

# Solving Cubic and Quartic Equations\*

René Thiemann

March 19, 2025

## Abstract

We formalize Cardano's formula to solve a cubic equation

$$ax^3 + bx^2 + cx + d = 0,$$

as well as Ferrari's formula to solve a quartic equation [1]. We further turn both formulas into executable algorithms based on the algebraic number implementation in the AFP [2]. To this end we also slightly extended this library, namely by making the minimal polynomial of an algebraic number executable, and by defining and implementing  $n$ -th roots of complex numbers.

## Contents

<b>1</b>	<b>Ferrari's formula for solving quartic equations</b>	<b>2</b>
1.1	Translation to depressed case . . . . .	2
1.2	Solving the depressed case via Ferrari's formula . . . . .	2
<b>2</b>	<b>Cardano's formula for solving cubic equations</b>	<b>4</b>
2.1	Translation to depressed case . . . . .	4
2.2	Solving the depressed case in arbitrary fields . . . . .	4
2.3	Solving the depressed case for complex numbers . . . . .	5
2.4	Solving the depressed case for real numbers . . . . .	7
<b>3</b>	<b><math>n</math>-th roots of complex numbers</b>	<b>11</b>
3.1	An algorithm to compute all complex roots of (algebraic) complex numbers . . . . .	11
3.2	A definition of <i>the</i> complex root of a complex number . . . .	13
<b>4</b>	<b>Algorithms to compute all complex and real roots of a cubic polynomial</b>	<b>22</b>

---

\*Supported by FWF (Austrian Science Fund) project Y757.

## 1 Ferrari's formula for solving quartic equations

**theory** *Ferraris-Formula*

**imports**

*Polynomial-Factorization.Explicit-Roots*  
*Polynomial-Interpolation.Ring-Hom-Poly*  
*Complex-Geometry.More-Complex*

**begin**

### 1.1 Translation to depressed case

Solving an arbitrary quartic equation can easily be turned into the depressed case, i.e., where there is no cubic part.

**lemma** *to-depressed-quartic*: **fixes**  $a_4 :: 'a :: \text{field-char-0}$

**assumes**  $a_4: a_4 \neq 0$

**and**  $b: b = a_3 / a_4$

**and**  $c: c = a_2 / a_4$

**and**  $d: d = a_1 / a_4$

**and**  $e: e = a_0 / a_4$

**and**  $p: p = c - (3/8) * b^2$

**and**  $q: q = (b^3 - 4*b*c + 8 * d) / 8$

**and**  $r: r = (-3 * b^4 + 256 * e - 64 * b * d + 16 * b^2 * c) / 256$

**and**  $x: x = y - b/4$

**shows**  $a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0 = 0$

$\longleftrightarrow y^4 + p * y^2 + q * y + r = 0$

**proof** -

**have**  $a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0 = 0 \longleftrightarrow$

$(a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0) / a_4 = 0$  **using**  $a_4$  **by**

*auto*

**also have**  $(a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0) / a_4$

$= x^4 + b * x^3 + c * x^2 + d * x + e$

**unfolding**  $b c d e$  **using**  $a_4$  **by** (*simp add: field-simps*)

**also have**  $\dots = y^4 + p * y^2 + q * y + r$

**unfolding**  $x p q r$

**by** (*simp add: field-simps power4-eq-xxxx power3-eq-cube power2-eq-square*)

**finally show** *?thesis* .

**qed**

**lemma** *biquadratic-solution*: **fixes**  $p q :: 'a :: \text{field-char-0}$

**shows**  $y^4 + p * y^2 + q = 0 \longleftrightarrow (\exists z. z^2 + p * z + q = 0 \wedge z = y^2)$

**by** (*auto simp: field-simps power4-eq-xxxx power2-eq-square*)

### 1.2 Solving the depressed case via Ferrari's formula

**lemma** *depressed-quartic-Ferrari*: **fixes**  $p q r :: 'a :: \text{field-char-0}$

**assumes** *cubic-root*:  $8 * m^3 + (8 * p) * m^2 + (2 * p^2 - 8 * r) * m - q^2 = 0$   
**and** *q0*:  $q \neq 0$  — otherwise  $m$  might be zero, so  $a$  is zero and then there is a division by zero in *b1* and *b2*  
**and** *sqrt*:  $a * a = 2 * m$   
**and** *b1*:  $b1 = p / 2 + m - q / (2 * a)$   
**and** *b2*:  $b2 = p / 2 + m + q / (2 * a)$   
**shows**  $y^4 + p * y^2 + q * y + r = 0 \iff \text{poly } [:b1, a, 1:] y = 0 \vee \text{poly } [:b2, -a, 1:] y = 0$   
**proof** —  
**let**  $?N = y^2 + p / 2 + m$   
**let**  $?M = a * y - q / (2 * a)$   
**from** *cubic-root* *q0* **have** *m0*:  $m \neq 0$  **by** *auto*  
**from** *sqrt* *m0* **have** *a0*:  $a \neq 0$  **by** *auto*  
**define** *N* **where**  $N = ?N$   
**define** *M* **where**  $M = ?M$   
**note** *powers* = *field-simps power4-eq-xxxx power3-eq-cube power2-eq-square*  
**from** *cubic-root* **have**  $8 * m^3 = -(8 * p) * m^2 - (2 * p^2 - 8 * r) * m + q^2$   
**by** (*simp add: powers*)  
**from** *arg-cong[OF this, of (\*) 4]*  
**have** *id*:  $32 * m^3 = 4 * (- (8 * p) * m^2 - (2 * p^2 - 8 * r) * m + q^2)$   
**by** *simp*  
**let**  $?add = 2 * y^2 * m + p * m + m^2$   
**have**  $y^4 + p * y^2 + q * y + r = 0 \iff (y^2 + p / 2)^2 = -q * y - r + p^2 / 4$   
**by** (*simp add: powers, algebra*)  
**also** **have**  $\dots \iff (y^2 + p / 2)^2 + ?add = -q * y - r + p^2 / 4 + ?add$   
**by** *simp*  
**also** **have**  $\dots \iff ?N^2 = 2 * m * y^2 - q * y + m^2 + m * p + p^2 / 4 - r$   
**by** (*simp add: powers*)  
**also** **have**  $2 * m * y^2 - q * y + m^2 + m * p + p^2 / 4 - r = ?M^2$  **using** *m0 id a0 sqrt* **by** (*simp add: powers, algebra*)  
**also** **have**  $?N^2 = ?M^2 \iff (?N + ?M) * (?N - ?M) = 0$   
**unfolding** *N-def[symmetric] M-def[symmetric]* **by** *algebra*  
**also** **have**  $\dots \iff ?N + ?M = 0 \vee ?N - ?M = 0$  **by** *simp*  
**also** **have**  $?N + ?M = y^2 + a * y + b1$   
**by** (*simp add: b1*)  
**also** **have**  $?N - ?M = y^2 - a * y + b2$   
**by** (*simp add: b2*)  
**also** **have**  $y^2 + a * y + b1 = 0 \iff \text{poly } [:b1, a, 1:] y = 0$   
**by** (*simp add: powers*)  
**also** **have**  $y^2 - a * y + b2 = 0 \iff \text{poly } [:b2, -a, 1:] y = 0$   
**by** (*simp add: powers*)  
**finally** **show** *?thesis* .  
**qed**

end

## 2 Cardano's formula for solving cubic equations

theory *Cardanos-Formula*

imports

*Polynomial-Factorization.Explicit-Roots*

*Polynomial-Interpolation.Ring-Hom-Poly*

*Complex-Geometry.More-Complex*

*Algebraic-Numbers.Complex-Roots-Real-Poly*

begin

### 2.1 Translation to depressed case

Solving an arbitrary cubic equation can easily be turned into the depressed case, i.e., where there is no quadratic part.

**lemma** *to-depressed-cubic*: fixes  $a :: 'a :: \text{field-char-0}$

assumes  $a: a \neq 0$

and  $xy: x = y - b / (3 * a)$

and  $e: e = (c - b^2 / (3 * a)) / a$

and  $f: f = (d + 2 * b^3 / (27 * a^2) - b * c / (3 * a)) / a$

shows  $(a * x^3 + b * x^2 + c * x + d = 0) \longleftrightarrow y^3 + e * y + f = 0$

**proof** -

let  $?yexp = y^3 + e * y + f$

have  $a * x^3 + b * x^2 + c * x + d = 0 \longleftrightarrow (a * x^3 + b * x^2 + c * x + d) / a = 0$

using  $a$  by *auto*

also have  $(a * x^3 + b * x^2 + c * x + d) / a = ?yexp$  **unfolding**  $xy\ e\ f$   
*power3-eq-cube power2-eq-square using a*

by (*simp add: field-simps*)

finally show *?thesis* .

qed

### 2.2 Solving the depressed case in arbitrary fields

**lemma** *cubic-depressed*: fixes  $e :: 'a :: \text{field-char-0}$

assumes  $yz: e \neq 0 \implies z^2 - y * z - e / 3 = 0$

and  $u: e \neq 0 \implies u = z^3$

and  $v: v = -(e^3 / 27)$

shows  $y^3 + e * y + f = 0 \longleftrightarrow (\text{if } e = 0 \text{ then } y^3 = -f \text{ else } u^2 + f * u + v = 0)$

**proof** -

let  $?yexp = y^3 + e * y + f$

show *?thesis*

**proof** (*cases e = 0*)

case *False*

note  $yz = yz[OF\ False]$

from  $yz$  have  $eyz: e = 3 * (z^2 - y * z)$  **by auto**

```

from  $yz$  False have  $z0: z \neq 0$  by auto
have  $?yexp = 0 \iff z^3 * ?yexp = 0$  using  $z0$  by simp
also have  $z^3 * ?yexp = z^6 + f * z^3 - e^3/27$  unfolding  $eyz$  by algebra
also have  $\dots = u^2 + f * u + v$  unfolding  $u[OF\ False]$   $v$  by algebra
finally show  $?thesis$  using False by auto
next
  case True
  show  $?thesis$  unfolding True by (auto, algebra)
qed
qed

```

## 2.3 Solving the depressed case for complex numbers

In the complex-numbers-case, the quadratic equation for  $u$  is always solvable, and the main challenge here is prove that it does not matter which solution of the quadratic equation is considered (this is the `diff:False` case in the proof below.)

```

lemma solve-cubic-depressed-Cardano-complex: fixes  $e :: complex$ 
  assumes  $e0: e \neq 0$ 
  and  $v: v = -(e^3 / 27)$ 
  and  $u: u^2 + f * u + v = 0$ 
shows  $y^3 + e * y + f = 0 \iff (\exists z. z^3 = u \wedge y = z - e / (3 * z))$ 
proof -
  from  $v\ e0$  have  $v0: v \neq 0$  by auto
  from  $e0$  have (if  $e = 0$  then  $x$  else  $y$ ) =  $y$  for  $x\ y :: bool$  by auto
  note  $main = cubic-depressed[OF\ -\ v, unfolded\ this]$ 
  show  $?thesis$  (is  $?l = ?r$ )
  proof
    assume  $?r$ 
    then obtain  $z$  where  $z: z^3 = u$  and  $y: y = z - e / (3 * z)$  by auto
    from  $u\ v0$  have  $u0: u \neq 0$  by auto
    from  $z\ u0$  have  $z0: z \neq 0$  by auto
    show  $?l$ 
    proof (subst main)
      show  $u^2 + f * u + v = 0$  by fact
      show  $u = z^3$  unfolding  $z$  by simp
      show  $z^2 - y * z - e / 3 = 0$  unfolding  $y$  using  $z0$ 
      by (auto simp: field-simps power2-eq-square)
    qed
  next
    assume  $?l$ 
    let  $?yexp = y^3 + e * y + f$ 
    have  $y0: ?yexp = 0$  using  $\langle ?l \rangle$  by auto
    define  $p$  where  $p = [-e/3, -y, 1:]$ 
    have  $deg: degree\ p = 2$  unfolding  $p-def$  by auto
    define  $z$  where  $z = hd\ (roots2\ p)$ 
    have  $z \in set\ (roots2\ p)$  unfolding  $roots2-def$   $Let-def\ z-def$  by auto
    with  $roots2[OF\ deg]$  have  $pz: poly\ p\ z = 0$  by auto

```

```

from pz e0 have z0:  $z \neq 0$  unfolding p-def by auto
from pz have yz:  $y * z = z * z - e / \mathfrak{z}$  unfolding p-def by (auto simp:
field-simps)
from arg-cong[OF this, of  $\lambda x. x / z$ ] z0 have  $y = z - e / (\mathfrak{z} * z)$ 
by (auto simp: field-simps)
have  $\exists u z. u^2 + f * u + v = 0 \wedge z^{\mathfrak{z}} = u \wedge y = z - e / (\mathfrak{z} * z)$ 
proof (intro exI conjI)
  show  $y = z - e / (\mathfrak{z} * z)$  by fact
  from y0 have  $0 = ?yexp * z^{\mathfrak{z}}$  by auto
  also have  $\dots = (y * z)^{\mathfrak{z}} + e * (y * z) * z^{\mathfrak{z}-2} + f * z^{\mathfrak{z}}$  by algebra
  also have  $\dots = (z^{\mathfrak{z}})^{\mathfrak{z}} + f * (z^{\mathfrak{z}}) + v$  unfolding yz v by algebra
  finally show  $(z^{\mathfrak{z}})^{\mathfrak{z}} + f * (z^{\mathfrak{z}}) + v = 0$  by simp
qed simp
then obtain uu z where
  *:  $uu^2 + f * uu + v = 0$   $z^{\mathfrak{z}} = uu$   $y = z - e / (\mathfrak{z} * z)$  by blast
  show ?r
proof (cases uu = u)
  case True
  thus ?thesis using * by auto
next
  case diff: False
  define p where  $p = [:v, f, 1:]$ 
  have p2: degree p = 2 unfolding p-def by auto
  have poly: poly p u = 0 poly p uu = 0 using u *(1) unfolding p-def
by (auto simp: field-simps power2-eq-square)
  have u0:  $u \neq 0$   $uu \neq 0$  using poly v0 unfolding p-def by auto
  {
    from poly(1) have  $[: -u, 1:]$  dvd p by (meson poly-eq-0-iff-dvd)
    then obtain q where  $pq: p = q * [: -u, 1:]$  by auto
    from poly(2)[unfolded pq poly-mult] diff have poly q uu = 0 by auto
    hence  $[: -uu, 1:]$  dvd q by (meson poly-eq-0-iff-dvd)
    then obtain q' where  $qq': q = q' * [: -uu, 1:]$  by auto
    with pq have  $pq: p = q' * [: -uu, 1:] * [: -u, 1:]$  by auto
    from pq[unfolded p-def] have q':  $q' \neq 0$  by auto
    from arg-cong[OF pq, of degree, unfolded p2]
    have  $2 = \text{degree } (q' * [: -uu, 1:] * [: -u, 1:])$  .
    also have  $\dots = \text{degree } q' + \text{degree } [: -uu, 1:] + \text{degree } [: -u, 1:]$ 
      apply (subst degree-mult-eq)
      subgoal using q' by (metis mult-eq-0-iff pCons-eq-0-iff zero-neq-one)
      subgoal by force
      by (subst degree-mult-eq[OF q'], auto)
    also have  $\dots = \text{degree } q' + 2$  by simp
    finally have dq: degree q' = 0 by simp
    from dq obtain c where  $q': q' = [: c:]$  by (metis degree-eq-zeroE)
    from pq[unfolded q' p-def] have c = 1 by auto
    with q' have  $q' = 1$  by simp
    with pq have  $[: -u, 1:] * [: -uu, 1 :] = p$  by simp
  }
from this[unfolded p-def, simplified] have prod:  $uu * u = v$  by simp

```

```

    hence uu: u = v / uu using u0 by (simp add: field-simps)
    define zz where zz = - e / (3 * z)
    show ?r using *(2-) uu unfolding v using u0
      by (intro exI[of - zz], auto simp: zz-def field-simps)
  qed
qed
qed

```

## 2.4 Solving the depressed case for real numbers

**definition** *discriminant-cubic-depressed* :: 'a :: comm-ring-1  $\Rightarrow$  'a  $\Rightarrow$  'a where  
*discriminant-cubic-depressed* e f =  $-(4 * e^3 + 27 * f^2)$

**lemma** *discriminant-cubic-depressed*: assumes  $[-x,1:] * [-y,1:] * [-z,1:] = [-f,e,0,1:]$   
 shows *discriminant-cubic-depressed* e f =  $(x-y)^2 * (x-z)^2 * (y-z)^2$   
**proof** –  
 from *assms* have f:  $f = -(z * (y * x))$  and e:  $e = y * x - z * (-y - x)$  and  
 z:  $z = -y - x$  by *auto*  
 show *thesis* unfolding *discriminant-cubic-depressed-def* e f z  
 by (simp add: *power2-eq-square power3-eq-cube field-simps*)  
**qed**

If the discriminant is negative, then there is exactly one real root

**lemma** *solve-cubic-depressed-Cardano-real*: fixes e f v u :: real  
 defines  $y1 \equiv \text{root } 3 \ u - e / (3 * \text{root } 3 \ u)$   
 and  $\Delta \equiv \text{discriminant-cubic-depressed } e \ f$   
 assumes e0:  $e \neq 0$   
 and v:  $v = -(e^3 / 27)$   
 and u:  $u^2 + f * u + v = 0$   
 shows  $y1^3 + e * y1 + f = 0$   
 $\Delta \neq 0 \implies y^3 + e * y + f = 0 \implies y = y1$   
**proof** –  
 let ?c = *complex-of-real*  
 let ?y = ?c y  
 let ?e = ?c e  
 let ?u = ?c u  
 let ?v = ?c v  
 let ?f = ?c f  
 {  
 fix y :: real  
 let ?y = ?c y  
 have  $y^3 + e * y + f = 0 \iff ?c (y^3 + e * y + f) = ?c 0$   
 using *of-real-eq-iff* by *blast*  
 also have  $\dots \iff ?y^3 + ?e * ?y + ?f = 0$  by *simp*  
 also have  $\dots \iff (\exists z. z^3 = ?u \wedge ?y = z - ?e / (3 * z))$   
**proof** (*rule solve-cubic-depressed-Cardano-complex*)  
 show  $?e \neq 0$  using e0 by *auto*  
 show  $?v = -(?e^3 / 27)$  unfolding v by *simp*
 }

```

    show ?u2 + ?f * ?u + ?v = 0 using arg-cong[OF u, of ?c] by simp
  qed
  finally have y3 + e * y + f = 0  $\longleftrightarrow$  ( $\exists z. z^3 = ?u \wedge ?y = z - ?e / (3 * z)$ ) .
} note pre = this
show y1: y13 + e * y1 + f = 0 unfolding pre y1-def
  by (intro exI[of - ?c (root 3 u)], simp only: of-real-power[symmetric],
    simp del: of-real-power add: odd-real-root-pow)
from u have {z. poly [:v,f,1:] z = 0}  $\neq$  {}
  by (auto simp add: field-simps power2-eq-square)
hence set (rroots2 [:v,f,1:])  $\neq$  {}
  by (subst rroots2[symmetric], auto)
hence rroots2 [:v,f,1:]  $\neq$  [] by simp
from this[unfolded rroots2-def Let-def, simplified]
have f2 - 4 * v  $\geq$  0
  by (auto split: if-splits simp: numeral-2-eq-2 field-simps power2-eq-square)
hence delta-le-0:  $\Delta \leq 0$  unfolding  $\Delta$ -def discriminant-cubic-depressed-def v by
auto

```

```

assume Delta-non-0:  $\Delta \neq 0$ 
with delta-le-0 have delta-neg:  $\Delta < 0$  by simp
let ?p = [:f,e,0,1:]
have poly: poly ?p y = 0  $\longleftrightarrow$  y3 + e * y + f = 0 for y
  by (simp add: field-simps power2-eq-square power3-eq-cube)
from y1 have poly ?p y1 = 0 unfolding poly .
hence [-y1,1:] dvd ?p using poly-eq-0-iff-dvd by blast
then obtain q where pq: ?p = [-y1,1:] * q by blast
{
  fix y2
  assume poly ?p y2 = 0 y2  $\neq$  y1
  from this[unfolded pq] poly-mult have poly q y2 = 0 by auto
  from this[unfolded poly-eq-0-iff-dvd] obtain r where qr: q = [-y2,1:] * r by
blast
{
  have r0: r  $\neq$  0 using pq unfolding qr poly-mult by auto
  have 3 = degree ?p by simp
  also have ... = 2 + degree r unfolding pq qr
    apply (subst degree-mult-eq, force)
    subgoal using r0 pq qr by force
    by (subst degree-mult-eq[OF - r0], auto)
  finally have degree r = 1 by simp
  from degree1-coeffs[OF this] obtain yy a where r: r = [:yy,a:] by metis
  define y3 where y3 = -yy
  with r have r: r = [-y3,a:] by auto
  from pq[unfolded qr r] have a = 1 by auto
  with r have  $\exists y3. r = [-y3,1:]$  by auto
}
}
then obtain y3 where r: r = [-y3,1:] by auto
have py: ?p = [-y1,1:] * [-y2,1:] * [-y3,1:] unfolding pq qr r by algebra

```



```

    from discriminant-cubic-depressed[OF this[symmetric], folded  $\Delta$ -def]
    have delta:  $\Delta = (y1 - y2)^2 * (y1 - y3)^2 * (y2 - y3)^2$  .
    have d0:  $\Delta \geq 0$  unfolding delta by auto
    with delta-neg have False by auto
  }
  with y1 show  $y^3 + e * y + f = 0 \implies y = y1$  unfolding poly by auto
qed

```

If the discriminant is non-negative, then all roots are real

**lemma** *solve-cubic-depressed-Cardano-all-real-roots*: fixes  $e f v :: \text{real}$  and  $y :: \text{complex}$

```

  defines  $\Delta \equiv \text{discriminant-cubic-depressed } e f$ 
  assumes Delta:  $\Delta \geq 0$ 
  and rt:  $y^3 + e * y + f = 0$ 
shows  $y \in \mathbb{R}$ 
proof -
  note powers = field-simps power3-eq-cube power2-eq-square
  let ?c = complex-of-real
  let ?e = ?c e
  let ?f = ?c f
  let ?cp = [?:?f, ?e, 0, 1:]
  let ?p = [?:f, e, 0, 1:]
  from odd-degree-imp-real-root[of ?p] obtain x1 where poly ?p x1 = 0 by auto
  hence [:-x1, 1:] dvd ?p using poly-eq-0-iff-dvd by blast
  then obtain q where pq: ?p = [:-x1, 1:] * q by auto
  from arg-cong[OF pq, of degree]
  have  $\mathfrak{3} = \text{degree} ([:-x1, 1:] * q)$  by simp
  also have  $\dots = 1 + \text{degree } q$ 
  by (subst degree-mult-eq, insert pq, auto)
  finally have dq: degree q = 2 by auto
  let ?cc = map-poly ?c
  let ?q = ?cc q
  have cpq: ?cc ?p = ?cc [:-x1, 1:] * ?q unfolding pq hom-distrib by simp
  let ?x1 = ?c x1
  have dq': degree ?q = 2 using dq by simp
  have  $\neg \text{constant } (\text{poly } ?q)$  using dq by (simp add: constant-degree)
  from fundamental-theorem-of-algebra[OF this] obtain x2 where x2: poly ?q x2
= 0 by blast
  have x2  $\in \mathbb{R}$ 
  proof (rule ccontr)
    assume x2r: x2  $\notin \mathbb{R}$ 
    define x3 where x3 = cnj x2
    from x2r have x23: x2  $\neq$  x3 unfolding x3-def using Reals-cnj-iff by force
    have x3: poly ?q x3 = 0 unfolding x3-def
    by (rule complex-conjugate-root[OF - x2], auto)
    from x2[unfolded poly-eq-0-iff-dvd] obtain r where qr: ?q = [:-x2, 1:] * r by
auto
    from arg-cong[OF this[symmetric], of  $\lambda x. \text{poly } x x3$ , unfolded poly-mult x3
mult-eq-0-iff] x23

```

**have**  $x3$ : *poly*  $r$   $x3 = 0$  **by** *auto*  
**from** *arg-cong*[*OF*  $qr$ , *of degree*]  
**have**  $2 = \text{degree } ([: -x2, 1:] * r)$  **using** *dq'* **by** *simp*  
**also have**  $\dots = 1 + \text{degree } r$  **by** (*subst degree-mult-eq, insert pq qr, auto*)  
**finally have**  $\text{degree } r = 1$  **by** *simp*  
**then obtain**  $a$   $b$  **where**  $r: r = [: a, b:]$  **by** (*metis degree1-coeffs*)  
**from** *cpq*[*unfolded qr r*] **have**  $b1: b = 1$  **by** *simp*  
**with**  $x3$   $r$  **have**  $a + x3 = 0$  **by** *simp*  
**hence**  $a = -x3$  **by** *algebra*  
**with**  $b1$   $r$  **have**  $r: r = [: -x3, 1:]$  **by** *auto*  
**have**  $?cc$   $?p = ?cc$   $[: -x1, 1:] * [: -x2, 1:] * [: -x3, 1:]$  **unfolding** *cpq qr r* **by**  
*algebra*  
**also have**  $?cc$   $[: -x1, 1:] = [: -?x1, 1:]$  **by** *simp*  
**also have**  $?cc$   $?p = ?cp$  **by** *simp*  
**finally have** *id*:  $[: -?x1, 1:] * [: -x2, 1:] * [: -x3, 1:] = ?cp$  **by** *simp*  
**define**  $x23$  **where**  $x23 = -4 * (Im\ x2)^2$   
**define**  $x12c$  **where**  $x12c = ?x1 - x2$   
**define**  $x12$  **where**  $x12 = (Re\ x12c)^2 + (Im\ x12c)^2$   
**have**  $x23-0: x23 < 0$  **unfolding**  $x23$ -*def* **using**  $x2r$  **using** *complex-is-Real-iff*  
**by** *force*  
**have**  $Im\ x12c \neq 0$  **unfolding**  $x12c$ -*def* **using**  $x2r$  **using** *complex-is-Real-iff*  
**by** *force*  
**hence**  $(Im\ x12c)^2 > 0$  **by** *simp*  
**hence**  $x12: x12 > 0$  **unfolding**  $x12$ -*def* **using** *sum-power2-gt-zero-iff* **by** *auto*  
**from** *discriminant-cubic-depressed*[*OF id*]  
**have**  $?c$   $\Delta = ((?x1 - x2)^2 * (?x1 - x3)^2) * (x2 - x3)^2$   
**unfolding**  $\Delta$ -*def* *discriminant-cubic-depressed-def* **by** *simp*  
**also have**  $(x2 - x3)^2 = ?c$   $x23$  **unfolding**  $x3$ -*def*  $x23$ -*def* **by** (*simp add:*  
*complex-eq-iff power2-eq-square*)  
**also have**  $(?x1 - x2)^2 * (?x1 - x3)^2 = ((?x1 - x2) * (?x1 - x3))^2$   
**by** (*simp add: power2-eq-square*)  
**also have**  $?x1 - x3 = cnj$   $(?x1 - x2)$  **unfolding**  $x3$ -*def* **by** *simp*  
**also have**  $(?x1 - x2) = x12c$  **unfolding**  $x12c$ -*def* **..**  
**also have**  $x12c * cnj\ x12c = ?c$   $x12$  **by** (*simp add: x12-def complex-eq-iff*  
*power2-eq-square*)  
**finally have**  $?c$   $\Delta = ?c$   $(x12^2 * x23)$  **by** *simp*  
**hence**  $\Delta = x12^2 * x23$  **by** (*rule of-real-hom.injectivity*)  
**also have**  $\dots < 0$  **using**  $x12$   $x23-0$  **by** (*meson mult-pos-neg zero-less-power*)  
**finally show** *False* **using** *Delta* **by** *simp*  
**qed**  
**with**  $x2$  **obtain**  $x2$  **where** *poly*  $?q$   $(?c\ x2) = 0$  **unfolding** *Reals-def* **by** *auto*  
**hence**  $x2: \text{poly } q\ x2 = 0$  **by** *simp*  
**from**  $x2$ [*unfolded poly-eq-0-iff-dvd*] **obtain**  $r$  **where**  $qr: q = [: -x2, 1:] * r$  **by**  
*auto*  
**from** *arg-cong*[*OF*  $qr$ , *of degree*]  
**have**  $2 = \text{degree } ([: -x2, 1:] * r)$  **using** *dq'* **by** *simp*  
**also have**  $\dots = 1 + \text{degree } r$  **by** (*subst degree-mult-eq, insert pq qr, auto*)  
**finally have**  $\text{degree } r = 1$  **by** *simp*  
**then obtain**  $a$   $b$  **where**  $r: r = [: a, b:]$  **by** (*metis degree1-coeffs*)

```

from pq[unfolded qr r] have b1: b = 1 by simp
define x3 where x3 = -a
have r: r = [-x3,1:] unfolding r b1 x3-def by simp
let ?pp = [-x1,1:] * [-x2,1:] * [-x3,1:]
have id: ?p = ?pp unfolding pq qr r by linarith
have True  $\longleftrightarrow$   $y^3 + e * y + f = 0$  using rt by auto
also have  $y^3 + e * y + f = \text{poly } (?cc ?p) y$  by (simp add: powers)
also have ... = poly (?cc ?pp) y unfolding id by simp
also have ?cc ?pp = [-?c x1, 1:] * [-?c x2, 1:] * [-?c x3, 1:]
  by simp
also have poly ... y = 0  $\longleftrightarrow$  y = ?c x1  $\vee$  y = ?c x2  $\vee$  y = ?c x3
  unfolding poly-mult mult-eq-0-iff by auto
finally show y  $\in$   $\mathbb{R}$  by auto
qed

end

```

### 3 $n$ -th roots of complex numbers

**theory** Complex-Roots

**imports**

Complex-Geometry.More-Complex  
 Algebraic-Numbers.Complex-Algebraic-Numbers  
 Factor-Algebraic-Polynomial.Roots-via-IA  
 HOL-Library.Product-Lexorder

**begin**

#### 3.1 An algorithm to compute all complex roots of (algebraic) complex numbers

**definition** all-roots :: nat  $\Rightarrow$  complex  $\Rightarrow$  complex list **where**

```

all-roots n x = (if n = 0 then [] else
  if algebraic x then
    (let p = min-int-poly x;
      q = poly-nth-root n p;
      xs = complex-roots-of-int-poly q
      in filter ( $\lambda y. y^n = x$ ) xs)
  else (SOME ys. set ys = {y.  $y^n = x$ }))

```

**lemma** all-roots: **assumes** n0:  $n \neq 0$  **shows** set (all-roots n x) = {y.  $y^n = x$ }

**proof** (cases algebraic x)

**case** True

**hence** id: (if n = 0 then y else if algebraic x then z else u) = z

**for** y z u :: complex list **using** n0 **by** auto

**define** p **where** p = poly-nth-root n (min-int-poly x)

**show** ?thesis **unfolding** Let-def p-def[symmetric] all-roots-def id

**proof** (standard, force, standard, simp)

**fix** y

```

assume  $y: y \hat{=}^n = x$ 
have min-int-poly  $x$  represents  $x$  using True by auto
from represents-nth-root[OF  $n0$   $y$  this]
have  $p$  represents  $y$  unfolding p-def by auto
thus  $y \in \text{set } (\text{complex-roots-of-int-poly } p)$ 
by (subst complex-roots-of-int-poly, auto)
qed
next
case False
hence id: (if  $n = 0$  then  $y$  else if algebraic  $x$  then  $z$  else  $u$ ) =  $u$ 
for  $y z u :: \text{complex list}$  using  $n0$  by auto
show ?thesis unfolding Let-def all-roots-def id
by (rule someI-ex, rule finite-list, insert n0, blast)
qed

```

TODO: One might change *complex-roots-of-int-poly* to *complex-roots-of-int-poly3* in order to avoid an unnecessary factorization of an integer polynomial. However, then this change already needs to be performed within the definition of *all-roots*.

```

lift-definition all-roots-part1 ::  $\text{nat} \Rightarrow \text{complex} \Rightarrow \text{complex}$  genuine-roots-aux is
 $\lambda n x. \text{if } n = 0 \vee x = 0 \vee \neg \text{algebraic } x \text{ then } (1, [], 0, \text{filter-fun-complex } 1)$ 
  else let  $p = \text{min-int-poly } x;$ 
     $q = \text{poly-nth-root } n p;$ 
     $\text{zeros} = \text{complex-roots-of-int-poly } q;$ 
     $r = \text{Polynomial.monom } 1 n - [:x:]$ 
  in ( $r, \text{zeros}, n, \text{filter-fun-complex } r$ )
subgoal for  $n x$ 
proof (cases  $n = 0 \vee x = 0 \vee \neg \text{algebraic } x$ )
case True
thus ?thesis by (simp add: filter-fun-complex)
next
case False
hence  $*$ : algebraic  $x$   $n \neq 0$   $x \neq 0$  by auto
{
fix  $z$ 
assume  $zn: z \hat{=}^n = x$ 
from  $*(1)$  have repr: min-int-poly  $x$  represents  $x$  by auto
from represents-nth-root[OF  $*(2)$   $zn$  repr]
have poly-nth-root  $n$  (min-int-poly  $x$ ) represents  $z$  .
}
moreover have  $\text{card } \{z. z \hat{=}^n = x\} = n$ 
by (rule card-nth-roots) (use  $*$  in auto)
ultimately show ?thesis using  $*$ 
by (auto simp: Let-def complex-roots-of-int-poly filter-fun-complex poly-monom)
qed
done

```

```

lemma all-roots-code[code]:
 $\text{all-roots } n x = (\text{if } n = 0 \text{ then } [] \text{ else if } x = 0 \text{ then } [0])$ 

```

```

      else if algebraic x then genuine-roots-impl (all-roots-part1 n x)
      else Code.abort (STR "all-roots invoked on non-algebraic number") (λ -.
all-roots n x))
proof (cases n = 0)
  case True
  thus ?thesis unfolding all-roots-def by simp
next
  case n: False
  show ?thesis
  proof (cases x = 0)
    case x: False
    show ?thesis
    proof (cases algebraic x)
      case False
      with n x show ?thesis by simp
    next
    case True
    define t where t = ?thesis
    have t  $\longleftrightarrow$  filter (λy. y ^ n = x)
      (complex-roots-of-int-poly (poly-nth-root n (min-int-poly x)))
      = genuine-roots-impl (all-roots-part1 n x)
    unfolding t-def
    by (subst all-roots-def[of n x], unfold Let-def, insert n x True, auto)
    also have ... using n x True unfolding genuine-roots-impl-def
    by (transfer, simp add: Let-def genuine-roots-def poly-monom)
    finally show ?thesis unfolding t-def by simp
  qed
next
  case x: True
  have set (all-roots n 0) = {0} unfolding all-roots[OF n] using n by simp
  moreover have distinct (all-roots n 0) unfolding all-roots-def using n
    by (auto intro!: distinct-filter complex-roots-of-int-poly)
  ultimately have all-roots n 0 = [0]
    by (smt (verit, del-insts) distinct.simps(2) distinct-singleton insert-ident
list.set-cases list.set-intros(1) list.simps(15) mem-Collect-eq set-empty singleton-conv)
  moreover have ?thesis  $\longleftrightarrow$  all-roots n 0 = [0] using n x by simp
  ultimately show ?thesis by auto
  qed
qed

```

### 3.2 A definition of *the* complex root of a complex number

While the definition of the complex root is quite natural and easy, the main task is a criterion to determine which of all possible roots of a complex number is the chosen one.

**definition** *croot* :: nat  $\Rightarrow$  complex  $\Rightarrow$  complex **where**  
*croot* n x = (rcis (root n (cmod x)) (Arg x / of-nat n))

**lemma** *croot-0*[simp]: *croot* n 0 = 0 *croot* 0 x = 0

**unfolding** *croot-def* **by** *auto*

**lemma** *croot-power*: **assumes**  $n: n \neq 0$   
**shows**  $(\text{croot } n \ x) \wedge n = x$   
**unfolding** *croot-def DeMoivre2*  
**by** (*subst real-root-pow-pos2, insert n, auto simp: rcis-cmod-Arg*)

**lemma** *Arg-of-real*:  $\text{Arg} (\text{of-real } x) =$   
*(if  $x < 0$  then  $\pi$  else  $0$ )*  
**proof** (*cases  $x = 0$* )  
**case** *False*  
**hence**  $x < 0 \vee x > 0$  **by** *auto*  
**thus** *?thesis* **by** (*intro cis-Arg-unique, auto*  
*simp: complex-sgn-def scaleR-complex.ctr complex-eq-iff*)  
**qed** (*auto simp: Arg-def*)

**lemma** *Arg-rcis-cis[simp]*: **assumes**  $x > 0$   
**shows**  $\text{Arg} (\text{rcis } x \ y) = \text{Arg} (\text{cis } y)$   
**using** *assms* **unfolding** *rcis-def* **by** *simp*

**lemma** *cis-Arg-1[simp]*:  $\text{cis} (\text{Arg } 1) = 1$   
**using** *Arg-of-real[of 1]* **by** *simp*

**lemma** *cis-Arg-power[simp]*: **assumes**  $x \neq 0$   
**shows**  $\text{cis} (\text{Arg} (x \wedge n)) = \text{cis} (\text{Arg } x * \text{real } n)$   
**proof** (*induct n*)  
**case** (*Suc n*)  
**show** *?case* **unfolding** *power.simps*  
**proof** (*subst cis-arg-mult*)  
**show**  $\text{cis} (\text{Arg } x + \text{Arg} (x \wedge n)) = \text{cis} (\text{Arg } x * \text{real} (\text{Suc } n))$   
**unfolding** *mult.commute[of Arg x] DeMoivre[symmetric]*  
**unfolding** *power.simps* **using** *Suc*  
**by** (*metis DeMoivre cis-mult mult.commute*)  
**show**  $x * x \wedge n \neq 0$  **using** *assms* **by** *auto*  
**qed**  
**qed** *simp*

**lemma** *Arg-croot[simp]*:  $\text{Arg} (\text{croot } n \ x) = \text{Arg } x / \text{real } n$   
**proof** (*cases  $n = 0 \vee x = 0$* )  
**case** *True*  
**thus** *?thesis* **by** (*auto simp: Arg-def*)  
**next**  
**case** *False*  
**hence**  $n: n \neq 0$  **and**  $x: x \neq 0$  **by** *auto*  
**let** *?root = croot n x*  
**from** *n* **have**  $n1: \text{real } n \geq 1 \ \text{real } n > 0 \ \text{real } n \neq 0$  **by** *auto*  
**have** *bounded*:  $-\pi < \text{Arg } x / \text{real } n \wedge \text{Arg } x / \text{real } n \leq \pi$   
**proof** (*cases  $\text{Arg } x < 0$* )

```

case True
from Arg-bounded[of x] have  $-pi < Arg\ x$  by auto
also have  $\dots \leq Arg\ x / real\ n$  using n1 True
  by (smt (verit) div-by-1 divide-minus-left frac-le)
finally have one:  $-pi < Arg\ x / real\ n$  .
have  $Arg\ x / real\ n \leq 0$  using True n1
  by (smt (verit) divide-less-0-iff)
also have  $\dots \leq pi$  by simp
finally show ?thesis using one by auto
next
case False
hence ax: Arg x ≥ 0 by auto
have  $Arg\ x / real\ n \leq Arg\ x$  using n1 ax
  by (smt (verit) div-by-1 frac-le)
also have  $\dots \leq pi$  using Arg-bounded[of x] by simp
finally have one:  $Arg\ x / real\ n \leq pi$  .
have  $-pi < 0$  by simp
also have  $\dots \leq Arg\ x / real\ n$  using ax n1 by simp
finally show ?thesis using one by auto
qed
have  $Arg\ ?root = Arg\ (cis\ (Arg\ x / real\ n))$ 
  unfolding croot-def using x n by simp
also have  $\dots = Arg\ x / real\ n$ 
  by (rule cis-Arg-unique, force, insert bounded, auto)
finally show ?thesis .
qed

lemma cos-abs[simp]: cos (abs x :: real) = cos x
proof (cases x < 0)
  case True
    hence abs: abs x = - x by simp
    show ?thesis unfolding abs by simp
  qed simp

lemma cos-mono-le: assumes abs x ≤ pi
  and abs y ≤ pi
shows  $cos\ x \leq cos\ y \iff abs\ y \leq abs\ x$ 
proof -
  have  $cos\ x \leq cos\ y \iff cos\ (abs\ x) \leq cos\ (abs\ y)$  by simp
  also have  $\dots \iff abs\ y \leq abs\ x$ 
  by (subst cos-mono-le-eq, insert assms, auto)
  finally show ?thesis .
qed

lemma abs-add-2-mult-bound: fixes x :: 'a :: linordered-idom
  assumes xy: |x| ≤ y
  shows  $|x| \leq |x + 2 * of-int\ i * y|$ 
proof (cases i = 0)
  case i: False

```

```

let ?oi = of-int :: int ⇒ 'a
from xy have y: y ≥ 0 by auto
consider (pp) x ≥ 0 i ≥ 0
  | (nn) x ≤ 0 i ≤ 0
  | (pn) x ≥ 0 i ≤ 0
  | (np) x ≤ 0 i ≥ 0
by linarith
thus ?thesis
proof cases
  case pp
  thus ?thesis using y by simp
next
  case nn
  have x ≥ x + 2 * ?oi i * y
    using nn y by (simp add: mult-nonneg-nonpos2)
  with nn show ?thesis by linarith
next
  case pn
  with i have 0 ≤ x i < 0 by auto
  define j where j = nat (-i) - 1
  define z where z = x - 2 * y
  define u where u = 2 * ?oi (nat j) * y
  have u: u ≥ 0 unfolding u-def using y by auto
  have i: i = - int (Suc j)
    using ⟨i < 0⟩ unfolding j-def by simp
  have id: x + 2 * ?oi i * y = z - u
    unfolding i z-def u-def by (simp add: field-simps)
  have z: z ≤ 0 abs z ≥ x using xy y pn(1)
    unfolding z-def by auto
  show ?thesis unfolding id using pn(1) z u by simp
next
  case np
  with i have 0 ≥ x i > 0 by auto
  define j where j = nat i - 1
  have i: i = int (Suc j)
    using ⟨i > 0⟩ unfolding j-def by simp
  define u where u = 2 * ?oi (nat j) * y
  have u: u ≥ 0 unfolding u-def using y by auto
  define z where z = - x - 2 * y
  have id: x + 2 * ?oi i * y = - z + u
    unfolding i z-def u-def by (simp add: field-simps)
  have z: z ≤ 0 abs z ≥ - x using xy y np(1)
    unfolding z-def by auto
  show ?thesis unfolding id using np(1) z u by simp
qed
qed simp

```

**lemma** abs-eq-add-2-mult: **fixes** y :: 'a :: linordered-idom  
**assumes** abs-id: |x| = |x + 2 \* of-int i \* y|



```

and  $xy: -y < x \ x \leq y$ 
and  $i: i \neq 0$ 
shows  $x = y \wedge i = -1$ 
proof -
  let  $?oi = of-int :: int \Rightarrow 'a$ 
  from  $xy$  have  $y: y > 0$  by auto
  consider  $(pp) \ x \geq 0 \ i \geq 0$ 
    |  $(nn) \ x < 0 \ i \leq 0$ 
    |  $(pn) \ x \geq 0 \ i \leq 0$ 
    |  $(np) \ x < 0 \ i \geq 0$ 
  by linarith
  hence  $?thesis \vee x = ?oi (-i) * y$ 
  proof cases
    case pp
    thus  $?thesis$  using  $y \ abs-id \ xy \ i$  by simp
  next
    case nn
    hence  $|x + 2 * ?oi \ i * y| =$ 
       $- (x + 2 * ?oi \ i * y)$ 
    using  $y \ nn$ 
    by  $(intro \ abs-of-nonpos \ add-nonpos-nonpos,$ 
       $force, \ simp, \ intro \ mult-nonneg-nonpos, \ auto)$ 
    thus  $?thesis$  using  $y \ abs-id \ xy \ i \ nn$ 
    by auto
  next
    case pn
    with  $i$  have  $0 \leq x \ i < 0$  by auto
    define  $j$  where  $j = nat (-i) - 1$ 
    define  $z$  where  $z = x - 2 * y$ 
    define  $u$  where  $u = 2 * ?oi (nat \ j) * y$ 
    have  $u: u \geq 0$  unfolding  $u-def$  using  $y$  by auto
    have  $i: i = -int (Suc \ j)$ 
    using  $\langle i < 0 \rangle$  unfolding  $j-def$  by simp
    have  $id: x + 2 * ?oi \ i * y = z - u$ 
    unfolding  $i \ z-def \ u-def$  by  $(simp \ add: \ field-simps)$ 
    have  $z: z \leq 0 \ abs \ z \geq x$  using  $xy \ y \ pn(1)$ 
    unfolding  $z-def$  by auto
    from  $abs-id[unfolded \ id]$  have  $z - u = -x$ 
    using  $z \ u \ pn$  by auto
    from  $this[folded \ id]$  have  $x = of-int (-i) * y$ 
    by auto
    thus  $?thesis$  by auto
  next
    case np
    with  $i$  have  $0 \geq x \ i > 0$  by auto
    define  $j$  where  $j = nat \ i - 1$ 
    have  $i: i = int (Suc \ j)$ 
    using  $\langle i > 0 \rangle$  unfolding  $j-def$  by simp
    define  $u$  where  $u = 2 * ?oi (nat \ j) * y$ 

```

```

have u: u ≥ 0 unfolding u-def using y by auto
define z where z = - x - 2 * y
have id: x + 2 * ?oi i * y = - z + u
  unfolding i z-def u-def by (simp add: field-simps)
have z: z ≤ 0
  using xy y np(1) unfolding z-def by auto
from abs-id[unfolded id] have - z + u = - x
  using u z np by auto
from this[folded id] have x = of-int (- i) * y
  by auto
thus ?thesis by auto
qed
thus ?thesis
proof
  assume x = ?oi (- i) * y
  with xy i y
  show ?thesis
  by (smt (verit, ccfv-SIG) less-le minus-less-iff mult-le-cancel-right2 mult-minus1-right
mult-minus-left mult-of-int-commute of-int-hom.hom-one of-int-le-1-iff of-int-minus)
qed
qed

```

This is the core lemma. It tells us that *croot* will choose the principal root, i.e. the root with largest real part and if there are two roots with identical real part, then the largest imaginary part. This criterion will be crucial for implementing *croot*.

```

lemma croot-principal: assumes n: n ≠ 0
  and y: y ^ n = x
  and neq: y ≠ croot n x
shows Re y < Re (croot n x) ∨ Re y = Re (croot n x) ∧ Im y < Im (croot n x)
proof (cases x = 0)
  case True
  with neq y have False by auto
  thus ?thesis ..
next
  case x: False
  let ?root = croot n x
  from n have n1: real n ≥ 1 real n > 0 real n ≠ 0 by auto
  from x y n have y0: y ≠ 0 by auto
  from croot-power[OF n, of x] y
  have id: ?root ^ n = y ^ n by simp
  hence cmod (?root ^ n) = cmod (y ^ n) by simp
  hence norm-eq: cmod ?root = cmod y using n unfolding norm-power
  by (meson gr-zeroI norm-ge-zero power-eq-imp-eq-base)
  have cis (Arg y * real n) = cis (Arg (y ^ n)) by (subst cis-Arg-power[OF y0],
simp)
  also have ... = cis (Arg x) using y by simp
  finally have ciseq: cis (Arg y * real n) = cis (Arg x) by simp
  from cis-eq[OF ciseq] obtain i where

```

$Arg\ y * real\ n - Arg\ x = 2 * real-of-int\ i * pi$   
**by auto**  
**hence**  $Arg\ y * real\ n = Arg\ x + 2 * real-of-int\ i * pi$  **by auto**  
**from**  $arg-cong[OF\ this,\ of\ \lambda\ x.\ x / real\ n]$   $n1$   
**have**  $Argy: Arg\ y = Arg\ ?root + 2 * real-of-int\ i * pi / real\ n$   
**by** (*auto simp: field-simps*)  
**have**  $i0: i \neq 0$   
**proof**  
**assume**  $i = 0$   
**hence**  $Arg\ y = Arg\ ?root$  **unfolding**  $Argy$  **by** *simp*  
**with** *norm-eq* **have**  $?root = y$  **by** (*metis rcis-cmod-Arg*)  
**with** *neq* **show** *False* **by** *simp*  
**qed**  
**from**  $y0$  **have**  $cy0: cmod\ y > 0$  **by** *auto*  
**from**  $Arg-bounded[of\ x]$  **have**  $abs-pi: abs\ (Arg\ x) \leq pi$  **by** *auto*  
**have**  $Re\ y \leq Re\ ?root \iff Re\ y / cmod\ y \leq Re\ ?root / cmod\ y$   
**using**  $cy0$  **unfolding** *divide-le-cancel* **by** *simp*  
**also** **have**  $cosy: Re\ y / cmod\ y = cos\ (Arg\ y)$  **unfolding**  $cos-arg[OF\ y0]$  **..**  
**also** **have**  $cosrt: Re\ ?root / cmod\ y = cos\ (Arg\ ?root)$   
**unfolding** *norm-eq[symmetric]* **by** (*subst cos-arg, insert norm-eq cy0, auto*)  
**also** **have**  $cos\ (Arg\ y) \leq cos\ (Arg\ ?root) \iff abs\ (Arg\ ?root) \leq abs\ (Arg\ y)$   
**by** (*rule cos-mono-le, insert Arg-bounded[of\ y] Arg-bounded[of\ ?root], auto*)  
**also** **have**  $\dots \iff abs\ (Arg\ ?root) * real\ n \leq abs\ (Arg\ y) * real\ n$   
**unfolding** *mult-le-cancel-right* **using**  $n1$  **by** *simp*  
**also** **have**  $\dots \iff abs\ (Arg\ x) \leq |Arg\ x + 2 * real-of-int\ i * pi|$   
**unfolding**  $Argy$  **using**  $n1$  **by** (*simp add: field-simps*)  
**also** **have**  $\dots$  **using** *abs-pi*  
**by** (*rule abs-add-2-mult-bound*)  
**finally** **have**  $le: Re\ y \leq Re\ (croot\ n\ x)$  .  
**show** *?thesis*  
**proof** (*cases Re\ y = Re\ (croot\ n\ x)*)  
**case** *False*  
**with**  $le$  **show** *?thesis* **by** *auto*  
**next**  
**case** *True*  
**hence**  $Re\ y / cmod\ y = Re\ ?root / cmod\ y$  **by** *simp*  
**hence**  $cos\ (Arg\ y) = cos\ (Arg\ ?root)$  **unfolding**  $cosy\ cosrt$  .  
**hence**  $cos\ (abs\ (Arg\ y)) = cos\ (abs\ (Arg\ ?root))$  **unfolding** *cos-abs* .  
**from** *cos-inj-pi[OF - - - this]*  
**have**  $abs\ (Arg\ y) = abs\ (Arg\ ?root)$   
**using**  $Arg-bounded[of\ y] Arg-bounded[of\ ?root]$  **by** *auto*  
**hence**  $abs\ (Arg\ y) * real\ n = abs\ (Arg\ ?root) * real\ n$  **by** *simp*  
**hence**  $abs\ (Arg\ x) = |Arg\ x + 2 * real-of-int\ i * pi|$  **unfolding**  $Argy$   
**using**  $n1$  **by** (*simp add: field-simps*)  
**from** *abs-eq-add-2-mult[OF this - - <i>\neq 0</i>]*  $Arg-bounded[of\ x]$   
**have**  $Argx: Arg\ x = pi$  **and**  $i: i = -1$  **by** *auto*  
**have**  $Argy: Arg\ y = -pi / real\ n$   
**unfolding**  $Argy\ Arg-croot\ i\ Argx$  **by** *simp*  
**have**  $Im\ ?root > Im\ y \iff Im\ ?root / cmod\ ?root > Im\ y / cmod\ y$

```

    unfolding norm-eq using cy0
    by (meson divide-less-cancel divide-strict-right-mono)
  also have ...  $\longleftrightarrow \sin (\text{Arg } ?\text{root}) > \sin (\text{Arg } y)$ 
    by (subst (1 2) sin-arg, insert y0 norm-eq, auto)
  also have ...  $\longleftrightarrow \sin (-\pi / \text{real } n) < \sin (\pi / \text{real } n)$ 
    unfolding Argy Arg-croot Argx by simp
  also have ...
  proof -
    have  $\sin (-\pi / \text{real } n) < 0$ 
      using n1 by (smt (verit) Arg-bounded Argy divide-neg-pos sin-gt-zero
        sin-minus)
    also have ...  $< \sin (\pi / \text{real } n)$ 
      using n1 calculation by fastforce
    finally show ?thesis .
  qed
  finally show ?thesis using le by auto
  qed
  qed

```

```

lemma croot-unique: assumes n:  $n \neq 0$ 
  and y:  $y^n = x$ 
  and y-max-Re-Im:  $\bigwedge z. z^n = x \implies \text{Re } z < \text{Re } y \vee \text{Re } z = \text{Re } y \wedge \text{Im } z \leq \text{Im } y$ 
shows  $\text{croot } n x = y$ 
proof (rule ccontr)
  assume  $\text{croot } n x \neq y$ 
  from croot-principal[OF n y this[symmetric]]
  have  $\text{Re } y < \text{Re } (\text{croot } n x) \vee$ 
     $\text{Re } y = \text{Re } (\text{croot } n x) \wedge \text{Im } y < \text{Im } (\text{croot } n x)$  .
  with y-max-Re-Im[OF croot-power[OF n]]
  show False by auto
qed

```

```

lemma csqrt-is-croot-2:  $\text{csqrt} = \text{croot } 2$ 
proof
  fix x
  show  $\text{csqrt } x = \text{croot } 2 x$ 
  proof (rule sym, rule croot-unique, force, force)
    let ?p =  $[-x, 0, 1:]$ 
    let ?cx =  $\text{csqrt } x$ 
    have p:  $?p = [?:?cx, 1:] * [:-?cx, 1:]$ 
      by (simp add: power2-eq-square[symmetric])
    fix y
    assume  $y^2 = x$ 
    hence  $\text{True} \longleftrightarrow \text{poly } ?p y = 0$ 
      by (auto simp: power2-eq-square)
    also have ...  $\longleftrightarrow y = -?cx \vee y = ?cx$ 
      unfolding p poly-mult mult-eq-0-iff poly-root-factor by auto
    finally have  $y = -?cx \vee y = ?cx$  by simp
  qed

```

```

thus  $Re\ y < Re\ ?cx \vee Re\ y = Re\ ?cx \wedge Im\ y \leq Im\ ?cx$ 
proof
  assume  $y: y = -\ ?cx$ 
  show ?thesis
  proof (cases  $Re\ ?cx = 0$ )
    case False
      with csqrt-principal[of  $x$ ] have  $Re\ ?cx > 0$  by simp
      thus ?thesis unfolding  $y$  by simp
    next
      case True
      with csqrt-principal[of  $x$ ] have  $Im\ ?cx \geq 0$  by simp
      thus ?thesis unfolding  $y$  using True by auto
  qed
qed auto
qed
qed

```

```

lemma croot-via-root-selection: assumes roots:  $set\ ys = \{ y. y^{\wedge}n = x \}$ 
  and  $n: n \neq 0$ 
shows  $croot\ n\ x = arg\ min\ list\ (\lambda\ y. (-\ Re\ y, -\ Im\ y))\ ys$ 
  (is  $- = arg\ min\ list\ ?f\ ys$ )
proof (rule croot-unique[OF  $n$ ])
  let  $?y = arg\ min\ list\ ?f\ ys$ 
  have  $rt: croot\ n\ x^{\wedge}n = x$  using  $n$  by (rule croot-power)
  hence  $croot\ n\ x \in set\ ys$  unfolding roots by auto
  hence  $ys: ys \neq []$  by auto
  from arg-min-list-in[OF this] have  $?y \in set\ ys$  by auto
  from this[unfolded roots]
  show  $?y^{\wedge}n = x$  by auto
  fix  $z$ 
  assume  $z^{\wedge}n = x$ 
  hence  $z: z \in set\ ys$  unfolding roots by auto
  from f-arg-min-list-f[OF  $ys$ , of  $?f$ ]  $z$ 
  have  $?f\ ?y \leq ?f\ z$  by simp
  thus  $Re\ z < Re\ ?y \vee Re\ z = Re\ ?y \wedge Im\ z \leq Im\ ?y$  by auto
qed

```

```

lemma croot-impl[code]:  $croot\ n\ x = (if\ n = 0\ then\ 0\ else$ 
   $arg\ min\ list\ (\lambda\ y. (-\ Re\ y, -\ Im\ y))\ (all\ roots\ n\ x))$ 
proof (cases  $n = 0$ )
  case  $n0: False$ 
  hence id:  $(if\ n = 0\ then\ y\ else\ z) = z$ 
  for  $y\ z\ u :: complex$  by auto
  show ?thesis unfolding id Let-def
  by (rule croot-via-root-selection[OF  $-n0$ ], rule all-roots[OF  $n0$ ])
qed auto

```

**end**

## 4 Algorithms to compute all complex and real roots of a cubic polynomial

**theory** *Cubic-Polynomials*

**imports**

*Cardanos-Formula*

*Complex-Roots*

**begin**

The real case where a result is only delivered if the discriminant is negative

**definition** *solve-depressed-cubic-Cardano-real* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real option* **where**

*solve-depressed-cubic-Cardano-real* *e f* = (

if *e* = 0 then *Some* (*root 3* (-*f*)) else

let *v* = - (*e* <sup>3</sup> / 27) in

case *rroots2* [:*v,f,1*:] of

[*u,-*]  $\Rightarrow$  let *rt* = *root 3 u* in *Some* (*rt* - *e* / (3 \* *rt*))

| -  $\Rightarrow$  *None*)

**lemma** *solve-depressed-cubic-Cardano-real*:

**assumes** *solve-depressed-cubic-Cardano-real* *e f* = *Some y*

**shows** {*y*. *y* <sup>3</sup> + *e* \* *y* + *f* = 0} = {*y*}

**proof** (*cases e* = 0)

case *True*

**have** {*y*. *y* <sup>3</sup> + *e* \* *y* + *f* = 0} = {*y*. *y* <sup>3</sup> = -*f*} **unfolding** *True*

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

**also have** ... = {*root 3* (-*f*)}

using *odd-real-root-unique*[of 3 - *f*] *odd-real-root-pow*[of 3] **by** *auto*

**also have** *root 3* (-*f*) = *y* **using** *assms* **unfolding** *True solve-depressed-cubic-Cardano-real-def*

by *auto*

**finally show** *?thesis* .

**next**

case *False*

**define** *v* **where** *v* = - (*e* <sup>3</sup> / 27)

**note** \* = *assms*[*unfolded solve-depressed-cubic-Cardano-real-def Let-def, folded v-def*]

let *?rr* = *rroots2* [:*v,f,1*:]

**from** \* *False* **obtain** *u u'* **where** *rr*: *?rr* = [*u,u'*]

by (*cases ?rr; cases tl ?rr; cases tl (tl ?rr); auto split: if-splits*)

**from** \* [*unfolded rr list.simps*] *False*

**have** *y*: *y* = *root 3 u* - *e* / (3 \* *root 3 u*) **by** *auto*

**have** *u*  $\in$  *set* (*rroots2* [:*v,f,1*:]) **unfolding** *rr* **by** *auto*

**also have** *set* (*rroots2* [:*v,f,1*:]) = {*u*. *poly* [:*v,f,1*:] *u* = 0}

by (*subst rroots2, auto*)

**finally have** *u*: *u* <sup>2</sup> + *f* \* *u* + *v* = 0 **by** (*simp add: field-simps power2-eq-square*)

**note** *Cardano* = *solve-cubic-depressed-Cardano-real*[*OF False v-def u*]

**have** 2: 2 = *Suc* (*Suc* 0) **by** *simp*

**from** *rr* **have** 0: *f* <sup>2</sup> - 4 \* *v*  $\neq$  0 **unfolding** *rroots2-def Let-def*

by (*auto split: if-splits simp: 2*)

**hence** 0: *discriminant-cubic-depressed e f*  $\neq$  0

**unfolding** *discriminant-cubic-depressed-def v-def* **by** *auto*  
**show** *?thesis* **using** *Cardano(1) Cardano(2)[OF 0]* **unfolding** *y[symmetric]* **by**  
*blast*  
**qed**

The complex case

**definition** *solve-depressed-cubic-complex* :: *complex*  $\Rightarrow$  *complex*  $\Rightarrow$  *complex list*  
**where**

*solve-depressed-cubic-complex e f* = (let  
   *ys* = (if *e* = 0 then *all-roots* 3 (- *f*) else (let  
     *u* = *hd* (*roots2* [: - (*e* ^ 3 / 27) ,*f*,1:]);  
     *zs* = *all-roots* 3 *u*  
     in *map* ( $\lambda z. z - e / (3 * z)$ ) *zs*)  
 in *remdups ys*)

**lemma** *solve-depressed-cubic-complex-code[code]*:

*solve-depressed-cubic-complex e f* = (let  
   *ys* = (if *e* = 0 then *all-roots* 3 (- *f*) else (let  
     *f2* = *f* / 2;  
     *u* = - *f2* + *csqrt* (*f2*^2 + *e* ^ 3 / 27);  
     *zs* = *all-roots* 3 *u*  
     in *map* ( $\lambda z. z - e / (3 * z)$ ) *zs*)  
 in *remdups ys*)

**unfolding** *solve-depressed-cubic-complex-def* *Let-def roots2-def*  
**by** (*simp add: numeral-2-eq-2*)

**lemma** *solve-depressed-cubic-complex*: *y*  $\in$  *set* (*solve-depressed-cubic-complex e f*)

$\longleftrightarrow$  (*y*^3 + *e* \* *y* + *f* = 0)

**proof** (*cases e = 0*)

**case** *True*

**thus** *?thesis* **by** (*simp add: solve-depressed-cubic-complex-def Let-def all-roots*  
*eq-neg-iff-add-eq-0*)

**next**

**case** *e0: False*

**hence** *id*: (if *e* = 0 then *x* else *y*) = *y* **for** *x y* :: *complex list* **by** *simp*

**define** *v* **where** *v* = - (*e* ^ 3 / 27)

**define** *p* **where** *p* = [:*v*, *f*, 1:]

**have** *p2*: *degree p* = 2 **unfolding** *p-def* **by** *auto*

**let** *?u* = *hd* (*roots2 p*)

**define** *u* **where** *u* = *?u*

**have** *u*  $\in$  *set* (*roots2 p*) **unfolding** *roots2-def Let-def u-def* **by** *auto*

**with** *roots2[OF p2]* **have** *poly p u* = 0 **by** *auto*

**hence** *u*: *u*^2 + *f* \* *u* + *v* = 0 **unfolding** *p-def*

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

**note** *cube-roots* = *all-roots[of 3, simplified]*

**show** *?thesis* **unfolding** *solve-depressed-cubic-complex-def Let-def set-remdups*  
*set-map id cube-roots*

```

unfolding v-def[symmetric] p-def[symmetric] set-concat set-map
  u-def[symmetric]
proof -
  have p: {x. poly p x = 0} = {u. u2 + f * u + v = 0} unfolding p-def by
(auto simp: field-simps power2-eq-square)
  have cube:  $\bigcup$  (set 'all-roots 3' {x. poly p x = 0}) = {z.  $\exists$  u. u2 + f * u +
v = 0  $\wedge$  z3 = u}
  unfolding p by (auto simp: cube-roots)
  show (y  $\in$  ( $\lambda$ z. z - e / (3 * z))) ' {y. y3 = u} = (y3 + e * y + f = 0)
  using solve-cubic-depressed-Cardano-complex[OF e0 v-def u] cube by blast
qed
qed

```

For the general real case, we first try Cardano with negative discriminant and only if it is not applicable, then we go for the calculation using complex numbers. Note that for non-negative delta no filter is required to identify the real roots from the list of complex roots, since in that case we already know that all roots are real.

**definition** solve-depressed-cubic-real :: real  $\Rightarrow$  real  $\Rightarrow$  real list **where**  
 solve-depressed-cubic-real e f = (case solve-depressed-cubic-Cardano-real e f  
 of Some y  $\Rightarrow$  [y]  
 | None  $\Rightarrow$  map Re (solve-depressed-cubic-complex (of-real e) (of-real f)))

**lemma** solve-depressed-cubic-real-code[code]: solve-depressed-cubic-real e f =  
 (if e = 0 then [root 3 (-f)] else  
 let v = e<sup>3</sup> / 27;  
 f2 = f / 2;  
 f2v = f2<sup>2</sup> + v in  
 if f2v > 0 then  
 let u = -f2 + sqrt f2v;  
 rt = root 3 u  
 in [rt - e / (3 \* rt)]  
 else  
 let ce3 = of-real e / 3;  
 u = - of-real f2 + csqrt (of-real f2v) in  
 map Re (remdups (map ( $\lambda$ rt. rt - ce3 / rt) (all-roots 3 u))))

```

proof -
have id: rroots2 [v, f, 1:] = (let
  f2 = f / 2;
  bac = f22 - v in
  if bac = 0 then [- f2] else
  if bac < 0 then [] else let e = sqrt bac in [- f2 + e, - f2 - e]) for v
unfolding rroots2-def Let-def numeral-2-eq-2 by auto
define foo :: real  $\Rightarrow$  real  $\Rightarrow$  real option where
  foo f2v f2 = (case (if f2v = 0 then [- f2] else []) of []  $\Rightarrow$  None | -  $\Rightarrow$  None)
for f2v f2
have solve-depressed-cubic-real e f = (if e = 0 then [root 3 (-f)] else
  let v = e3 / 27;
  f2 = f / 2;

```



```

      f2v = f22 + v in
    if f2v > 0 then
      let u = -f2 + sqrt f2v;
          rt = root 3 u
      in [rt - e / (3 * rt)]
    else
      (case foo f2v f2 of
        None => let u = - cor f2 + csqrt (cor f2v) in
          map Re
            (remdups (map (λz. z - cor e / (3 * z)) (all-roots 3 u)))
        | Some y => [])
      unfolding solve-depressed-cubic-real-def solve-depressed-cubic-Cardano-real-def

      solve-depressed-cubic-complex-code
      Let-def id foo-def
      by (auto split: if-splits)
    also have id: foo f2v f2 = None
      for f2v f2 unfolding foo-def by auto
    ultimately show ?thesis by (auto simp: Let-def)
  qed

lemma solve-depressed-cubic-real: y ∈ set (solve-depressed-cubic-real e f)
  ↔ (y3 + e * y + f = 0)
proof (cases solve-depressed-cubic-Cardano-real e f)
  case (Some x)
  show ?thesis unfolding solve-depressed-cubic-real-def Some option.simps
    using solve-depressed-cubic-Cardano-real[OF Some] by auto
next
  case None
  from this[unfolded solve-depressed-cubic-Cardano-real-def Let-def rroots2-def]
  have disc: 0 ≤ discriminant-cubic-depressed e f unfolding discriminant-cubic-depressed-def
    by (auto split: if-splits simp: numeral-2-eq-2)
  let ?c = complex-of-real
  let ?y = ?c y
  let ?e = ?c e
  let ?f = ?c f
  have sub: set (solve-depressed-cubic-complex ?e ?f) ⊆ ℝ
  proof
    fix y
    assume y: y ∈ set (solve-depressed-cubic-complex ?e ?f)
    show y ∈ ℝ
    by (rule solve-cubic-depressed-Cardano-all-real-roots[OF disc y[unfolded solve-depressed-cubic-complex]])
  qed
  have y3 + e * y + f = 0 ↔ (?c (y3 + e * y + f) = ?c 0) unfolding
of-real-eq-iff by simp
  also have ... ↔ ?y3 + ?e * ?y + ?f = 0 by simp
  also have ... ↔ ?y ∈ set (solve-depressed-cubic-complex ?e ?f)
  unfolding solve-depressed-cubic-complex ..
  also have ... ↔ y ∈ Re ' set (solve-depressed-cubic-complex ?e ?f) using sub

```

by force  
 finally show ?thesis unfolding solve-depressed-cubic-real-def None by auto  
 qed

Combining the various algorithms

**lemma** degree3-coeffs: degree  $p = 3 \implies$   
 $\exists a b c d. p = [: d, c, b, a :] \wedge a \neq 0$   
 by (metis One-nat-def Suc-1 degree2-coeffs degree-pCons-eq-if nat.inject numeral-3-eq-3  
 pCons-cases zero-neq-numeral)

**definition** roots3-generic :: ('a :: field-char-0  $\Rightarrow$  'a  $\Rightarrow$  'a list)  $\Rightarrow$  'a poly  $\Rightarrow$  'a list  
 where

roots3-generic depressed-solver  $p =$  (let  
 cs = coeffs  $p$ ;  
 $a = cs ! 3$ ;  $b = cs ! 2$ ;  $c = cs ! 1$ ;  $d = cs ! 0$ ;  
 $a3 = 3 * a$ ;  
 $ba3 = b / a3$ ;  
 $b2 = b * b$ ;  
 $b3 = b2 * b$ ;  
 $e = (c - b2 / a3) / a$ ;  
 $f = (d + 2 * b3 / (27 * a^2) - b * c / a3) / a$ ;  
 roots = depressed-solver  $e f$   
 in map ( $\lambda y. y - ba3$ ) roots)

**lemma** roots3-generic: assumes deg: degree  $p = 3$   
 and solver:  $\bigwedge e f y. y \in \text{set } (\text{depressed-solver } e f) \iff y^3 + e * y + f = 0$   
 shows set (roots3-generic depressed-solver  $p$ ) = { $x. \text{poly } p x = 0$ }

**proof** –

**note** powers = field-simps power3-eq-cube power2-eq-square  
**from** degree3-coeffs[OF deg] **obtain**  $a b c d$  **where**  
 $p: p = [:d,c,b,a:]$  **and**  $a: a \neq 0$  **by** auto  
**have** coeffs: coeffs  $p ! 3 = a$  coeffs  $p ! 2 = b$  coeffs  $p ! 1 = c$  coeffs  $p ! 0 = d$   
**unfolding**  $p$  **using**  $a$  **by** auto  
**define**  $e$  **where**  $e = (c - b^2 / (3 * a)) / a$   
**define**  $f$  **where**  $f = (d + 2 * b^3 / (27 * a^2) - b * c / (3 * a)) / a$   
**note** def = roots3-generic-def[of depressed-solver  $p$ , unfolded Let-def coeffs,  
 folded power3-eq-cube, folded power2-eq-square, folded e-def f-def]  
 {  
 fix  $x :: 'a$   
**define**  $y$  **where**  $y = x + b / (3 * a)$   
**have**  $xy: x = y - b / (3 * a)$  **unfolding**  $y$ -def **by** auto  
**have** poly  $p x = 0 \iff a * x^3 + b * x^2 + c * x + d = 0$  **unfolding**  $p$   
**by** (simp add: powers)  
**also** **have** ...  $\iff (y^3 + e * y + f = 0)$   
**unfolding** to-depressed-cubic[OF  $a xy e$ -def  $f$ -def] ..  
**also** **have** ...  $\iff y \in \text{set } (\text{depressed-solver } e f)$   
**unfolding** solver ..  
**also** **have** ...  $\iff x \in \text{set } (\text{roots3-generic depressed-solver } p)$  **unfolding**  $xy$  def  
**by** auto

**finally have**  $\text{poly } p \ x = 0 \iff x \in \text{set } (\text{roots3-generic depressed-solver } p)$  **by**  
*auto*  
**}**  
**thus** *?thesis* **by** *auto*  
**qed**

**definition**  $\text{roots3} :: \text{complex poly} \Rightarrow \text{complex list}$  **where**  
 $\text{roots3} = \text{roots3-generic solve-depressed-cubic-complex}$

**lemma**  $\text{roots3}$ : **assumes**  $\text{deg: degree } p = 3$   
**shows**  $\text{set } (\text{roots3 } p) = \{ x. \text{poly } p \ x = 0 \}$   
**unfolding**  $\text{roots3-def}$  **by**  $(\text{rule } \text{roots3-generic}[\text{OF deg solve-depressed-cubic-complex}])$

**definition**  $\text{rroots3} :: \text{real poly} \Rightarrow \text{real list}$  **where**  
 $\text{rroots3} = \text{roots3-generic solve-depressed-cubic-real}$

**lemma**  $\text{rroots3}$ : **assumes**  $\text{deg: degree } p = 3$   
**shows**  $\text{set } (\text{rroots3 } p) = \{ x. \text{poly } p \ x = 0 \}$   
**unfolding**  $\text{rroots3-def}$  **by**  $(\text{rule } \text{roots3-generic}[\text{OF deg solve-depressed-cubic-real}])$

**end**

## 5 Algorithms to compute all complex and real roots of a quartic polynomial

**theory** *Quartic-Polynomials*  
**imports**  
*Ferraris-Formula*  
*Cubic-Polynomials*  
**begin**

The complex case is straight-forward

**definition**  $\text{solve-depressed-quartic-complex} :: \text{complex} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{complex list}$  **where**

$\text{solve-depressed-quartic-complex } p \ q \ r = \text{remdups } (\text{if } q = 0 \text{ then}$   
 $(\text{concat } (\text{map } (\lambda z. \text{let } y = \text{csqrt } z \text{ in } [y, -y]) (\text{roots2 } [:r, p, 1:]))) \text{ else}$   
 $\text{let cubics} = \text{roots3 } [:(q^2), 2 * p^2 - 8 * r, 8 * p, 8:];$   
 $m = \text{hd cubics};$  — select any root of the cubic polynomial  
 $a = \text{csqrt } (2 * m);$   
 $p2m = p / 2 + m;$   
 $q2a = q / (2 * a);$   
 $b1 = p2m - q2a;$   
 $b2 = p2m + q2a$   
 $\text{in } (\text{roots2 } [:b1, a, 1:] @ \text{roots2 } [:b2, -a, 1:])))$

**lemma**  $\text{solve-depressed-quartic-complex}$ :  $x \in \text{set } (\text{solve-depressed-quartic-complex } p \ q \ r)$   
 $\iff (x^4 + p * x^2 + q * x + r = 0)$

```

proof –
  note powers = field-simps power4-eq-xxxx power3-eq-cube power2-eq-square
  show ?thesis
  proof (cases q = 0)
    case True
      have csqrt:  $z = x^{\wedge}2 \iff (x = \text{csqrt } z \vee x = - \text{csqrt } z)$  for z
        by (metis power2-csqrt power2-eq-iff)
      have  $(x^{\wedge}4 + p * x^2 + q * x + r = 0) \iff (x^{\wedge}4 + p * x^2 + r = 0)$ 
        unfolding True by simp
      also have ...  $\iff (\exists z. z^2 + p * z + r = 0 \wedge z = x^2)$  unfolding bi-
        quadratic-solution by simp
      also have ...  $\iff (\exists z. \text{poly } [:r,p,1:] z = 0 \wedge z = x^2)$ 
        by (simp add: powers)
      also have ...  $\iff (\exists z \in \text{set } (\text{croots2 } [:r,p,1:]). z = x^2)$ 
        by (subst roots2[symmetric], auto)
      also have ...  $\iff (\exists z \in \text{set } (\text{croots2 } [:r,p,1:]). x = \text{csqrt } z \vee x = - \text{csqrt } z)$ 
        unfolding csqrt ..
      also have ...  $\iff (x \in \text{set } (\text{solve-depressed-quartic-complex } p \ q \ r))$ 
        unfolding solve-depressed-quartic-complex-def id unfolding True Let-def by
        auto
      finally show ?thesis ..
    next
      case q0: False
        hence id: (if q = 0 then x else y) = y for x y :: complex list by auto
        note powers = field-simps power4-eq-xxxx power3-eq-cube power2-eq-square
        let ?poly = [:- q2, 2 * p2 - 8 * r, 8 * p, 8:]
        from roots3[of ?poly] have roots: set (roots3 ?poly) = {x. poly ?poly x =
        0} by auto
        from fundamental-theorem-of-algebra-alt[of ?poly]
        have {x. poly ?poly x = 0}  $\neq \{\}$  by auto
        with roots have roots3 ?poly  $\neq []$  by auto
        then obtain m rest where rts: roots3 ?poly = m # rest by (cases roots3
        ?poly, auto)
        hence hd: hd (roots3 ?poly) = m by auto
        from roots[unfolded rts] have poly ?poly m = 0 by auto
        hence mrt:  $8 * m^3 + (8 * p) * m^2 + (2 * p^2 - 8 * r) * m - q^2 = 0$ 
          and m0:  $m \neq 0$  using q0
          by (auto simp: powers)
        define b1 where  $b1 = p / 2 + m - q / (2 * \text{csqrt } (2 * m))$ 
        define b2 where  $b2 = p / 2 + m + q / (2 * \text{csqrt } (2 * m))$ 
        have csqrt:  $\text{csqrt } x * \text{csqrt } x = x$  for x by (metis power2-csqrt power2-eq-square)
        show ?thesis unfolding solve-depressed-quartic-complex-def id Let-def set-remdups
        set-append hd
          unfolding b1-def[symmetric] b2-def[symmetric]
          apply (subst depressed-quartic-Ferrari[OF mrt q0 csqrt b1-def b2-def])
          apply (subst (1 2) roots2[symmetric], auto)
          done
      qed
    qed

```

The main difference in the real case is that a specific cubic root has to be used, namely a positive one. In the soundness proof we show that such a cubic root always exists.

**definition** *solve-depressed-quartic-real* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real list* **where**  
*solve-depressed-quartic-real* *p q r* = *remdups* (if *q* = 0 then  
 (*concat* (*map* ( $\lambda z$ . *rroots2* [:-*z*,0,1:]) (*rroots2* [:*r*,*p*,1:]))) else  
 let *cubics* = *rroots3* [:- (*q*<sup>2</sup>), 2 \* *p*<sup>2</sup> - 8 \* *r*, 8 \* *p*, 8:];  
     *m* = *the* (*find* ( $\lambda m$ . *m* > 0) *cubics*); — select any positive root of the  
 cubic polynomial  
     *a* = *sqrt* (2 \* *m*);  
     *p2m* = *p* / 2 + *m*;  
     *q2a* = *q* / (2 \* *a*);  
     *b1* = *p2m* - *q2a*;  
     *b2* = *p2m* + *q2a*  
 in (*rroots2* [:*b1*,*a*,1:] @ *rroots2* [:*b2*, -*a*,1:])))

**lemma** *solve-depressed-quartic-real*: *x*  $\in$  *set* (*solve-depressed-quartic-real* *p q r*)  
 $\longleftrightarrow$  (*x*<sup>4</sup> + *p* \* *x*<sup>2</sup> + *q* \* *x* + *r* = 0)

**proof** —

**note** *powers* = *field-simps* *power4-eq-xxxx* *power3-eq-cube* *power2-eq-square*  
**show** *?thesis*

**proof** (*cases* *q* = 0)

**case** *True*

**have** *sqrt*: *z* = *x*<sup>2</sup>  $\longleftrightarrow$  (*x*  $\in$  *set* (*rroots2* [:-*z*,0,1:]))) **for** *z*

**by** (*subst* *rroots2*[*symmetric*], *auto* *simp*: *powers*)

**have** (*x*<sup>4</sup> + *p* \* *x*<sup>2</sup> + *q* \* *x* + *r* = 0)  $\longleftrightarrow$  (*x*<sup>4</sup> + *p* \* *x*<sup>2</sup> + *r* = 0)

**unfolding** *True* **by** *simp*

**also have** ...  $\longleftrightarrow$  ( $\exists z$ . *z*<sup>2</sup> + *p* \* *z* + *r* = 0  $\wedge$  *z* = *x*<sup>2</sup>) **unfolding** *bi-quadratic-solution* **by** *simp*

**also have** ...  $\longleftrightarrow$  ( $\exists z$ . *poly* [:*r*,*p*,1:] *z* = 0  $\wedge$  *z* = *x*<sup>2</sup>)

**by** (*simp* *add*: *powers*)

**also have** ...  $\longleftrightarrow$  ( $\exists z \in$  *set* (*rroots2* [:*r*,*p*,1:]). *z* = *x*<sup>2</sup>)

**by** (*subst* *rroots2*[*symmetric*], *auto*)

**also have** ...  $\longleftrightarrow$  ( $\exists z \in$  *set* (*rroots2* [:*r*,*p*,1:]). *x*  $\in$  *set* (*rroots2* [:-*z*,0,1:])))

**unfolding** *sqrt* ..

**also have** ...  $\longleftrightarrow$  (*x*  $\in$  *set* (*solve-depressed-quartic-real* *p q r*))

**unfolding** *solve-depressed-quartic-real-def* *id* **unfolding** *True* *Let-def* **by** *auto*

**finally show** *?thesis* ..

**next**

**case** *q0*: *False*

**hence** *id*: (if *q* = 0 then *x* else *y*) = *y* **for** *x y* :: *real list* **by** *auto*

**note** *powers* = *field-simps* *power4-eq-xxxx* *power3-eq-cube* *power2-eq-square*

**let** *?poly* = [:- *q*<sup>2</sup>, 2 \* *p*<sup>2</sup> - 8 \* *r*, 8 \* *p*, 8:]

**define** *cubics* **where** *cubics* = *rroots3* *?poly*

**from** *rroots3*[*of* *?poly*, *folded* *cubics-def*]

**have** *rroots*: *set* *cubics* = {*x*. *poly* *?poly* *x* = 0} **by** *auto*

**from** *odd-degree-imp-real-root*[*of* *?poly*]

**have** {*x*. *poly* *?poly* *x* = 0}  $\neq$  {} **by** *auto*

```

with roots have cubics ≠ [] by auto
have ∃ m. m ∈ set cubics ∧ m > 0
proof (rule ccontr)
  assume ¬ ?thesis
  from this[unfolded roots] have rt: poly ?poly m = 0 ⇒ m ≤ 0 for m by
auto
  have poly ?poly 0 = - (q^2) by simp
  also have ... < 0 using q0 by auto
  finally have lt: poly ?poly 0 ≤ 0 by simp
  from poly-pinfty-gt-lc[of ?poly] obtain n0 where ∧ n. n ≥ n0 ⇒ 8 ≤ poly
?poly n by auto
  from this[of max n0 0] have poly ?poly (max n0 0) ≥ 0 by auto
  from IVT[of poly ?poly, OF lt this] obtain m where m ≥ 0 and poly: poly
?poly m = 0 by auto
  from rt[OF this(2)] this(1) have m = 0 by auto
  thus False using poly q0 by simp
qed
hence find (λ m. m > 0) cubics ≠ None unfolding find-None-iff by auto
then obtain m where find: find (λ m. m > 0) cubics = Some m by auto
from find-Some-D[OF this] have m: m ∈ set cubics and m-0: m > 0 by auto
with roots have poly ?poly m = 0 by auto
hence mrt: 8*m^3 + (8 * p) * m^2 + (2 * p^2 - 8 * r) * m - q^2 = 0
  by (auto simp: powers)
from m-0 have sqrt: sqrt (2 * m) * sqrt (2 * m) = 2 * m by simp
define b1 where b1 = p / 2 + m - q / (2 * sqrt (2 * m))
define b2 where b2 = p / 2 + m + q / (2 * sqrt (2 * m))
show ?thesis unfolding solve-depressed-quartic-real-def id Let-def set-remdups
set-append
  cubics-def[symmetric] find option.sel
  unfolding b1-def[symmetric] b2-def[symmetric]
  apply (subst depressed-quartic-Ferrari[OF mrt q0 sqrt b1-def b2-def])
  apply (subst (1 2) rroots2[symmetric], auto)
  done
qed
qed

```

Combining the various algorithms

```

lemma numeral-4-eq-4: 4 = Suc (Suc (Suc (Suc 0)))
  by (simp add: eval-nat-numeral)

```

```

lemma degree4-coeffs: degree p = 4 ⇒
  ∃ a b c d e. p = [: e, d, c, b, a :] ∧ a ≠ 0
  using degree3-coeffs degree-pCons-eq-if nat.inject numeral-3-eq-3 numeral-4-eq-4
pCons-cases zero-neq-numeral
  by metis

```

```

definition roots4-generic :: ('a :: field-char-0 ⇒ 'a ⇒ 'a ⇒ 'a list) ⇒ 'a poly ⇒
'a list where
  roots4-generic depressed-solver p = (let

```

```

cs = coeffs p;
cs = coeffs p;
a4 = cs ! 4; a3 = cs ! 3; a2 = cs ! 2; a1 = cs ! 1; a0 = cs ! 0;
b = a3 / a4;
c = a2 / a4;
d = a1 / a4;
e = a0 / a4;
b2 = b * b;
b3 = b2 * b;
b4 = b3 * b;
b4' = b / 4;
p = c - 3/8 * b2;
q = (b3 - 4*b*c + 8 * d) / 8;
r = (-3 * b4 + 256 * e - 64 * b * d + 16 * b2 * c) / 256;
roots = depressed-solver p q r
in map (λ y. y - b4') roots)

```

**lemma** *roots4-generic*: **assumes** *deg*: degree  $p = 4$   
**and** *solver*:  $\bigwedge p q r y. y \in \text{set } (\text{depressed-solver } p q r) \longleftrightarrow y^4 + p * y^2 + q * y + r = 0$   
**shows**  $\text{set } (\text{roots4-generic depressed-solver } p) = \{x. \text{poly } p x = 0\}$   
**proof** –  
**note** *powers* = *field-simps power4-eq-xxxx power3-eq-cube power2-eq-square*  
**from** *degree4-coeffs*[*OF deg*] **obtain**  $a_4 a_3 a_2 a_1 a_0$  **where**  
 $p = [:a_0, a_1, a_2, a_3, a_4:]$  **and**  $a_4: a_4 \neq 0$  **by** *auto*  
**have** *coeffs*:  $\text{coeffs } p ! 4 = a_4 \text{ coeffs } p ! 3 = a_3 \text{ coeffs } p ! 2 = a_2 \text{ coeffs } p ! 1 = a_1 \text{ coeffs } p ! 0 = a_0$   
**unfolding**  $p$  **using**  $a_4$  **by** *auto*  
**define**  $b$  **where**  $b = a_3 / a_4$   
**define**  $c$  **where**  $c = a_2 / a_4$   
**define**  $d$  **where**  $d = a_1 / a_4$   
**define**  $e$  **where**  $e = a_0 / a_4$   
**note** *def* = *roots4-generic-def*[*of depressed-solver p, unfolded Let-def coeffs, folded b-def c-def d-def e-def,*  
*folded power4-eq-xxxx, folded power3-eq-cube, folded power2-eq-square*]  
**let**  $?p = p$   
{  
**fix**  $x$   
**define**  $y$  **where**  $y = x + b / 4$   
**define**  $p$  **where**  $p = c - (3/8) * b^2$   
**define**  $q$  **where**  $q = (b^3 - 4*b*c + 8 * d) / 8$   
**define**  $r$  **where**  $r = (-3 * b^4 + 256 * e - 64 * b * d + 16 * b^2 * c) / 256$   
**note** *def* = *def*[*folded p-def q-def r-def*]  
**have** *xy*:  $x = y - b / 4$  **unfolding** *y-def* **by** *auto*  
**have** *poly*  $?p x = 0 \longleftrightarrow a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0 = 0$  **unfolding**  $p$   
**by** (*simp add: powers*)  
**also have**  $\dots \longleftrightarrow (y^4 + p * y^2 + q * y + r = 0)$

**unfolding** *to-depressed-quartic*[*OF a4 b-def c-def d-def e-def p-def q-def r-def xy*] ..  
**also have** ...  $\longleftrightarrow y \in \text{set } (\text{depressed-solver } p \ q \ r)$   
**unfolding** *solver* ..  
**also have** ...  $\longleftrightarrow x \in \text{set } (\text{roots4-generic depressed-solver } ?p)$  **unfolding** *xy*  
**def by auto**  
**finally have** *poly ?p x = 0*  $\longleftrightarrow x \in \text{set } (\text{roots4-generic depressed-solver } ?p)$   
**by auto**  
**}**  
**thus** *?thesis by simp*  
**qed**

**definition** *roots4* :: *complex poly*  $\Rightarrow$  *complex list* **where**  
*roots4* = *roots4-generic solve-depressed-quartic-complex*

**lemma** *roots4*: **assumes** *deg: degree p = 4*  
**shows** *set (roots4 p) = { x. poly p x = 0 }*  
**unfolding** *roots4-def* **by** (*rule roots4-generic*[*OF deg solve-depressed-quartic-complex*])

**definition** *rroots4* :: *real poly*  $\Rightarrow$  *real list* **where**  
*rroots4* = *roots4-generic solve-depressed-quartic-real*

**lemma** *rroots4*: **assumes** *deg: degree p = 4*  
**shows** *set (rroots4 p) = { x. poly p x = 0 }*  
**unfolding** *rroots4-def* **by** (*rule roots4-generic*[*OF deg solve-depressed-quartic-real*])

end

## References

- [1] G. Cardano. *Ars Magna, The Great Art or the Rules of Algebra*. 1545. [https://en.wikipedia.org/wiki/Ars\\_Magna\\_\(Cardano\\_book\)](https://en.wikipedia.org/wiki/Ars_Magna_(Cardano_book)).
- [2] R. Thiemann, A. Yamada, and S. Joosten. Algebraic numbers in Isabelle/HOL. *Archive of Formal Proofs*, Dec. 2015. [https://isa-afp.org/entries/Algebraic\\_Numbers.html](https://isa-afp.org/entries/Algebraic_Numbers.html), Formal proof development.