n$ 
    using snd-sqrt-remainder-surd-pos[OF nonsquare] by (simp add:  $Q\text{-def}$ )
  have  $k\text{-pos}$ :  $k \, n > 0$  for  $n$ 
    by (auto simp:  $k\text{-def}$  intro!:  $\text{conv-denom-pos}$ )
  have  $k\text{-nonneg}$ :  $k \, n \geq 0$  for  $n$ 
    by (auto simp:  $k\text{-def}$  intro!:  $\text{conv-denom-nonneg}$ )

  let  $?A = (\text{sqrt } D + P \, (n + 1)) * h \, (n + 1) + Q \, (n + 1) * h \, n$ 
  let  $?B = (\text{sqrt } D + P \, (n + 1)) * k \, (n + 1) + Q \, (n + 1) * k \, n$ 
  have  $?B > 0$  using  $k\text{-pos } Q\text{-pos } k\text{-nonneg}$ 
    by (intro add-nonneg-pos mult-nonneg-nonneg add-nonneg-nonneg) auto

  have  $\text{sqrt } D = \text{conv}' \, c \, (\text{Suc } (\text{Suc } n)) \, (\alpha \, (\text{Suc } (\text{Suc } n)))$ 
    unfolding  $\alpha\text{-def}$  by (subst  $\text{conv}'\text{-cfrac-remainder}$ ) auto
  also have  $\dots = (\alpha \, (n + 2) * h \, (n + 1) + h \, n) / (\alpha \, (n + 2) * k \, (n + 1) + k \, n)$ 
    using wf  $\alpha\text{-pos}$  by (subst  $\text{conv}'\text{-num-denom}$ ) (simp-all add:  $h\text{-def } k\text{-def}$ )
  also have  $\alpha \, (n + 2) = \text{surd-to-real } D \, (\alpha' \, (\text{Suc } n))$ 
    using  $\text{surd-to-real-sqrt-remainder-surd}[OF \, \text{nonsquare}, of \, \text{Suc } n]$ 
    by (simp add:  $\alpha'\text{-def } \alpha\text{-def } c\text{-def}$ )
  also have  $\dots = (\text{sqrt } D + P \, (\text{Suc } n)) / Q \, (\text{Suc } n)$  (is - =  $?A$ )
    by (simp add:  $\alpha'\text{-surd-to-real-def}$ )
  also have  $?A * h \, (n + 1) + h \, n =$ 
     $1 / Q \, (n + 1) * ((\text{sqrt } D + P \, (n + 1)) * h \, (n + 1) + Q \, (n + 1) * h \, n)$ 
    using  $Q\text{-pos}$  by (simp add:  $\text{field-simps}$ )
  also have  $?A * k \, (n + 1) + k \, n =$ 
     $1 / Q \, (n + 1) * ((\text{sqrt } D + P \, (n + 1)) * k \, (n + 1) + Q \, (n + 1) * k \, n)$ 
    (is - =  $?f \, k$ ) using  $Q\text{-pos}$  by (simp add:  $\text{field-simps}$ )
  also have  $?f \, h / ?f \, k = ((\text{sqrt } D + P \, (n + 1)) * h \, (n + 1) + Q \, (n + 1) * h \, n) /$ 
     $((\text{sqrt } D + P \, (n + 1)) * k \, (n + 1) + Q \, (n + 1) * k \, n)$ 
    (is - =  $?A / ?B$ ) using  $Q\text{-pos}$  by (intro  $\text{mult-divide-mult-cancel-left}$ ) auto
  finally have  $\text{sqrt } D * ?B = ?A$ 
    using  $\langle ?B > 0 \rangle$  by (simp add:  $\text{divide-simps}$ )
  moreover have  $\text{sqrt } D * \text{sqrt } D = D$  by simp

```

ultimately have $\text{sqrt } D * (P (n + 1) * k (n + 1) + Q (n + 1) * k n - h (n + 1)) =$
 $P (n + 1) * h (n + 1) + Q (n + 1) * h n - k (n + 1) * D$
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: $h (Suc n) = P (Suc n) * k (Suc n) + Q (Suc n) * k n$
and 2: $D * k (Suc n) = P (Suc n) * h (Suc n) + Q (Suc n) * h n$
by (*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 $h m ^ 2 - D * k m ^ 2 = (-1) ^ {Suc m} * Q m$
by (*simp add: m*)
qed

Every non-trivial solution to Pell's equation is a convergent in the expansion of \sqrt{D} :

theorem *pell-solution-is-conv*:
assumes $x^2 = Suc (D * y^2)$ **and** $y > 0$
shows $(int x, int y) \in \text{range } (\lambda n. (conv-num c n, conv-denom c n))$
proof –
have $\exists n. enat n \leq \text{cfraction-length } c \wedge (int x, int y) = (conv-num c n, conv-denom c n)$
proof (*rule frac-is-convergentI*)
have $\text{gcd } (x^2) (y^2) = 1$ **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*)
next
from *assms* **have** $D > 1$
using *nonsquare* **by** (*cases D*) (*auto intro!: Nat.gr0I*)
hence *pos*: $x + y * \text{sqrt } D > 0$ **using** *assms*
by (*intro add-nonneg-pos*) *auto*

from *assms* **have** $\text{real } (x^2) = \text{real } (Suc (D * y^2))$
by (*simp only: of-nat-eq-iff*)
hence $1 = \text{real } x ^ 2 - D * \text{real } y ^ 2$
unfolding *of-nat-power* **by** *simp*
also have $\dots = (x - y * \text{sqrt } D) * (x + y * \text{sqrt } D)$
by (*simp add: field-simps power2-eq-square*)
finally have $*$: $x - y * \text{sqrt } D = 1 / (x + y * \text{sqrt } D)$
using *pos* **by** (*simp add: field-simps*)

```

from pos have  $0 < 1 / (x + y * \text{sqrt } D)$ 
  by (intro divide-pos-pos) auto
also have  $\dots = x - y * \text{sqrt } D$  by (rule * [symmetric])
finally have less:  $y * \text{sqrt } D < x$  by simp

have  $\text{sqrt } D - x / y = -((x - y * \text{sqrt } D) / y)$ 
  using  $\langle y > 0 \rangle$  by (simp add: field-simps)
also have  $|\dots| = (x - y * \text{sqrt } D) / y$ 
  using less by simp
also have  $(x - y * \text{sqrt } D) / y = 1 / (y * (x + y * \text{sqrt } D))$ 
  using  $\langle y > 0 \rangle$  by (subst *) auto
also have  $\dots \leq 1 / (y * (y * \text{sqrt } D + y * \text{sqrt } D))$ 
  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  $\dots = 1 / (2 * y^2 * \text{sqrt } D)$ 
  by (simp add: power2-eq-square)
also have  $\dots < 1 / (\text{real } (2 * y^2) * 1)$  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  $|\text{cfrac-lim } c - \text{int } x / \text{int } y| < 1 / (2 * \text{int } y ^ 2)$ 
  unfolding c-def using irrat-sqrt-nonsquare[of D]  $\langle \neg \text{is-square } D \rangle$  by simp
qed (insert assms irrat-sqrt-nonsquare[of D], auto simp: c-def)
thus ?thesis by auto
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:

```

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

```

```

have  $(h (l * m - 1))^2 - D * (k (l * m - 1))^2 =$ 
   $(-1) ^ \text{Suc } (l * m - 1) * \text{int } (Q (l * m - 1))$ 
  using lm' by (intro conv-num-denom-pell) auto
also have  $(-1) ^ \text{Suc } (l * m - 1) = (1 :: \text{int})$ 
  using lm l m by (subst neg-one-even-power) auto
also have  $Q (l * m - 1) = Q ((l * m - 1) \bmod l)$ 
  unfolding Q-def l-def o-def by (subst sqrt-remainder-surd-periodic[OF non-square]) 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 ... = l - 1
      using l by (intro mod-less) auto
    also have Q ... = 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) ^ 2 = D * k (l * m - 1) ^ 2 + 1
    unfolding of-nat-Suc by (simp add: algebra-simps)
  hence h (l * m - 1) ^ 2 = D * k (l * m - 1) ^ 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
next
  assume nontriv-solution (x, y)
  hence asm: x ^ 2 = Suc (D * y ^ 2) y > 0
    by (auto simp: nontriv-solution-def abs-square-eq-1 intro!: Nat.gr0I)
  from asm have asm': int x ^ 2 = int D * int y ^ 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 > 0
    using asm' conv-num-denom-pell(1) xy by (intro Nat.gr0I) auto
  have 1 = h m ^ 2 - D * k m ^ 2
    using asm' xy by simp
  also have ... = (- 1) ^ Suc m * int (Q m)
    using conv-num-denom-pell(2)[OF m] .
  finally have *: (- 1) ^ Suc m * int (Q m) = 1 ..
  from * have m': odd m ∧ 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 *: ¬(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 non-
square]) simp
    also {
      have m mod l < l using ⟨l > 0⟩ by simp
      moreover have Suc (m mod l) ≠ l using * l ⟨m > 0⟩
        using mod-Suc[of m l] by auto
    }

```

```

    ultimately have  $m \bmod l < l - 1$  by simp
    hence  $Q (m \bmod l) > 1$  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:  $Suc\ m = n * l \wedge m = n * l - 1$ 
by (simp-all add:  $n$ -def)
hence  $n > 0$  by (auto intro!: Nat.gr0I)
thus  $\exists n > 0. \text{int } x = h (l * n - 1) \wedge \text{int } y = k (l * n - 1) \wedge \text{even } (l * n)$ 
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  $n \equiv \text{if even } l \text{ then } l - 1 \text{ else } 2 * l - 1$ 
  shows fund-sol = (nat (h  $n$ ), nat (k  $n$ ))
proof (rule fund-sol-eq-sndI)
  have [simp]:  $h\ n \geq 0 \wedge k\ n \geq 0$  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  $x\ y :: \text{nat}$  assume nontriv-solution ( $x, y$ )
  then obtain  $m$  where  $m > 0 \wedge x = h (l * m - 1) \wedge y = k (l * m - 1) \wedge \text{even } (l * m)$ 
  by (subst (asm) nontriv-solution-iff-conv-num-denom) auto
  have  $l > 0$  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  $n \leq l * m - 1$  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 info

```

```

in if l = 0 then (0, 0) else
  let
    c = cfrac-sqrt-nth info;
    n = if even l then l - 1 else 2 * l - 1
  in
    (nat (conv-num-fun c n), nat (conv-denom-fun c n)))
proof -
  have *: is-cfrac (cfrac-sqrt-nth (sqrt-cfrac-info-array D)) if ¬is-square D
  using that cfrac-sqrt-nth[of D] unfolding is-cfrac-def
  by (metis cfrac-nth-nonzero neg0-conv of-nat-0 of-nat-0-less-iff)
  have **: cfrac (λx. int (cfrac-sqrt-nth (sqrt-cfrac-info-array D) x)) = cfrac-of-real
  (sqrt D)
  if ¬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 ⇒ find-nth-solution D n =
(0, 0)
by (simp add: find-nth-solution-def)

```

```

lemma fst-find-fund-sol-eq-0-iff [simp]: fst (find-fund-sol D) = 0 ⟷ 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)
qed (auto simp: find-fund-sol-def square-test-correct)

```

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

```

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

```

```

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

proof (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 (λx. 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

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

proof (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 (λx. 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

end

```

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

```

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

```

```

code-lazy-type stream

```

```

lemma lnth-code [code]:
  lnth xs 0 = (if lnull xs then undefined (0 :: nat) else lhd xs)
  lnth xs (Suc n) = (if lnull xs then undefined (Suc n) else lnth (ltl xs) n)
  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) =
  foldl (λ(x, y) x'. let y' = f x' in if y' > y then (x', y') else (x, y)) (x, f x) xs

```

```

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

```

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

```

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

```


7.1 Fundamental solutions of Pell's equation

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

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

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

The same for $D < 100000$:

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

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

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

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

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

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

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

7.2 Tests for other operations

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

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

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

end

8 Computing continued fraction expansions through interval arithmetic

theory *Continued-Fraction-Approximation*

```

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

```

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

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

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

```

definition float-to-rat :: float  $\Rightarrow$  int  $\times$  int where
  float-to-rat f = (if exponent f  $\geq$  0 then
    (mantissa f * 2  $^$  nat (exponent f), 1) else (mantissa f, 2  $^$  nat (-exponent f)))

```

```

lemma float-to-rat: fst (float-to-rat f) / snd (float-to-rat f) = real-of-float f
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  $\vee$  nu = 0  $\vee$  dl = 0  $\vee$  du = 0 then []
     else let l = nl div dl; u = nu div du
        in if l  $\neq$  u then []
           else l  $\#$  (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  $x \in \{\text{fst } l / \text{snd } l, \text{fst } u / \text{snd } u\}$  and  $\text{snd } l > 0$  and  $\text{snd } u > 0$ 
  assumes  $i < \text{length } (\text{cfrac-from-approx } l \ u)$ 
  shows  $\text{cfrac-nth } (\text{cfrac-of-real } x) \ i = \text{cfrac-from-approx } l \ u \ ! \ i$ 
  using assms
proof (induction  $l \ u$  arbitrary:  $i \ x$  rule: cfrac-from-approx.induct)
  case (1  $nl \ dl \ nu \ du \ i \ x$ )
  from 1.prem1 have *:  $nl \ \text{div} \ dl = nu \ \text{div} \ du \ \wedge \ nl \neq 0 \ \wedge \ nu \neq 0 \ \wedge \ dl > 0 \ \wedge \ du > 0$ 
  by (auto simp: cfrac-from-approx.simps Let-def split: if-splits)
  have  $\lfloor nl / dl \rfloor \leq \lfloor x \rfloor \ \wedge \ \lfloor x \rfloor \leq \lfloor nu / du \rfloor$ 
  using 1.prem1 by (intro floor-mono; simp)
  hence  $nl \ \text{div} \ dl \leq \lfloor x \rfloor \ \wedge \ \lfloor x \rfloor \leq nu \ \text{div} \ du$ 
  by (simp-all add: floor-divide-of-int-eq)
  with * have  $\lfloor x \rfloor = nu \ \text{div} \ du$ 
  by linarith

  show ?case
proof (cases  $i$ )
  case 0
  with 0 and  $\langle \lfloor x \rfloor = \rangle$  show ?thesis using 1.prem1
  by (auto simp: Let-def cfrac-from-approx.simps)
next
  case [simp]: (Suc i)
  from 1.prem1 * have  $nl \ \text{mod} \ dl \neq 0$ 
  by (subst (asm) cfrac-from-approx.simps) (auto split: if-splits)
  have frac-eq:  $\text{frac } x = x - nu \ \text{div} \ du$ 
  using  $\langle \lfloor x \rfloor = \rangle$  by (simp add: frac-def)

  have  $\text{frac } x \geq nl / dl - nl \ \text{div} \ dl$ 
  using * 1.prem1 by (simp add: frac-eq)
  also have  $nl / dl - nl \ \text{div} \ dl = (nl - dl * (nl \ \text{div} \ dl)) / dl$ 
  using * by (simp add: field-simps)
  also have  $nl - dl * (nl \ \text{div} \ dl) = nl \ \text{mod} \ dl$ 
  by (subst minus-div-mult-eq-mod [symmetric]) auto
  finally have  $\text{frac } x \geq (nl \ \text{mod} \ dl) / dl$  .

  have  $nl \ \text{mod} \ dl \geq 0$ 
  using * by (intro pos-mod-sign) auto
  with  $\langle nl \ \text{mod} \ dl \neq 0 \rangle$  have  $nl \ \text{mod} \ dl > 0$ 
  by linarith
  hence  $0 < (nl \ \text{mod} \ dl) / dl$ 
  using * by (intro divide-pos-pos) auto
  also have  $\dots \leq \text{frac } x$ 
  by fact
  finally have  $\text{frac } x > 0$  .

  have  $\text{frac } x \leq nu / du - nu \ \text{div} \ du$ 
  using * 1.prem1 by (simp add: frac-eq)
  also have  $\dots = (nu - du * (nu \ \text{div} \ du)) / du$ 

```

```

    using * by (simp add: field-simps)
  also have  $nu - du * (nu \text{ div } du) = nu \text{ mod } du$ 
    by (subst minus-div-mult-eq-mod [symmetric]) auto
  finally have  $\text{frac } x \leq \text{real-of-int } (nu \text{ mod } du) / \text{real-of-int } du$  .

  have  $0 < \text{frac } x$ 
    by fact
  also have  $\dots \leq (nu \text{ mod } du) / du$ 
    by fact
  finally have  $nu \text{ mod } du > 0$ 
    using * by (auto simp: field-simps)

  have  $\text{cfrac-nth } (\text{cfrac-of-real } x) \ i = \text{cfrac-nth } (\text{cfrac-tl } (\text{cfrac-of-real } x)) \ i'$ 
    by simp
  also have  $\text{cfrac-tl } (\text{cfrac-of-real } x) = \text{cfrac-of-real } (1 / \text{frac } x)$ 
    using  $\langle \text{frac } x > 0 \rangle$  by (intro cfrac-tl-of-real) auto
  also have  $\text{cfrac-nth } (\text{cfrac-of-real } (1 / \text{frac } x)) \ i' =$ 
     $\text{cfrac-from-approx } (du, nu \text{ mod } du) \ (dl, nl \text{ mod } dl) \ ! \ i'$ 
  proof (rule 1.IH[OF - refl refl - refl])
    show  $\neg (nl = 0 \vee nu = 0 \vee dl = 0 \vee du = 0) \rightarrow nl \text{ div } dl \neq nu \text{ div } du$ 
      using 1.prem by (auto split: if-splits simp: Let-def cfrac-from-approx.simps)
    next
      show  $i' < \text{length } (\text{cfrac-from-approx } (du, nu \text{ mod } du) \ (dl, nl \text{ mod } dl))$  using
        1.prem by
        by (subst (asm) cfrac-from-approx.simps) (auto split: if-splits simp: Let-def)
    next
      have  $1 / \text{frac } x \leq dl / (nl \text{ mod } dl)$ 
        using  $\langle \text{frac } x > 0 \rangle$  and  $\langle nl \text{ mod } dl > 0 \rangle$  and  $\langle \text{frac } x \geq (nu \text{ mod } du) / du \rangle$ 
      and *
        by (auto simp: field-simps)
      moreover have  $1 / \text{frac } x \geq du / (nu \text{ mod } du)$ 
        using  $\langle \text{frac } x > 0 \rangle$  and  $\langle nu \text{ mod } du > 0 \rangle$  and  $\langle \text{frac } x \leq (nu \text{ mod } du) / du \rangle$ 
      and *
        by (auto simp: field-simps)
      ultimately show
         $1 / \text{frac } x \in \{\text{real-of-int } (\text{fst } (du, nu \text{ mod } du)) / \text{real-of-int } (\text{snd } (du, nu \text{ mod } du))..$ 
           $\text{real-of-int } (\text{fst } (dl, nl \text{ mod } dl)) / \text{real-of-int } (\text{snd } (dl, nl \text{ mod } dl))\}$ 
        by simp
      show  $\text{snd } (du, nu \text{ mod } du) > 0 \wedge \text{snd } (dl, nl \text{ mod } dl) > 0$  and  $nl \text{ mod } dl \neq 0$ 
        using  $\langle nu \text{ mod } du > 0 \rangle$  and  $\langle nl \text{ mod } dl > 0 \rangle$  by simp-all
    qed
  also have  $\text{cfrac-from-approx } (du, nu \text{ mod } du) \ (dl, nl \text{ mod } dl) \ ! \ i' =$ 
     $\text{cfrac-from-approx } (nl, dl) \ (nu, du) \ ! \ i$ 
    using 1.prem *  $\langle nl \text{ mod } dl \neq 0 \rangle$  by (subst (2) cfrac-from-approx.simps) auto
  finally 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  $x \in \{\text{real-of-float } l.. \text{real-of-float } u\}$ 
  assumes  $i < \text{length } (\text{cfrac-from-approx}' l u)$ 
  shows  $\text{cfrac-nth } (\text{cfrac-of-real } x) i = \text{cfrac-from-approx}' l u ! i$ 
  using 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 [] of
      None  $\Rightarrow$  []
    | 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 ( $\lambda n. (\text{conv-num-fun } (!) c) n, \text{conv-denom-fun } (!) c) n$ ) [0.. $\text{length } c$ ]

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

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

end

```

References

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