

Computing N-th Roots using the Babylonian Method*

René Thiemann

May 26, 2024

Abstract

We implement the Babylonian method [1] to compute n-th roots of numbers. We provide precise algorithms for naturals, integers and rationals, and offer an approximation algorithm for square roots within linear ordered fields. Moreover, there are precise algorithms to compute the floor and the ceiling of n-th roots.

Contents

1	Auxiliary lemmas which might be moved into the Isabelle distribution.	2
2	A Fast Logarithm Algorithm	4
3	Executable algorithms for p-th roots	8
3.1	Logarithm	8
3.2	Computing the p -th root of an integer number	9
3.3	Floor and ceiling of roots	20
3.4	Downgrading algorithms to the naturals	23
3.5	Upgrading algorithms to the rationals	25
4	Executable algorithms for square roots	28
4.1	The Babylonian method	28
4.2	The Babylonian method using integer division	28
4.3	Square roots for the naturals	31
4.4	Square roots for the rationals	31
4.5	Approximating square roots	33
4.6	Some tests	36

*This research is supported by FWF (Austrian Science Fund) project P22767-N13.

1 Auxiliary lemmas which might be moved into the Isabelle distribution.

```

theory Sqrt-Babylonian-Auxiliary
imports
  Complex-Main
begin

lemma mod-div-equality-int:  $(n :: int) \text{ div } x * x = n - n \text{ mod } x$ 
  using div-mult-mod-eq[of  $n \ x$ ] by arith

lemma div-is-floor-divide-rat:  $n \text{ div } y = \lfloor \text{rat-of-int } n / \text{rat-of-int } y \rfloor$ 
  unfolding Fract-of-int-quotient[symmetric] floor-Fract by simp

lemma div-is-floor-divide-real:  $n \text{ div } y = \lfloor \text{real-of-int } n / \text{of-int } y \rfloor$ 
  unfolding div-is-floor-divide-rat[of  $n \ y$ ]
  by (metis Ratreal-def of-rat-divide of-rat-of-int-eq real-floor-code)

lemma floor-div-pos-int:
  fixes  $r :: 'a :: \text{floor-ceiling}$ 
  assumes  $n: n > 0$ 
  shows  $\lfloor r / \text{of-int } n \rfloor = \lfloor r \rfloor \text{ div } n$  (is  $?l = ?r$ )
proof -
  let  $?of-int = \text{of-int} :: int \Rightarrow 'a$ 
  define  $rhs$  where  $rhs = \lfloor r \rfloor \text{ div } n$ 
  let  $?n = ?of-int \ n$ 
  define  $m$  where  $m = \lfloor r \rfloor \text{ mod } n$ 
  let  $?m = ?of-int \ m$ 
  from div-mult-mod-eq[of  $\text{floor } r \ n$ ] have  $dm: rhs * n + m = \lfloor r \rfloor$  unfolding
rhs-def m-def by simp
  have  $mn: m < n$  and  $m0: m \geq 0$  using  $n$  m-def by auto
  define  $e$  where  $e = r - ?of-int \ \lfloor r \rfloor$ 
  have  $e0: e \geq 0$  unfolding e-def
  by (metis diff-self eq-iff floor-diff-of-int zero-le-floor)
  have  $e1: e < 1$  unfolding e-def
  by (metis diff-self dual-order.refl floor-diff-of-int floor-le-zero)
  have  $r = ?of-int \ \lfloor r \rfloor + e$  unfolding e-def by simp
  also have  $\lfloor r \rfloor = rhs * n + m$  using  $dm$  by simp
  finally have  $r = ?of-int \ (rhs * n + m) + e$  .
  hence  $r / ?n = ?of-int \ (rhs * n) / ?n + (e + ?m) / ?n$  using  $n$  by (simp add:
field-simps)
  also have  $?of-int \ (rhs * n) / ?n = ?of-int \ rhs$  using  $n$  by auto
  finally have  $*$ :  $r / ?of-int \ n = (e + ?of-int \ m) / ?of-int \ n + ?of-int \ rhs$  by
simp
  have  $?l = rhs + \text{floor} \ ((e + ?m) / ?n)$  unfolding  $*$  by simp
  also have  $\text{floor} \ ((e + ?m) / ?n) = 0$ 
  proof (rule floor-unique)
  show  $?of-int \ 0 \leq (e + ?m) / ?n$  using  $e0 \ m0 \ n$ 
  by (metis add-increasing2 divide-nonneg-pos of-int-0 of-int-0-le-iff of-int-0-less-iff)

```

```

show  $(e + ?m) / ?n < ?of-int\ 0 + 1$ 
proof (rule ccontr)
  from  $n$  have  $n'$ :  $?n > 0$   $?n \geq 0$  by simp-all
  assume  $\neg ?thesis$ 
  hence  $(e + ?m) / ?n \geq 1$  by auto
  from mult-right-mono[OF this n'(2)]
  have  $?n \leq e + ?m$  using  $n'(1)$  by simp
  also have  $?m \leq ?n - 1$  using  $mn$ 
    by (metis of-int-1 of-int-diff of-int-le-iff zle-diff1-eq)
  finally have  $?n \leq e + ?n - 1$  by auto
  with  $e1$  show False by arith
qed
qed
finally show  $?thesis$  unfolding rhs-def by simp
qed

```

```

lemma floor-div-neg-int:
  fixes  $r :: 'a :: floor-ceiling$ 
  assumes  $n: n < 0$ 
  shows  $\lfloor r / of-int\ n \rfloor = \lceil r \rceil\ div\ n$ 
proof -
  from  $n$  have  $n'$ :  $-n > 0$  by auto
  have  $\lfloor r / of-int\ n \rfloor = \lfloor -r / of-int\ (-n) \rfloor$  using  $n$ 
    by (metis floor-of-int floor-zero less-int-code(1) minus-divide-left minus-minus
nonzero-minus-divide-right of-int-minus)
  also have  $\dots = \lfloor -r \rfloor\ div\ (-n)$  by (rule floor-div-pos-int[OF n'])
  also have  $\dots = \lceil r \rceil\ div\ n$  using  $n$ 
    by (metis ceiling-def div-minus-right)
  finally show  $?thesis$  .
qed

```

```

lemma divide-less-floor1:  $n / y < of-int\ (floor\ (n / y)) + 1$ 
  by (metis floor-less-iff less-add-one of-int-1 of-int-add)

```

```

context linordered-idom
begin

```

```

lemma sgn-int-pow-if [simp]:
   $sgn\ x \wedge^p = (if\ even\ p\ then\ 1\ else\ sgn\ x)$  if  $x \neq 0$ 
  using that by (induct p) simp-all

```

```

lemma compare-pow-le-iff:  $p > 0 \implies (x :: 'a) \geq 0 \implies y \geq 0 \implies (x \wedge^p \leq y \wedge^p) = (x \leq y)$ 
  by (rule power-mono-iff)

```

```

lemma compare-pow-less-iff:  $p > 0 \implies (x :: 'a) \geq 0 \implies y \geq 0 \implies (x \wedge^p < y \wedge^p) = (x < y)$ 
  using compare-pow-le-iff [of p x y]

```

```

using local.dual-order.order-iff-strict local.power-strict-mono by blast

end

lemma quotient-of-int[simp]: quotient-of (of-int i) = (i,1)
  by (metis Rat.of-int-def quotient-of-int)

lemma quotient-of-nat[simp]: quotient-of (of-nat i) = (int i,1)
  by (metis Rat.of-int-def Rat.quotient-of-int of-int-of-nat-eq)

lemma square-lesseq-square:  $\bigwedge x y. 0 \leq (x :: 'a :: \text{linordered-field}) \implies 0 \leq y \implies (x * x \leq y * y) = (x \leq y)$ 
  by (metis mult-mono mult-strict-mono' not-less)

lemma square-less-square:  $\bigwedge x y. 0 \leq (x :: 'a :: \text{linordered-field}) \implies 0 \leq y \implies (x * x < y * y) = (x < y)$ 
  by (metis mult-mono mult-strict-mono' not-less)

lemma sqrt-sqrt[simp]:  $x \geq 0 \implies \text{sqrt } x * \text{sqrt } x = x$ 
  by (metis real-sqrt-pow2 power2-eq-square)

lemma abs-lesseq-square:  $\text{abs } (x :: \text{real}) \leq \text{abs } y \longleftrightarrow x * x \leq y * y$ 
  using square-lesseq-square[of abs x abs y] by auto

end

```

2 A Fast Logarithm Algorithm

```

theory Log-Impl
imports
  Sqrt-Babylonian-Auxiliary
begin

```

We implement the discrete logarithm function in a manner similar to a repeated squaring exponentiation algorithm.

In order to prove termination of the algorithm without intermediate checks we need to ensure that we only use proper bases, i.e., values of at least 2. This will be encoded into a separate type.

```

typedef proper-base = {x :: int. x  $\geq$  2} by auto

setup-lifting type-definition-proper-base

lift-definition get-base :: proper-base  $\Rightarrow$  int is  $\lambda x. x$  .

lift-definition square-base :: proper-base  $\Rightarrow$  proper-base is  $\lambda x. x * x$ 
proof –
  fix i :: int
  assume i: 2  $\leq$  i

```

```

have 2 * 2 ≤ i * i
  by (rule mult-mono[OF i i], insert i, auto)
thus 2 ≤ i * i by auto
qed

```

lift-definition into-base :: int ⇒ proper-base is λ x. if x ≥ 2 then x else 2 **by** auto

```

lemma square-base: get-base (square-base b) = get-base b * get-base b
  by (transfer, auto)

```

```

lemma get-base-2: get-base b ≥ 2
  by (transfer, auto)

```

```

lemma b-less-square-base-b: get-base b < get-base (square-base b)
  unfolding square-base using get-base-2[of b] by simp

```

```

lemma b-less-div-base-b: assumes xb: ¬ x < get-base b
  shows x div get-base b < x
proof –
  from get-base-2[of b] have b: get-base b ≥ 2 .
  with xb have x2: x ≥ 2 by auto
  with b int-div-less-self[of x (get-base b)]
  show ?thesis by auto
qed

```

We now state the main algorithm.

```

function log-main :: proper-base ⇒ int ⇒ nat × int where
  log-main b x = (if x < get-base b then (0,1) else
    case log-main (square-base b) x of
      (z, bz) ⇒
        let l = 2 * z; bz1 = bz * get-base b
        in if x < bz1 then (l,bz) else (Suc l,bz1))
  by pat-completeness auto

```

```

termination by (relation measure (λ (b,x). nat (1 + x - get-base b)),
  insert b-less-square-base-b, auto)

```

```

lemma log-main: x > 0 ⇒ log-main b x = (y,by) ⇒ by = (get-base b) ^ y ∧
  (get-base b) ^ y ≤ x ∧ x < (get-base b) ^ (Suc y)

```

```

proof (induct b x arbitrary: y by rule: log-main.induct)
  case (1 b x y by)
  note x = 1(2)
  note y = 1(3)
  note IH = 1(1)
  let ?b = get-base b
  show ?case
  proof (cases x < ?b)
    case True
    with x y show ?thesis by auto

```

```

next
  case False
  obtain z bz where zz: log-main (square-base b) x = (z,bz)
    by (cases log-main (square-base b) x, auto)
  have id: get-base (square-base b) ^k = ?b ^ (2 * k) for k unfolding square-base
    by (simp add: power-mult semiring-normalization-rules(29))
  from IH[OF False x zz, unfolded id]
  have z: ?b ^ (2 * z) ≤ x x < ?b ^ (2 * Suc z) and bz: bz = get-base b ^ (2 * z) by auto
  from y[unfolded log-main.simps[of b x] Let-def zz split] bz False
  have yy: (if x < bz * ?b then (2 * z, bz) else (Suc (2 * z), bz * ?b)) =
    (y, by) by auto
  show ?thesis
  proof (cases x < bz * ?b)
    case True
    with yy have yz: y = 2 * z by = bz by auto
    from True z(1) bz show ?thesis unfolding yz by (auto simp: ac-simps)
  next
  case False
  with yy have yz: y = Suc (2 * z) by = ?b * bz by auto
  from False have ?b ^ Suc (2 * z) ≤ x by (auto simp: bz ac-simps)
  with z(2) bz show ?thesis unfolding yz by auto
qed
qed
qed

```

We then derive the floor- and ceiling-log functions.

```

definition log-floor :: int ⇒ int ⇒ nat where
  log-floor b x = fst (log-main (into-base b) x)

```

```

definition log-ceiling :: int ⇒ int ⇒ nat where
  log-ceiling b x = (case log-main (into-base b) x of
    (y,by) ⇒ if x = by then y else Suc y)

```

```

lemma log-floor-sound: assumes b > 1 x > 0 log-floor b x = y
  shows b ^ y ≤ x x < b ^ (Suc y)

```

```

proof –
  from assms(1,3) have id: get-base (into-base b) = b by transfer auto
  obtain yy bb where log: log-main (into-base b) x = (yy,bb)
    by (cases log-main (into-base b) x, auto)
  from log-main[OF assms(2) log] assms(3)[unfolded log-floor-def log] id
  show b ^ y ≤ x x < b ^ (Suc y) by auto
qed

```

```

lemma log-ceiling-sound: assumes b > 1 x > 0 log-ceiling b x = y
  shows x ≤ b ^ y y ≠ 0 ⇒ b ^ (y - 1) < x

```

```

proof –
  from assms(1,3) have id: get-base (into-base b) = b by transfer auto
  obtain yy bb where log: log-main (into-base b) x = (yy,bb)

```

by (cases log-main (into-base b) x, auto)
 from log-main[OF assms(2) log, unfolded id] assms(3)[unfolded log-ceiling-def
 log split]
 have bnd: $b^{\wedge} yy \leq x < b^{\wedge} \text{Suc } yy$ and
 y: $y = (\text{if } x = b^{\wedge} yy \text{ then } yy \text{ else } \text{Suc } yy)$ by auto
 have $x \leq b^{\wedge} y \wedge (y \neq 0 \longrightarrow b^{\wedge}(y - 1) < x)$
 proof (cases $x = b^{\wedge} yy$)
 case True
 with y bnd assms(1) show ?thesis by (cases yy, auto)
 next
 case False
 with y bnd show ?thesis by auto
 qed
 thus $x \leq b^{\wedge} y \wedge y \neq 0 \implies b^{\wedge}(y - 1) < x$ by auto
 qed

Finally, we connect it to the *log* function working on real numbers.

lemma log-floor[simp]: **assumes** $b > 1$ **and** $x > 0$
shows $\text{log-floor } b \ x = \lfloor \log b \ x \rfloor$
proof –
obtain y **where** $y: \text{log-floor } b \ x = y$ **by** auto
note $\text{main} = \text{log-floor-sound}$ [OF assms y]
from $b \ x$ **have** *: $1 < \text{real-of-int } b \ 0 < \text{real-of-int } (b^{\wedge} y) \ 0 < \text{real-of-int } x$
and **: $1 < \text{real-of-int } b \ 0 < \text{real-of-int } x \ 0 < \text{real-of-int } (b^{\wedge} \text{Suc } y)$
by auto
show ?thesis **unfolding** y
proof (rule sym, rule floor-unique)
show $\text{real-of-int } (int \ y) \leq \log (\text{real-of-int } b) (\text{real-of-int } x)$
using main(1)[folded log-le-cancel-iff[OF *, unfolded of-int-le-iff]]
using log-pow-cancel[of b y] b **by** auto
show $\log (\text{real-of-int } b) (\text{real-of-int } x) < \text{real-of-int } (int \ y) + 1$
using main(2)[folded log-less-cancel-iff[OF **, unfolded of-int-less-iff]]
using log-pow-cancel[of b Suc y] b **by** auto
 qed
 qed

lemma log-ceiling[simp]: **assumes** $b > 1$ **and** $x > 0$
shows $\text{log-ceiling } b \ x = \lceil \log b \ x \rceil$
proof –
obtain y **where** $y: \text{log-ceiling } b \ x = y$ **by** auto
note $\text{main} = \text{log-ceiling-sound}$ [OF assms y]
from $b \ x$ **have** *: $1 < \text{real-of-int } b \ 0 < \text{real-of-int } (b^{\wedge}(y - 1)) \ 0 < \text{real-of-int } x$
and **: $1 < \text{real-of-int } b \ 0 < \text{real-of-int } x \ 0 < \text{real-of-int } (b^{\wedge} y)$
by auto
show ?thesis **unfolding** y
proof (rule sym, rule ceiling-unique)
show $\log (\text{real-of-int } b) (\text{real-of-int } x) \leq \text{real-of-int } (int \ y)$
using main(1)[folded log-le-cancel-iff[OF **, unfolded of-int-le-iff]]

```

    using log-pow-cancel[of b y] b by auto
  from x have x: x ≥ 1 by auto
  show real-of-int (int y) - 1 < log (real-of-int b) (real-of-int x)
  proof (cases y = 0)
    case False
    thus ?thesis
      using main(2)[folded log-less-cancel-iff[OF *, unfolded of-int-less-iff]]
      using log-pow-cancel[of b y - 1] b x by auto
  next
  case True
  have real-of-int (int y) - 1 = log b (1/b) using True b
    by (subst log-divide, auto)
  also have ... < log b 1
    by (subst log-less-cancel-iff, insert b, auto)
  also have ... ≤ log b x
    by (subst log-le-cancel-iff, insert b x, auto)
  finally show real-of-int (int y) - 1 < log (real-of-int b) (real-of-int x) .
qed
qed
qed

end

```

3 Executable algorithms for p -th roots

```

theory NthRoot-Impl
imports
  Log-Impl
  Cauchy.CauchysMeanTheorem
begin

```

We implemented algorithms to decide $\sqrt[p]{n} \in \mathbb{Q}$ and to compute $\lfloor \sqrt[p]{n} \rfloor$. To this end, we use a variant of Newton iteration which works with integer division instead of floating point or rational division. To get suitable starting values for the Newton iteration, we also implemented a function to approximate logarithms.

3.1 Logarithm

For computing the p -th root of a number n , we must choose a starting value in the iteration. Here, we use $(2::'a)^{\text{nat } \lceil \log 2 n \rceil / p}$.

We use a partial efficient algorithm, which does not terminate on corner-cases, like $b = 0$ or $p = 1$, and invoke it properly afterwards. Then there is a second algorithm which terminates on these corner-cases by additional guards and on which we can perform induction.

3.2 Computing the p -th root of an integer number

Using the logarithm, we can define an executable version of the intended starting value. Its main property is the inequality $x \leq (\text{start-value } x \ p)^p$, i.e., the start value is larger than the p -th root. This property is essential, since our algorithm will abort as soon as we fall below the p -th root.

definition *start-value* :: *int* \Rightarrow *nat* \Rightarrow *int* **where**

start-value $n \ p = 2 \wedge (\text{nat } \lceil \text{of-nat } (\text{log-ceiling } 2 \ n) / \text{rat-of-nat } p \rceil)$

lemma *start-value-main*: **assumes** $x: x \geq 0$ **and** $p: p > 0$

shows $x \leq (\text{start-value } x \ p) \wedge \text{start-value } x \ p \geq 0$

proof (*cases* $x = 0$)

case *True*

with p **show** *?thesis* **unfolding** *start-value-def* *True* **by** *simp*

next

case *False*

with x **have** $x: x > 0$ **by** *auto*

define $l2x$ **where** $l2x = \lceil \text{log } 2 \ x \rceil$

define pow **where** $pow = \text{nat } \lceil \text{rat-of-int } l2x / \text{of-nat } p \rceil$

have $\text{root } p \ x = x \ \text{powr } (1 / p)$ **by** (*rule* *root-powr-inverse*, *insert* $x \ p$, *auto*)

also have $\dots = (2 \ \text{powr } (\text{log } 2 \ x)) \ \text{powr } (1 / p)$ **using** *powr-log-cancel*[*of* $2 \ x$] x

by *auto*

also have $\dots = 2 \ \text{powr } (\text{log } 2 \ x * (1 / p))$ **by** (*rule* *powr-powr*)

also have $\text{log } 2 \ x * (1 / p) = \text{log } 2 \ x / p$ **using** p **by** *auto*

finally have $r: \text{root } p \ x = 2 \ \text{powr } (\text{log } 2 \ x / p)$.

have $lp: \text{log } 2 \ x \geq 0$ **using** x **by** *auto*

hence $l2pos: l2x \geq 0$ **by** (*auto* *simp*: *l2x-def*)

have $\text{log } 2 \ x / p \leq l2x / p$ **using** $x \ p$ **unfolding** *l2x-def*

by (*metis* *divide-right-mono* *le-of-int-ceiling* *of-nat-0-le-iff*)

also have $\dots \leq \lceil l2x / (p :: \text{real}) \rceil$ **by** (*simp* *add*: *ceiling-correct*)

also have $l2x / \text{real } p = l2x / \text{real-of-rat } (\text{of-nat } p)$

by (*metis* *of-rat-of-nat-eq*)

also have $\text{of-int } l2x = \text{real-of-rat } (\text{of-int } l2x)$

by (*metis* *of-rat-of-int-eq*)

also have $\text{real-of-rat } (\text{of-int } l2x) / \text{real-of-rat } (\text{of-nat } p) = \text{real-of-rat } (\text{rat-of-int } l2x / \text{of-nat } p)$

by (*metis* *of-rat-divide*)

also have $\lceil \text{real-of-rat } (\text{rat-of-int } l2x / \text{rat-of-nat } p) \rceil = \lceil \text{rat-of-int } l2x / \text{of-nat } p \rceil$

by *simp*

also have $\lceil \text{rat-of-int } l2x / \text{of-nat } p \rceil \leq \text{real } pow$ **unfolding** *pow-def* **by** *auto*

finally have $le: \text{log } 2 \ x / p \leq pow$.

from *powr-mono*[*OF* le , *of* 2 , *folded* r]

have $\text{root } p \ x \leq 2 \ \text{powr } pow$ **by** *auto*

also have $\dots = 2 \wedge pow$ **by** (*rule* *powr-realpow*, *auto*)

also have $\dots = \text{of-int } ((2 :: \text{int}) \wedge pow)$ **by** *simp*

also have $pow = (\text{nat } \lceil \text{of-int } (\text{log-ceiling } 2 \ x) / \text{rat-of-nat } p \rceil)$

unfolding *pow-def* *l2x-def* **using** x **by** *simp*

also have $\text{real-of-int } ((2 :: \text{int}) \wedge \dots) = \text{start-value } x \ p$ **unfolding** *start-value-def* **by** *simp*

```

finally have less:  $\text{root } p \ x \leq \text{start-value } x \ p$  .
have  $0 \leq \text{root } p \ x$  using  $p \ x$  by auto
also have  $\dots \leq \text{start-value } x \ p$  by (rule less)
finally have start:  $0 \leq \text{start-value } x \ p$  by simp
from power-mono[OF less, of p] have  $\text{root } p \ (\text{of-int } x) \wedge^p \leq \text{of-int } (\text{start-value } x \ p) \wedge^p$  using  $p \ x$  by auto
also have  $\dots = \text{start-value } x \ p \wedge^p$  by simp
also have  $\text{root } p \ (\text{of-int } x) \wedge^p = x$  using  $p \ x$  by force
finally have  $x \leq (\text{start-value } x \ p) \wedge^p$  by presburger
with start show ?thesis by auto
qed

```

```

lemma start-value: assumes  $x: x \geq 0$  and  $p: p > 0$  shows  $x \leq (\text{start-value } x \ p) \wedge^p$ 
start-value  $x \ p \geq 0$ 
using start-value-main[OF x p] by auto

```

We now define the Newton iteration to compute the p -th root. We are working on the integers, where every $(/)$ is replaced by (div) . We are proving several things within a locale which ensures that $p > 0$, and where $pm = p - 1$.

```

locale fixed-root =
  fixes  $p \ pm :: \text{nat}$ 
  assumes  $p: p = \text{Suc } pm$ 
begin

```

```

function root-newton-int-main ::  $\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \times \text{bool}$  where
  root-newton-int-main  $x \ n = (\text{if } (x < 0 \vee n < 0) \text{ then } (0, \text{False}) \text{ else } (\text{if } x \wedge^p \leq n \text{ then } (x, x \wedge^p = n) \text{ else } \text{root-newton-int-main } ((n \text{ div } (x \wedge^p \text{ pm}) + x * \text{int } pm) \text{ div } (\text{int } p)) \ n))$ 
  by pat-completeness auto
end

```

For the executable algorithm we omit the guard and use a let-construction

```

partial-function (tailrec) root-int-main' ::  $\text{nat} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \times \text{bool}$  where
  [code]: root-int-main'  $pm \ ipm \ ip \ x \ n = (\text{let } xpm = x \wedge^p \text{ pm}; xp = xpm * x \text{ in } \text{if } xp \leq n \text{ then } (x, xp = n) \text{ else } \text{root-int-main}' \ pm \ ipm \ ip \ ((n \text{ div } xpm + x * ipm) \text{ div } ip) \ n)$ 

```

In the following algorithm, we start the iteration. It will compute $\lfloor \text{root } p \ n \rfloor$ and a boolean to indicate whether the root is exact.

```

definition root-int-main ::  $\text{nat} \Rightarrow \text{int} \Rightarrow \text{int} \times \text{bool}$  where
  root-int-main  $p \ n \equiv \text{if } p = 0 \text{ then } (1, n = 1) \text{ else } \text{let } pm = p - 1 \text{ in } \text{root-int-main}' \ pm \ (\text{int } pm) \ (\text{int } p) \ (\text{start-value } n \ p) \ n$ 

```

Once we have proven soundness of *fixed-root.root-newton-int-main* and equivalence to *root-int-main*, it is easy to assemble the following algorithm which computes all roots for arbitrary integers.

definition *root-int* :: nat \Rightarrow int \Rightarrow int list **where**
root-int p x \equiv if p = 0 then [] else
 if x = 0 then [0] else
 let e = even p; s = sgn x; x' = abs x
 in if x < 0 \wedge e then [] else case *root-int-main* p x' of (y, True) \Rightarrow if e then
 [y, -y] else [s * y] | - \Rightarrow []

We start with proving termination of *fixed-root.root-newton-int-main*.

context *fixed-root*

begin

lemma *iteration-mono-eq*: **assumes** xn: x \wedge p = (n :: int)

shows (n div x \wedge pm + x * int pm) div int p = x

proof -

have [simp]: \wedge n. (x + x * n) = x * (1 + n) **by** (auto simp: field-simps)

show ?thesis **unfolding** xn[symmetric] p **by** simp

qed

lemma p0: p \neq 0 **unfolding** p **by** auto

The following property is the essential property for proving termination of *root-newton-int-main*.

lemma *iteration-mono-less*: **assumes** x: x \geq 0

and n: n \geq 0

and xn: x \wedge p > (n :: int)

shows (n div x \wedge pm + x * int pm) div int p < x

proof -

let ?sx = (n div x \wedge pm + x * int pm) div int p

from xn **have** xn-le: x \wedge p \geq n **by** auto

from xn x n **have** x0: x > 0

using not-le p **by** fastforce

from p **have** xp: x \wedge p = x * x \wedge pm **by** auto

from x n **have** n div x \wedge pm * x \wedge pm \leq n

by (auto simp add: minus-mod-eq-div-mult [symmetric] mod-int-pos-iff not-less power-le-zero-eq)

also **have** ... \leq x \wedge p **using** xn **by** auto

finally **have** le: n div x \wedge pm \leq x **using** x x0 **unfolding** xp **by** simp

have ?sx \leq (x \wedge p div x \wedge pm + x * int pm) div int p

by (rule zdiv-mono1, insert le p0, unfold xp, auto)

also **have** x \wedge p div x \wedge pm = x **unfolding** xp **by** auto

also **have** x + x * int pm = x * int p **unfolding** p **by** (auto simp: field-simps)

also **have** x * int p div int p = x **using** p **by** force

finally **have** le: ?sx \leq x .

{

assume ?sx = x

from arg-cong[OF this, of λ x. x * int p]

have x * int p \leq (n div x \wedge pm + x * int pm) div (int p) * int p **using** p0 **by** simp

also **have** ... \leq n div x \wedge pm + x * int pm

unfolding mod-div-equality-int **using** p **by** auto

```

    finally have  $n \operatorname{div} x^{\wedge} pm \geq x$  by (auto simp: p field-simps)
    from mult-right-mono[OF this, of  $x^{\wedge} pm$ ]
    have ge:  $n \operatorname{div} x^{\wedge} pm * x^{\wedge} pm \geq x^{\wedge} p$  unfolding xp using x by auto
    from div-mult-mod-eq[of  $n x^{\wedge} pm$ ] have  $n \operatorname{div} x^{\wedge} pm * x^{\wedge} pm = n - n \operatorname{mod} x^{\wedge} pm$ 
  by arith
    from ge[unfolded this]
    have le:  $x^{\wedge} p \leq n - n \operatorname{mod} x^{\wedge} pm$  .
    from x n have ge:  $n \operatorname{mod} x^{\wedge} pm \geq 0$ 
      by (auto simp add: mod-int-pos-iff not-less power-le-zero-eq)
    from le ge
    have  $n \geq x^{\wedge} p$  by auto
    with xn have False by auto
  }
  with le show ?thesis unfolding p by fastforce
qed

```

```

lemma iteration-mono-lesseq: assumes  $x: x \geq 0$  and  $n: n \geq 0$  and  $xn: x^{\wedge} p \geq$ 
( $n :: int$ )
  shows  $(n \operatorname{div} x^{\wedge} pm + x * int pm) \operatorname{div} int p \leq x$ 
proof (cases  $x^{\wedge} p = n$ )
  case True
    from iteration-mono-eq[OF this] show ?thesis by simp
  next
    case False
      with assms have  $x^{\wedge} p > n$  by auto
      from iteration-mono-less[OF x n this]
      show ?thesis by simp
qed

```

termination

proof –

```

  let ?mm =  $\lambda x n :: int. nat x$ 
  let ?m1 =  $\lambda (x,n). ?mm x n$ 
  let ?m = measures [?m1]
  show ?thesis
  proof (relation ?m)
    fix  $x n :: int$ 
    assume  $\neg x^{\wedge} p \leq n$ 
    hence  $x: x^{\wedge} p > n$  by auto
    assume  $\neg (x < 0 \vee n < 0)$ 
    hence  $x-n: x \geq 0 n \geq 0$  by auto
    from x x-n have  $x0: x > 0$  using p by (cases  $x = 0$ , auto)
    from iteration-mono-less[OF x-n x] x0
    show  $((n \operatorname{div} x^{\wedge} pm + x * int pm) \operatorname{div} int p, n), (x, n) \in ?m$  by auto
  qed auto
qed

```

We next prove that *root-int-main'* is a correct implementation of *root-newton-int-main*. We additionally prove that the result is always positive, a lower bound, and that the returned boolean indicates whether the result has a root or not. We

prove all these results in one go, so that we can share the inductive proof.

abbreviation *root-main'* **where** *root-main'* \equiv *root-int-main'* *pm* (*int pm*) (*int p*)

lemmas *root-main'-simps* = *root-int-main'.simps*[*of pm int pm int p*]

lemma *root-main'-newton-pos*: $x \geq 0 \implies n \geq 0 \implies$

$\text{root-main}' x n = \text{root-newton-int-main} x n \wedge (\text{root-main}' x n = (y,b) \longrightarrow y \geq 0$
 $\wedge y \hat{\ }^p \leq n \wedge b = (y \hat{\ }^p = n))$

proof (*induct x n rule: root-newton-int-main.induct*)

case (*1 x n*)

have *pm-x[simp]*: $x \hat{\ }^{pm} * x = x \hat{\ }^p$ **unfolding** *p* **by** *simp*

from *1* **have** *id*: $(x < 0 \vee n < 0) = \text{False}$ **by** *auto*

note *d* = *root-main'-simps*[*of x n*] *root-newton-int-main.simps*[*of x n*] *id if-False*

Let-def

show *?case*

proof (*cases x \hat{\ }^p \leq n*)

case *True*

thus *?thesis* **unfolding** *d* **using** *1(2)* **by** *auto*

next

case *False*

hence *id*: $(x \hat{\ }^p \leq n) = \text{False}$ **by** *simp*

from *1(3)* *1(2)* **have** *not*: $\neg (x < 0 \vee n < 0)$ **by** *auto*

then have *x*: $x > 0 \vee x = 0$

by *auto*

with $\langle 0 \leq n \rangle$ **have** $0 \leq (n \text{ div } x \hat{\ }^{pm} + x * \text{int } pm) \text{ div } \text{int } p$

by (*auto simp add: p algebra-simps pos-imp-zdiv-nonneg-iff power-0-left*)

then show *?thesis* **unfolding** *d id pm-x*

by (*rule 1(1)[OF not False - 1(3)]*)

qed

qed

lemma *root-main'*: $x \geq 0 \implies n \geq 0 \implies \text{root-main}' x n = \text{root-newton-int-main}$
 $x n$

using *root-main'-newton-pos* **by** *blast*

lemma *root-main'-pos*: $x \geq 0 \implies n \geq 0 \implies \text{root-main}' x n = (y,b) \implies y \geq 0$

using *root-main'-newton-pos* **by** *blast*

lemma *root-main'-sound*: $x \geq 0 \implies n \geq 0 \implies \text{root-main}' x n = (y,b) \implies b =$
 $(y \hat{\ }^p = n)$

using *root-main'-newton-pos* **by** *blast*

In order to prove completeness of the algorithms, we provide sharp upper and lower bounds for *root-main'*. For the upper bounds, we use Cauchy's mean theorem where we added the non-strict variant to Porter's formalization of this theorem.

lemma *root-main'-lower*: $x \geq 0 \implies n \geq 0 \implies \text{root-main}' x n = (y,b) \implies y \hat{\ }^p$
 $\leq n$

using *root-main'-newton-pos* **by** *blast*

```

lemma root-newton-int-main-upper:
  shows  $y^{\wedge} p \geq n \implies y \geq 0 \implies n \geq 0 \implies \text{root-newton-int-main } y \ n = (x,b)$ 
   $\implies n < (x + 1)^{\wedge} p$ 
proof (induct y n rule: root-newton-int-main.induct)
  case (1 y n)
  from 1(3) have  $y0: y \geq 0$  .
  then have  $y > 0 \vee y = 0$ 
    by auto
  from 1(4) have  $n0: n \geq 0$  .
  define  $y'$  where  $y' = (n \text{ div } (y^{\wedge} pm) + y * \text{int } pm) \text{ div } (\text{int } p)$ 
  from  $\langle y > 0 \vee y = 0 \rangle \langle n \geq 0 \rangle$  have  $y'0: y' \geq 0$ 
    by (auto simp add: y'-def p algebra-simps pos-imp-zdiv-nonneg-iff power-0-left)
  let  $?rt = \text{root-newton-int-main}$ 
  from 1(5) have  $rt: ?rt \ y \ n = (x,b)$  by auto
  from  $y0 \ n0$  have  $\text{not}: \neg (y < 0 \vee n < 0) (y < 0 \vee n < 0) = \text{False}$  by auto
  note  $rt = rt[\text{unfolded } \text{root-newton-int-main.simps}[\text{of } y \ n] \ \text{not}(2) \ \text{if-False}, \ \text{folded } y'\text{-def}]$ 
  note  $IH = 1(1)[\text{folded } y'\text{-def}, \ \text{OF } \text{not}(1) \ - \ y'0 \ n0]$ 
  show  $?case$ 
proof (cases y^{\wedge} p \leq n)
  case False note  $yyn = \text{this}$ 
  with  $rt$  have  $rt: ?rt \ y' \ n = (x,b)$  by simp
  show  $?thesis$ 
proof (cases n \leq y'^{\wedge} p)
  case True
  show  $?thesis$ 
    by (rule IH[OF False True rt])
  next
  case False
  with  $rt$  have  $x: x = y'$  unfolding root-newton-int-main.simps[\text{of } y' \ n]
    using  $n0 \ y'0$  by simp
  from  $yyn$  have  $yyn: y^{\wedge} p > n$  by simp
  from False have  $yyn': n > y'^{\wedge} p$  by auto
  {
    assume  $pm: pm = 0$ 
    have  $y': y' = n$  unfolding  $y'\text{-def } p \ pm$  by simp
    with  $yyn'$  have False unfolding  $p \ pm$  by auto
  }
  hence  $pm0: pm > 0$  by auto
  show  $?thesis$ 
proof (cases n = 0)
  case True
  thus  $?thesis$  unfolding  $p$ 
    by (metis False y'0 zero-le-power)
  next
  case False note  $n00 = \text{this}$ 
  let  $?y = \text{of-int } y :: \text{real}$ 
  let  $?n = \text{of-int } n :: \text{real}$ 

```

```

from  $yyn$   $n0$  have  $y00$ :  $y \neq 0$  unfolding  $p$  by auto
from  $y00$   $y0$  have  $y0$ :  $?y > 0$  by auto
from  $n0$  False have  $n0$ :  $?n > 0$  by auto
define  $Y$  where  $Y = ?y * of-int\ pm$ 
define  $NY$  where  $NY = ?n / ?y \wedge pm$ 
note  $pos-intro = divide-nonneg-pos\ add-nonneg-nonneg\ mult-nonneg-nonneg$ 
have  $NY0$ :  $NY > 0$  unfolding  $NY-def$  using  $y0\ n0$ 
by (metis  $NY-def\ zero-less-divide-iff\ zero-less-power$ )
let  $?ls = NY \# replicate\ pm\ ?y$ 
have  $prod$ :  $\prod : replicate\ pm\ ?y = ?y \wedge pm$ 
by (induct  $pm, auto$ )
have  $sum$ :  $\sum : replicate\ pm\ ?y = Y$  unfolding  $Y-def$ 
by (induct  $pm, auto\ simp: field-simps$ )
have  $pos$ :  $pos\ ?ls$  unfolding  $pos-def$  using  $NY0\ y0$  by auto
have  $root\ p\ ?n = gmean\ ?ls$  unfolding  $gmean-def$  using  $y0$ 
by (auto simp: p\ NY-def\ prod)
also have  $\dots < mean\ ?ls$ 
proof (rule  $CauchysMeanTheorem-Less[OF\ pos\ het-gt-0I]$ )
show  $NY \in set\ ?ls$  by simp
from  $pm0$  show  $?y \in set\ ?ls$  by simp
have  $NY < ?y$ 
proof -
from  $yyn$  have  $less$ :  $?n < ?y \wedge Suc\ pm$  unfolding  $p$ 
by (metis  $of-int-less-iff\ of-int-power$ )
have  $NY < ?y \wedge Suc\ pm / ?y \wedge pm$  unfolding  $NY-def$ 
by (rule  $divide-strict-right-mono[OF\ less], insert\ y0, auto$ )
thus  $?thesis$  using  $y0$  by auto
qed
thus  $NY \neq ?y$  by blast
qed
also have  $\dots = (NY + Y) / real\ p$ 
by (simp add: mean-def\ sum\ p)
finally have  $*$ :  $root\ p\ ?n < (NY + Y) / real\ p$  .
have  $?n = (root\ p\ ?n) \wedge p$  using  $n0$ 
by (metis  $neq0-conv\ p0\ real-root-pow-pos$ )
also have  $\dots < ((NY + Y) / real\ p) \wedge p$ 
by (rule  $power-strict-mono[OF\ *], insert\ n0\ p, auto$ )
finally have  $ineq1$ :  $?n < ((NY + Y) / real\ p) \wedge p$  by auto
{
define  $s$  where  $s = n\ div\ y \wedge pm + y * int\ pm$ 
define  $S$  where  $S = NY + Y$ 
have  $Y0$ :  $Y \geq 0$  using  $y0$  unfolding  $Y-def$ 
by (metis  $1.premis(2)\ mult-nonneg-nonneg\ of-int-0-le-iff\ of-nat-0-le-iff$ )
have  $S0$ :  $S > 0$  using  $NY0\ Y0$  unfolding  $S-def$  by auto
from  $p$  have  $p0$ :  $p > 0$  by auto
have  $?n / ?y \wedge pm < of-int\ (floor\ (?n / ?y \wedge pm)) + 1$ 
by (rule  $divide-less-floor1$ )
also have  $floor\ (?n / ?y \wedge pm) = n\ div\ y \wedge pm$ 
unfolding  $div-is-floor-divide-real$  by (metis  $of-int-power$ )

```

```

finally have  $NY < \text{of-int } (n \text{ div } y \wedge pm) + 1$  unfolding  $NY\text{-def}$  by  $\text{simp}$ 
hence less:  $S < \text{of-int } s + 1$  unfolding  $Y\text{-def } s\text{-def } S\text{-def}$  by  $\text{simp}$ 
{
  have  $f1: \forall x_0. \text{rat-of-int } \lfloor \text{rat-of-nat } x_0 \rfloor = \text{rat-of-nat } x_0$ 
    using  $\text{of-int-of-nat-eq}$  by  $\text{simp}$ 
  have  $f2: \forall x_0. \text{real-of-int } \lfloor \text{rat-of-nat } x_0 \rfloor = \text{real } x_0$ 
    using  $\text{of-int-of-nat-eq}$  by  $\text{auto}$ 
  have  $f3: \forall x_0 x_1. \lfloor \text{rat-of-int } x_0 / \text{rat-of-int } x_1 \rfloor = \lfloor \text{real-of-int } x_0 /$ 
 $\text{real-of-int } x_1 \rfloor$ 
    using  $\text{div-is-floor-divide-rat div-is-floor-divide-real}$  by  $\text{simp}$ 
  have  $f4: 0 < \lfloor \text{rat-of-nat } p \rfloor$ 
    using  $p$  by  $\text{simp}$ 
  have  $\lfloor S \rfloor \leq s$  using  $\text{less floor-le-iff}$  by  $\text{auto}$ 
  hence  $\lfloor \text{rat-of-int } \lfloor S \rfloor / \text{rat-of-nat } p \rfloor \leq \lfloor \text{rat-of-int } s / \text{rat-of-nat } p \rfloor$ 
    using  $f1 f3 f4$  by  $(\text{metis div-is-floor-divide-real zdiv-mono1})$ 
  hence  $\lfloor S / \text{real } p \rfloor \leq \lfloor \text{rat-of-int } s / \text{rat-of-nat } p \rfloor$ 
    using  $f1 f2 f3 f4$  by  $(\text{metis div-is-floor-divide-real floor-div-pos-int})$ 
  hence  $S / \text{real } p \leq \text{real-of-int } (s \text{ div int } p) + 1$ 
    using  $f1 f3$  by  $(\text{metis div-is-floor-divide-real floor-le-iff floor-of-nat}$ 
 $\text{less-eq-real-def})$ 
}
hence  $S / \text{real } p \leq \text{of-int}(s \text{ div } p) + 1$  .
note  $\text{this}[\text{unfolded } S\text{-def } s\text{-def}]$ 
}
hence  $ge: \text{of-int } y' + 1 \geq (NY + Y) / p$  unfolding  $y'\text{-def}$ 
by  $\text{simp}$ 
have  $pos1: (NY + Y) / p \geq 0$  unfolding  $Y\text{-def } NY\text{-def}$ 
by  $(\text{intro divide-nonneg-pos add-nonneg-nonneg mult-nonneg-nonneg,}$ 
 $\text{insert } y0 \text{ n0 } p0, \text{ auto})$ 
have  $pos2: \text{of-int } y' + (1 :: \text{rat}) \geq 0$  using  $y'0$  by  $\text{auto}$ 
have  $ineq2: (\text{of-int } y' + 1) \wedge p \geq ((NY + Y) / p) \wedge p$ 
by  $(\text{rule power-mono}[OF ge pos1])$ 
from  $\text{order.strict-trans2}[OF ineq1 ineq2]$ 
have  $?n < \text{of-int } ((x + 1) \wedge p)$  unfolding  $x$ 
by  $(\text{metis of-int-1 of-int-add of-int-power})$ 
thus  $n < (x + 1) \wedge p$  using  $\text{of-int-less-iff}$  by  $\text{blast}$ 
qed
qed
next
case  $\text{True}$ 
with  $rt$  have  $x: x = y$  by  $\text{simp}$ 
with  $1(2) \text{ True}$  have  $n: n = y \wedge p$  by  $\text{auto}$ 
show  $?thesis$  unfolding  $n x$  using  $y0$  unfolding  $p$ 
by  $(\text{metis add-le-less-mono add-less-cancel-left lessI less-add-one add.right-neutral}$ 
 $\text{le-iff-add power-strict-mono})$ 
qed
qed

```

lemma $\text{root-main}'\text{-upper}$:

$x \wedge p \geq n \implies x \geq 0 \implies n \geq 0 \implies \text{root-main}' x n = (y, b) \implies n < (y + 1) \wedge p$
using *root-newton-int-main-upper*[of $n x y b$] *root-main'*[of $x n$] **by** *auto*
end

Now we can prove all the nice properties of *root-int-main*.

lemma *root-int-main-all*: **assumes** $n: n \geq 0$
and $rm: \text{root-int-main } p n = (y, b)$
shows $y \geq 0 \wedge b = (y \wedge p = n) \wedge (p > 0 \implies y \wedge p \leq n \wedge n < (y + 1) \wedge p)$
 $\wedge (p > 0 \implies x \geq 0 \implies x \wedge p = n \implies y = x \wedge b)$
proof (*cases* $p = 0$)
case *True*
with $rm[\text{unfolded } \text{root-int-main-def}]$
have $y: y = 1$ **and** $b: b = (n = 1)$ **by** *auto*
show *?thesis unfolding True y b using n by auto*
next
case *False*
from *False* **have** $p=0: p > 0$ **by** *auto*
from *False* **have** $(p = 0) = \text{False}$ **by** *simp*
from $rm[\text{unfolded } \text{root-int-main-def this Let-def}]$
have $rm: \text{root-int-main}' (p - 1) (\text{int } (p - 1)) (\text{int } p) (\text{start-value } n p) n = (y, b)$
by *simp*
from $\text{start-value}[OF n p-0]$ **have** $\text{start}: n \leq (\text{start-value } n p) \wedge p \leq \text{start-value } n p$ **by** *auto*
interpret *fixed-root p p - 1*
by (*unfold-locales, insert False, auto*)
from $\text{root-main}'\text{-pos}[OF \text{start}(2) n rm]$ **have** $y: y \geq 0$.
from $\text{root-main}'\text{-sound}[OF \text{start}(2) n rm]$ **have** $b: b = (y \wedge p = n)$.
from $\text{root-main}'\text{-lower}[OF \text{start}(2) n rm]$ **have** $\text{low}: y \wedge p \leq n$.
from $\text{root-main}'\text{-upper}[OF \text{start } n rm]$ **have** $\text{up}: n < (y + 1) \wedge p$.
{
assume $n: x \wedge p = n$ **and** $x: x \geq 0$
with low up **have** $\text{low}: y \wedge p \leq x \wedge p$ **and** $\text{up}: x \wedge p < (y + 1) \wedge p$ **by** *auto*
from $\text{power-strict-mono}[of x y, OF - x p-0]$ low **have** $x: x \geq y$ **by** *arith*
from $\text{power-mono}[of (y + 1) x p]$ $y \text{ up}$ **have** $y: y \geq x$ **by** *arith*
from $x y$ **have** $x = y$ **by** *auto*
with $b n$
have $y = x \wedge b$ **by** *auto*
}
thus *?thesis using b low up y by auto*
qed

lemma *root-int-main*: **assumes** $n: n \geq 0$
and $rm: \text{root-int-main } p n = (y, b)$
shows $y \geq 0 \wedge b = (y \wedge p = n) \wedge p > 0 \implies y \wedge p \leq n \wedge p > 0 \implies n < (y + 1) \wedge p$
 $p > 0 \implies x \geq 0 \implies x \wedge p = n \implies y = x \wedge b$
using *root-int-main-all*[*OF n rm, of x*] **by** *blast+*

lemma *root-int[simp]*: **assumes** $p: p \neq 0 \vee x \neq 1$

```

shows set (root-int p x) = {y . y ^ p = x}
proof (cases p = 0)
  case True
  with p have x ≠ 1 by auto
  thus ?thesis unfolding root-int-def True by auto
next
  case False
  hence p: (p = 0) = False and p0: p > 0 by auto
  note d = root-int-def p if-False Let-def
  show ?thesis
  proof (cases x = 0)
    case True
    thus ?thesis unfolding d using p0 by auto
  next
    case False
    hence x: (x = 0) = False by auto
    show ?thesis
    proof (cases x < 0 ∧ even p)
      case True
      hence left: set (root-int p x) = {} unfolding d by auto
      {
        fix y
        assume x: y ^ p = x
        with True have y ^ p < 0 ∧ even p by auto
        hence False by presburger
      }
      with left show ?thesis by auto
    next
      case False
      with x p have cond: (x = 0) = False (x < 0 ∧ even p) = False by auto
      obtain y b where rt: root-int-main p |x| = (y,b) by force
      have abs x ≥ 0 by auto
      note rm = root-int-main[OF this rt]
      have ?thesis =
        (set (case root-int-main p |x| of (y, True) ⇒ if even p then [y, - y] else
[sgn x * y] | (y, False) ⇒ [])) =
        {y. y ^ p = x} unfolding d cond by blast
      also have (case root-int-main p |x| of (y, True) ⇒ if even p then [y, - y]
else [sgn x * y] | (y, False) ⇒ [])
        = (if b then if even p then [y, - y] else [sgn x * y] else []) (is - = ?lhs)
      unfolding rt by auto
      also have set ?lhs = {y. y ^ p = x} (is - = ?rhs)
    proof -
      {
        fix z
        assume idx: z ^ p = x
        hence eq: (abs z) ^ p = abs x by (metis power-abs)
        from idx x p0 have z: z ≠ 0 unfolding p by auto
        have (y, b) = (|z|, True)

```

```

    using rm(5)[OF p0 - eq] by auto
    hence id:  $y = \text{abs } z \text{ } b = \text{True}$  by auto
    have  $z \in \text{set } ?\text{lhs}$  unfolding id using z by (auto simp: idx[symmetric]),
cases  $z < 0$ , auto)
}
moreover
{
  fix z
  assume  $z: z \in \text{set } ?\text{lhs}$ 
  hence  $b: b = \text{True}$  by (cases b, auto)
  note  $z = z[\text{unfolded } b \text{ if-True}]$ 
  from rm(2) b have  $yx: y^p = |x|$  by auto
  from rm(1) have  $y: y \geq 0$  .
  from False have  $\text{odd } p \vee \text{even } p \wedge x \geq 0$  by auto
  hence  $z \in ?\text{rhs}$ 
  proof
    assume odd:  $\text{odd } p$ 
    with z have  $z = \text{sgn } x * y$  by auto
    hence  $z^p = (\text{sgn } x * y)^p$  by auto
    also have  $\dots = \text{sgn } x^p * y^p$  unfolding power-mult-distrib by auto
    also have  $\dots = \text{sgn } x^p * \text{abs } x$  unfolding yx by simp
    also have  $\text{sgn } x^p = \text{sgn } x$  using x odd by auto
    also have  $\text{sgn } x * \text{abs } x = x$  by (rule mult-sgn-abs)
    finally show  $z \in ?\text{rhs}$  by auto
  next
    assume even:  $\text{even } p \wedge x \geq 0$ 
    from z even have  $z = y \vee z = -y$  by auto
    hence id:  $\text{abs } z = y$  using y by auto
    with yx x even have  $z: z \neq 0$  using p0 by (cases y = 0, auto)
    have  $z^p = (\text{sgn } z * \text{abs } z)^p$  by (simp add: mult-sgn-abs)
    also have  $\dots = (\text{sgn } z * y)^p$  using id by auto
    also have  $\dots = (\text{sgn } z)^p * y^p$  unfolding power-mult-distrib by
simp
    also have  $\dots = \text{sgn } z^p * x$  unfolding yx using even by auto
    also have  $\text{sgn } z^p = 1$  using even z by (auto)
    finally show  $z \in ?\text{rhs}$  by auto
  qed
}
ultimately show ?thesis by blast
qed
finally show ?thesis by auto
qed
qed
qed
qed
lemma root-int-pos: assumes  $x: x \geq 0$  and ri:  $\text{root-int } p \ x = y \ \# \ ys$ 
shows  $y \geq 0$ 
proof -
  from x have abs:  $\text{abs } x = x$  by auto

```

```

note  $ri = ri$ [unfolded root-int-def Let-def abs]
from  $ri$  have  $p: (p = 0) = False$  by (cases p, auto)
note  $ri = ri$ [unfolded p if-False]
show ?thesis
proof (cases x = 0)
  case True
    with  $ri$  show ?thesis by auto
  next
    case False
      hence  $(x = 0) = False$   $(x < 0 \wedge \text{even } p) = False$  using  $x$  by auto
      note  $ri = ri$ [unfolded this if-False]
      obtain  $y' b'$  where  $r: \text{root-int-main } p \ x = (y', b')$  by force
      note  $ri = ri$ [unfolded this]
      hence  $y: y = (\text{if even } p \ \text{then } y' \ \text{else } \text{sgn } x * y')$  by (cases b', auto)
      from  $\text{root-int-main}(1)$ [OF x r] have  $y': 0 \leq y'$  .
      thus ?thesis unfolding  $y$  using  $x$  False by auto
qed
qed

```

3.3 Floor and ceiling of roots

Using the bounds for *root-int-main* we can easily design algorithms which compute $\lfloor \text{root } p \ x \rfloor$ and $\lceil \text{root } p \ x \rceil$. To this end, we first develop algorithms for non-negative x , and later on these are used for the general case.

definition *root-int-floor-pos* $p \ x = (\text{if } p = 0 \ \text{then } 0 \ \text{else } \text{fst } (\text{root-int-main } p \ x))$

definition *root-int-ceiling-pos* $p \ x = (\text{if } p = 0 \ \text{then } 0 \ \text{else } (\text{case } \text{root-int-main } p \ x \ \text{of } (y, b) \Rightarrow \text{if } b \ \text{then } y \ \text{else } y + 1))$

lemma *root-int-floor-pos-lower*: **assumes** $p0: p \neq 0$ **and** $x: x \geq 0$
shows *root-int-floor-pos* $p \ x \hat{=} p \leq x$
using $\text{root-int-main}(3)$ [*OF x, of p*] $p0$ **unfolding** *root-int-floor-pos-def*
by (*cases root-int-main p x, auto*)

lemma *root-int-floor-pos-pos*: **assumes** $x: x \geq 0$
shows *root-int-floor-pos* $p \ x \geq 0$
using $\text{root-int-main}(1)$ [*OF x, of p*]
unfolding *root-int-floor-pos-def*
by (*cases root-int-main p x, auto*)

lemma *root-int-floor-pos-upper*: **assumes** $p0: p \neq 0$ **and** $x: x \geq 0$
shows $(\text{root-int-floor-pos } p \ x + 1) \hat{=} p > x$
using $\text{root-int-main}(4)$ [*OF x, of p*] $p0$ **unfolding** *root-int-floor-pos-def*
by (*cases root-int-main p x, auto*)

lemma *root-int-floor-pos*: **assumes** $x: x \geq 0$
shows *root-int-floor-pos* $p \ x = \text{floor } (\text{root } p \ (\text{of-int } x))$
proof (*cases p = 0*)
case *True*
thus *?thesis* **by** (*simp add: root-int-floor-pos-def*)

```

next
  case False
  hence  $p: p > 0$  by auto
  let  $?s1 = \text{real-of-int } (\text{root-int-floor-pos } p \ x)$ 
  let  $?s2 = \text{root } p \ (\text{of-int } x)$ 
  from  $x$  have  $s1: ?s1 \geq 0$ 
    by (metis of-int-0-le-iff root-int-floor-pos-pos)
  from  $x$  have  $s2: ?s2 \geq 0$ 
    by (metis of-int-0-le-iff real-root-pos-pos-le)
  from  $s1$  have  $s11: ?s1 + 1 \geq 0$  by auto
  have  $\text{id}: ?s2 \wedge^p = \text{of-int } x$  using  $x$ 
    by (metis p of-int-0-le-iff real-root-pow-pos2)
  show ?thesis
  proof (rule floor-unique[symmetric])
    show  $?s1 \leq ?s2$ 
      unfolding compare-pow-le-iff[OF p s1 s2, symmetric]
      unfolding id
      using root-int-floor-pos-lower[OF False x]
      by (metis of-int-le-iff of-int-power)
    show  $?s2 < ?s1 + 1$ 
      unfolding compare-pow-less-iff[OF p s2 s11, symmetric]
      unfolding id
      using root-int-floor-pos-upper[OF False x]
      by (metis of-int-add of-int-less-iff of-int-power of-int-1)
  qed
qed

```

```

lemma root-int-ceiling-pos: assumes  $x: x \geq 0$ 
  shows  $\text{root-int-ceiling-pos } p \ x = \text{ceiling } (\text{root } p \ (\text{of-int } x))$ 
proof (cases p = 0)
  case True
  thus ?thesis by (simp add: root-int-ceiling-pos-def)
next
  case False
  hence  $p: p > 0$  by auto
  obtain  $y \ b$  where  $s: \text{root-int-main } p \ x = (y, b)$  by force
  note  $rm = \text{root-int-main}$ [OF x s]
  note  $rm = rm(1-2) \ rm(3-5)$ [OF p]
  from  $rm(1)$  have  $y: y \geq 0$  by simp
  let  $?s = \text{root-int-ceiling-pos } p \ x$ 
  let  $?sx = \text{root } p \ (\text{of-int } x)$ 
  note  $d = \text{root-int-ceiling-pos-def}$ 
  show ?thesis
  proof (cases b)
    case True
    hence  $\text{id}: ?s = y$  unfolding  $s \ d$  using  $p$  by auto
    from  $rm(2)$  True have  $xy: x = y \wedge^p$  by auto
    show ?thesis unfolding id unfolding  $xy$  using  $y$ 
      by (simp add: p real-root-power-cancel)

```

```

next
  case False
  hence id:  $?s = \text{root-int-floor-pos } p \ x + 1$  unfolding d root-int-floor-pos-def
    using s p by simp
  from False have  $x0: x \neq 0$  using rm(5)[of 0] using s unfolding root-int-main-def
Let-def using p
  by (cases  $x = 0$ , auto)
  show ?thesis unfolding id root-int-floor-pos[OF x]
  proof (rule ceiling-unique[symmetric])
    show  $?sx \leq \text{real-of-int } (\lfloor \text{root } p \ (\text{of-int } x) \rfloor + 1)$ 
      by (metis of-int-add real-of-int-floor-add-one-ge of-int-1)
    let  $?l = \text{real-of-int } (\lfloor \text{root } p \ (\text{of-int } x) \rfloor + 1) - 1$ 
    let  $?m = \text{real-of-int } \lfloor \text{root } p \ (\text{of-int } x) \rfloor$ 
    have  $?l = ?m$  by simp
    also have  $\dots < ?sx$ 
  proof -
    have le:  $?m \leq ?sx$  by (rule of-int-floor-le)
    have neq:  $?m \neq ?sx$ 
  proof
    assume  $?m = ?sx$ 
    hence  $?m \wedge p = ?sx \wedge p$  by auto
    also have  $\dots = \text{of-int } x$  using x False
  by (metis p real-root-ge-0-iff real-root-pow-pos2 root-int-floor-pos root-int-floor-pos-pos
zero-le-floor zero-less-Suc)
  finally have xs:  $x = \lfloor \text{root } p \ (\text{of-int } x) \rfloor \wedge p$ 
    by (metis floor-power floor-of-int)
  hence  $\lfloor \text{root } p \ (\text{of-int } x) \rfloor \in \text{set } (\text{root-int } p \ x)$  using p by simp
  hence  $\text{root-int } p \ x \neq []$  by force
  with s False  $\langle p \neq 0 \rangle$  x x0 show False unfolding root-int-def
    by (cases p, auto)
  qed
  from le neq show ?thesis by arith
  qed
  finally show  $?l < ?sx$  .
  qed
  qed
  qed
  qed

```

definition *root-int-floor* $p \ x = (\text{if } x \geq 0 \text{ then } \text{root-int-floor-pos } p \ x \text{ else } - \text{root-int-ceiling-pos } p \ (-x))$

definition *root-int-ceiling* $p \ x = (\text{if } x \geq 0 \text{ then } \text{root-int-ceiling-pos } p \ x \text{ else } - \text{root-int-floor-pos } p \ (-x))$

lemma *root-int-floor[simp]*: $\text{root-int-floor } p \ x = \text{floor } (\text{root } p \ (\text{of-int } x))$

```

proof -
  note d = root-int-floor-def
  show ?thesis
  proof (cases  $x \geq 0$ )

```

```

    case True
    with root-int-floor-pos[OF True, of p] show ?thesis unfolding d by simp
next
    case False
    hence  $-x \geq 0$  by auto
    from False root-int-ceiling-pos[OF this] show ?thesis unfolding d
    by (simp add: real-root-minus ceiling-minus)
qed
qed

```

lemma *root-int-ceiling[simp]*: $\text{root-int-ceiling } p \ x = \text{ceiling } (\text{root } p \ (\text{of-int } x))$

```

proof -
  note d = root-int-ceiling-def
  show ?thesis
  proof (cases  $x \geq 0$ )
    case True
    with root-int-ceiling-pos[OF True] show ?thesis unfolding d by simp
  next
    case False
    hence  $-x \geq 0$  by auto
    from False root-int-floor-pos[OF this, of p] show ?thesis unfolding d
    by (simp add: real-root-minus floor-minus)
  qed
qed

```

3.4 Downgrading algorithms to the naturals

definition *root-nat-floor* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{int}$ **where**
 $\text{root-nat-floor } p \ x = \text{root-int-floor-pos } p \ (\text{int } x)$

definition *root-nat-ceiling* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{int}$ **where**
 $\text{root-nat-ceiling } p \ x = \text{root-int-ceiling-pos } p \ (\text{int } x)$

definition *root-nat* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list}$ **where**
 $\text{root-nat } p \ x = \text{map nat } (\text{take } 1 \ (\text{root-int } p \ x))$

lemma *root-nat-floor [simp]*: $\text{root-nat-floor } p \ x = \text{floor } (\text{root } p \ (\text{real } x))$
unfolding *root-nat-floor-def* **using** *root-int-floor-pos*[of $\text{int } x \ p$]
by *auto*

lemma *root-nat-floor-lower*: **assumes** $p0: p \neq 0$
shows $\text{root-nat-floor } p \ x \wedge^p \leq x$
using *root-int-floor-pos-lower*[OF $p0, \text{ of } x$] **unfolding** *root-nat-floor-def* **by** *auto*

lemma *root-nat-floor-upper*: **assumes** $p0: p \neq 0$
shows $(\text{root-nat-floor } p \ x + 1) \wedge^p > x$
using *root-int-floor-pos-upper*[OF $p0, \text{ of } x$] **unfolding** *root-nat-floor-def* **by** *auto*

lemma *root-nat-ceiling [simp]*: $\text{root-nat-ceiling } p \ x = \text{ceiling } (\text{root } p \ x)$

```

unfolding root-nat-ceiling-def using root-int-ceiling-pos[of x p]
by auto

lemma root-nat: assumes p0:  $p \neq 0 \vee x \neq 1$ 
shows set (root-nat p x) =  $\{y. y \hat{\ }^p = x\}$ 
proof -
  {
    fix y
    assume y  $\in$  set (root-nat p x)
    note y = this[unfolded root-nat-def]
    then obtain yi ys where ri: root-int p x = yi # ys by (cases root-int p x,
auto)
    with y have y: y = nat yi by auto
    from root-int-pos[OF - ri] have yi:  $0 \leq yi$  by auto
    from root-int[of p int x] p0 ri have yi  $\hat{\ }^p = x$  by auto
    from arg-cong[OF this, of nat] yi have nat yi  $\hat{\ }^p = x$ 
      by (metis nat-int nat-power-eq)
    hence y  $\in$   $\{y. y \hat{\ }^p = x\}$  using y by auto
  }
moreover
  {
    fix y
    assume yx: y  $\hat{\ }^p = x$ 
    hence y: int y  $\hat{\ }^p =$  int x
      by (metis of-nat-power)
    hence set (root-int p (int x))  $\neq$  {} using root-int[of p int x] p0
    by (metis (mono-tags) One-nat-def  $\langle y \hat{\ }^p = x \rangle$  empty-Collect-eq nat-power-eq-Suc-0-iff)
    then obtain yi ys where ri: root-int p (int x) = yi # ys
      by (cases root-int p (int x), auto)
    from root-int-pos[OF - this] have yip: yi  $\geq 0$  by auto
    from root-int[of p int x, unfolded ri] p0 have yi: yi  $\hat{\ }^p =$  int x by auto
    with y have int y  $\hat{\ }^p =$  yi  $\hat{\ }^p$  by auto
    from arg-cong[OF this, of nat] have id: y  $\hat{\ }^p =$  nat yi  $\hat{\ }^p$ 
      by (metis  $\langle y \hat{\ }^p = x \rangle$  nat-int nat-power-eq yi yip)
  }
  {
    assume p: p  $\neq 0$ 
    hence p0: p  $> 0$  by auto
    obtain yy b where rm: root-int-main p (int x) = (yy,b) by force
    from root-int-main(5)[OF - rm p0 - y] have yy = int y and b = True by
auto
    note rm = rm[unfolded this]
    hence y  $\in$  set (root-nat p x)
      unfolding root-nat-def p root-int-def using p0 p yx
      by auto
  }
moreover
  {
    assume p: p = 0
    with p0 have x  $\neq 1$  by auto
  }

```



```

    with y p have False by auto
  }
  ultimately have y ∈ set (root-nat p x) by auto
}
ultimately show ?thesis by blast
qed

```

3.5 Upgrading algorithms to the rationals

The main observation to lift everything from the integers to the rationals is the fact, that one can reformulate $\frac{a}{b}^{1/p}$ as $\frac{(ab^{p-1})^{1/p}}{b}$.

definition *root-rat-floor* :: nat ⇒ rat ⇒ int **where**
root-rat-floor p x ≡ case quotient-of x of (a,b) ⇒ root-int-floor p (a * b^{^(p - 1)) div b}

definition *root-rat-ceiling* :: nat ⇒ rat ⇒ int **where**
root-rat-ceiling p x ≡ - (root-rat-floor p (-x))

definition *root-rat* :: nat ⇒ rat ⇒ rat list **where**
root-rat p x ≡ case quotient-of x of (a,b) ⇒ concat
 (map (λ rb. map (λ ra. of-int ra / rat-of-int rb) (root-int p a)) (take 1 (root-int p b)))

lemma *root-rat-reform*: **assumes** q: quotient-of x = (a,b)
shows root p (real-of-rat x) = root p (of-int (a * b^{^(p - 1)) / of-int b}

proof (cases p = 0)

case False

from quotient-of-denom-pos[OF q] have b: 0 < b by auto

hence b: 0 < real-of-int b by auto

from quotient-of-div[OF q] have x: root p (real-of-rat x) = root p (a / b)

by (metis of-rat-divide of-rat-of-int-eq)

also have a / b = a * real-of-int b^{^(p - 1)} / of-int b^{^p} using b False

by (cases p, auto simp: field-simps)

also have root p ... = root p (a * real-of-int b^{^(p - 1)) / root p (of-int b^{^p})}

by (rule real-root-divide)

also have root p (of-int b^{^p}) = of-int b using False b

by (metis neq0-conv real-root-pow-pos real-root-power)

also have a * real-of-int b^{^(p - 1)} = of-int (a * b^{^(p - 1))}

by (metis of-int-mult of-int-power)

finally show ?thesis .

qed auto

lemma *root-rat-floor [simp]*: root-rat-floor p x = floor (root p (of-rat x))

proof -

obtain a b **where** q: quotient-of x = (a,b) **by** force

from quotient-of-denom-pos[OF q] **have** b: b > 0 .

show ?thesis

unfolding root-rat-floor-def q split root-int-floor

unfolding *root-rat-reform*[*OF q*] *floor-div-pos-int*[*OF b*] ..
qed

lemma *root-rat-ceiling* [*simp*]: *root-rat-ceiling* *p x* = *ceiling* (*root p* (*of-rat x*))

unfolding
root-rat-ceiling-def
ceiling-def
real-root-minus
root-rat-floor
of-rat-minus
..

lemma *root-rat*[*simp*]: **assumes** *p*: $p \neq 0 \vee x \neq 1$

shows *set* (*root-rat p x*) = { *y*. $y \wedge^p = x$ }

proof (*cases p = 0*)

case *False*

note *p* = *this*

obtain *a b* **where** *q*: *quotient-of x = (a,b)* **by force**

note *x* = *quotient-of-div*[*OF q*]

have *b*: $b > 0$ **by** (*rule quotient-of-denom-pos*[*OF q*])

note *d* = *root-rat-def q split set-concat set-map*

{

fix *q*

assume $q \in \text{set } (\text{root-rat } p \ x)$

note *mem* = *this*[*unfolded d*]

from *mem* **obtain** *rb xs* **where** *rb*: *root-int p b = Cons rb xs* **by** (*cases root-int p b, auto*)

note *mem* = *mem*[*unfolded this*]

from *mem* **obtain** *ra* **where** *ra*: $ra \in \text{set } (\text{root-int } p \ a)$ **and** *q*: $q = \text{of-int } ra / \text{of-int } rb$

by (*cases root-int p a, auto*)

from *rb* **have** $rb \in \text{set } (\text{root-int } p \ b)$ **by auto**

with *ra p* **have** *rb*: $b = rb \wedge^p$ **and** *ra*: $a = ra \wedge^p$ **by auto**

have $q \in \{y. y \wedge^p = x\}$ **unfolding** *q x ra rb*

by (*auto simp: power-divide*)

}

moreover

{

fix *q*

assume $q \in \{y. y \wedge^p = x\}$

hence $q \wedge^p = \text{of-int } a / \text{of-int } b$ **unfolding** *x* **by auto**

hence *eq*: $\text{of-int } b * q \wedge^p = \text{of-int } a$ **using** *b* **by auto**

obtain *z n* **where** *quo*: *quotient-of q = (z,n)* **by force**

note *qzn* = *quotient-of-div*[*OF quo*]

have *n*: $n > 0$ **using** *quotient-of-denom-pos*[*OF quo*].

from *eq*[*unfolded qzn*] **have** $\text{rat-of-int } b * \text{of-int } z \wedge^p / \text{of-int } n \wedge^p = \text{of-int } a$

unfolding *power-divide* **by simp**

from *arg-cong*[*OF this, of λ x. x * of-int n ^ p*] *n* **have** $\text{rat-of-int } b * \text{of-int } z \wedge^p = \text{of-int } a * \text{of-int } n \wedge^p$

```

    by auto
    also have  $\text{rat-of-int } b * \text{of-int } z^p = \text{rat-of-int } (b * z^p)$  unfolding of-int-mult
of-int-power ..
    also have  $\text{of-int } a * \text{rat-of-int } n^p = \text{of-int } (a * n^p)$  unfolding of-int-mult
of-int-power ..
    finally have  $\text{id: } a * n^p = b * z^p$  by linarith
    from quotient-of-coprime[OF quo] have  $\text{cop: coprime } (z^p) (n^p)$ 
    by simp
    from coprime-crossproduct-int[OF quotient-of-coprime[OF q] this] arg-cong[OF
id, of abs]
    have  $|n^p| = |b|$ 
    by (simp add: field-simps abs-mult)
    with  $n\ b$  have  $\text{bnp: } b = n^p$  by auto
    hence  $\text{rn: } n \in \text{set } (\text{root-int } p\ b)$  using  $p$  by auto
    then obtain  $\text{rb rs where rb: root-int } p\ b = \text{Cons } \text{rb } \text{rs}$  by (cases root-int } p\ b,
auto)
    from id[folded bnp]  $b$  have  $a = z^p$  by auto
    hence  $a: z \in \text{set } (\text{root-int } p\ a)$  using  $p$  by auto
    from root-int-pos[OF - rb]  $b$  have  $\text{rb0: } \text{rb} \geq 0$  by auto
    from root-int[OF disjI1[OF p], of b]  $\text{rb}$  have  $\text{rb}^p = b$  by auto
    with  $\text{bnp}$  have  $\text{id: } \text{rb}^p = n^p$  by auto
    have  $\text{rb} = n$  by (rule power-eq-imp-eq-base[OF id], insert n rb0 p, auto)
    with  $\text{rb}$  have  $b: n \in \text{set } (\text{take } 1\ (\text{root-int } p\ b))$  by auto
    have  $q \in \text{set } (\text{root-rat } p\ x)$  unfolding  $d\ \text{qzn}$  using  $b\ a$  by auto
  }
  ultimately show ?thesis by blast
next
case True
with  $p$  have  $x: x \neq 1$  by auto
obtain  $a\ b$  where  $q: \text{quotient-of } x = (a,b)$  by force
show ?thesis unfolding True root-rat-def q split root-int-def using  $x$ 
by auto
qed
end

```

```

theory Sqrt-Babylonian
imports
  Sqrt-Babylonian-Auxiliary
  NthRoot-Impl
begin

```

4 Executable algorithms for square roots

This theory provides executable algorithms for computing square-roots of numbers which are all based on the Babylonian method (which is also known as Heron's method or Newton's method).

For integers / naturals / rationals precise algorithms are given, i.e., here `sqrt x` delivers a list of all integers / naturals / rationals y where $y^2 = x$. To this end, the Babylonian method has been adapted by using integer-divisions.

In addition to the precise algorithms, we also provide approximation algorithms. One works for arbitrary linear ordered fields, where some number y is computed such that $|y^2 - x| < \varepsilon$. Moreover, for the naturals, integers, and rationals we provide algorithms to compute $\lfloor \text{sqrt } x \rfloor$ and $\lceil \text{sqrt } x \rceil$ which are all based on the underlying algorithm that is used to compute the precise square-roots on integers, if these exist.

The major motivation for developing the precise algorithms was given by CeTA [2], a tool for certifying termination proofs. Here, non-linear equations of the form $(a_1x_1 + \dots a_nx_n)^2 = p$ had to be solved over the integers, where p is a concrete polynomial. For example, for the equation $(ax + by)^2 = 4x^2 - 12xy + 9y^2$ one easily figures out that $a^2 = 4$, $b^2 = 9$, and $ab = -6$, which results in a possible solution $a = \sqrt{4} = 2$, $b = -\sqrt{9} = -3$.

4.1 The Babylonian method

The Babylonian method for computing \sqrt{n} iteratively computes

$$x_{i+1} = \frac{\frac{n}{x_i} + x_i}{2}$$

until $x_i^2 \approx n$. Note that if $x_0^2 \geq n$, then for all i we have both $x_i^2 \geq n$ and $x_i \geq x_{i+1}$.

4.2 The Babylonian method using integer division

First, the algorithm is developed for the non-negative integers. Here, the division operation $\frac{x}{y}$ is replaced by $x \text{ div } y = \lfloor \text{of-int } x / \text{of-int } y \rfloor$. Note that replacing $\lfloor \text{of-int } x / \text{of-int } y \rfloor$ by $\lceil \text{of-int } x / \text{of-int } y \rceil$ would lead to non-termination in the following algorithm.

We explicitly develop the algorithm on the integers and not on the naturals, as the calculations on the integers have been much easier. For example, $y - x + x = y$ on the integers, which would require the side-condition $y \geq x$ for the naturals. These conditions will make the reasoning much more tedious—as we have experienced in an earlier state of this development where everything was based on naturals.

Since the elements x_0, x_1, x_2, \dots are monotone decreasing, in the main algorithm we abort as soon as $x_i^2 \leq n$.

Since in the meantime, all of these algorithms have been generalized to arbitrary p -th roots in *Sqrt-Babylonian.NthRoot-Impl*, we just instantiate the general algorithms by $p = 2$ and then provide specialized code equations which are more efficient than the general purpose algorithms.

definition *sqrt-int-main'* :: *int* \Rightarrow *int* \Rightarrow *int* \times *bool* **where**
 [simp]: *sqrt-int-main'* *x n* = *root-int-main'* 1 1 2 *x n*

lemma *sqrt-int-main'-code*[code]: *sqrt-int-main'* *x n* = (let *x2* = *x * x* in if *x2* \leq *n* then (*x*, *x2* = *n*)
 else *sqrt-int-main'* ((*n div x* + *x*) *div* 2) *n*)
using *root-int-main'.simps*[of 1 1 2 *x n*]
unfolding *Let-def* **by** *auto*

definition *sqrt-int-main* :: *int* \Rightarrow *int* \times *bool* **where**
 [simp]: *sqrt-int-main* *x* = *root-int-main* 2 *x*

lemma *sqrt-int-main-code*[code]: *sqrt-int-main* *x* = *sqrt-int-main'* (*start-value* *x* 2) *x*
by (*simp add: root-int-main-def Let-def*)

definition *sqrt-int* :: *int* \Rightarrow *int* *list* **where**
sqrt-int *x* = *root-int* 2 *x*

lemma *sqrt-int-code*[code]: *sqrt-int* *x* = (if *x* < 0 then [] else case *sqrt-int-main* *x* of (*y*, *True*) \Rightarrow if *y* = 0 then [0] else [*y*, -*y*] | - \Rightarrow [])

proof –

interpret *fixed-root* 2 1 **by** (*unfold-locales*, *auto*)
obtain *b y* **where** *res*: *root-int-main* 2 *x* = (*b*, *y*) **by** *force*
show *?thesis*
unfolding *sqrt-int-def root-int-def Let-def*
using *root-int-main[OF - res]*
using *res*
by *simp*

qed

lemma *sqrt-int*[simp]: *set* (*sqrt-int* *x*) = {*y*. *y* * *y* = *x*}
unfolding *sqrt-int-def* **by** (*simp add: power2-eq-square*)

lemma *sqrt-int-pos*: **assumes** *res*: *sqrt-int* *x* = *Cons* *s* *ms*
shows *s* \geq 0

proof –

note *res* = *res*[*unfolded sqrt-int-code Let-def*, *simplified*]
from *res* **have** *x0*: *x* \geq 0 **by** (*cases ?thesis*, *auto*)
obtain *ss b* **where** *call*: *sqrt-int-main* *x* = (*ss*, *b*) **by** *force*

from *res*[*unfolded call*] *x0* **have** $ss = s$
by (*cases b, cases ss = 0, auto*)
from *root-int-main*(1)[*OF x0 call[unfolded this sqrt-int-main-def]*]
show *?thesis* .
qed

definition [*simp*]: *sqrt-int-floor-pos* $x = \text{root-int-floor-pos } 2 \ x$

lemma *sqrt-int-floor-pos-code*[*code*]: *sqrt-int-floor-pos* $x = \text{fst } (\text{sqrt-int-main } x)$
by (*simp add: root-int-floor-pos-def*)

lemma *sqrt-int-floor-pos*: **assumes** $x: x \geq 0$
shows *sqrt-int-floor-pos* $x = \lfloor \text{sqrt } (\text{of-int } x) \rfloor$
using *root-int-floor-pos*[*OF x, of 2*] **by** (*simp add: sqrt-def*)

definition [*simp*]: *sqrt-int-ceiling-pos* $x = \text{root-int-ceiling-pos } 2 \ x$

lemma *sqrt-int-ceiling-pos-code*[*code*]: *sqrt-int-ceiling-pos* $x = (\text{case } \text{sqrt-int-main } x \text{ of } (y,b) \Rightarrow \text{if } b \text{ then } y \text{ else } y + 1)$
by (*simp add: root-int-ceiling-pos-def*)

lemma *sqrt-int-ceiling-pos*: **assumes** $x: x \geq 0$
shows *sqrt-int-ceiling-pos* $x = \lceil \text{sqrt } (\text{of-int } x) \rceil$
using *root-int-ceiling-pos*[*OF x, of 2*] **by** (*simp add: sqrt-def*)

definition *sqrt-int-floor* $x = \text{root-int-floor } 2 \ x$

lemma *sqrt-int-floor-code*[*code*]: *sqrt-int-floor* $x = (\text{if } x \geq 0 \text{ then } \text{sqrt-int-floor-pos } x \text{ else } - \text{sqrt-int-ceiling-pos } (- x))$
unfolding *sqrt-int-floor-def* *root-int-floor-def* **by** *simp*

lemma *sqrt-int-floor*[*simp*]: *sqrt-int-floor* $x = \lfloor \text{sqrt } (\text{of-int } x) \rfloor$
by (*simp add: sqrt-int-floor-def sqrt-def*)

definition *sqrt-int-ceiling* $x = \text{root-int-ceiling } 2 \ x$

lemma *sqrt-int-ceiling-code*[*code*]: *sqrt-int-ceiling* $x = (\text{if } x \geq 0 \text{ then } \text{sqrt-int-ceiling-pos } x \text{ else } - \text{sqrt-int-floor-pos } (- x))$
unfolding *sqrt-int-ceiling-def* *root-int-ceiling-def* **by** *simp*

lemma *sqrt-int-ceiling*[*simp*]: *sqrt-int-ceiling* $x = \lceil \text{sqrt } (\text{of-int } x) \rceil$
by (*simp add: sqrt-int-ceiling-def sqrt-def*)

lemma *sqrt-int-ceiling-bound*: $0 \leq x \implies x \leq (\text{sqrt-int-ceiling } x)^2$
unfolding *sqrt-int-ceiling* **using** *le-of-int-ceiling* *sqrt-le-D*
by (*metis of-int-power-le-of-int-cancel-iff*)

4.3 Square roots for the naturals

definition *sqrt-nat* :: *nat* \Rightarrow *nat list*
where *sqrt-nat* *x* = *root-nat* 2 *x*

lemma *sqrt-nat-code*[*code*]: *sqrt-nat* *x* \equiv *map nat (take 1 (sqrt-int (int x)))*
unfolding *sqrt-nat-def* *root-nat-def* *sqrt-int-def* **by** *simp*

lemma *sqrt-nat[simp]*: *set (sqrt-nat x)* = { *y. y * y = x* }
unfolding *sqrt-nat-def* **using** *root-nat[of 2 x]* **by** (*simp add: power2-eq-square*)

definition *sqrt-nat-floor* :: *nat* \Rightarrow *int* **where**
sqrt-nat-floor *x* = *root-nat-floor* 2 *x*

lemma *sqrt-nat-floor-code*[*code*]: *sqrt-nat-floor* *x* = *sqrt-int-floor-pos (int x)*
unfolding *sqrt-nat-floor-def* *root-nat-floor-def* **by** *simp*

lemma *sqrt-nat-floor[simp]*: *sqrt-nat-floor* *x* = $\lfloor \text{sqrt} (\text{real } x) \rfloor$
unfolding *sqrt-nat-floor-def* **by** (*simp add: sqrt-def*)

definition *sqrt-nat-ceiling* :: *nat* \Rightarrow *int* **where**
sqrt-nat-ceiling *x* = *root-nat-ceiling* 2 *x*

lemma *sqrt-nat-ceiling-code*[*code*]: *sqrt-nat-ceiling* *x* = *sqrt-int-ceiling-pos (int x)*
unfolding *sqrt-nat-ceiling-def* *root-nat-ceiling-def* **by** *simp*

lemma *sqrt-nat-ceiling[simp]*: *sqrt-nat-ceiling* *x* = $\lceil \text{sqrt} (\text{real } x) \rceil$
unfolding *sqrt-nat-ceiling-def* **by** (*simp add: sqrt-def*)

4.4 Square roots for the rationals

definition *sqrt-rat* :: *rat* \Rightarrow *rat list* **where**
sqrt-rat *x* = *root-rat* 2 *x*

lemma *sqrt-rat-code*[*code*]: *sqrt-rat* *x* = (*case quotient-of x of (z,n) \Rightarrow (case sqrt-int*
n of

$\square \Rightarrow \square$

| *sn # xs \Rightarrow map ($\lambda sz. \text{of-int } sz / \text{of-int } sn$) (sqrt-int z))*

proof –

obtain *z n* **where** *q: quotient-of x = (z,n)* **by** *force*

show *?thesis*

unfolding *sqrt-rat-def* *root-rat-def* *q split sqrt-int-def*

by (*cases root-int 2 n, auto*)

qed

lemma *sqrt-rat[simp]*: *set (sqrt-rat x)* = { *y. y * y = x* }
unfolding *sqrt-rat-def* **using** *root-rat[of 2 x]*
by (*simp add: power2-eq-square*)

lemma *sqrt-rat-pos*: **assumes** *sqrt: sqrt-rat x = Cons s ms*

shows $s \geq 0$
proof –
obtain $z\ n$ **where** q : *quotient-of* $x = (z,n)$ **by** *force*
note $\text{sqrt} = \text{sqrt}[\text{unfolded } \text{sqrt-rat-code } q, \text{simplified}]$
let $?sz = \text{sqrt-int } z$
let $?sn = \text{sqrt-int } n$
from q **have** n : $n > 0$ **by** (*rule quotient-of-denom-pos*)
from sqrt **obtain** $sz\ mz$ **where** sz : $?sz = sz \# mz$ **by** (*cases ?sn, auto*)
from sqrt **obtain** $sn\ mn$ **where** sn : $?sn = sn \# mn$ **by** (*cases ?sn, auto*)
from $\text{sqrt-int-pos}[OF\ sz]\ \text{sqrt-int-pos}[OF\ sn]$ **have** pos : $0 \leq sz \leq sn$ **by** *auto*
from $\text{sqrt } sz\ sn$ **have** s : $s = \text{of-int } sz / \text{of-int } sn$ **by** *auto*
show *?thesis unfolding s using pos*
by (*metis of-int-0-le-iff zero-le-divide-iff*)
qed

definition $\text{sqrt-rat-floor} :: \text{rat} \Rightarrow \text{int}$ **where**
 $\text{sqrt-rat-floor } x = \text{root-rat-floor } 2\ x$

lemma $\text{sqrt-rat-floor-code}[code]$: $\text{sqrt-rat-floor } x = (\text{case } \text{quotient-of } x \text{ of } (a,b) \Rightarrow \text{sqrt-int-floor } (a * b) \text{ div } b)$
unfolding $\text{sqrt-rat-floor-def } \text{root-rat-floor-def}$ **by** (*simp add: sqrt-def*)

lemma $\text{sqrt-rat-floor}[simp]$: $\text{sqrt-rat-floor } x = \lfloor \text{sqrt } (\text{of-rat } x) \rfloor$
unfolding $\text{sqrt-rat-floor-def}$ **by** (*simp add: sqrt-def*)

definition $\text{sqrt-rat-ceiling} :: \text{rat} \Rightarrow \text{int}$ **where**
 $\text{sqrt-rat-ceiling } x = \text{root-rat-ceiling } 2\ x$

lemma $\text{sqrt-rat-ceiling-code}[code]$: $\text{sqrt-rat-ceiling } x = -(\text{sqrt-rat-floor } (-x))$
unfolding $\text{sqrt-rat-ceiling-def } \text{sqrt-rat-floor-def } \text{root-rat-ceiling-def}$ **by** *simp*

lemma sqrt-rat-ceiling : $\text{sqrt-rat-ceiling } x = \lceil \text{sqrt } (\text{of-rat } x) \rceil$
unfolding $\text{sqrt-rat-ceiling-def}$ **by** (*simp add: sqrt-def*)

lemma sqrt-rat-of-int : **assumes** x : $x * x = \text{rat-of-int } i$
shows $\exists j :: \text{int}. j * j = i$

proof –
from x **have** mem : $x \in \text{set } (\text{sqrt-rat } (\text{rat-of-int } i))$ **by** *simp*
from x **have** $\text{rat-of-int } i \geq 0$ **by** (*metis zero-le-square*)
hence $*$: $\text{quotient-of } (\text{rat-of-int } i) = (i,1)$ **by** (*metis quotient-of-int*)
have 1 : $\text{sqrt-int } 1 = [1,-1]$ **by** *code-simp*
from mem $\text{sqrt-rat-code } *$ *split 1*
have x : $x \in \text{rat-of-int } \{y. y * y = i\}$ **by** *auto*
thus *?thesis by auto*
qed

4.5 Approximating square roots

The difference to the previous algorithms is that now we abort, once the distance is below ϵ . Moreover, here we use standard division and not integer division. This part is not yet generalized by *Sqrt-Babylonian.NthRoot-Impl*.

We first provide the executable version without guard $(0::'a) < x$ as partial function, and afterwards prove termination and soundness for a similar algorithm that is defined within the upcoming locale.

```
partial-function (tailrec) sqrt-approx-main-impl :: 'a :: linordered-field  $\Rightarrow$  'a  $\Rightarrow$ 
'a  $\Rightarrow$  'a where
  [code]: sqrt-approx-main-impl  $\epsilon$  n x = (if x * x - n <  $\epsilon$  then x else sqrt-approx-main-impl
 $\epsilon$  n
  ((n / x + x) / 2))
```

We setup a locale where we ensure that we have standard assumptions: positive ϵ and positive n . We require sort *floor-ceiling*, since $\lfloor x \rfloor$ is used for the termination argument.

```
locale sqrt-approximation =
  fixes  $\epsilon$  :: 'a :: {linordered-field,floor-ceiling}
  and n :: 'a
  assumes  $\epsilon$  :  $\epsilon > 0$ 
  and n:  $n > 0$ 
begin
```

```
function sqrt-approx-main :: 'a  $\Rightarrow$  'a where
  sqrt-approx-main x = (if x > 0 then (if x * x - n <  $\epsilon$  then x else sqrt-approx-main
  ((n / x + x) / 2)) else 0)
  by pat-completeness auto
```

Termination essentially is a proof of convergence. Here, one complication is the fact that the limit is not always defined. E.g., if $'a$ is *rat* then there is no square root of 2. Therefore, the error-rate $\frac{x}{\sqrt{n}} - 1$ is not expressible. Instead we use the expression $\frac{x^2}{n} - 1$ as error-rate which does not require any square-root operation.

```
termination
proof -
  define er where er x = (x * x / n - 1) for x
  define c where c = 2 * n /  $\epsilon$ 
  define m where m x = nat  $\lfloor c * er\ x \rfloor$  for x
  have c:  $c > 0$  unfolding c-def using n  $\epsilon$  by auto
  show ?thesis
proof
  show wf (measures [m]) by simp
next
  fix x
  assume x:  $0 < x$  and xe:  $\neg x * x - n < \epsilon$ 
```

```

define y where y = (n / x + x) / 2
show ((n / x + x) / 2, x) ∈ measures [m] unfolding y-def[symmetric]
proof (rule measures-less)
  from n have inv-n: 1 / n > 0 by auto
  from xe have x * x - n ≥ ε by simp
  from this[unfolded mult-le-cancel-left-pos[OF inv-n, of ε, symmetric]]
  have erxen: er x ≥ ε / n unfolding er-def using n by (simp add: field-simps)
  have en: ε / n > 0 and ne: n / ε > 0 using ε n by auto
  from en erxen have erx: er x > 0 by linarith
  have pos: er x * 4 + er x * (er x * 4) > 0 using erx
    by (auto intro: add-pos-nonneg)
  have er y = 1 / 4 * (n / (x * x) - 2 + x * x / n) unfolding er-def y-def
using x n
  by (simp add: field-simps)
  also have ... = 1 / 4 * er x * er x / (1 + er x) unfolding er-def using x n
  by (simp add: field-simps)
  finally have er y = 1 / 4 * er x * er x / (1 + er x) .
  also have ... < 1 / 4 * (1 + er x) * er x / (1 + er x) using erx erx pos
  by (auto simp: field-simps)
  also have ... = er x / 4 using erx by (simp add: field-simps)
  finally have er-y-x: er y ≤ er x / 4 by linarith
  from erxen have c * er x ≥ 2 unfolding c-def mult-le-cancel-left-pos[OF ne,
of - er x, symmetric]
  using n ε by (auto simp: field-simps)
  hence pos: ⌊c * er x⌋ > 0 ⌊c * er x⌋ ≥ 2 by auto
  show m y < m x unfolding m-def nat-mono-iff[OF pos(1)]
  proof -
    have ⌊c * er y⌋ ≤ ⌊c * (er x / 4)⌋
      by (rule floor-mono, unfold mult-le-cancel-left-pos[OF c], rule er-y-x)
    also have ... < ⌊c * er x / 4 + 1⌋ by auto
    also have ... ≤ ⌊c * er x⌋
      by (rule floor-mono, insert pos(2), simp add: field-simps)
    finally show ⌊c * er y⌋ < ⌊c * er x⌋ .
  qed
qed
qed
qed
qed

```

Once termination is proven, it is easy to show equivalence of *sqrt-approx-main-impl* and *sqrt-approx-main*.

```

lemma sqrt-approx-main-impl: x > 0 ⇒ sqrt-approx-main-impl ε n x = sqrt-approx-main
x
proof (induct x rule: sqrt-approx-main.induct)
  case (1 x)
  hence x: x > 0 by auto
  hence nx: 0 < (n / x + x) / 2 using n by (auto intro: pos-add-strict)
  note simps = sqrt-approx-main-impl.simps[of - x] sqrt-approx-main.simps[of x]
  show ?case
  proof (cases x * x - n < ε)

```

```

    case True
    thus ?thesis unfolding_simps using x by auto
  next
    case False
    show ?thesis using 1(1)[OF x False nx] unfolding_simps using x False by
  auto
  qed
  qed

```

Also soundness is not complicated.

```

lemma sqrt-approx-main-sound: assumes x: x > 0 and xx: x * x > n
  shows sqrt-approx-main x * sqrt-approx-main x > n ∧ sqrt-approx-main x *
sqrt-approx-main x - n < ε
  using assms
proof (induct x rule: sqrt-approx-main.induct)
  case (1 x)
  from 1 have x: x > 0 (x > 0) = True by auto
  note simp = sqrt-approx-main_simps[of x, unfolded x if-True]
  show ?case
  proof (cases x * x - n < ε)
    case True
    with 1 show ?thesis unfolding simp by simp
  next
    case False
    let ?y = (n / x + x) / 2
    from False simp have simp: sqrt-approx-main x = sqrt-approx-main ?y by
  simp
  from n x have y: ?y > 0 by (auto intro: pos-add-strict)
  note IH = 1(1)[OF x(1) False y]
  from x have x4: 4 * x * x > 0 by (auto intro: mult-sign-intros)
  show ?thesis unfolding simp
  proof (rule IH)
    show n < ?y * ?y
      unfolding mult-less-cancel-left-pos[OF x4, of n, symmetric]
    proof -
      have id: 4 * x * x * (?y * ?y) = 4 * x * x * n + (n - x * x) * (n - x *
  x) using x(1)
        by (simp add: field_simps)
      from 1(3) have x * x - n > 0 by auto
      from mult-pos-pos[OF this this]
      show 4 * x * x * n < 4 * x * x * (?y * ?y) unfolding id
        by (simp add: field_simps)
    qed
  qed
  qed
  qed
  qed
end

```

It remains to assemble everything into one algorithm.

definition *sqrt-approx* :: 'a :: {linordered-field, floor-ceiling} \Rightarrow 'a \Rightarrow 'a **where**
sqrt-approx ε x \equiv if $\varepsilon > 0$ then (if x = 0 then 0 else let xpos = abs x in
sqrt-approx-main-impl ε xpos (xpos + 1)) else 0

lemma *sqrt-approx*: **assumes** $\varepsilon: \varepsilon > 0$
shows $|\text{sqrt-approx } \varepsilon x * \text{sqrt-approx } \varepsilon x - |x|| < \varepsilon$
proof (cases x = 0)
 case True
 with ε **show** ?thesis **unfolding** *sqrt-approx-def* **by** auto
next
 case False
 let ?x = |x|
 let ?sqrti = *sqrt-approx-main-impl* ε ?x (?x + 1)
 let ?sqrt = *sqrt-approximation.sqrt-approx-main* ε ?x (?x + 1)
 define sqrt **where** sqrt = ?sqrt
 from False **have** x: ?x > 0 ?x + 1 > 0 **by** auto
 interpret *sqrt-approximation* ε ?x
 by (unfold-locales, insert x ε , auto)
 from False ε **have** *sqrt-approx* ε x = ?sqrti **unfolding** *sqrt-approx-def* **by** (simp
add: Let-def)
 also **have** ?sqrti = ?sqrt
 by (rule *sqrt-approx-main-impl*, auto)
 finally **have** id: *sqrt-approx* ε x = sqrt **unfolding** *sqrt-def* .
 have sqrt: sqrt * sqrt > ?x \wedge sqrt * sqrt - ?x < ε **unfolding** *sqrt-def*
 by (rule *sqrt-approx-main-sound*[OF x(2)], insert x *mult-pos-pos*[OF x(1) x(1)],
auto simp: field-simps)
 show ?thesis **unfolding** id **using** sqrt **by** auto
qed

4.6 Some tests

Testing executability and show that sqrt 2 is irrational

lemma $\neg (\exists i :: \text{rat. } i * i = 2)$
proof -
 have set (*sqrt-rat* 2) = {} **by** eval
 thus ?thesis **by** simp
qed

Testing speed

lemma $\neg (\exists i :: \text{int. } i * i = 12345678901234567890123456789012345678901234567890)$
proof -
 have set (*sqrt-int* 12345678901234567890123456789012345678901234567890) = {} **by** eval
 thus ?thesis **by** simp
qed

The following test

value let $\varepsilon = 1 / 100000000 :: \text{rat}$; s = *sqrt-approx* ε 2 in (s, s * s - 2, |s * s - 2| < ε)

results in (1.4142135623731116, 4.738200762148612e-14, True).
end

Acknowledgements

We thank Bertram Felgenhauer for mentioning Cauchy's mean theorem during the formalization of the algorithms for computing n-th roots.

References

- [1] T. Heath. *A History of Greek Mathematics*, volume 2, pages 323–326. Clarendon Press, 1921.
- [2] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468, 2009.