The Mason–Stothers theorem

Manuel Eberl

March 17, 2025

Abstract

This article provides a formalisation of Snyder's simple and elegant proof of the Mason–Stothers theorem [2, 1], which is the polynomial analogue of the famous *abc* Conjecture for integers. Remarkably, Snyder found this very elegant proof when he was still a high-school student.

In short, the statement of the theorem is that three non-zero coprime polynomials A, B, C over a field which sum to 0 and do not all have vanishing derivatives fulfil $\max\{\deg(A), \deg(B), \deg(C)\} < \deg(\mathrm{rad}(ABC))$ where $\mathrm{rad}(P)$ denotes the *radical* of P, i. e. the product of all unique irreducible factors of P.

This theorem also implies a kind of polynomial analogue of Fermat's Last Theorem for polynomials: except for trivial cases, $A^n + B^n + C^n = 0$ implies $n \leq 2$ for coprime polynomials A, B, C over a field.

Contents

1 The Mason–Stother's Theorem

```
theory Mason-Stothers
imports
  HOL−Computational-Algebra.Computational-Algebra
  HOL−Computational-Algebra.Polynomial-Factorial
begin
```

1.1 Auxiliary material

```
hide-const (open) Formal-Power-Series.radical

lemma degree-div:
  assumes a dvd b
  shows   degree (b div a) = degree b − degree a
 using assms by (cases a = 0; cases b = 0) (auto elim!: dvdE simp: degree-mult-eq)

lemma degree-pderiv-le:
  shows   degree (pderiv p) ≤ degree p − 1
  by (rule degree-le, cases degree p = 0) (auto simp: coeff-pderiv coeff-eq-0)

lemma degree-pderiv-less:
  assumes pderiv p ≠ 0
  shows   degree (pderiv p) < degree p
proof −
  have degree (pderiv p) ≤ degree p − 1
    by (rule degree-pderiv-le)
  also have degree p ≠ 0
    using assms by (auto intro!: Nat.gr0I elim!: degree-eq-zeroE)
  hence degree p − 1 < degree p by simp
  finally show ?thesis .
qed

lemma pderiv-eq-0:
  assumes degree p = 0
  shows   pderiv p = 0
  using assms by (auto elim!: degree-eq-zeroE)
```

1.2 Definition of a radical

The following definition of a radical is generic for any factorial semiring.

```
context factorial-semiring
begin

definition radical :: 'a ⇒ 'a where
  radical x = (if x = 0 then 0 else ∏(prime-factors x))

lemma radical-0 [simp]: radical 0 = 0
  by (simp add: radical-def)
```

```
lemma radical-nonzero: x ≠ 0 ⟹ radical x = ∏(prime-factors x)
  by (simp add: radical-def)

lemma radical-eq-0-iff [simp]: radical x = 0 ⟷ x = 0
  by (auto simp: radical-def)

lemma prime-factorization-radical [simp]:
  assumes x ≠ 0
  shows   prime-factorization (radical x) = mset-set (prime-factors x)
proof −
  have prime-factorization (radical x) = (∑ p∈prime-factors x. prime-factorization
p)
    unfolding radical-def using assms by (auto intro!: prime-factorization-prod)
  also have ... = (∑ p∈prime-factors x. {#p#})
    by (intro Groups-Big.sum.cong) (auto intro!: prime-factorization-prime)
  also have ... = mset-set (prime-factors x) by simp
  finally show ?thesis .
qed

lemma prime-factors-radical [simp]: x ≠ 0 ⟹ prime-factors (radical x) = prime-factors
x
  by simp

lemma radical-dvd [simp, intro]: radical x dvd x
  by (cases x = 0) (force intro: prime-factorization-subset-imp-dvd mset-set-set-mset-msubset)+

lemma multiplicity-radical-prime:
  assumes prime p x ≠ 0
  shows   multiplicity p (radical x) = (if p dvd x then 1 else 0)
proof −
  have multiplicity p (radical x) = (∑ q∈prime-factors x. multiplicity p q)
    using assms unfolding radical-def
    by (auto simp: prime-elem-multiplicity-prod-distrib)
  also have ... = (∑ q∈prime-factors x. if p = q then 1 else 0)
    using assms by (intro Groups-Big.sum.cong) (auto intro!: prime-multiplicity-other)
  also have ... = (if p ∈ prime-factors x then 1 else 0) by simp
  also have ... = (if p dvd x then 1 else 0)
    using assms by (auto simp: prime-factors-dvd)
  finally show ?thesis .
qed

lemma radical-1 [simp]: radical 1 = 1
  by (simp add: radical-def)

lemma radical-unit [simp]: is-unit x ⟹ radical x = 1
  by (auto simp: radical-def prime-factorization-unit)

lemma prime-factors-power:
```

assumes *n* > *0*
shows *prime-factors* (*x* ^ *n*) = *prime-factors x*
using *assms* **by** (*cases x* = *0*) (*auto simp*: *prime-factors-dvd zero-power prime-dvd-power-iff*)

lemma *radical-power* [*simp*]: *n* > *0* ⟹ *radical* (*x* ^ *n*) = *radical x*
by (*auto simp add*: *radical-def prime-factors-power*)

end

context *factorial-semiring-gcd*
begin

lemma *radical-mult-coprime*:
assumes *coprime a b*
shows *radical* (*a* ∗ *b*) = *radical a* ∗ *radical b*
proof (*cases a* = *0* ∨ *b* = *0*)
case *False*
with *assms* **have** *prime-factors a* ∩ *prime-factors b* = {}
using *not-prime-unit coprime-common-divisor* **by** (*auto simp*: *prime-factors-dvd*)
hence ∏ (*prime-factors a* ∪ *prime-factors b*) = ∏ (*prime-factors a*) ∗ ∏ (*prime-factors b*)
by (*intro prod.union-disjoint*) *auto*
with *False* **show** *?thesis* **by** (*simp add*: *radical-def prime-factorization-mult*)
qed *auto*

lemma *multiplicity-le-imp-dvd′*:
assumes *x* ≠ *0* ⋀*p*. *p* ∈ *prime-factors x* ⟹ *multiplicity p x* ≤ *multiplicity p y*
shows *x dvd y*
proof (*rule multiplicity-le-imp-dvd*)
fix *p* **assume** *prime p*
thus *multiplicity p x* ≤ *multiplicity p y* **using** *assms*(*1*) *assms*(*2*)[*of p*]
by (*cases p dvd x*) (*auto simp*: *prime-factors-dvd not-dvd-imp-multiplicity-0*)
qed *fact*+

end

1.3 Main result

The following proofs are basically a one-to-one translation of Franz Lemmermeyer's presentation [1] of Snyder's proof of the Mason–Stothers theorem.

lemma *prime-power-dvd-pderiv*:
fixes *f p* :: ′*a* :: *field-gcd poly*
assumes *prime-elem p*
defines *n* ≡ *multiplicity p f* − *1*
shows *p* ^ *n dvd pderiv f*
proof (*cases p dvd f* ∧ *f* ≠ *0*)
case *True*
hence *multiplicity p f* > *0* **using** *assms*
by (*subst prime-multiplicity-gt-zero-iff*) *auto*

hence *Suc-n*: *Suc n* = *multiplicity p f* **by** (*simp add*: *n-def*)
define *g* **where** *g* = *f div p* ^ *Suc n*
have *p* ^ *Suc n dvd f* **unfolding** *Suc-n* **by** (*rule multiplicity-dvd*)
hence *f-eq*: *f* = *p* ^ *Suc n* ∗ *g* **by** (*simp add*: *g-def*)
also have *pderiv* ... = *p* ^ *n* ∗ (*smult* (*of-nat* (*Suc n*)) (*pderiv p* ∗ *g*) + *p* ∗ *pderiv g*)
by (*simp only*: *pderiv-mult pderiv-power-Suc*) (*simp add*: *algebra-simps*)
also have *p* ^ *n dvd* ... **by** *simp*
finally show *?thesis* **.**
qed (*auto simp*: *n-def not-dvd-imp-multiplicity-0*)

lemma *poly-div-radical-dvd-pderiv*:
fixes *p* :: *'a* :: *field-gcd poly*
shows *p div radical p dvd pderiv p*
proof (*cases pderiv p* = *0*)
case *False*
hence *p* ≠ *0* **by** *auto*
show *?thesis*
proof (*rule multiplicity-le-imp-dvd'*)
fix *q* :: *'a poly* **assume** *q*: *q* ∈ *prime-factors* (*p div radical p*)
hence *q dvd p div radical p* **by** *auto*
also from ‹*p* ≠ *0*› **have** ... *dvd p* **by** (*subst div-dvd-iff-mult*) *auto*
finally have *q dvd p* **.**

have *p* = *p div radical p* ∗ *radical p* **by** *simp*
also from *q* **and** ‹*p* ≠ *0*› **have** *multiplicity q* ... = *Suc* (*multiplicity q* (*p div radical p*))
by (*subst prime-elem-multiplicity-mult-distrib*)
(*auto simp*: *dvd-div-eq-0-iff multiplicity-radical-prime* ‹*q dvd p*› *prime-factors-dvd*)
finally have *multiplicity q* (*p div radical p*) ≤ *multiplicity q p* − *1* **by** *simp*
also have ... ≤ *multiplicity q* (*pderiv p*) **using** ‹*pderiv p* ≠ *0*› **and** *q* **and** ‹*p* ≠ *0*›
by (*intro multiplicity-geI prime-power-dvd-pderiv*)
(*auto simp*: *prime-factors-dvd dvd-div-eq-0-iff*)
finally show *multiplicity q* (*p div radical p*) ≤ *multiplicity q* (*pderiv p*) **.**
qed (*insert* ‹*p* ≠ *0*›, *auto simp*: *dvd-div-eq-0-iff*)
qed *auto*

lemma *degree-pderiv-mult-less*:
assumes *pderiv C* ≠ *0*
shows *degree* (*pderiv C* ∗ *B*) < *degree B* + *degree C*
proof −
have *degree* (*pderiv C* ∗ *B*) ≤ *degree* (*pderiv C*) + *degree B*
by (*rule degree-mult-le*)
also from *assms* **have** *degree* (*pderiv C*) < *degree C* **by** (*rule degree-pderiv-less*)
finally show *?thesis* **by** *simp*
qed

lemma *Mason-Stothers-aux*:

fixes *A B C* :: *'a* :: *field-gcd poly*
assumes *nz*: *A* ≠ *0 B* ≠ *0 C* ≠ *0* **and** *sum*: *A* + *B* + *C* = *0* **and** *coprime*: *Gcd* {*A*, *B*, *C*} = *1*
and *deg-ge*: *degree A* ≥ *degree* (*radical* (*A* ∗ *B* ∗ *C*))
shows *pderiv A* = *0 pderiv B* = *0 pderiv C* = *0*
proof −
have *C-eq*: *C* = −*A* − *B* −*C* = *A* + *B* **using** *sum* **by** *algebra*+
from *coprime* **have** *gcd A* (*gcd B* (−*C*)) = *1* **by** *simp*
also note *C-eq*(*2*)
finally have *coprime A B* **by** (*simp add*: *gcd.commute add.commute*[*of A B*] *coprime-iff-gcd-eq-1*)
hence *coprime A* (−*C*) *coprime B* (−*C*)
unfolding *C-eq* **by** (*simp-all add*: *gcd.commute*[*of B A*] *gcd.commute*[*of B A* + *B*]
add.commute coprime-iff-gcd-eq-1)
hence *coprime A C coprime B C* **by** *simp-all*
note *coprime* = *coprime* ‹*coprime A B*› *this*
have *coprime1*: *coprime* (*A div radical A*) (*B div radical B*)
by (*rule coprime-divisors*[*OF* - - ‹*coprime A B*›]) (*insert nz*, *auto simp*: *div-dvd-iff-mult*)
have *coprime2*: *coprime* (*A div radical A*) (*C div radical C*)
by (*rule coprime-divisors*[*OF* - - ‹*coprime A C*›]) (*insert nz*, *auto simp*: *div-dvd-iff-mult*)
have *coprime3*: *coprime* (*B div radical B*) (*C div radical C*)
by (*rule coprime-divisors*[*OF* - - ‹*coprime B C*›]) (*insert nz*, *auto simp*: *div-dvd-iff-mult*)
have *coprime4*: *coprime* (*A div radical A* ∗ (*B div radical B*)) (*C div radical C*)
using *coprime2 coprime3* **by** (*subst coprime-mult-left-iff*) *auto*

have *eq*: *A* ∗ *pderiv B* − *pderiv A* ∗ *B* = *pderiv C* ∗ *B* − *C* ∗ *pderiv B*
by (*simp add*: *C-eq pderiv-add pderiv-diff pderiv-minus algebra-simps*)

have *A div radical A dvd* (*A* ∗ *pderiv B* − *pderiv A* ∗ *B*)
using *nz* **by** (*intro dvd-diff dvd-mult2 poly-div-radical-dvd-pderiv*) (*auto simp*: *div-dvd-iff-mult*)
with *eq* **have** *A div radical A dvd* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*) **by** *simp*
moreover have *C div radical C dvd* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*)
using *nz* **by** (*intro dvd-diff dvd-mult2 poly-div-radical-dvd-pderiv*) (*auto simp*: *div-dvd-iff-mult*)
moreover have *B div radical B dvd* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*)
using *nz* **by** (*intro dvd-diff dvd-mult poly-div-radical-dvd-pderiv*) (*auto simp*: *div-dvd-iff-mult*)
ultimately have (*A div radical A*) ∗ (*B div radical B*) ∗ (*C div radical C*) *dvd*
(*pderiv C* ∗ *B* − *C* ∗ *pderiv B*) **using** *coprime coprime1 coprime4*
by (*intro divides-mult*) *auto*
also have (*A div radical A*) ∗ (*B div radical B*) ∗ (*C div radical C*) =
(*A* ∗ *B* ∗ *C*) *div* (*radical A* ∗ *radical B* ∗ *radical C*)
by (*simp add*: *div-mult-div-if-dvd mult-dvd-mono*)
also have *radical A* ∗ *radical B* ∗ *radical C* = *radical* (*A* ∗ *B*) ∗ *radical C*
using *coprime* **by** (*subst radical-mult-coprime*) *auto*

also have . . . = *radical* (*A* ∗ *B* ∗ *C*)
using *coprime* **by** (*subst radical-mult-coprime* [*symmetric*]) *auto*
finally have *dvd*: ((*A* ∗ *B* ∗ *C*) *div radical* (*A* ∗ *B* ∗ *C*)) *dvd* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*) **.**

have *pderiv B* = *0* ∧ *pderiv C* = *0*
proof (*rule ccontr*)
assume ¬(*pderiv B* = *0* ∧ *pderiv C* = *0*)
hence ∗: *pderiv B* ≠ *0* ∨ *pderiv C* ≠ *0* **by** *blast*

have *degree* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*) ≤
max (*degree* (*pderiv C* ∗ *B*)) (*degree* (*C* ∗ *pderiv B*)) **by** (*rule degree-diff-le-max*)
also have . . . < *degree B* + *degree C*
using *degree-pderiv-mult-less*[*of B C*] *degree-pderiv-mult-less*[*of C B*] ∗
by (*cases pderiv B* = *0*; *cases pderiv C* = *0*) (*auto simp add*: *algebra-simps*)
also have *degree B* + *degree C* = *degree* (*B* ∗ *C*)
using *nz* **by** (*subst degree-mult-eq*) *auto*
also have . . . = *degree* (*A* ∗ (*B* ∗ *C*)) − *degree A*
using *nz* **by** (*subst* (*2*) *degree-mult-eq*) *auto*
also have . . . ≤ *degree* (*A* ∗ *B* ∗ *C*) − *degree* (*radical* (*A* ∗ *B* ∗ *C*)) **unfolding** *mult.assoc*
using *assms* **by** (*intro diff-le-mono2*) (*auto simp*: *mult-ac*)
also have . . . = *degree* ((*A* ∗ *B* ∗ *C*) *div radical* (*A* ∗ *B* ∗ *C*))
by (*intro degree-div* [*symmetric*]) *auto*
finally have *less*: *degree* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*) <
degree (*A* ∗ *B* ∗ *C div radical* (*A* ∗ *B* ∗ *C*)) **by** *simp*

have *eq'*: *pderiv C* ∗ *B* − *C* ∗ *pderiv B* = *0*
proof (*rule ccontr*)
assume *pderiv C* ∗ *B* − *C* ∗ *pderiv B* ≠ *0*
hence *degree* (*A* ∗ *B* ∗ *C div radical* (*A* ∗ *B* ∗ *C*)) ≤ *degree* (*pderiv C* ∗ *B* − *C* ∗ *pderiv B*)
using *dvd* **by** (*intro dvd-imp-degree-le*) *auto*
with *less* **show** *False* **by** *linarith*
qed
from ∗ **show** *False*
proof (*elim disjE*)
assume [*simp*]: *pderiv C* ≠ *0*
have *C dvd C* ∗ *pderiv B* **by** *simp*
also from *eq'* **have** . . . = *pderiv C* ∗ *B* **by** *simp*
finally have *C dvd pderiv C* **using** *coprime*
by (*subst* (*asm*) *coprime-dvd-mult-left-iff*) (*auto simp*: *coprime-commute*)
hence *degree C* ≤ *degree* (*pderiv C*) **by** (*intro dvd-imp-degree-le*) *auto*
moreover have *degree* (*pderiv C*) < *degree C* **by** (*intro degree-pderiv-less*) *auto*
ultimately show *False* **by** *simp*
next
assume [*simp*]: *pderiv B* ≠ *0*

have *B dvd B ∗ pderiv C* **by** *simp*
also from *eq′* **have** . . . = *pderiv B ∗ C* **by** (*simp add*: *mult-ac*)
finally have *B dvd pderiv B* **using** *coprime*
by (*subst* (*asm*) *coprime-dvd-mult-left-iff*) *auto*
hence *degree B* ≤ *degree* (*pderiv B*) **by** (*intro dvd-imp-degree-le*) *auto*
moreover have *degree* (*pderiv B*) < *degree B* **by** (*intro degree-pderiv-less*) *auto*
ultimately show *False* **by** *simp*
qed
qed
with *eq* **and** *nz* **show** *pderiv A = 0 pderiv B = 0 pderiv C = 0* **by** *auto*
qed

theorem *Mason-Stothers*:
fixes *A B C* :: *′a* :: *field-gcd poly*
assumes *nz*: *A ≠ 0 B ≠ 0 C ≠ 0* ∃ *p*∈{*A*,*B*,*C*}. *pderiv p ≠ 0*
and *sum*: *A + B + C = 0* **and** *coprime*: *Gcd* {*A*, *B*, *C*} = *1*
shows *Max* {*degree A*, *degree B*, *degree C*} < *degree* (*radical* (*A ∗ B ∗ C*))
proof −
have *degree A* < *degree* (*radical* (*A ∗ B ∗ C*))
if ∀ *p*∈{*A*,*B*,*C*}. *p ≠ 0* ∃ *p*∈{*A*,*B*,*C*}. *pderiv p ≠ 0 sum-mset* {#*A*,*B*,*C*#} = *0 Gcd* {*A*, *B*, *C*} = *1*
for *A B C* :: *′a poly*
proof (*rule ccontr*)
assume ¬(*degree A* < *degree* (*radical* (*A ∗ B ∗ C*)))
hence *degree A* ≥ *degree* (*radical* (*A ∗ B ∗ C*)) **by** *simp*
with *Mason-Stothers-aux*[*of A B C*] *that* **show** *False* **by** (*auto simp*: *add-ac*)
qed
from *this*[*of A B C*] *this*[*of B C A*] *this*[*of C A B*] *assms* **show** *?thesis*
by (*simp only*: *insert-commute mult-ac add-ac*) (*auto simp*: *add-ac mult-ac*)
qed

The result can be simplified a bit more in fields of characteristic 0:

corollary *Mason-Stothers-char-0*:
fixes *A B C* :: *′a* :: {*field-gcd*, *field-char-0*} *poly*
assumes *nz*: *A ≠ 0 B ≠ 0 C ≠ 0* **and** *deg*: ∃ *p*∈{*A*,*B*,*C*}. *degree p ≠ 0*
and *sum*: *A + B + C = 0* **and** *coprime*: *Gcd* {*A*, *B*, *C*} = *1*
shows *Max* {*degree A*, *degree B*, *degree C*} < *degree* (*radical* (*A ∗ B ∗ C*))
proof −
from *deg* **have** ∃ *p*∈{*A*,*B*,*C*}. *pderiv p ≠ 0*
by (*auto simp*: *pderiv-eq-0-iff*)
from *Mason-Stothers*[*OF assms*(*1*−*3*) *this assms*(*5*−)] **show** *?thesis* **.**
qed

As a nice corollary, we get a kind of analogue of Fermat's last theorem for polynomials: Given non-zero polynomials A, B, C with $A^n + B^n + C^n = 0$ on lowest terms, we must either have $n \leq 2$ or $(A^n)' = (B^n)' = (C^n)' = 0$.

In the case of a field with characteristic 0, this last possibility is equivalent to A, B, and C all being constant.

```
corollary fermat-poly:
  fixes A B C :: 'a :: field-gcd poly
  assumes sum: A ^ n + B ^ n + C ^ n = 0 and cop: Gcd {A, B, C} = 1
  assumes nz: A ≠ 0 B ≠ 0 C ≠ 0   and deg: ∃ p∈{A,B,C}. pderiv (p ^ n) ≠ 0
  shows n ≤ 2
proof (rule ccontr)
  assume ¬(n ≤ 2)
  hence n > 2 by simp
  have Max {degree (A ^ n), degree (B ^ n), degree (C ^ n)} <
        degree (radical (A ^ n ∗ B ^ n ∗ C ^ n)) (is - < ?d)
    using assms by (intro Mason-Stothers) (auto simp: degree-power-eq gcd-exp)
  hence Max {degree (A ^ n), degree (B ^ n), degree (C ^ n)} + 1 ≤ ?d by linarith
  hence n ∗ degree A + 1 ≤ ?d n ∗ degree B + 1 ≤ ?d n ∗ degree C + 1 ≤ ?d
    using assms by (simp-all add: degree-power-eq)
  hence n ∗ (degree A + degree B + degree C) + 3 ≤ 3 ∗ ?d
    unfolding ring-distribs by linarith
  also have A ^ n ∗ B ^ n ∗ C ^ n = (A ∗ B ∗ C) ^ n by (simp add: mult-ac power-mult-distrib)
  also have radical ... = radical (A ∗ B ∗ C)
    using ‹n > 2› by simp
  also have degree (radical (A ∗ B ∗ C)) ≤ degree (A ∗ B ∗ C)
    using nz by (intro dvd-imp-degree-le) auto
  also have ... = degree A + degree B + degree C
    using nz by (simp add: degree-mult-eq)
  finally have (3 − n) ∗ (degree A + degree B + degree C) ≥ 3
    by (simp add: algebra-simps)
  hence 3 − n ≠ 0 by (intro notI) auto
  hence n < 3 by simp
  with ‹n > 2› show False by simp
qed

corollary fermat-poly-char-0:
  fixes A B C :: 'a :: {field-gcd,field-char-0} poly
  assumes sum: A ^ n + B ^ n + C ^ n = 0 and cop: Gcd {A, B, C} = 1
  assumes nz: A ≠ 0 B ≠ 0 C ≠ 0   and deg: ∃ p∈{A,B,C}. degree p > 0
  shows n ≤ 2
proof (rule ccontr)
  assume ∗: ¬(n ≤ 2)
  with nz and deg have ∃ p∈{A,B,C}. pderiv (p ^ n) ≠ 0
    by (auto simp: pderiv-eq-0-iff degree-power-eq)
  from fermat-poly[OF assms(1−5) this] and ∗ show False by simp
qed

end
```

References

[1] F. Lemmermeyer. Algebraic Geometry (lecture notes). http://www.fen.bilkent.edu.tr/~franz/ag05/ag-02.pdf, 2005.

[2] N. Snyder. An alternate proof of Mason's theorem. *Elemente der Mathematik*, 55(3):93–94, Aug 2000.