Continued Fractions in Lean

A Newbie's Adventure

Kevin Kappelmann

June 14, 2019

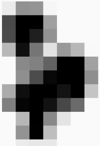
Vrije Universiteit Amsterdam

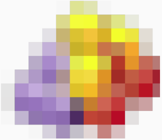
Let’s Go on an Adventure

Choose a Weapon

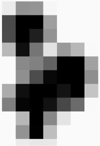
Choose a Weapon

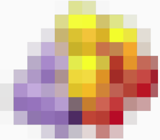
Choose a Weapon





Choose a Weapon





...perhaps because I am interning at VU Amsterdam

The Adventurer’s Skill Set

The Adventurer’s Skill Set

- Some experience using Isabelle

The Adventurer’s Skill Set

- Some experience using Isabelle
- First project with a dependent type theorem prover

The Adventurer's Skill Set

- Some experience using Isabelle
- First project with a dependent type theorem prover
- Basic maths and functional programming knowledge


S

Definitions

Generalized Continued Fractions

A generalized continued fraction is...

Generalized Continued Fractions

A generalized continued fraction is...

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

Generalized Continued Fractions

A generalized continued fraction is...

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

- b is called the *integer part*

Generalized Continued Fractions

A generalized continued fraction is...

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

- b is called the *integer part*
- each a_i is a *partial numerator*

Generalized Continued Fractions

A generalized continued fraction is...

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

- b is called the *integer part*
- each a_i is a *partial numerator*
- each b_i is a *partial denominator*

Generalized Continued Fractions of π

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \ddots}}}}}$$

Generalized Continued Fractions of π

Continued fraction

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \ddots}}}}}$$

Generalized Continued Fractions of π

Continued fraction

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \ddots}}}}}$$

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \ddots}}}}}$$

Generalized Continued Fractions of π

Continued fraction

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \ddots}}}}}$$

Generalized continued fraction

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \ddots}}}}}$$

Generalized Continued Fractions in Lean

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

Generalized Continued Fractions in Lean

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

```
/- Fix a type -/
variable (α : Type*)

/-- A gcf_pair consists of a partial numerator a and partial denominator b -/
structure gcf_pair := (a : α) (b : α)
```

Generalized Continued Fractions in Lean

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

```
/- Fix a type -/
variable (α : Type*)

/-- A gcf_pair consists of a partial numerator a
↪  and partial denominator b -/
structure gcf_pair := (a : α) (b : α)
```

```
-- Once a sequence hits none, it stays none
def seq := {f : ℕ → option α // ∀ {n}, f n = none →
↪  f (n + 1) = none}
```

Generalized Continued Fractions in Lean

$$b + \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \ddots}}}}$$

```
/- Fix a type -/
variable (α : Type*)

/-- A gcf_pair consists of a partial numerator a
↪  and partial denominator b -/
structure gcf_pair := (a : α) (b : α)
```

```
def seq := {f : ℕ → option α // ∀ {n}, f n = none →
 ↪  f (n + 1) = none}
/-- A generalized continued fraction consists of a
 ↪  leading head term (the "integer part") and a
 ↪  sequence of partial partial numerators aₙ and
 ↪  partial denominators bₙ -/
structure gcf := (head : α) (seq : seq (gcf_pair
 ↪  α))
```

Evaluate Generalized Continued Fractions

```
def convergents (g : gcf α) (n : ℕ) : α :=
g.head + if n = 0 then 0 else aux n g.seq
```

Evaluate Generalized Continued Fractions

```
def aux : ℕ → seq (gcf_pair α) → α
| 0 s := match s.head with
  | none := 0
  | some ⟨a, b⟩ := a / b
  end
| (n + 1) s := match s.head with
  | none := 0
  | some ⟨a, b⟩ := a / (b + aux n s.tail)
  end

def convergents (g : gcf α) (n : ℕ) : α :=
g.head + if n = 0 then 0 else aux n g.seq
```

Continued Fractions

$$b + \cfrac{1}{b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ddots}}}}$$

Continued Fractions

$$b + \cfrac{1}{b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ddots}}}}$$

```
/-- A continued fraction is a gcf whose partial numerators are equal to 1. -/
def cf := {g : gcf α // ∀ (n : ℕ) (a : α), (partial_numerators g).nth n = some a → a = 1}
```

Continued Fractions

$$b + \cfrac{1}{b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ddots}}}}$$

```
/-- A continued fraction is a gcf whose partial numerators are equal to 1. -/
def cf := {g : gcf α // ∀ (n : ℕ) (a : α), (partial_numerators g).nth n = some a → a = 1}
```

First impression:

Continued Fractions

$$b + \cfrac{1}{b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ddots}}}}$$

```
/-- A continued fraction is a gcf whose partial numerators are equal to 1. -/
def cf := {g : gcf α // ∀ (n : ℕ) (a : α), (partial_numerators g).nth n = some a → a = 1}
```

First impression: Pretty Sweet!

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent c 0
```

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent c 0
```

NOPE!

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent c 0
```

```
type mismatch at application
  continuants c
term
  c
has type
  cf α : Type u_1
but is expected to have type
  gcf ?m_1 : Type ?
```

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent c 0
```

```
type mismatch at application
  continuants c
term
  c
has type
  cf α : Type u_1
but is expected to have type
  gcf ?m_1 : Type ?
```

Oh, I see – I need to cast!

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent (c : gcf α) 0
```

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent (c : gcf α) 0
```

NOPE!

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent (c : gcf α) 0
```

```
invalid type ascription, term
has type
  cf α
but is expected to have type
  gcf α
```

Fun with Subtypes

So, since *cf* is a subtype of *gcf*, we can do

```
def convergents (g : gcf α) (n : ℕ) : α := ...

variable (c : cf α)
#check convergent (c : gcf α) 0
```

```
invalid type ascription, term
has type
  cf α
but is expected to have type
  gcf α
```

...alright, let's go on Zulip

Please Help Me

A few minutes and messages from *Kevin Buzzard* later...

The “Solution”

We first need to define the casting

The “Solution”

We first need to define the casting

```
instance cf_to_gcf : has_coe (cf β) (gcf β)
:= by {unfold cf, apply_instance}

/- Best practice: create a lemma for your cast -/
@[simp, elim_cast]
lemma coe_cf (c : cf β) : (↑c : gcf β) = c.val
:= by refl
```

The “Solution”

We first need to define the casting

```
instance cf_to_gcf : has_coe (cf β) (gcf β)
:= by {unfold cf, apply_instance}

/- Best practice: create a lemma for your cast -/
@[simp, elim_cast]
lemma coe_cf (c : cf β) : (↑c : gcf β) = c.val
:= by refl
```

Now this works:

```
variable (c : cf α)
#check convergent (c : gcf α) 0
```

The “Solution”

We first need to define the casting

```
instance cf_to_gcf : has_coe (cf β) (gcf β)
:= by {unfold cf, apply_instance}

/- Best practice: create a lemma for your cast -/
@[simp, elim_cast]
lemma coe_cf (c : cf β) : (↑c : gcf β) = c.val
:= by refl
```

This, however, still does not work:

```
variable (c : cf α)
#check convergent c 0
```

Proofs

The Proof Is Trivial

```
lemma floor_rat_eq_num_div_denom (n d : ℤ) :
  ⌊rat.mk n d⌋ = n / d
```

The Proof Is Trivial

```
lemma floor_rat_eq_num_div_denom (n d : ℤ) :
  ⌊rat.mk n d⌋ = n / d
```

Wait, let’s do some examples first…

The Proof Is Trivial

```
lemma floor_rat_eq_num_div_denom (n d : ℤ) :
  ⌊rat.mk n d⌋ = n / d
```

Alright, I am sold!

Proving... Please Wait

Proving... Please Wait

Something seems wrong

Now It Is Trivial

```
lemma floor_rat_eq_num_div_denom (n : ℤ) (d : ℕ) :
  ⌊rat.mk n d⌋ = n / d
```

That’s better!

A Short Note About Tactics

<Show two short examples in VS Code>

Results

Collected Treasures

Collected Treasures

Definition of (generalized) continued fractions and their evaluation

```
structure gcf := (head : α) (seq : seq (gcf_pair α))
def cf := {g : gcf α // ∀ (n : ℕ) (a : α), (partial_numerators g).nth n = some a → a = 1}
def convergents (g : gcf α) (n : ℕ) : α := ...
```

Collected Treasures

Computable continued fractions for discrete linear ordered floor fields

```
def get_cf [discrete_linear_ordered_field α]
 ↪  [floor_ring α] (v : α) : cf α := ...
```

Collected Treasures

Computable continued fractions for discrete linear ordered floor fields

```
def get_cf [discrete_linear_ordered_field α]
 ↪  [floor_ring α] (v : α) : cf α := ...
```

Also works for $\mathbb{R}$ – just not computable...

Collected Treasures

Termination proof for archimedean fields

```
theorem termination_iff_rat [archimedean α] (v : α) :
  Terminates (get_gcf v) ↔ ∃ (q : ℚ), v = (q : α)
```

Termination proof for archimedian fields

```
theorem termination_iff_rat [archimedean α] (v : α) :
  Terminates (get_gcf v) ↔ ∃ (q : ℚ), v = (q : α)
```

Including a theorem a mathematician would never prove:

Collected Treasures

Termination proof for archimedian fields

```
theorem termination_iff_rat [archimedean α] (v : α) :
  Terminates (get_gcf v) ↔ ∃ (q : ℚ), v = (q : α)
```

Including a theorem a mathematician would never prove:

```
theorem translate_rat_get_cf {q : ℚ}
(v_eq_q : v = q) :
  ((get_gcf q : gcf ℚ) : gcf α) = get_gcf v :=
```

Finite correctness of the computation

```
theorem get_gcf_finite_correctness
(terminates: Terminates (get_gcf v)) :
  ∃ (n : ℕ), v = convergents (get_gcf v) n
```

Some interesting inequalities, and finally:

```
theorem epsilon_convergence : ∀ (ε > (0 : α)),
  ∃ (N : ℕ), ∀ (n ≥ N),
  |v - convergents (get_gcf v) n| < ε :=
```

Collected Treasures

Some interesting inequalities, and finally:

```
theorem epsilon_convergence : ∀ (ε > (0 : α)),
  ∃ (N : ℕ), ∀ (n ≥ N),
  |v - convergents (get_gcf v) n| < ε :=
```

But sadly no library for sequence limits in Lean :(

End of the Story

Lessons Learnt

Lessons Learnt

- Lean's type system is very expressive and great for definitions...

Lessons Learnt

- Lean’s type system is very expressive and great for definitions…
    - …if one knows the gotchas.

Lessons Learnt

- Lean’s type system is very expressive and great for definitions…
    - …if one knows the gotchas.
- Support on Zulip is fantastic.

Lessons Learnt

- Lean's type system is very expressive and great for definitions...
    - ...if one knows the gotchas.
- Support on Zulip is fantastic.
- Existing tactics help a LOT...

Lessons Learnt

- Lean's type system is very expressive and great for definitions...
  - ...if one knows the gotchas.
- Support on Zulip is fantastic.
- Existing tactics help a LOT...
  - ...but no integration of automated theorem provers yet.

We Need You!

Help us making interactive theorem proving
an even better place!

Formalisation can be found at

`github.com/kappelmann/lean-continued-fractions`

Formalisation can be found at

`github.com/kappelmann/lean-continued-fractions`

$$\mathit{Thanks} + \cfrac{1}{\mathit{for} + \cfrac{1}{\mathit{your} + \cfrac{1}{\mathit{attention}!}}}$$

Formalisation can be found at

`github.com/kappelmann/lean-continued-fractions`

$$\mathit{Thanks} + \cfrac{1}{\mathit{for} + \cfrac{1}{\mathit{your} + \cfrac{1}{\mathit{attention}!}}}$$

Any questions?

Image Sources i

- Salt shaker: Modified from `bit.ly/2K8Jw8s`
- Link 1: `bit.ly/2wMGOwE`
- Link 2: `bit.ly/2RaypfX`
- Link 3: `bit.ly/2MNGUPt`
- Clock: `bit.ly/2HOc9GC`
- Melting clock: `bit.ly/2MKWknv`