

Continued Fractions in Lean

A Newbie's Adventure

Kevin Kappelmann

June 14, 2019

Vrije Universiteit Amsterdam

Let's Go on an Adventure

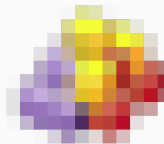
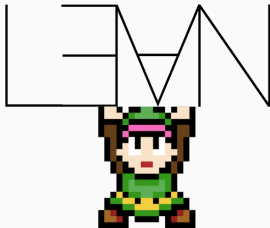
Choose a Weapon

Choose a Weapon

LEMN



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



Definitions

Generalized Continued Fractions

A generalized continued fraction is...

Generalized Continued Fractions

A generalized continued fraction is...

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

Generalized Continued Fractions

A generalized continued fraction is...

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

- b is called the *integer part*

Generalized Continued Fractions

A generalized continued fraction is...

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

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

Generalized Continued Fractions

A generalized continued fraction is...

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

- 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

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

Generalized Continued Fractions of π

Continued fraction

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

Generalized Continued Fractions of π

Continued fraction

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

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

Generalized Continued Fractions of π

Continued fraction

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

Generalized continued fraction

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

Generalized Continued Fractions in Lean

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

Generalized Continued Fractions in Lean

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

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

Generalized Continued Fractions in Lean

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

```
1  /- Fix a type -/  
2  variable (α : Type*)  
3  
4  /-- A gcf_pair consists of a partial numerator  
   ↪ a and partial denominator b -/  
5  structure gcf_pair := (a : α) (b : α)
```

Generalized Continued Fractions in Lean

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

/- 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

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

/- 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 a_n and
↪ partial denominators b_n -/*

structure gcf := (head : α) (seq : seq (gcf_pair
↪ α))

Evaluate Generalized Continued Fractions

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

Evaluate Generalized Continued Fractions

```
1 def aux :  $\mathbb{N}$   $\rightarrow$  seq (gcf_pair a)  $\rightarrow$  a
2 | 0 s := match s.head with
3   | none := 0
4   | some <a, b> := a / b
5   end
6 | (n + 1) s := match s.head with
7   | none := 0
8   | some <a, b> := a / (b + aux n s.tail)
9   end
10
11 def convergents (g : gcf a) (n :  $\mathbb{N}$ ) : a :=
12 g.head + if n = 0 then 0 else aux n g.seq
```

Continued Fractions

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

Continued Fractions

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

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

Continued Fractions

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

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

First impression:

Continued Fractions

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

```
1 /-- A continued fraction is a gcf whose partial
   ↪ numerators are equal to 1. -/
2 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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #check convergent c 0
```

Fun with Subtypes

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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #check convergent c 0
```

NOPE!

Fun with Subtypes

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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #check convergent (c : gcf a) 0
```

Fun with Subtypes

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

```
1 def convergents (g : gcf a) (n : ℕ) : a := ...
2
3 variable (c : cf a)
4 #check convergent (c : gcf a) 0
```

NOPE!

Fun with Subtypes

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

```
1 def convergents (g : gcf α) (n : ℕ) : α := ...
2
3 variable (c : cf α)
4 #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

```
1 def convergents (g : gcf α) (n : ℕ) : α := ...
2
3 variable (c : cf α)
4 #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

```
1 instance cf_to_gcf : has_coe (cf  $\beta$ ) (gcf  $\beta$ )
2 := by {unfold cf, apply_instance}
3
4 /- Best practice: create a lemma for your cast -/
5 @[simp, elim_cast]
6 lemma coe_cf (c : cf  $\beta$ ) : ( $\uparrow$ c : gcf  $\beta$ ) = c.val
7 := by refl
```

The “Solution”

We first need to define the casting

```
1 instance cf_to_gcf : has_coe (cf  $\beta$ ) (gcf  $\beta$ )
2 := by {unfold cf, apply_instance}
3
4 /- Best practice: create a lemma for your cast -/
5 @[simp, elim_cast]
6 lemma coe_cf (c : cf  $\beta$ ) : ( $\uparrow$ c : gcf  $\beta$ ) = c.val
7 := by refl
```

Now this works:

```
1 variable (c : cf  $\alpha$ )
2 #check convergent (c : gcf  $\alpha$ ) 0
```

The “Solution”

We first need to define the casting

```
1 instance cf_to_gcf : has_coe (cf  $\beta$ ) (gcf  $\beta$ )
2 := by {unfold cf, apply_instance}
3
4 /- Best practice: create a lemma for your cast -/
5 @[simp, elim_cast]
6 lemma coe_cf (c : cf  $\beta$ ) : ( $\uparrow$ c : gcf  $\beta$ ) = c.val
7 := by refl
```

This, however, still does not work:

```
1 variable (c : cf  $\alpha$ )
2 #check convergent c 0
```

Proofs

The Proof Is Trivial

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

The Proof Is Trivial

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

Wait, let's do some examples first...

The Proof Is Trivial

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

Alright, I am sold!

Proving... Please Wait



Proving... Please Wait



Something seems wrong

Now It Is Trivial

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

That's better!

A Short Note About Tactics

<Show two short examples in VS Code>

Results

Collected Treasures



Definition of (generalized) continued fractions and their evaluation

```
1 structure gcf := (head :  $\alpha$ ) (seq : seq (gcf_pair  
    $\hookrightarrow$   $\alpha$ ))  
2 def cf := {g : gcf  $\alpha$  //  $\forall$  (n :  $\mathbb{N}$ ) (a :  $\alpha$ ),  
    $\hookrightarrow$  (partial_numerators g).nth n = some a  $\rightarrow$  a = 1}  
3 def convergents (g : gcf  $\alpha$ ) (n :  $\mathbb{N}$ ) :  $\alpha$  := ...
```

Computable continued fractions for discrete linear ordered floor fields

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

Computable continued fractions for discrete linear ordered floor fields

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

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

Termination proof for archimedean fields

```
1 theorem termination_iff_rat [archimedean  $\alpha$ ] (v :  $\alpha$ )  
   $\hookrightarrow$  :  
2   Terminates (get_gcf v)  $\leftrightarrow \exists$  (q :  $\mathbb{Q}$ ), v = (q :  $\alpha$ )
```

Termination proof for archimedean fields

```
1 theorem termination_iff_rat [archimedean  $\alpha$ ] (v :  $\alpha$ )  
   $\hookrightarrow$  :  
2   Terminates (get_gcf v)  $\leftrightarrow \exists$  (q :  $\mathbb{Q}$ ), v = (q :  $\alpha$ )
```

Including a theorem a mathematician would never prove:

Collected Treasures

Termination proof for archimedean fields

```
1 theorem termination_iff_rat [archimedean  $\alpha$ ] (v :  $\alpha$ )  
   $\leftrightarrow$  :  
2   Terminates (get_gcf v)  $\leftrightarrow \exists$  (q :  $\mathbb{Q}$ ), v = (q :  $\alpha$ )
```

Including a theorem a mathematician would never prove:

```
1 theorem translate_rat_get_cf {q :  $\mathbb{Q}$ }  
2 (v_eq_q : v = q) :  
3   ((get_gcf q : gcf  $\mathbb{Q}$ ) : gcf  $\alpha$ ) = get_gcf v :=
```

Finite correctness of the computation

```
1 theorem get_gcf_finite_correctness
2 (terminates: Terminates (get_gcf v)) :
3    $\exists$  (n :  $\mathbb{N}$ ), v = convergents (get_gcf v) n
```

Some interesting inequalities, and finally:

```
1 theorem epsilon_convergence :  $\forall (\epsilon > (0 : \alpha)),$   
2    $\exists (N : \mathbb{N}), \forall (n \geq N),$   
3    $|v - \text{convergets } (\text{get\_gcf } v) \ n| < \epsilon :=$ 
```

Some interesting inequalities, and finally:

```
1 theorem epsilon_convergence :  $\forall (\epsilon > (0 : \alpha)),$   
2    $\exists (N : \mathbb{N}), \forall (n \geq N),$   
3    $|v - \text{convergets } (\text{get\_gcf } v) \ n| < \epsilon :=$ 
```

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`

$$\begin{array}{r} \textit{Thanks} + \frac{1}{\textit{for} + \frac{1}{\textit{your} + \frac{1}{\textit{attention!}}}} \end{array}$$

Formalisation can be found at

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

$$\begin{array}{c} \textit{Thanks} + \frac{1}{\textit{for} + \frac{1}{\textit{your} + \frac{1}{\textit{attention!}}}} \end{array}$$

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