Gmail

William DeMeo <williamdemeo@gmail.com>

# [lean-user] Question about mathlib/data/set/finite and mathlib/data/finsupp

10 messages

**Ching-Tsun Chou** <chingtsun.chou@gmail.com> Mon, Feb 12, 2018 at 12:25 AM
To: lean-user <lean-user@googlegroups.com>

In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable. Why? Intuitively, the use of strong classical logic does not seem necessary here. A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements. So it would seem that the same list can be used to make the finset that the finite set is converted to.

I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?

Thanks!
- Ching Tsun



**Mario Carneiro** <di.gama@gmail.com> Mon, Feb 12, 2018 at 12:37 AM
To: Ching-Tsun Chou <chingtsun.chou@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

Hi Ching-Tsun,

On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:

> In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable. Why? Intuitively, the use of strong classical logic does not seem necessary here. A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements. So it would seem that the same list can be used to make the finset that the finite set is converted to.

A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)

> I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?

Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.

Mario



---

**Ching-Tsun Chou** <chingtsun.chou@gmail.com> Mon, Feb 12, 2018 at 12:53 AM
To: Mario Carneiro <di.gama@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

But isn't the base logic of Lean a constructive logic, where "exists" means having a witness? I also observe that mathlib/data/finset is developed without "noncomputable". Why is "finite" substantially different from "finset"?

Thanks!
- Ching Tsun

On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail.com> wrote:

> Hi Ching-Tsun,
>
> On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>
>> In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable. Why? Intuitively, the use of strong classical logic does not seem necessary here. A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements. So it would seem that the same list can be used to make the finset that the finite set is converted to.
>
> A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)
>
>> I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?
>
> Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.
>
> Mario

--
You received this message because you are subscribed to the Google Groups "lean-user" group.
To unsubscribe from this group and stop receiving emails from it, send an email to lean-user+unsubscribe@googlegroups.com.
For more options, visit https://groups.google.com/d/optout.

---

**Mario Carneiro** <di.gama@gmail.com>
To: Ching-Tsun Chou <chingtsun.chou@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

Mon, Feb 12, 2018 at 1:29 AM

On Mon, Feb 12, 2018 at 2:53 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:

> But isn't the base logic of Lean a constructive logic, where "exists" means having a witness?

The purpose of the Prop universe is exactly to be able to forget about witnesses. A member of Exists x, p x does not contain an x such that p x; this would be the (data) type {x // p x}. In fact, if it did, it would fail proof irrelevance, because if h1 h2 : Exists x, p x, and there was a projection operation h1.1 : A such that h1.1 != h2.1, then this would imply h1 != h2, contradicting proof irrelevance.

It is true that the base logic of lean is constructive in the sense that if you can prove Exists x, p x without using the axiom of choice, then you can also prove {x // p x}; but this is a proof translation procedure (or a meta-function if you like), not a function you can apply to the proof of Exists x, p x.

> I also observe that mathlib/data/finset is developed without "noncomputable". Why is "finite" substantially different from "finset"?

Because finset (and fintype) are in Type, while finite is in Prop. If you need a computable witness to finiteness, use "fintype s" instead of "finite s". That's what it's there for.

Mario

> Thanks!
> \- Ching Tsun
>
> On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail.com> wrote:
>
>> Hi Ching-Tsun,
>>
>> On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>>
>>> In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable. Why? Intuitively, the use of strong classical logic does not seem necessary here. A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements. So it would seem that the same list can be used to make the finset that the finite set is converted to.
>>
>> A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)
>>
>>> I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?
>>
>> Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.
>>
>> Mario

---

**Ching-Tsun Chou** <chingtsun.chou@gmail.com>
To: Mario Carneiro <di.gama@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>
Mon, Feb 12, 2018 at 1:44 AM

"finite" is defined as:

```
def finite (s : set α) : Prop := nonempty (fintype s)
```

TPIL says nonempty is equivalent to exists:

https://leanprover.github.io/theorem_proving_in_lean/axioms_and_computation.html#choice

```
example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)
```

So, via the definition of fintype, s being finite gives a witness finset which enumerates the elements of s.  The detour through classical.choice seems unnecessary.

Thanks!
- Ching Tsun

On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:

> But isn't the base logic of Lean a constructive logic, where "exists" means having a witness?  I also observe that mathlib/data/finset is developed without "noncomputable".  Why is "finite" substantially different from "finset"?
>
> Thanks!
> - Ching Tsun
>
> On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail.com> wrote:
>
>> Hi Ching-Tsun,
>>
>> On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>>
>>> In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable.  Why?  Intuitively, the use of strong classical logic does not seem necessary here.  A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements.  So it would seem that the same list can be used to make the finset that the finite set is converted to.
>>
>> A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that

> the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)
>
> > I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?
>
> Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.
>
> Mario



---

**Mario Carneiro** <di.gama@gmail.com> Mon, Feb 12, 2018 at 2:01 AM
To: Ching-Tsun Chou <chingtsun.chou@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

Remember the definition of choice:

```
axiom choice {α : Sort u} : nonempty α → α
```

Given that "finite s" is literally just "nonempty (fintype s)", choice is *exactly* the function needed to produce "fintype s" from this evidence. It is not computable because in the VM, evidence of "finite s" is essentially just "unit.star", there is no information in it; and to produce a list from this is quite impossible.

Mario

On Mon, Feb 12, 2018 at 3:44 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:

> "finite" is defined as:
>
> def finite (s : set α) : Prop := nonempty (fintype s)
>
> TPIL says nonempty is equivalent to exists:
>
> https://leanprover.github.io/theorem_proving_in_lean/axioms_and_computation.html#choice
>
> ```
> example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
> iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)
> ```
>
> So, via the definition of fintype, s being finite gives a witness finset which enumerates the elements of s. The detour through classical.choice seems unnecessary.
>
> Thanks!
> - Ching Tsun

> > On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
> > > But isn't the base logic of Lean a constructive logic, where "exists" means having a witness?  I also observe that mathlib/data/finset is developed without "noncomputable".  Why is "finite" substantially different from "finset"?
> > >
> > > Thanks!
> > > - Ching Tsun
> > >
> > > On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail.com> wrote:
> > > > Hi Ching-Tsun,
> > > >
> > > > On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
> > > > > In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable.  Why?  Intuitively, the use of strong classical logic does not seem necessary here.  A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements.  So it would seem that the same list can be used to make the finset that the finite set is converted to.
> > > >
> > > > A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)
> > > >
> > > > > I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable.  For example, finsupp.support is noncomputable only because finite.to_finset is.  For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality.  No?
> > > >
> > > > Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.
> > > >
> > > > Mario

> --
> You received this message because you are subscribed to the Google Groups "lean-user" group.
> To unsubscribe from this group and stop receiving emails from it, send an email to lean-user+unsubscribe@googlegroups.com.
> For more options, visit https://groups.google.com/d/optout.

---

**Ching-Tsun Chou** <chingtsun.chou@gmail.com> Mon, Feb 12, 2018 at 2:26 AM
To: Mario Carneiro <di.gama@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

OK, I think I'm getting it now.  But I'm still wondering: if Prop is not proof-irrelevant, would the situation change?  For example, what if the proof object remembers so much information as to make (Exists x, p x) the same as {x // p x}?  I suppose that is not the design choice Lean made.  But I wonder what are the pros and cons of each design choice.

Thanks!
- Ching Tsun

On Mon, Feb 12, 2018 at 1:01 AM, Mario Carneiro <di.gama@gmail.com> wrote:

> Remember the definition of choice:
>
> ```
> axiom choice {α : Sort u} : nonempty α → α
> ```
>
> Given that "finite s" is literally just "nonempty (fintype s)", choice is *exactly* the function needed to produce "fintype s" from this evidence. It is not computable because in the VM, evidence of "finite s" is essentially just "unit.star", there is no information in it; and to produce a list from this is quite impossible.
>
> Mario
>
> On Mon, Feb 12, 2018 at 3:44 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>
>> "finite" is defined as:
>>
>> ```
>> def finite (s : set α) : Prop := nonempty (fintype s)
>> ```
>>
>> TPIL says nonempty is equivalent to exists:
>>
>> https://leanprover.github.io/theorem_proving_in_lean/axioms_and_computation.html#choice
>>
>> ```
>> example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
>> iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)
>> ```
>>
>> So, via the definition of fintype, s being finite gives a witness finset which enumerates the elements of s.  The detour through classical.choice seems unnecessary.
>>
>> Thanks!
>> - Ching Tsun
>>
>> On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>>
>>> But isn't the base logic of Lean a constructive logic, where "exists" means having a witness?  I also observe that mathlib/data/finset is developed without "noncomputable".  Why is "finite" substantially different from "finset"?
>>>
>>> Thanks!
>>> - Ching Tsun
>>>
>>> On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail.com> wrote:
>>>
>>>> Hi Ching-Tsun,
>>>>
>>>> On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.chou@gmail.com> wrote:
>>>>
>>>>> In mathlib/data/set/finite, finite.to_finset is defined to be noncomputable.  Why?  Intuitively, the use of strong classical logic does not seem necessary here.  A set is finite iff there is a finite list with no duplicate elements enumerating the set's elements.  So it would seem that the same list can be used to

> > > > make the finset that the finite set is converted to.
> > >
> > > A set is finite if THERE EXISTS a list (slash finset) enumerating its elements. That doesn't mean that you have a function that produces such a list or finset; since "finite s" is a Prop, it contains no data and no enumerating function. To go from this to an actual list enumerating the set requires choice. (It is true that the finset so defined is unique, so that this is in fact a definite description, but Lean makes no distinction - they are both noncomputable operations.)
> > >
> > > > I am also wondering how much of mathlib/data/finsupp, which is currently declared to be a noncomputable theory, actually needs to be noncomputable. For example, finsupp.support is noncomputable only because finite.to_finset is. For another example, I'd imagine that "noncomputable" can be removed from finsupp.single by assuming the domain type has decidable equality. No?
> > >
> > > Some of this can be made computable, like finsupp.single as you've observed, but finsupp.support can't be for the reasons mentioned above. For computational purposes, finsupp is not a good fit anyway, since it deals with functions where association lists or hashmaps would be more efficient.
> > >
> > > Mario



---

**Johannes Hölzl** <johannes.hoelzl@gmx.de> Mon, Feb 12, 2018 at 2:54 AM
To: Ching-Tsun Chou <chingtsun.chou@gmail.com>, Mario Carneiro <di.gama@gmail.com>
Cc: lean-user <lean-user@googlegroups.com>

`Prop` is the proof irrelevant version, everything else lives in
`Type`. Changing this would meant to loose `Prop` but not gain
anything. There are multiple reasons why we want to have Prop:

* it *impredicative*, i.e. closed under function space
  for a : Type u, p : Prop we have a -> p : Prop

  For a : Type u, b : Type v we have a -> b : Type (max u v).

* proof irrelevance is baked into the kernel, the kernel knows:
    h_1 =def= h_2 (for h_1, h_2 : p : Prop)
  This is an very important feature for e.g. type classes.

* many things which require truncation are for free just by
  working in Prop

* it is very natural to extend to classical logic (just add choice or
  at LEM)

So if we remove Prop, then our logic would be very different and it
would be difficult to write automation.

If the proof object is required to 'remember' all this informations,
then we it is not anymore proof irrelevant and we loose this property.

The cons of Prop are that you can not work in a setting a la HoTT. But
HoTT is very much in development people still explore how to implement
it in the right way.

- Johannes

Am Montag, den 12.02.2018, 01:26 -0800 schrieb Ching-Tsun Chou:
> OK, I think I'm getting it now.  But I'm still wondering: if Prop is
> not proof-irrelevant, would the situation change?  For example, what
> if the proof object remembers so much information as to make (Exists
> x, p x) the same as {x // p x}?  I suppose that is not the design
> choice Lean made.  But I wonder what are the pros and cons of each
> design choice.
>
> Thanks!
> - Ching Tsun
>
>
> On Mon, Feb 12, 2018 at 1:01 AM, Mario Carneiro <di.gama@gmail.com>
> wrote:
> > Remember the definition of choice:
> > axiom choice {α : Sort u} : nonempty α → α
> > Given that "finite s" is literally just "nonempty (fintype s)",
> > choice is exactly the function needed to produce "fintype s" from
> > this evidence. It is not computable because in the VM, evidence of
> > "finite s" is essentially just "unit.star", there is no information
> > in it; and to produce a list from this is quite impossible.
> > Mario
> >
> > On Mon, Feb 12, 2018 at 3:44 AM, Ching-Tsun Chou <chingtsun.chou@gm
> > ail.com> wrote:
> > > "finite" is defined as:
> > >
> > > def finite (s : set α) : Prop := nonempty (fintype s)
> > >
> > > TPIL says nonempty is equivalent to exists:
> > >
> > > https://leanprover.github.io/theorem_proving_in_lean/axioms_and_c
> > > omputation.html#choice
> > > example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
> > > iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)

> > > So, via the definition of fintype, s being finite gives a witness
> > > finset which enumerates the elements of s.  The detour through
> > > classical.choice seems unnecessary.
> > >
> > > Thanks!
> > > - Ching Tsun
> > >
> > >
> > >
> > > On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou
> > > @gmail.com> wrote:
> > > > But isn't the base logic of Lean a constructive logic, where
> > > > "exists" means having a witness?  I also observe that
> > > > mathlib/data/finset is developed without "noncomputable".  Why
> > > > is "finite" substantially different from "finset"?
> > > >
> > > > Thanks!
> > > > - Ching Tsun
> > > >
> > > >
> > > > On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail
> > > > .com> wrote:
> > > > > Hi Ching-Tsun,
> > > > >
> > > > > On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.c
> > > > > hou@gmail.com> wrote:
> > > > > > In mathlib/data/set/finite, finite.to_finset is defined to
> > > > > > be noncomputable.  Why?  Intuitively, the use of strong
> > > > > > classical logic does not seem necessary here.  A set is
> > > > > > finite iff there is a finite list with no duplicate
> > > > > > elements enumerating the set's elements.  So it would seem
> > > > > > that the same list can be used to make the finset that the
> > > > > > finite set is converted to.
> > > > > >
> > > > >
> > > > > A set is finite if THERE EXISTS a list (slash finset)
> > > > > enumerating its elements. That doesn't mean that you have a
> > > > > function that produces such a list or finset; since "finite
> > > > > s" is a Prop, it contains no data and no enumerating
> > > > > function. To go from this to an actual list enumerating the
> > > > > set requires choice. (It is true that the finset so defined
> > > > > is unique, so that this is in fact a definite description,
> > > > > but Lean makes no distinction - they are both noncomputable
> > > > > operations.)
> > > > >
> > > > > > I am also wondering how much of mathlib/data/finsupp, which
> > > > > > is currently declared to be a noncomputable theory,
> > > > > > actually needs to be noncomputable.  For example,
> > > > > > finsupp.support is noncomputable only because

> > > > > > finite.to_finset is.  For another example, I'd imagine that
> > > > > > "noncomputable" can be removed from finsupp.single by
> > > > > > assuming the domain type has decidable equality.  No?
> > > > > >
> > > > >
> > > > > Some of this can be made computable, like finsupp.single as
> > > > > you've observed, but finsupp.support can't be for the reasons
> > > > > mentioned above. For computational purposes, finsupp is not a
> > > > > good fit anyway, since it deals with functions where
> > > > > association lists or hashmaps would be more efficient.
> > > > >
> > > > > Mario
>


---

**Rob Lewis** <rob.y.lewis@gmail.com> Mon, Feb 12, 2018 at 3:16 AM
To: Johannes Hölzl <johannes.hoelzl@gmx.de>
Cc: Ching-Tsun Chou <chingtsun.chou@gmail.com>, Mario Carneiro <di.gama@gmail.com>, lean-user <lean-user@googlegroups.com>

As I understand it, impredicativity + excluded middle implies proof irrelevance (https://github.com/FStarLang/FStar/issues/360). Since the logic of Lean should remain consistent with classical axioms, a proof-relevant Prop would need to be predicative, which would make it the same as any other Type universe. Someone who wants to work in such a system can do so in Lean by ignoring Prop entirely (as I believe the Lean 3 HoTT development does).

-Rob

On Mon, Feb 12, 2018 at 10:54 AM, Johannes Hölzl <johannes.hoelzl@gmx.de> wrote:

`Prop` is the proof irrelevant version, everything else lives in
`Type`. Changing this would meant to loose `Prop` but not gain
anything. There are multiple reasons why we want to have Prop:

* it *impredicative*, i.e. closed under function space
  for a : Type u, p : Prop we have a -> p : Prop

  For a : Type u, b : Type v we have a -> b : Type (max u v).

* proof irrelevance is baked into the kernel, the kernel knows:
    h_1 =def= h_2 (for h_1, h_2 : p : Prop)
  This is an very important feature for e.g. type classes.

* many things which require truncation are for free just by
  working in Prop

* it is very natural to extend to classical logic (just add choice or
  at LEM)

So if we remove Prop, then our logic would be very different and it
would be difficult to write automation.

If the proof object is required to 'remember' all this informations,
then we it is not anymore proof irrelevant and we loose this property.

The cons of Prop are that you can not work in a setting a la HoTT. But
HoTT is very much in development people still explore how to implement
it in the right way.

 - Johannes

Am Montag, den 12.02.2018, 01:26 -0800 schrieb Ching-Tsun Chou:
> OK, I think I'm getting it now.  But I'm still wondering: if Prop is
> not proof-irrelevant, would the situation change?  For example, what
> if the proof object remembers so much information as to make (Exists
> x, p x) the same as {x // p x}?  I suppose that is not the design
> choice Lean made.  But I wonder what are the pros and cons of each
> design choice.
>
> Thanks!
> - Ching Tsun
>
>
> On Mon, Feb 12, 2018 at 1:01 AM, Mario Carneiro <di.gama@gmail.com>
> wrote:
> > Remember the definition of choice:
> > axiom choice {α : Sort u} : nonempty α → α
> > Given that "finite s" is literally just "nonempty (fintype s)",
> > choice is exactly the function needed to produce "fintype s" from
> > this evidence. It is not computable because in the VM, evidence of
> > "finite s" is essentially just "unit.star", there is no information
> > in it; and to produce a list from this is quite impossible.
> > Mario
> >
> > On Mon, Feb 12, 2018 at 3:44 AM, Ching-Tsun Chou <chingtsun.chou@gm
> > ail.com> wrote:
> > > "finite" is defined as:
> > >
> > > def finite (s : set α) : Prop := nonempty (fintype s)

> > >
> > > TPIL says nonempty is equivalent to exists:
> > >
> > > https://leanprover.github.io/theorem_proving_in_lean/axioms_and_c
> > > omputation.html#choice
> > > example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
> > > iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)
> > > So, via the definition of fintype, s being finite gives a witness
> > > finset which enumerates the elements of s.  The detour through
> > > classical.choice seems unnecessary.
> > >
> > > Thanks!
> > > - Ching Tsun
> > >
> > >
> > >
> > > On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou
> > > @gmail.com> wrote:
> > > > But isn't the base logic of Lean a constructive logic, where
> > > > "exists" means having a witness?  I also observe that
> > > > mathlib/data/finset is developed without "noncomputable".  Why
> > > > is "finite" substantially different from "finset"?
> > > >
> > > > Thanks!
> > > > - Ching Tsun
> > > >
> > > >
> > > > On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail
> > > > .com> wrote:
> > > > > Hi Ching-Tsun,
> > > > >
> > > > > On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.c
> > > > > hou@gmail.com> wrote:
> > > > > > In mathlib/data/set/finite, finite.to_finset is defined to
> > > > > > be noncomputable.  Why?  Intuitively, the use of strong
> > > > > > classical logic does not seem necessary here.  A set is
> > > > > > finite iff there is a finite list with no duplicate
> > > > > > elements enumerating the set's elements.  So it would seem
> > > > > > that the same list can be used to make the finset that the
> > > > > > finite set is converted to.
> > > > > >
> > > > >
> > > > > A set is finite if THERE EXISTS a list (slash finset)
> > > > > enumerating its elements. That doesn't mean that you have a
> > > > > function that produces such a list or finset; since "finite
> > > > > s" is a Prop, it contains no data and no enumerating
> > > > > function. To go from this to an actual list enumerating the
> > > > > set requires choice. (It is true that the finset so defined
> > > > > is unique, so that this is in fact a definite description,

> > > > > but Lean makes no distinction - they are both noncomputable
> > > > > operations.)
> > > > >
> > > > > > I am also wondering how much of mathlib/data/finsupp, which
> > > > > > is currently declared to be a noncomputable theory,
> > > > > > actually needs to be noncomputable.  For example,
> > > > > > finsupp.support is noncomputable only because
> > > > > > finite.to_finset is.  For another example, I'd imagine that
> > > > > > "noncomputable" can be removed from finsupp.single by
> > > > > > assuming the domain type has decidable equality.  No?
> > > > > >
> > > > >
> > > > > Some of this can be made computable, like finsupp.single as
> > > > > you've observed, but finsupp.support can't be for the reasons
> > > > > mentioned above. For computational purposes, finsupp is not a
> > > > > good fit anyway, since it deals with functions where
> > > > > association lists or hashmaps would be more efficient.
> > > > >
> > > > > Mario
>
> --


--


--


---

**Ching-Tsun Chou** <chingtsun.chou@gmail.com> Mon, Feb 12, 2018 at 6:24 PM
To: Rob Lewis <rob.y.lewis@gmail.com>
Cc: Johannes Hölzl <johannes.hoelzl@gmx.de>, Mario Carneiro <di.gama@gmail.com>, lean-user <lean-user@googlegroups.com>

Thanks to all who replied to my questions!  They have been very illuminating.

- Ching Tsun

On Mon, Feb 12, 2018 at 2:16 AM, Rob Lewis <rob.y.lewis@gmail.com> wrote:

As I understand it, impredicativity + excluded middle implies proof irrelevance (https://github.com/FStarLang/FStar/issues/360). Since the logic of Lean should remain consistent with classical axioms, a proof-relevant Prop would need to be predicative, which would make it the same as any other Type universe. Someone who wants to work in such a system can do so in Lean by ignoring Prop entirely (as I believe the Lean 3 HoTT development does).

-Rob

On Mon, Feb 12, 2018 at 10:54 AM, Johannes Hölzl <johannes.hoelzl@gmx.de> wrote:

> `Prop` is the proof irrelevant version, everything else lives in `Type`. Changing this would meant to loose `Prop` but not gain anything. There are multiple reasons why we want to have Prop:
>
> * it *impredicative*, i.e. closed under function space
>   for a : Type u, p : Prop we have a -> p : Prop
>
>   For a : Type u, b : Type v we have a -> b : Type (max u v).
>
> * proof irrelevance is baked into the kernel, the kernel knows:
>     h_1 =def= h_2 (for h_1, h_2 : p : Prop)
>   This is an very important feature for e.g. type classes.
>
> * many things which require truncation are for free just by
>   working in Prop
>
> * it is very natural to extend to classical logic (just add choice or
>   at LEM)
>
> So if we remove Prop, then our logic would be very different and it would be difficult to write automation.
>
> If the proof object is required to 'remember' all this informations, then we it is not anymore proof irrelevant and we loose this property.
>
> The cons of Prop are that you can not work in a setting a la HoTT. But HoTT is very much in development people still explore how to implement it in the right way.
>
>  - Johannes
>
> Am Montag, den 12.02.2018, 01:26 -0800 schrieb Ching-Tsun Chou:
> > OK, I think I'm getting it now.  But I'm still wondering: if Prop is
> > not proof-irrelevant, would the situation change?  For example, what
> > if the proof object remembers so much information as to make (Exists
> > x, p x) the same as {x // p x}?  I suppose that is not the design
> > choice Lean made.  But I wonder what are the pros and cons of each
> > design choice.
> >
> > Thanks!

> - Ching Tsun
>
>
> On Mon, Feb 12, 2018 at 1:01 AM, Mario Carneiro <di.gama@gmail.com>
> wrote:
> > Remember the definition of choice:
> > axiom choice {α : Sort u} : nonempty α → α
> > Given that "finite s" is literally just "nonempty (fintype s)",
> > choice is exactly the function needed to produce "fintype s" from
> > this evidence. It is not computable because in the VM, evidence of
> > "finite s" is essentially just "unit.star", there is no information
> > in it; and to produce a list from this is quite impossible.
> > Mario
> >
> > On Mon, Feb 12, 2018 at 3:44 AM, Ching-Tsun Chou <chingtsun.chou@gm
> > ail.com> wrote:
> > > "finite" is defined as:
> > >
> > > def finite (s : set α) : Prop := nonempty (fintype s)
> > >
> > > TPIL says nonempty is equivalent to exists:
> > >
> > > https://leanprover.github.io/theorem_proving_in_lean/axioms_and_c
> > > omputation.html#choice
> > > example (α : Type u) : nonempty α ↔ ∃ x : α, true :=
> > > iff.intro (λ ⟨a⟩, ⟨a, trivial⟩) (λ ⟨a, h⟩, ⟨a⟩)
> > > So, via the definition of fintype, s being finite gives a witness
> > > finset which enumerates the elements of s.  The detour through
> > > classical.choice seems unnecessary.
> > >
> > > Thanks!
> > > - Ching Tsun
> > >
> > >
> > >
> > > On Sun, Feb 11, 2018 at 11:53 PM, Ching-Tsun Chou <chingtsun.chou
> > > @gmail.com> wrote:
> > > > But isn't the base logic of Lean a constructive logic, where
> > > > "exists" means having a witness?  I also observe that
> > > > mathlib/data/finset is developed without "noncomputable".  Why
> > > > is "finite" substantially different from "finset"?
> > > >
> > > > Thanks!
> > > > - Ching Tsun
> > > >
> > > >
> > > > On Sun, Feb 11, 2018 at 11:37 PM, Mario Carneiro <di.gama@gmail
> > > > .com> wrote:
> > > > > Hi Ching-Tsun,

> > > > >
> > > > > On Mon, Feb 12, 2018 at 2:25 AM, Ching-Tsun Chou <chingtsun.c
> > > > > hou@gmail.com> wrote:
> > > > > > In mathlib/data/set/finite, finite.to_finset is defined to
> > > > > > be noncomputable.  Why?  Intuitively, the use of strong
> > > > > > classical logic does not seem necessary here.  A set is
> > > > > > finite iff there is a finite list with no duplicate
> > > > > > elements enumerating the set's elements.  So it would seem
> > > > > > that the same list can be used to make the finset that the
> > > > > > finite set is converted to.
> > > > > >
> > > > >
> > > > > A set is finite if THERE EXISTS a list (slash finset)
> > > > > enumerating its elements. That doesn't mean that you have a
> > > > > function that produces such a list or finset; since "finite
> > > > > s" is a Prop, it contains no data and no enumerating
> > > > > function. To go from this to an actual list enumerating the
> > > > > set requires choice. (It is true that the finset so defined
> > > > > is unique, so that this is in fact a definite description,
> > > > > but Lean makes no distinction - they are both noncomputable
> > > > > operations.)
> > > > >
> > > > > > I am also wondering how much of mathlib/data/finsupp, which
> > > > > > is currently declared to be a noncomputable theory,
> > > > > > actually needs to be noncomputable.  For example,
> > > > > > finsupp.support is noncomputable only because
> > > > > > finite.to_finset is.  For another example, I'd imagine that
> > > > > > "noncomputable" can be removed from finsupp.single by
> > > > > > assuming the domain type has decidable equality.  No?
> > > > > >
> > > > >
> > > > > Some of this can be made computable, like finsupp.single as
> > > > > you've observed, but finsupp.support can't be for the reasons
> > > > > mentioned above. For computational purposes, finsupp is not a
> > > > > good fit anyway, since it deals with functions where
> > > > > association lists or hashmaps would be more efficient.
> > > > >
> > > > > Mario
>
> --


--