How to Do Maths without Dependent Types

Anthony Bordg
jww Larry Paulson and Wenda Li

University of Cambridge

workshop Schlumberger, IHES, June 15th 2022

Outline

Intro

What can be done when formalising mathematics without dependent types?

I will give you new insights into this question by exploring the capability and possible limitations of the Isabelle/HOL proof assistant. I will explain what we learnt formalising Grothendieck's schemes using only Isabelle's module system called "locales".

I will finish with a question about the expressive power of "simple types + locales" wrt dependent types when it comes to doing "real mathematics" in practice.

Keywords: proof assistants; type theory; formalisation; algebraic geometry; schemes

Outline

The Isabelle framework

Isabelle/Pure is a *logical framework* with various instances including the *Isabelle/HOL* proof assistant based on *higher-order logic* (a classical logic).

Although there are other instances like Isabelle/FOL or Isabelle/ZF (an untyped set theory), I will exclusively focus in this talk on **Isabelle/HOL**.

Isabelle/HOL

Isabelle/HOL is based on a *simple type theory*, **no dependent types**.

A question from a student using Lean: "Does Isabelle/HOL have lists and products?"

Yes, of course!

The type $A \times B$ is not a dependent type. Isabelle doesn't have the *type of lists of length n* like Coq's *Mathematical Components* library for instance, but Isabelle has a type 'a list for the lists of elements of type 'a and a length function which, given a list, outputs its length.

Analysis vs algebra in Isabelle/HOL

Relatively well-developed library for analysis in Isabelle/HOL (integration, ODE *e.g.* Fabian Immler's ODE solver and the Lorenz attractor, some measure theory, a large chunk of complex analysis, differentiable manifolds, ergodic theory and Gromov hyperbolicity by Sébastien Gouëzel . . .).

Apostol's textbook *Introduction to Analytic Number Theory* formalised mainly by Manuel Eberl. Current work to formalise Apostol's graduate textbook *Modular Functions and Dirichlet Series in Number Theory* (jww Eberl, Li and Paulson).

Algebra? Not so well-developed, some material on groups, rings, modules, fields, vector spaces, multivariate polynomials. Kevin Buzzard: "Where is the algebraic number theory? Where are schemes?"

Problems with algebra in Isabelle/HOL

Several libraries for algebra in Isabelle/HOL. A bit confusing!

Records are used as a way to extend conveniently algebraic structures, but there are quirks. In the *HOL-Algebra* library **abelian groups use a ring record.** The unused fields of the record are left undefined.
McEnroe: "You cannot be serious!" Very confusing!

Isabelle's records only support *single inheritance*, *i.e.* a record declaration can be extended in only one way and this apparently led to this quirk; combining the ring structure from its additive group and multiplicative monoid substructures is not possible with records.

Outline

Problems with algebra in Isabelle/HOL (continued)

Is algebra in simple type theory doomed? Not yet!

Use *locales*, Isabelle's module system, to capture modern algebra, *i.e.* "abstract set + some structure + some axioms".

Pioneering work by Elsa Gunter for formalising algebra in simple type theory.
Isabelle's locales developed by Florian Kammüller, Makarius Wenzel and Larry Paulson, then later by Clemens Ballarin.

State-of-the-art locales dating back from 2008.

Algebra reborn in Isabelle/HOL?

In an experiment, Ballarin reformalised the algebraic hierarchy

$$\text{monoids} \rightarrow \text{groups} \rightarrow \text{rings},$$

as well as quotients, using locales (*cf. Exploring the Structure of an Algebra Text with Locales*, JAR, 2020).

Ballarin doesn't use records anymore and fixed the quirk in abelian groups previously mentioned.

```
locale abelian_group = group + commutative_monoid G "(·)" 1

locale ring = additive: abelian_group R "(+)" 0 + multiplicative: monoid R "(·)" 1
  for R and addition (infixl "+" 65) and multiplication (infixl "·" 70) and zero ("0")
      and unit ("1") +
  assumes distributive: "⟦ a ∈ R; b ∈ R; c ∈ R ⟧ ⟹ a · (b + c) = a · b + a · c"
    "⟦ a ∈ R; b ∈ R; c ∈ R ⟧ ⟹ (b + c) · a = b · a + c · a"
```

Locales vs type classes in Isabelle/HOL

Isabelle has a second, older, hierarchy to capture "abstract set + some structure + some axioms": *type classes*, a Haskell-like type system with ordered type classes due to Makarius Wenzel.

In Isabelle libraries, both locales and type classes coexist. A bit confusing!

One can even combine them, a good point to reuse code, but possibly confusing!

Locales vs type classes in Isabelle/HOL (continued)

Three limitations of type classes:

- a type class can't contain more than one type variable
- one can't pass explicit parameters to type classes called within a new type class declaration, in particular one can't declare more than one instance of a given type class simultaneously
- plus usually type classes have no carrier sets as parts of their data, but use the whole universe of the type variable involved.

What you need to know in a nutshell:

- The limitations outlined above rule out the possibility of formalising advanced algebra with type classes.
- locales are more flexible than type classes and overcome the above limitations.

Locales vs dependent types

Still, Isabelle has no dependent types and **"algebraic geometry has dependent types everywhere"**.

Can Isabelle do sheaves? "A sheaf is a dependent type" , "one would be forever battling the fact that you are dealing with rings but you can't access the API for rings directly because the ring is a value of a dependent function and any workaround would basically leave you screwed in practice."

Is "simple type theory is too simple" a good mantra?

Are the Isabelle folks screwed and Isabelle doomed?

Outline

Grothendieck's schemes with locales

Using locales, with Larry Paulson and Wenda Li, we defined sheaves of rings, locally ringed spaces, affine schemes and schemes in Isabelle following Hartshorne's textbook.

See our article *Simple Type Theory is not too Simple: Grothendieck's Schemes Without Dependent Types*, published in Experimental Mathematics (2022, open access), for all the details of the formalisation.

I will show you a few snippets of code with their textbook counterparts, just for comparison.

Let's start directly with schemes.

Grothendieck's schemes with locales (continued)

Definition (scheme)

A scheme is a locally ringed space $(X, \mathscr{O}_X)$ in which every point has an open neighborhood U such that the topological space U, together with the sheaf $\mathscr{O}_X|_U$, is an affine scheme.

```
locale scheme = locally_ringed_space for univ +
  assumes are_affine_schemes: "⋀x. x ∈ S ⟹ (∃U. x∈U ∧ is_open U ∧
(∃R add mult zero one f φf. R ⊆ univ ∧ comm_ring R add mult zero one ∧
affine_scheme R add mult zero one U
(ind_topology.ind_is_open S is_open U) (ind_sheaf.ind_sheaf ℱ U)
(ind_sheaf.ind_ring_morphisms ϱ U) b (ind_sheaf.ind_add_str add_str U)
(ind_sheaf.ind_mult_str mult_str U) (ind_sheaf.ind_zero_str zero_str U)
(ind_sheaf.ind_one_str one_str U) f φf))"
```

Grothendieck's schemes with locales (continued)

With locales formalising the definition of schemes is relatively straightforward (no dead ends met).

There are tree type variables in the locale for schemes (hence, schemes can't be defined using only type classes).

Note that locales have implicit parameters inherited, along with their notations, from the previous locale declarations (here locally_ringed_space).
One also inherits the definitions and theorems from these previous locales.
If one wants to change the notations for these implicit parameters, then one has to give all the parameters explicitly, for instance as follows.

Grothendieck's schemes with locales (continued)

```
locale scheme =
locally_ringed_space X is_open 𝒪ₓ ϱ b add_str mult_str zero_str one_str
for X is_open 𝒪ₓ ϱ b add_str mult_str zero_str one_str univ +
  assumes are_affine_schemes: "⋀x. x ∈ X ⟹ (∃U. x∈U ∧ is_open U ∧
(∃R add mult zero one f φf. R ⊆ univ ∧ comm_ring R add mult zero one ∧
affine_scheme R add mult zero one U
(ind_topology.ind_is_open X is_open U) (ind_sheaf.ind_sheaf 𝒪ₓ U)
(ind_sheaf.ind_ring_morphisms ϱ U) b (ind_sheaf.ind_add_str add_str U)
(ind_sheaf.ind_mult_str mult_str U) (ind_sheaf.ind_zero_str zero_str U)
(ind_sheaf.ind_one_str one_str U) f φf))"
```

It becomes more verbose.
No implicit arguments when passing a list of arguments to a locale nor a way to bundle arguments to avoid long lists. These are engineering issues which could be tackled.

At least, when interpreting a locale a name can be attached to this interpretation and then used as an abbreviation.

Presheaves of rings with locales

Let's backpedal and discuss presheaves of rings on topological spaces. The textbook definition is as follows.

Definition (presheaf of rings)

Let X be a topological space. A presheaf $\mathscr{F}$ of rings on X consists of the following data:

- for every open set U, a ring $\mathscr{F}(U)$
- for every inclusion $V \subseteq U$ of open subsets, a morphism of rings $\rho_{UV} : \mathscr{F}(U) \to \mathscr{F}(V)$

satisfying

1. $\mathscr{F}(\emptyset) = \{0\}$
2. ρ_{UU} is the identity map for every open subset U
3. If $W \subseteq V \subseteq U$ are three open subsets, then $\rho_{UW} = \rho_{VW} \circ \rho_{UV}$.

Problems and solution

Problems:

- Without type classes, no type Rings where $\mathscr{F}$ could take its values.
- No dependent type of the open subsets of a given topological space for the domain of $\mathscr{F}$.

Solution: **use "big" types to declare the structure explicitly and relativise the axioms using predicates.**

Every locale gives you a predicate for free

Indeed, we have predicates.

Every locale gives you a predicate for free outside its scope.

Hence, we have a predicate `is_open` for relativising our axioms to the open subsets of the underlying topological space.

We have also predicates like `ring` and `ring_homomorphism`.

Presheaves of rings with locales

```
locale presheaf_of_rings = topological_space
  + fixes 𝔉:: "'a set ⇒ 'b set"
  and ϱ:: "'a set ⇒ 'a set ⇒ ('b ⇒ 'b)" and b:: "'b"
  and add_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)" ("+↘_↙")
  and mult_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)" ("·↘_↙")
  and zero_str:: "'a set ⇒ 'b" ("0↘_↙") and one_str:: "'a set ⇒ 'b" ("1↘_↙")
assumes is_ring_morphism:
  "⋀U V. ⟦is_open U; is_open V; V ⊆ U⟧ ⟹ ring_homomorphism (ϱ U V)
                                                (𝔉 U) (+↘U↙) (·↘U↙) 0↘U↙ 1↘U↙
                                                (𝔉 V) (+↘V↙) (·↘V↙) 0↘V↙ 1↘V↙"
  and ring_of_empty: "𝔉 {} = {b}"
  and identity_map [simp]: "⋀U. is_open U ⟹ (⋀x. x ∈ 𝔉 U ⟹ ϱ U U x = x)"
  and assoc_comp: "⋀U V W. ⟦is_open U; is_open V; is_open W; V ⊆ U; W ⊆ V⟧ ⟹
                              (⋀x. x ∈ (𝔉 U) ⟹ ϱ U W x = (ϱ V W ∘ ϱ U V) x)"
```

Within the context of the locale presheaf_of_rings, we can immediately prove the following.

```
lemma is_ring_from_is_homomorphism:
  shows "⋀U. is_open U ⟹ ring (𝔉 U) (+↘U↙) (·↘U↙) 0↘U↙ 1↘U↙"
  using is_ring_morphism ring_homomorphism.axioms(2) by fastforce
```

Presheaves of rings with locales (continued)

A more technical remark (only one slide).

We have a type variable `'a` for the carrier of the topological space which is typed `'a set` and a second type variable `'b` for the carriers of the rings which are all typed `'b set`.
We can't have a family of type variables indexed by the open subsets.

Could this be a problem?

No problem with the structure sheaf $\mathscr{O}$ on Spec R. The carrier of Spec R has type `'a set set`, where `'a set` is the type of the carrier of R, and all the carriers of the rings $\mathscr{O}(U)$ have the same type `('a set ⇒ ('a × 'a) set) set`.

Sanity checks for the definition of schemes in Isabelle/HOL

Schemes have also been formalised in Lean by Kevin Buzzard, Chris Hughes, Kenny Lau, Amelia Livingston, Ramon Fernández and Scott Morrison, with additional insights from Reid Barton, Mario Carneiro and Neil Strickland. See their article *Schemes in Lean* in Experimental Mathematics (Nov. 2021, open access).

As a sanity check we proved that the sprectrum of a commutative ring is a locally ringed space as well as the benchmark set by Kevin Buzzard *et al.*: an affine scheme is a scheme.

$\sim$ 7000 lines of code including topological spaces and material on commutative algebra (prime and maximal ideals, localisation).

No need to be a type whizz to do maths in Isabelle

In Isabelle none of the problems mentioned by Buzzard *et al.* with identity types in Lean, *e.g.*. $R\,[1/fg] \cong R\,[1/f]\,[1/g]$.

Actually, there are no identity types in Isabelle, since identity types are dependent types.

In Lean, to avoid a tedious rewrite along the above isomorphism followed by a tedious diagram chase, they **had to** introduce a **predicate** for the localisation of R at S in a second iteration of their formalisation.

In Isabelle, this is the **natural** way to proceed. Formalising localisations of rings with a locale gives you the corresponding predicate `localisation R S + × 0 1`. In Isabelle, there is nothing like a (dependent) type of localisations of R at S.

Outline

Questions we were asked, sheaves of modules

"Do you think you could do sheaves of modules in Isabelle?"

Answer: Let's do it!

```
locale module = ring + abelian_group M add z
  for M add z +
  fixes scalar_mult:: "'a ⇒ 'b ⇒ 'b" (infixl ‹•› 60)
  assumes scalar_mult_closed: "⟦r ∈ R; x ∈ M⟧ ⟹ r • x ∈ M"
    and scalar_mult_unit: "⋀x. x ∈ M ⟹ 1 • x = x"
    and scalar_mult_l_distr:
      "⟦r ∈ R; s ∈ R; x ∈ M⟧ ⟹ (r + s) • x = add (r • x) (s • x)"
    and scalar_mult_r_distr:
      "⟦r ∈ R; x ∈ M; y ∈ M⟧ ⟹ r • (add x y) = add (r • x) (r • y)"
    and scalar_mult_assoc:
      "⟦r ∈ R; s ∈ R; x ∈ M⟧ ⟹ (r · s) • x = r • (s • x)"
```

Questions we were asked, sheaves of modules (continued)

```
(* Should be read as a sheaf 𝔉' of 𝔉-modules *)
locale sheaf_of_module =
    ringed_space  +
    sheaf_of_ab_groups S is_open 𝔉' ϱ' c add_str' zero_str'
  for 𝔉' ϱ' c add_str' zero_str' +
  fixes scalar_mult_str:: "'a set ⇒ ('b ⇒ 'c ⇒ 'c)"
  assumes is_module: "⋀U. is_open U
    ⟹ module (𝔉 U) (add_str U) (mult_str U) (zero_str U) (one_str U)
                (𝔉' U) (add_str' U) (zero_str' U) (scalar_mult_str U)"
  and restriction_maps_are_compatible:
    "⋀U V f s. ⟦is_open U; is_open V; V ⊆ U; f ∈ 𝔉 U; s ∈ 𝔉' U⟧ ⟹
          (ϱ' U V ) (scalar_mult_str U f s)
        = scalar_mult_str V (ϱ U V f) (ϱ' U V s)"
```

Questions we were asked, families of schemes

"What about an indexed family of schemes?"

One can certainly represent this kind of objects. Below is a mock locale to declare and work with an infinite family of schemes.

```
locale inf_fam_of_schemes =
  fixes T:: "nat ⇒ 'a set" and t:: "nat ⇒ ('a set ⇒ bool)" and
    𝒪:: "nat ⇒ ('a set ⇒ 'b set)" and ϱ:: "nat ⇒ ('a set ⇒ 'a set ⇒ 'b ⇒ 'b)" and
    b:: "'b" and add_str:: "nat ⇒ 'a set ⇒ 'b ⇒ 'b ⇒ 'b" and
    mult_str:: "nat ⇒ 'a set ⇒ 'b ⇒ 'b ⇒ 'b" and zero_str:: "nat ⇒ 'a set ⇒ 'b" and
    one_str:: "nat ⇒ 'a set ⇒ 'b" and univ::"'c set"
  assumes are_schemes:
    "⋀n. scheme (T n) (t n) (𝒪 n) (ϱ n) b (add_str n) (mult_str n) (zero_str n) (one_str n)
      (UNIV::'c set)"
begin
(* Here you can start working with this infinite family of schemes and maybe later
instantiate it with your favorite example. *)
end
```

Questions we were asked, categories

"Can we define categories in Isabelle and state that sheaves on a topological space form a category?"

Yes. There are currently several formalisations of categories, functors and natural transformations in Isabelle (and many other things formalised by Eugene W. Stark: Yoneda's lemma; limits; adjunction; bicategories; monoidal categories . . .).

Questions we were asked, proof automation

"Very efficient proof automation is a strong point of Isabelle/HOL, does it work well even in such a complicated context?"

In the context of locales and the scheme formalisation, we benefited from the automation in Isabelle/HOL.

However, given the nested algebraic structures involved, it's not always great.
Given a prime ideal $\mathfrak{p}$, proving that the stalk $\mathscr{O}_{\mathsf{Spec}R,\mathfrak{p}}$ is isomorphic to the local ring $R_{\mathfrak{p}}$ (whatever that means!) is 14-line long in Hartshorne's textbook, while it's 200-line of code in Isabelle (a factor of 14).

Questions we were asked, analysis library refactor

"Most of Isabelle/HOL analysis library is written using typeclasses. In advanced mathematics, algebra and analysis interact all the time: should all analysis be redone using locales instead of typeclasses? Does this scheme formalization warrant a big library refactor?"

Locales capture the kind of flexibility that one needs for formalising more advanced mathematics in the simple type theory of Isabelle.

The HOL-Algebra library definitely needs an overhaul.

For the analysis library it's less clear; type classes are special cases of locales and new locale declarations can call, not only previous locales, but also type classes.

Outline

Take-Home Message(s)

- No need to be a type whizz to do maths in Isabelle, defining schemes in Isabelle/HOL was straightforward using locales.
- Isabelle's locale mechanism could probably be further improved to better express mathematical structures.
- One can replace dependent types with locales/predicates to express nested structures.
- **dependent types** $\overset{?}{\simeq}$ **simple types** $+$ **locales/predicates**

Thank You