Cubical Methods in Homotopy Type Theory and Univalent Foundations

Anders Mörtberg

Topos Institute Colloquium, October 7, 2021

What is type theory?

Type theory considers a class of formal systems which forms an alternative to set theory for the foundations of mathematics

The type theories which we will discuss today are *dependent* type theories, often referred to as "Martin-Löf type theories" (MLTT)

MLTT uses the syntax of λ-calculus, just like functional programming languages (Lisp, Scheme, Haskell, OCaml...), hence it can be used as a functional programming language with a very powerful type system

Because of this it is well suited as a system for computer implementation as demonstrated by systems like Agda, Coq, Lean...

What is equality?

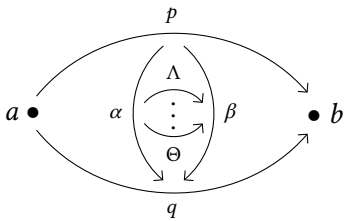
Equality plays a central role in type theory: there is both a builtin notion of strict/judgmental/definitional equality and a definable notion of weak/propositional/typal equality

Questions about equality puzzled type theorists for many years...

- What is the meaning of $f = g$ for functions $f\ g : A \to B$?
- What is the meaning of $P = Q$ of propositions P and Q?
- What is the meaning of $A = B$ for arbitrary types A and B?
- Is the type $x = y$ for all x and y always a proposition?

In 1998 Hofmann-Streicher proved that it is consistent to assume that $x = y$ is **not** a proposition using a model of type theory in groupoids!

Groupoidal structure of type theoretic equality

Type theory	Groupoids
$a, b : A$	
$p, q : a = b$	
$\alpha, \beta : p = q$	
$\Lambda, \Theta : \alpha = \beta$	
$\vdots$	

Crucial idea: proof relevant equality can have topological meaning! (Awodey, Garner, Lumsdaine, Streicher, van den Berg, Warren, ...)

Outline

Outline

Homotopy Type Theory and Univalent Foundations

Around the time Vladimir Voevodsky received the 2002 Field's medal for his work in algebraic geometry he became interested in formalization of mathematics:

> *I think it was at this moment that I largely stopped doing what is called "curiosity-driven research" and started to think seriously about the future. I didn't have the tools to explore the areas where curiosity was leading me and the areas that I considered to be of value and of interest and of beauty. So I started to look into what I could do to create such tools. And it soon became clear that the only long-term solution was somehow to make it possible for me to use computers to verify my abstract, logical, and mathematical constructions.*

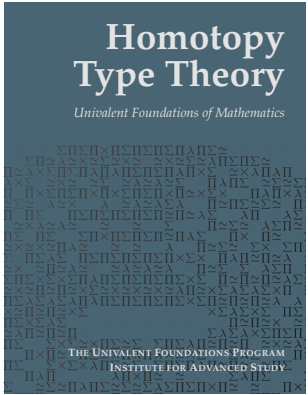
Homotopy Type Theory and Univalent Foundations

This eventually led to *Homotopy Type Theory and Univalent Foundations*

Aims to provide a *practical* foundations for computer formalization of mathematics

Builds on deep connections between type theory, homotopy theory and (higher) category theory

In 2013–2014 the Institute For Advanced Study in Princeton arranged a special year devoted to this topic



The homotopical interpretation of type theory

Type theory	Homotopy theory
A type	A space
$a : A$	a point in A
$x : A \vdash B(x)$	Fibration
$(x : A) \to B(x)$	Space of sections
$(x : A) \times B(x)$	Total space
$x = y$	Path space $A^{\mathbb{I}}$

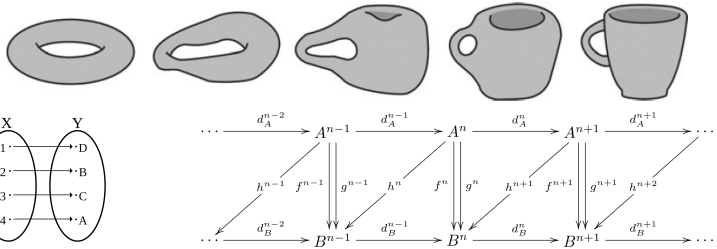
This informal correspondence forms the basis of HoTT/UF. Made formal by model Kan simplicial sets (classically), later also by models in cubical sets (constructively)

Homotopy Type Theory and Univalent Foundations

From a logical perspective HoTT/UF is about equality/identifications:

"When are two mathematical objects the same?"

Mathematics is full of different notions of *"sameness"*



These notions are elegantly captured by *equivalences* of types in HoTT/UF

Homotopy Type Theory and Univalent Foundations

Inspired by the simplicial set model Voevodsky realized that one can extend MLTT with a new axiom:

Univalence: equality of types is equivalent to equivalences of types

One consequence is that it is possible to **transport** structures along equivalences (cf. Bourbaki: Theory of sets, 1968)

This has consequences for data abstraction and representation independence when considering HoTT/UF as a functional programming language

Homotopy levels of types

Voevodsky also realized that it is very useful to stratify types by the complexity of their equality

He called the levels of this stratification “homotopy-levels”, or “h-levels”, as they correspond to objects truncated at a fixed level

The lowest level are the contractible types:

Definition (Contractible types)

We say that a type A is *contractible* is the following type is inhabited:

$$\text{isContr } A := (a : A) \times ((b : A) \to a = b)$$

Voevodsky's internal hierarchy of homotopy levels

Definition (H-levels)

Given n : nat and a type A, the h-levels are defined as:

$$\text{isOfHLevel } 0\ A := \text{isContr } A$$
$$\text{isOfHLevel } (\text{suc } n)\ A := (a\ b : A) \to \text{isOfHLevel } n\ (a = b)$$

Warning: the HoTT book talks about "n-types" instead and the hierarchy is shifted down by 2, so it starts from $n = -2$ instead of $n = 0$.

Voevodsky's hierarchy of homotopy levels

H-level	n-type	Name	Examples
0	-2	contractible	unit
1	-1	proposition	empty, unit
2	0	set	empty, unit, bool, nat
3	1	groupoid	$\mathbb{S}^1$, Set
4	2	2-groupoid	Gpd
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Voevodsky developed a `Coq` library called `Foundations` (now `UniMath`) centered around these notions and proved that this gives a very elegant way of organizing mathematics

Example 1: unique existence as contractibility

Given $x : A \vdash B$ we can define

$$\exists!(x : A)\ B(x) := \text{isContr}\ ((x : A) \times B(x))$$

This says that there exists a center of contraction $(a, b) : (x : A) \times B(x)$ with $a : A$ and $b : B(a)$, such that all other (a', b') are equal to it, i.e. that a is the unique element satisfying B

This is very useful when formalizing category theory in HoTT/UF as we can express universal properties using it

This also generalizes to higher categories (see Riehl's Topos Institute talk: *Contractibility as uniqueness*)

Example 2: surjectivity

We have just seen how to express unique existence using isContr, but how can we express for example that $f : A \to B$ is surjective?

First attempt:

$$\text{isSurjective}\ f := (y : B) \to (x : A) \times (f\ x = y)$$

Too strong! (this is the space of sections)

The solution in HoTT/UF is to "squash" $(x : A) \times B(x)$ into a proposition, written $\|(x : A) \times B(x)\|$

This squashing operation forces all elements to be typally equal

Propositional truncation

This allows us to define a weaker existential quantifier than the one given by Σ-types:

$$\exists(x : A)\ B(x) := \| \ (x : A) \times B(x) \ \|$$

This is a proposition by definition and we can define surjectivity as:

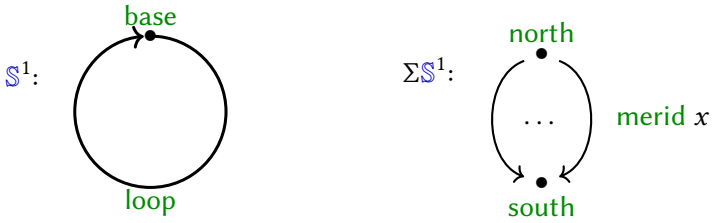
$$\text{isSurjective}\ f := (y : B) \to \exists(x : A)\ (f\ x = y)$$

The truncation restricts how we can eliminate out of the type, making this behave the way we expect

There are also corresponding higher truncation operations $\|A\|_n$ which forces A to become an n-type; these are all examples of *higher inductive types*

Higher inductive types

Datatypes generated by regular "point" constructors and (higher) path constructors:



These are usually added axiomatically and justified semantically in "sufficiently nice" model categories (Lumsdaine-Shulman 2017)

Allows homotopy theory to be developed *synthetically* in HoTT/UF

Computing with HoTT/UF

Problem: by adding axioms like univalence and existence of various HITs we break the good computational properties of type theory

This effectively turns it into a programming language where we cannot run all programs...

This has been a major source of open problems in HoTT/UF and a lot of progress has happened in recent years using *cubical* methods

Theorem (Cohen, Coquand, Huber, M. 2015 (CCHM))

HoTT/UF has a constructive model in structural cubical sets with connections and reversals

This built on a previous model of Bezem-Coquand-Huber (2014), based on *substructural* cubes. Having a model based on structural cubes let us formulate a *cubical* type theory, which provides a computational formulation of HoTT/UF

Outline

Cubical sets

Cubical sets are presheaf categories:

$$\widehat{\square} = [\square^{\mathrm{op}}, \mathrm{Set}]$$

Here $\square$ is a *cube category*—there are many possible choices (BCH, cartesian, Dedekind, De Morgan, Boolean...)

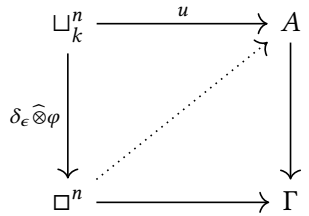
In this talk: $\square$ has a bi-pointed interval object $\mathbb{I}$ and closed under products

Hence similar to simplex category Δ, but Δ is *not* closed under products!

As this is a presheaf category it forms a constructive model of type theory (in the form of Categories with Families/Attributes), but in order to model all of HoTT/UF we need more structure

Kan structure

Just like in Kan simplicial sets we need to be able to fill open boxes (cf. horns in $\widehat{\Delta}$):



The dashed line gives an element $\texttt{fill}_\epsilon\ \varphi\ u$ defined on all of $\square^n$ from a partial element $u : A$ defined on $\sqcup^n_k$

(Remark: at this point we haven't assumed that we have a model structure)

Kan structure

The Kan operations are *structures* on types and not properties

The main difficulty in the construction of a cubical model for HoTT/UF is in proving (constructively) that all type formers preserve this structure: if A has Kan filling and B is a family of types dependent on A which all have Kan filling, then $\Pi\ A\ B$ has Kan filling

Furthermore, this structure has to respect substitutions $\sigma : \Delta \to \Gamma$ (called *uniformity* in BCH)

All of this can be elegantly expressed and formalized axiomatically using the internal language of the presheaf topos $\widehat{\square}$ (Orton-Pitts 2017)

Univalent universes

In order to prove that cubical set models support univalent universes we introduced "Glue types" in CCHM. These lets us define:

$$\mathsf{ua} : (A\ B : \mathcal{U}) \to A \simeq B \to \mathsf{Path}_{\mathcal{U}}\ A\ B$$

Universe construction can be elegantly presented using *tinyness* of the interval (Licata-Orton-Pitts-Spitters 2018)

The interval $\mathbb{I}$ is *tiny* if exponentiating by it has a right adjoint:

$$\mathbb{I} \times \_ \dashv \_^{\mathbb{I}} \dashv \sqrt{\_}$$

If the base category is closed under products then the interval in the presheaf category is tiny

Variations cubical set models

There are many variations on cubical set models (different cube categories, different fibrations, generating trivial cofibrations, etc.)

Many of the structural variations are special cases of a general version of the cartesian model (Cavallo-M.-Swan 2020)

Some of them also give rise to Quillen model structures (Sattler 2017)

There is a recent equivariant variation of the cartesian model for which the induced model structure is Quillen equivalent to spaces (Awodey-Cavallo-Coquand-Riehl-Sattler 2020)

Why cubes?

It is curious that we can get a constructive model out of cubes, but it is still open for simplices (although progress has been made in recent years)

Some reasons why cubes are nice:

- Close correspondence between morphisms in cube category and structure of cubical contexts
- Tiny interval very useful to construct univalent universes

These properties make cubical set models useful as blueprints for new *cubical* type theories and proof assistants based on them

Outline

Cubical proof assistants

The cubical set models can be made syntactic and turned into cubical type theories

`Cubical Agda` was implemented by Andrea Vezzosi, building on a series of experimental typecheckers developed at Chalmers: `cubical`, `cubicaltt`...

There are also many other cubical and cubically-inspired systems: `Arend`, **RedPRL**, **redtt**, **cooltt**, `yacctt`, `mlang`...

These are all implementations of (HoTT/UF) that provide extensionality principles like univalence and function extensionality to dependent type theory, without sacrificing computation

Making Agda cubical

New features:

- Interval (pre-)type I with endpoints i0 : I and i1 : I
- Kan structure (via transp and hcomp)
- Computational univalence (via Glue types)
- General schema for higher inductive types

Builds on an already existing proof assistant, so it differs from all the other cubical systems that are implemented from scratch

Pros: get all of the Agda infrastructure and code base for free (dependent pattern-matching, Agda emacs interaction mode, lots of users...)

Cubical type theory

Key idea: replace Agda's inductive _≡_ with paths

A path $p : x \equiv y$ is a function $p : \mathbb{I} \to A$ with endpoints x and y:

$$p\ \mathsf{i0} = x \qquad\qquad p\ \mathsf{i1} = y$$

Get cubes by iteration: $p : \mathbb{I} \to \mathbb{I} \to A$ is a square, $q : \mathbb{I} \to \mathbb{I} \to \mathbb{I} \to A$ is a cube, etc...

All of the cubical systems mentioned earlier builds on this idea, but some variations in how things are set up (variations in structure on $\mathbb{I}$, Kan operations, formulation of Glue types, etc.)—reflecting the cubical set model on which they are based

Cubical Agda

The cubical mode has been part of Agda since version 2.6.0 (April 2019)

To activate it just open an `.agda` file and add

```
{-# OPTIONS --cubical #-}
```

Since October 2018 Andrea Vezzosi and I have been maintaining the `agda/cubical` library:

`https://github.com/agda/cubical/`

By now 57 contributors, 61k LOC, 530 files

Cubical Agda: examples

The library contains many interesting things:

- Computer science: datastructures, representation independence results
- Algebra: algebraic structures (SIP), ring localization, matrices, polynomials (soon), towards affine schemes (constructively)
- Synthetic homotopy theory: some homotopy groups of spheres, Freudenthal suspension theorem, Hopf fibration, Eckmann-Hilton (with syllepsis), cohomology groups and ring...
- ...

General observation: many notions can be defined constructively without relying on decidable equality

Elementary example: polynomials

In my intro to Python programming class I define elements of the polynomial ring $R[x]$ using lists:

$$[1, -2, 4] = 1 - 2x + 4x^2$$

However, not well-suited for formalization as many lists represent same polynomial as we allow trailing zeroes:

$$[1, -2, 4, 0, 0] = 1 - 2x + 4x^2 + 0x^3 + 0x^4 = 1 - 2x + 4x^2 = [1, -2, 4]$$

Pre-HoTT solution: assume that R has decidable equality:

$$\texttt{Poly}\ R := (p : \texttt{List}\ R) \times (\texttt{last}\ p \neq 0)$$

Polynomials in Cubical Agda

In `Cubical Agda` we instead use a HIT:

```
data Poly : Type ℓ where
  []   : Poly
  _::_ : R → Poly → Poly
  drop0 : 0r :: [] ≡ []
```

Here drop0 is a *path* constructor which ensures that trailing zeroes can be ignored

This is equivalent to $(\texttt{List}\ R)/\sim$ where $\sim$ equates lists after removing trailing zeroes

Polynomials in Cubical Agda

We can define operations on Poly using pattern-matching equations:

```
negPoly : Poly → Poly
negPoly [] = []
negPoly (a :: p) = (- a) :: negPoly p
negPoly (drop0 i) = (cong (_:: []) 0Selfinverse ∙ drop0) i
```

In the last case we convince `Cubical Agda` that negPoly is well-defined by proving that
`- 0r :: [] ≡ []`

We can now prove a lot of properties of Poly by pattern-matching, without assuming that R has decidable equality. Many other examples of this in the library (e.g. finite multisets)

Synthetic homotopy theory

One of the appealing aspects of HoTT/UF is that we can do synthetic homotopy theory—classic results of homotopy theory can be proved in HoTT/UF and then interpreted into appropriate models

We have implemented many of the results in the HoTT book in `Cubical Agda`—often leading to big simplifications as things compute better (e.g. 3×3 lemma for pushouts in 200LOC compared to 3000LOC in `HoTT-Agda`)

However, homotopy groups are still difficult to compute... So we have recently also added cohomology groups

More complex example: integral cohomology

Intuitively $H^n(A)$ counts the number of n-dimensional holes in A, e.g. $H^1(\mathbb{T}^2) = \mathbb{Z} \times \mathbb{Z}$

Cohomology is often defined algebraically using chain complexes, however this is not invariant up to homotopy equivalence and hence not suited as a synthetic definition

Instead we work with a definition using Eilenberg-MacLane spaces $K(G, n)$—spaces with $\pi_n(K(G, n)) = G$ and all other homotopy groups trivial

For simplicity we will focus on integral cohomology, so $G = \mathbb{Z}$

More complex example: integral cohomology

Definition ($K(\mathbb{Z}, n)$)

The n:th Eilenberg-MacLane space of $\mathbb{Z}$, written K_n, is a pointed type:

$$\mathsf{K}_n = \begin{cases} (\mathbb{Z}, 0) & \text{if } n = 0 \\ (\| \mathbb{S}^n \|_n, | *_{\mathbb{S}^n} |) & \text{if } n \geq 1 \end{cases}$$

Using this we define the n:th integral cohomology group of A as:

$$H^n(A) = \| A \to \mathsf{K}_n \|_0$$

Group operations

We can equip K_n with a group structure. For $n = 1$:

$$|x| +_k |\mathsf{base}| = |x| \qquad -_k |\mathsf{base}| = |\mathsf{base}|$$
$$|\mathsf{base}| +_k |\mathsf{loop}\, j| = |\mathsf{loop}\, j| \qquad -_k |\mathsf{loop}\, i| = |\mathsf{loop}\, (\sim i)|$$
$$|\mathsf{loop}\, i| +_k |\mathsf{loop}\, j| = |\mathsf{Q}\, i\, j|$$

(Where Q fills a suitable square)

For $n \geq 2$ the definition is more complex and uses the Freudenthal equivalence

This then induces a group structure on $H^n(A)$:

$$|f| +_h |g| = |\lambda x \to f\ x +_k g\ x| \qquad -_h |f| = |\lambda x \to -_k f\ x|$$

Characterizations of cohomology groups

We have characterized the cohomology groups of various spaces defined as HITs. For example, the Klein bottle can be defined as:

```
data 𝕂² : Type where
  pt : 𝕂²
  ℓ₁ ℓ₂ : pt ≡ pt
  □ : PathP (λ i → ℓ₂ (~ i) ≡ ℓ₂ i) ℓ₁ ℓ₁
```

And we have proved that $H^1(\mathbb{K}^2) \simeq \mathbb{Z}$ and $H^2(\mathbb{K}^2) \simeq \mathbb{Z}/2\mathbb{Z}$

Many of these proofs are direct by analyzing function spaces, but some require more elaborate techniques. To this end we have formalized the Eilenberg-Steenrod axioms as well as the Mayer-Vietoris sequence

Computations in proofs of cohomology groups

One of the appealing aspects of developing these results in `Cubical Agda` is that we can do some proofs simply by computation

For example, some of the base cases when verifying the group laws involve path algebra in loop spaces over the spheres which can typically be reduced to integer computations

Furthermore, all of the proofs are constructive and have computational content. We can hence compute both with the group structure as well as the equivalences between $H^n(A)$ and some simple concrete type (e.g. $\mathbb{Z}$)

Cup product and cohomology ring

Cohomology allows us to distinguish many spaces, but it is sometimes a bit too coarse

For example, $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ has the same cohomology groups as $\mathbb{T}^2$, but they are not equivalent

To be able to distinguish between spaces like these one equips the cohomology groups also with a graded multiplication operations $\smile : H^n(A) \to H^m(A) \to H^{n+m}(A)$ which turn them into a graded commutative ring $H^*(A)$

A HoTT construction of $H^*(A)$ was described in Guillaume Brunerie's PhD thesis, however the construction is quite complex and some parts are a bit sketchy

Cup product and cohomology ring

With Axel Ljungström and Guillaume Brunerie, we have foundj an alternative definition which is easier to work with formally:

$$
\begin{array}{cc}
n = 1: & n \geq 2: \\
|\,\mathsf{base}\,| \smile_k y = 0_k & |\,\mathsf{north}\,| \smile_k y = 0_k \\
|\,\mathsf{loop}\ i\,| \smile_k y = \sigma_m\ y\ i & |\,\mathsf{south}\,| \smile_k y = 0_k \\
 & |\,\mathsf{merid}\ x\ i\,| \smile_k y = \sigma_{(n-1)+m}\ (|\,x\,| \smile_k y)\ i
\end{array}
$$

Here σ_n is one of the maps in the Freudenthal equivalence defined as:

$$
\begin{aligned}
&\sigma_n : \mathsf{K}_n \to \Omega\,\mathsf{K}_{n+1} \\
&\sigma_n\,|\,x\,| = \mathsf{cong}\,|\_|\,(\mathsf{merid}\ x \cdot (\mathsf{merid}\ *_{\mathbb{S}^n})^{-1})
\end{aligned}
$$

Again $\smile_k$ induces an operation $\smile$ on the cohomology groups

Cup product and cohomology ring

My PhD student Axel Ljungström has formalized that $\smile$ satisfies the graded ring axioms

Graded commutativity is the most difficult and takes $\sim$ 900LOC, however Tim Baumann's HoTT-Agda proof is $\sim$ 5000LOC

As everything is constructive we can use the cohomology ring to distinguish spaces purely by computation

Computing with the cohomology ring

To distinguish $\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1$ and $\mathbb{T}^2$ we define a predicate $P : \mathsf{Type} \to \mathsf{Type}$:

$$P(A) := (x\ y : H^1(A)) \to x \smile y \equiv 0_h$$

In `Cubical Agda`, we have defined isomorphisms:

$$f_1 : H^1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z} \qquad g_1 : H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \mathbb{Z} \times \mathbb{Z}$$
$$f_2 : H^2(\mathbb{T}^2) \cong \mathbb{Z} \qquad g_2 : H^2(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1) \cong \mathbb{Z}$$

We will now disprove $P(\mathbb{T}^2)$ and prove $P(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$, which establishes that they are not equivalent

Computing with the cohomology ring

To disprove $P(\mathbb{T}^2)$ we need $x, y : H^1(\mathbb{T}^2)$ such that $x \smile y \not\equiv 0_h$. Let $x = f_1^{-1}(0, 1)$ and $y = f_1^{-1}(1, 0)$. In `Cubical Agda`, $f_2(x \smile y) \equiv 1$ holds by refl and thus $x \smile y \not\equiv 0_h$.

To prove $P(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$ we let $x, y : H^1(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$. In `Cubical Agda`, we have that $g_2(g_1^{-1}(g_1\ x) \smile g_1^{-1}(g_1\ y)) \equiv 0$, again by refl (modulo truncation elimination). Thus $g_1^{-1}(g_1\ x) \smile g_1^{-1}(g_1\ y) \equiv x \smile y \equiv 0_h$.

So $P(\mathbb{T}^2)$ holds while $P(\mathbb{S}^2 \vee \mathbb{S}^1 \vee \mathbb{S}^1)$ doesn't, so these types are not equivalent

Further computations with cohomology ring

For a more ambitious computation, consider Chapter 6 of Guillaume Brunerie's PhD thesis. This chapter is devoted to proving that the generator $e : H^2(\mathbb{C}P^2)$ when multiplied with itself yields a generator of $H^4(\mathbb{C}P^2)$

Let $g : \mathbb{Z} \to \mathbb{Z}$ be the map given by composing:

$$\mathbb{Z} \xrightarrow{\cong} H^2(\mathbb{C}P^2) \xrightarrow{\lambda\, x \to x \smile x} H^4(\mathbb{C}P^2) \xrightarrow{\cong} \mathbb{Z}$$

The number $g(1)$ should reduce to ± 1 for $e \smile e$ to generate $H^4(\mathbb{C}P^2)$ and by evaluating it in `Cubical Agda` we should be able to reduce the whole chapter to a single computation... However, `Cubical Agda` is currently stuck on computing $g(1)$

Brunerie number

The main result of Brunerie's PhD thesis is a HoTT proof that $\pi_4(\mathbb{S}^3) \simeq \mathbb{Z}/2\mathbb{Z}$. The appendix contains a number $\beta \in \mathbb{Z}$ which should reduce to this 2, however no one has been able to compute it using an implementation of HoTT/UF yet

$g(1)$ is hence another "Brunerie number" which should hopefully be more feasible to compute

As $g(1)$ is much simpler than the original Brunerie number we are optimistic that it should be possible to analyze it and find bottlenecks in the implementation

Brunerie number

An alternative approach to proving these results purely by computation is to formalize them using a mix of proofs and computations, this seems much more feasible

Axel recently formalized the Gysin sequence which allows us to establish that $e \smile e$ generates $H^4(\mathbb{C}P^2)$, so despite not being able to solve the problem purely by computation we can still formalize it!

This essentially constitutes the second part of Brunerie's thesis (which hasn't been formalized before), so we are optimistic that a complete formal proof of $\pi_4(\mathbb{S}^3) \simeq \mathbb{Z}/2\mathbb{Z}$ is now within reach

Outline

Conclusion

`Cubical Agda` gives us many useful things without sacrificing computation: function extensionality, propositional extensionality, univalence, quotients, HITs...

Computations simplify proofs! Even if something is not proved by refl we can often discharge subgoals by computation

Some computations are infeasible... Optimize implementations?

Connect synthetic homotopy theory with computational algebraic topology?

Synthetic homotopy theory has so far mainly been used to verify well-known results, can it be used to verify some recent results in homotopy theory?

Thank you for your attention!

Questions?

https://github.com/agda/cubical/