Continued Fractions in Lean

A Newbie's Adventure

Kevin Kappelmann

June 14, 2019

Vrije Universiteit Amsterdam

Let's Go on an Adventure
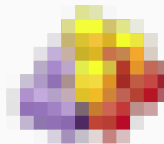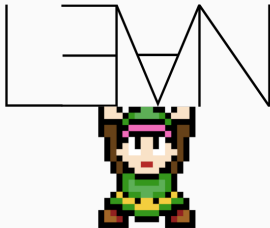
...perhaps because I am interning at VU Amsterdam

The Adventurer's Skill Set

- Some experience using Isabelle

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

- 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...

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

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*

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

Continued fraction

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

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

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

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

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

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

Generalized Continued Fractions in Lean

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

```
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)
```

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

```
/- 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

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

Evaluate Generalized Continued Fractions

```
1 def aux : ℕ → seq (gcf_pair α) → α
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 α) (n : ℕ) : α :=
12 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

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

```
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

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

```
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:

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

```
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!

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 convergents c 0
```

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 convergents c 0
```

NOPE!

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 convergents c 0
```

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

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 convergents c 0
```

```
type mismatch at
application
  convergents 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 α) (n : ℕ) : α := ...
2
3 variable (c : cf α)
4 #check convergents (c : gcf α) 0
```

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 convergents (c : gcf α) 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 convergents (c : gcf α) 0
```

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

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 convergents (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 β) (gcf β)
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 β) : (↑c : gcf β) = c.val
7 := by refl
```

The "Solution"

We first need to define the casting

```
1 instance cf_to_gcf : has_coe (cf β) (gcf β)
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 β) : (↑c : gcf β) = c.val
7 := by refl
```

Now this works:

```
1 variable (c : cf α)
2 #check convergents (c : gcf α) 0
```

We first need to define the casting

```
1 instance cf_to_gcf : has_coe (cf β) (gcf β)
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 β) : (↑c : gcf β) = c.val
7 := by refl
```

This, however, still does not work:

```
1 variable (c : cf α)
2 #check convergents c 0
```

Proofs

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

```
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...

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

Alright, I am sold!

Something seems wrong

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

Collected Treasures

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 := ...
```

Collected Treasures

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 archimedian fields

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

Collected Treasures

Termination proof for archimedian fields

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

Including a theorem a mathematician would never prove:

Collected Treasures

Termination proof for archimedian fields

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

Including a theorem a mathematician would never prove:

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

Finite correctness of the computation

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

Some interesting inequalities, and finally:

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

Collected Treasures

Some interesting inequalities, and finally:

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

But sadly no library for sequence limits in Lean :(

End of the Story

Lessons Learnt

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

- 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.

- 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.

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

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

Formalisation can be found at
`github.com/kappelmann/lean-continued-fractions`

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

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`