Computational Difficulties in Cubical Type Theory: a Case Study

VeriDis Seminar

Johann Rosain
j.w.w. Thierry Coquand and Jönas Hœfer

Computer Science Department
ENS Lyon
France

LORIA, Oct. 28, 2024

Will you Take Some Howard with your Curry?

The (very hard) “composition problem”.

- Formula: $(A \to B) \to (B \to C) \to A \to C$.
- Proof: assume (i) that $A \to B$, (ii) that $B \to C$ and (iii) that A. By (i), we have B. By (ii), we have C.
- Program: given $f : A \to B$, $g : B \to C$, $x : A$, return $g(f(x))$.
- Type: $(f : A \to B) \to (g : B \to C) \to (x : A) \to C$.

Will you Take Some Howard with your Curry?

The (very hard) “composition problem”.

- Formula: $(A \to B) \to (B \to C) \to A \to C$.
- Proof: assume (i) that $A \to B$, (ii) that $B \to C$ and (iii) that A. By (i), we have B. By (ii), we have C.
- Program: given $f : A \to B$, $g : B \to C$, $x : A$, return $g(f(x))$.
- Type: $(f : A \to B) \to (g : B \to C) \to (x : A) \to C$.

Fig. 1: Curry’s Realization

(I swear this figure is from his paper)

Coq Rock(q)s

Immediate consequence: not all of computer science is useless*!! ☺

Coq Rock(q)s

Immediate consequence: not all of computer science is useless*!! ☺

What do you mean?

- Proving a formula is building an algorithm.
- This algorithm is typed.
- $\Longrightarrow$ checking a proof = type-checking an algorithm!

*Under the (trivial, of course) condition that type-checking is decidable

Coq Rock(q)s

Immediate consequence: not all of computer science is useless*!! ☺

What do you mean?

- Proving a formula is building an algorithm.
- This algorithm is typed.
- $\Longrightarrow$ checking a proof = type-checking an algorithm!

*Under the (trivial, of course) condition that type-checking is decidable

And what about the rooster?

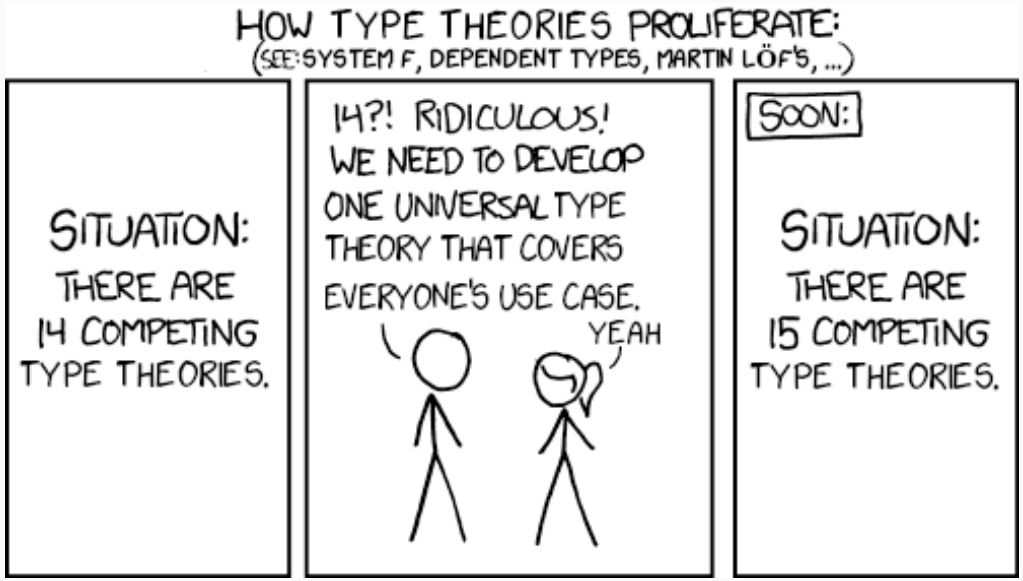
- Softwares (ITPs) do that for us.
- Coq, Cubical Agda, ...
- Each one runs on a particular type theory.

Yet Another Type Theory...

Did you say hot? No, I said HoTT!

- Get all the useful tools of others type theories.
- + isomorphic structures are equal!! ☺
- Can formalize proofs up to isomorphism.

+ everything is secretly geometry, but shhh! Don't tell it too loud, mathematicians will hear you!



Original image: `https://xkcd.com/927/`

Why Would *I* Want This?

We can compute things up to isomorphism!! What the hell?

Why Would *I* Want This?

We can compute things up to isomorphism!! What the hell?

$\Longrightarrow$ let's do some dumb things ☺

Why Would *I* Want This?

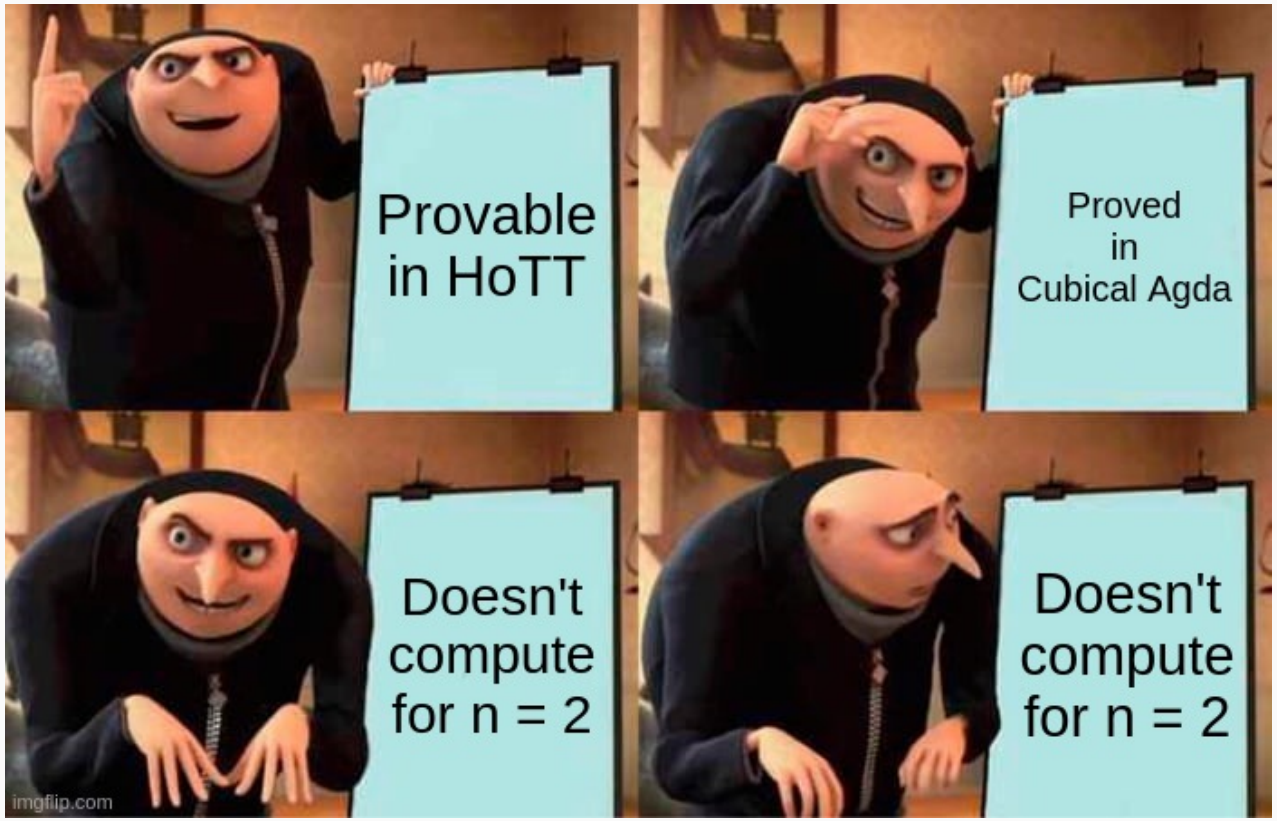
We can compute things up to isomorphism!! What the hell?

$\Longrightarrow$ let's do some dumb things ☺

OEIS-A000001

- Surely very important seeing that's it's OEIS-A000001.
- i.e., the number of groups of finite order.
- We shall compute that naively.
- Group = monoid where all elements are invertible.
- Expected complexity for naive algorithm: $\mathcal{O}(n^{n^2} n!)$.
- For $n = 2$, 32α operations, good! (α shouldn't be too big)

First Disappointment



I Hear Confusion in the Room

What happened?!? Let's take a look at the algorihm.

```
FinSemiGroupStr : FinSet ℓ → FinSet ℓ
FinSemiGroupStr X .fst =
  Σ[ p ∈ (X .fst → X .fst → X .fst) ]
    ((x y z : X .fst) → p (p x y) z ≡ p x (p y z))
FinSemiGroupStr X .snd =
  isFinSetΣ (_ , isFinSetΠ2 X (λ _ → X) (λ _ _ → X))
    (λ p → _ ,
      isFinSetΠ3 X
        (λ _ → X)
        (λ _ _ → X)
        (λ _ _ _ → _ , isFinSet≡ X _ _))
```

I Hear Confusion in the Room

What happened?!? Let's take a look at the algorihm.

```
FinSemiGroupStr : FinSet ℓ → FinSet ℓ
FinSemiGroupStr X .fst =
  Σ[ p ∈ (X .fst → X .fst → X .fst) ]
    ((x y z : X .fst) → p (p x y) z ≡ p x (p y z))
FinSemiGroupStr X .snd =
  isFinSetΣ (_ , isFinSetΠ2 X (λ _ → X) (λ _ _ → X))
    (λ p → _ ,
      isFinSetΠ3 X
        (λ _ → X)
        (λ _ _ → X)
        (λ _ _ _ → _ , isFinSet≡ X _ _))
```

OK, maybe it was a bad idea...

A Talk about Disappointments

Our Goal

Find the reason(s) that make(s) it so bad.

Our Tool

postt, experimental type-checker s.t. HoTT computes (+ analysis).

Today

- Proof that structures over finite types are finite (a fragment).
- A computational analysis of the proof with ~~magma~~ semigroups.

Homotopical Type Theory

Martin-Löf Type Theories

Dependent Types

$$\lambda x.t : \prod_{x\,:\,A} B(x) \qquad (x, p) : \sum_{x\,:\,A} P(x)$$

think of as $\forall x, B(x)$ — think of as $\exists x, P(x)$ + explicit witness

$A \to B$ shortcut for $\prod_{(\_\,:\,A)} B$ and $A \times B$ for $\sum_{(\_\,:\,A)} B$.

Martin-Löf Type Theories

Dependent Types

$$\lambda x.t : \prod_{x\colon A} B(x) \qquad (x, p) : \sum_{x\colon A} P(x)$$

think of as $\forall x, B(x)$ — think of as $\exists x, P(x)$ + explicit witness

$A \to B$ shortcut for $\prod_{(\_\colon A)} B$ and $A \times B$ for $\sum_{(\_\colon A)} B$.

Inductive Types

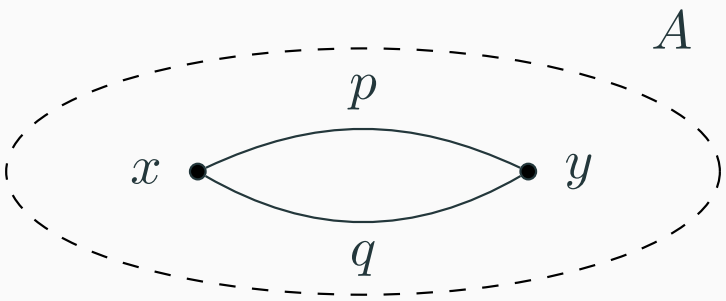
$\mathsf{inl}(x), \mathsf{inr}(y) \colon A + B$ — $\star \colon \mathbf{1}$, $\mathbf{0}$

think of as $A \vee B$ — think of as $\top$, $\bot$

$0 : \mathbb{N}, \mathsf{suc}_{\mathbb{N}}(n) : \mathbb{N}$ unary encoding of integers

Identity Types

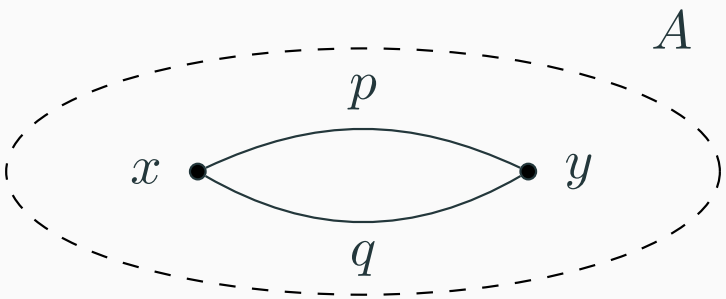
- Defined inductively by: $\mathsf{refl}_x : x =_A x$.
- Multiple proofs of identity $\rightsquigarrow$ types are spaces.



$p, q : x =_A y$ continuous paths

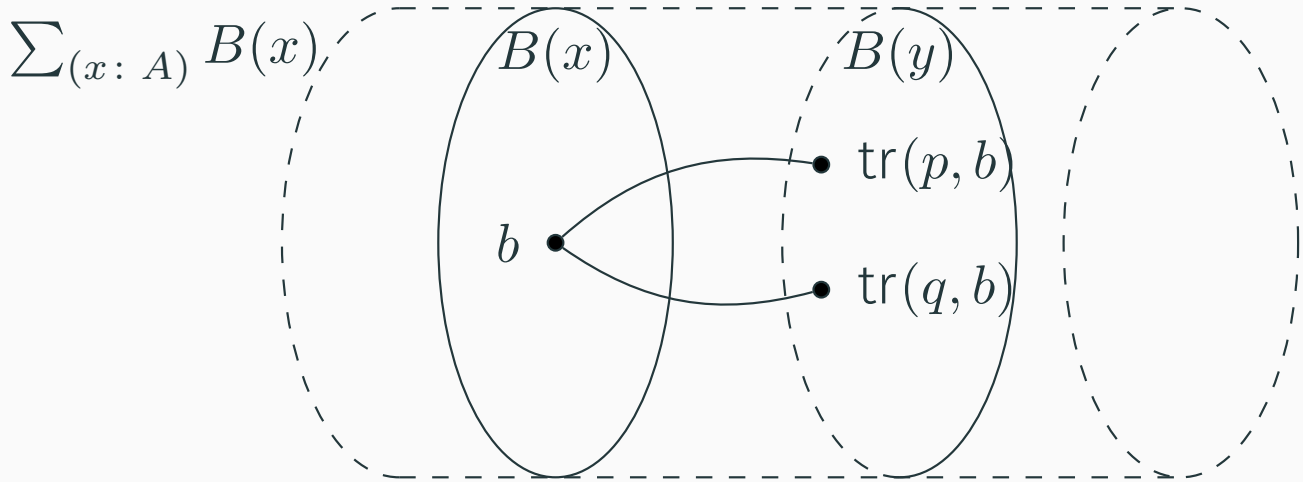
Identity Types

- Defined inductively by: $\mathsf{refl}_x : x =_A x$.
- Multiple proofs of identity $\rightsquigarrow$ types are spaces.



$p, q : x =_A y$ continuous paths

Transporting through a path:



Exploring Space

Definition (Contractible Type)

A type A is contractible if it comes together with a term

$$\text{is-contr}(A) :\equiv \prod_{x,y\,:\,A} x = y$$

Exploring Space

Definition (Contractible Type)

A type A is contractible if it comes together with a term

$$\text{is-contr}(A) :\equiv \prod_{x,y:\ A} x = y$$

Actually:

Definition (Homotopy Levels)

A type A is an n-type if it comes equipped with a term is-n-type(A):

$$\text{is-}(-2)\text{-type}(A) :\equiv \text{is-contr}(A)$$
$$\text{is-}(n+1)\text{-type}(A) :\equiv \prod_{x,y:\ A} \text{is-}n\text{-type}(x = y)$$

An Amusing Fact

We can define usual structures using h-levels:

- A type is a proposition if it is a (-1)-type (proof irrelevance).
- A type is a set if it is a 0-type (axiom K).
- Notations: $\mathsf{Prop}_{\mathcal{U}}$ and $\mathsf{Set}_{\mathcal{U}}$.

Mindblowing, right?

An Amusing Fact

We can define usual structures using h-levels:

- A type is a proposition if it is a (-1)-type (proof irrelevance).
- A type is a set if it is a 0-type (axiom K).
- Notations: $\mathsf{Prop}_{\mathcal{U}}$ and $\mathsf{Set}_{\mathcal{U}}$.

Mindblowing, right? But let's come back to our protagonist.

Bi-inverses or Inverses, That is the Question

Definition (Equivalence)

$A \simeq B$ if there exists maps $f : A \to B, g, h : B \to A$ s.t.

$$f(g(x)) = x \qquad \text{and} \qquad h(f(x)) = x$$

Theorem

$A \simeq B$ iff there exists $f : A \to B$ and $g : B \to A$ such that

$$f(g(x)) = x \qquad \text{and} \qquad g(f(x)) = x$$

Proof: blackboard.

Bi-inverses or Inverses, That is the Question

Definition (Equivalence)

$A \simeq B$ if there exists maps $f : A \to B, g, h : B \to A$ s.t.

$$f(g(x)) = x \qquad \text{and} \qquad h(f(x)) = x$$

Theorem

$A \simeq B$ iff there exists $f : A \to B$ and $g : B \to A$ such that

$$f(g(x)) = x \qquad \text{and} \qquad g(f(x)) = x$$

Proof: blackboard.

$\rightsquigarrow$ we use these notions interchangeably.

The Additional Axiom that Changes Everything

HoTT: everything we have seen before, plus:

$$(A \simeq B) \simeq (A = B)$$

and HITs, but eh, don't spoil the mood.

The Additional Axiom that Changes Everything

HoTT: everything we have seen before, plus:

$$(A \simeq B) \simeq (A = B)$$

and HITs, but eh, don't spoil the mood.

Great consequence: isomorphic types are equal!

The Additional Axiom that Changes Everything

HoTT: everything we have seen before, plus:

$$(A \simeq B) \simeq (A = B)$$

and HITs, but eh, don't spoil the mood.

Great consequence: isomorphic types are equal!

Dumb consequence: all singletons are equal (⚠ contradicts set theory (but who cares, nobody still believes in set theory in 2024, right[1]?)).

[1] Otherwise, they are *clearly* wrong, don't listen to them.

Another Amusing Fact

Another dumb consequence: two elements that are equal are not... "the same elements"!

Can you guess why?

Another Amusing Fact

Another dumb consequence: two elements that are equal are not... "the same elements"!

Can you guess why?

Here, "the same" means that the path between the elements is refl. But we can have "holes" in the space.

Examples: circles, semigroups, monoids, groups, ...

Another Amusing Fact

Another dumb consequence: two elements that are equal are not... "the same elements"!

Can you guess why?

Here, "the same" means that the path between the elements is refl. But we can have "holes" in the space.

Examples: circles, semigroups, monoids, groups, ...

By this same argument: the type of all sets is not a set!

Filling the Holes

Filling the holes in the space: “truncating” to an h-level.

Truncation Levels

Depend on what we care about:

- Propositional truncation: inhabitation.
- Set truncation: connected components.

Filling the Holes

Filling the holes in the space: "truncating" to an h-level.

Truncation Levels

Depend on what we care about:

- Propositional truncation: inhabitation.
- Set truncation: connected components.

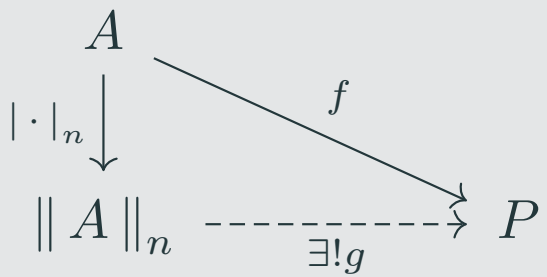
Can be defined as an HIT or by universal property. Here: the latter[2].

[2] HITs are *really* bad news: look, we even do categories over them!

I Think We're Getting Squashed, Mate

Theorem (*h*-Truncation)

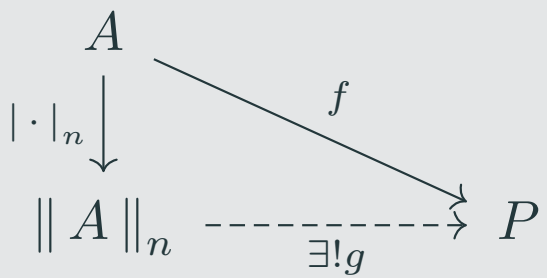
For every type A, there exists a type $\| A \|_n$ and a morphism $|\cdot|_n : A \to \| A \|_n$ such that for every n-type X and morphism $f : A \to X$, the following diagram commutes:



I Think We're Getting Squashed, Mate

Theorem (*h*-Truncation)

For every type A, there exists a type $\| A \|_n$ and a morphism $|\cdot|_n : A \to \| A \|_n$ such that for every n-type X and morphism $f : A \to X$, the following diagram commutes:



Note that I'm cheating here: this theorem does not hold for $n = -1$ in MLTT. We admit that it does though.

Come Forth, Examples!

Using propositional truncation, we get back classical logic:

- $\exists x : A, B(x)$ is $\Sigma_{(x\,:\,A)} B(x)$ without explicit inhabitant.
- Isn't that exactly $\left\| \Sigma_{(x\,:\,A)} B(x) \right\|$? (spoiler: yes it is).
- The law of excluded middle? No, but don't worry, it's false anyway.

Come Forth, Examples!

Using propositional truncation, we get back classical logic:

- $\exists x : A, B(x)$ is $\Sigma_{(x\,:\,A)} B(x)$ without explicit inhabitant.
- Isn't that exactly $\left\| \Sigma_{(x\,:\,A)} B(x) \right\|$? (spoiler: yes it is).
- The law of excluded middle? No, but don't worry, it's false anyway.

Using the magic of set truncation, we transform circles into disks!

A Total Coincidence[3]

Recall our objective (yes, it's tough, I know, it's been a long time).

- Isomorphic (semi)groups are equal!
- Can we count them like this? No: there are holes in their space.
- But look, we can fill holes now!

[3] I'm sure nobody believes me, I don't understand why...

A Total Coincidence[3]

Recall our objective (yes, it's tough, I know, it's been a long time).

- Isomorphic (semi)groups are equal!
- Can we count them like this? No: there are holes in their space.
- But look, we can fill holes now!

$\Longrightarrow$ counting up to isomorphism = counting the number of connected components, i.e., counting the set truncation.

[3] I'm sure nobody believes me, I don't understand why...

A Total Coincidence[3]

Recall our objective (yes, it's tough, I know, it's been a long time).

- Isomorphic (semi)groups are equal!
- Can we count them like this? No: there are holes in their space.
- But look, we can fill holes now!

$\Longrightarrow$ counting up to isomorphism = counting the number of connected components, i.e., counting the set truncation.

All our tools are finally ready, it's time for some fun things.

[3] I'm sure nobody believes me, I don't understand why...

Finiteness of Structures over Finite Types

Everyone Learns How to Count, Eventually

Definition (Standard Finite Types)

$$\mathsf{Fin}_0 :\equiv \mathbf{0}$$

$$\mathsf{Fin}_{\mathsf{suc}_{\mathbb{N}}(n)} :\equiv \mathsf{Fin}_n + \mathbf{1}$$

i.e., there are exactly n elements, ordered, in Fin_n.

Definition (Finite Type)

$$\text{is-finite}(A) :\equiv \sum_{(k\colon \mathbb{N})} \| \mathsf{Fin}_k \simeq A \|$$

Can you guess why we would want to use propositional truncation here?

Everyone Learns How to Count, Eventually

Definition (Standard Finite Types)

$$\mathsf{Fin}_0 :\equiv \mathbf{0}$$

$$\mathsf{Fin}_{\mathsf{suc}_{\mathbb{N}}(n)} :\equiv \mathsf{Fin}_n + \mathbf{1}$$

i.e., there are exactly n elements, ordered, in Fin_n.

Definition (Finite Type)

$\text{is-finite}(A) :\equiv \sum_{(k\,:\,\mathbb{N})} \| \mathsf{Fin}_k \simeq A \|$

Can you guess why we would want to use propositional truncation here? Without it, we get a fully ordered set.

Theorem

For every type A, $\text{is-finite}(A)$ is a proposition.

Proof: blackboard.

We Start Off Easy

Key Theorem 1: Finite Codomain

Let $f : A \to B$ a surjective function and A finite. Then B is finite whenever its equality is decidable[4].

[4] Classically, we wouldn't need this.

We Start Off Easy

Key Theorem 1: Finite Codomain

Let $f : A \to B$ a surjective function and A finite. Then B is finite whenever its equality is decidable[4].

Definition (Surjectivity)

is-surj$(f) :\equiv \prod_{(y:\ B)} \exists x, f(x) = y$

Exercise

Why use propositional truncation in the definition of surjectivity?

[4] Classically, we wouldn't need this.

We Start Off Easy

Key Theorem 1: Finite Codomain

Let $f : A \to B$ a surjective function and A finite. Then B is finite whenever its equality is decidable[4].

Definition (Surjectivity)

is-surj$(f) :\equiv \prod_{(y:\ B)} \exists x, f(x) = y$

Exercise

Why use propositional truncation in the definition of surjectivity?

Definition (Decidable Type)

A is decidable if $d\colon A + \neg A$.

[4] Classically, we wouldn't need this.

We Start Off Easy

Key Theorem 1: Finite Codomain

Let $f : A \to B$ a surjective function and A finite. Then B is finite whenever its equality is decidable[4].

Definition (Surjectivity)

is-surj$(f) :\equiv \prod_{(y:\ B)} \exists x, f(x) = y$

Exercise

Why use propositional truncation in the definition of surjectivity?

Definition (Decidable Type)

A is decidable if $d\colon A + \neg A$.

We can now prove and analyze the complexity of Key Thm. 1: blackboard.

[4] Classically, we wouldn't need this.

We Start Off Easy

Key Theorem 1: Finite Codomain

Let $f : A \to B$ a surjective function and A finite. Then B is finite whenever its equality is decidable[4].

Definition (Surjectivity)

is-surj$(f) :\equiv \prod_{(y:\ B)} \exists x, f(x) = y$

Exercise

Why use propositional truncation in the definition of surjectivity?

Definition (Decidable Type)

A is decidable if $d\colon A + \neg A$.

We can now prove and analyze the complexity of Key Thm. 1: blackboard. Cheatsheet: we should have found $\mathcal{O}(|A|^2 d)$ (d = complexity of one decidability check).

[4] Classically, we wouldn't need this.

Another Easy One

Key Theorem 2

If B is a family of finite type over a finite type A, then $\sum_{(x\colon A)} B(x)$ is also finite.

Proof and complexity analysis: blackboard.

Another Easy One

Key Theorem 2

If B is a family of finite type over a finite type A, then $\sum_{(x:\ A)} B(x)$ is also finite.

Proof and complexity analysis: blackboard. Cheatsheet: we should have found $\mathcal{O}(|A| \cdot \max_{(x:A)} |B(x)|)$.

Another Easy One

Key Theorem 2

If B is a family of finite type over a finite type A, then $\sum_{(x:\ A)} B(x)$ is also finite.

Proof and complexity analysis: blackboard. Cheatsheet: we should have found $\mathcal{O}(|A| \cdot \max_{(x:A)} |B(x)|)$.

That's more or less the only things we need to prove the main theorem.

We need, however, two more definitions.

Last Definitions

Connectedness

A is connected its set truncation is contractible.

Recall that we get disks from circles: the circle is connected.

Last Definitions

Connectedness

A is connected its set truncation is contractible.

Recall that we get disks from circles: the circle is connected.

A slightly more generic version of finiteness up to isomorphism:

Homotopy Finiteness

$$\text{is-}\pi_0\text{-finite}(A) :\equiv \text{is-finite} \parallel A \parallel_0$$

$$\text{is-}\pi_{\text{suc}_{\mathbb{N}}(n)}\text{-finite} :\equiv \text{is-finite} \parallel A \parallel_0 \times \prod_{x,y\,:\,A} \text{is-}\pi_n\text{-finite}(x = y)$$

Last Definitions

Connectedness

A is connected its set truncation is contractible.

Recall that we get disks from circles: the circle is connected.

A slightly more generic version of finiteness up to isomorphism:

Homotopy Finiteness

$$\text{is-}\pi_0\text{-finite}(A) :\equiv \text{is-finite} \parallel A \parallel_0$$

$$\text{is-}\pi_{\text{suc}_{\mathbb{N}}(n)}\text{-finite} :\equiv \text{is-finite} \parallel A \parallel_0 \times \prod_{x,y\,:\,A} \text{is-}\pi_n\text{-finite}(x = y)$$

Remark: is-π_n-finite is again a proposition (by closure under products).

Finiteness up to Isomorphism

Key Theorem 3

For B family of π_0-finite types over connected, π_1-finite type A, $\sum_{(x\,:\,A)} B(x)$ is π_0-finite.

Read: if B family of types finite up to isomorphism over A type with one connected component s.t. its identity types are finite up to isomorphism, then $\sum_{(x\,:\,A)} B(x)$ is finite up to isomorphism.

Finiteness up to Isomorphism

Key Theorem 3

For B family of π_0-finite types over connected, π_1-finite type A, $\sum_{(x\,:\,A)} B(x)$ is π_0-finite.

Read: if B family of types finite up to isomorphism over A type with one connected component s.t. its identity types are finite up to isomorphism, then $\sum_{(x\,:\,A)} B(x)$ is finite up to isomorphism.

Proof and complexity analysis: blackboard.

Finiteness up to Isomorphism

Key Theorem 3

For B family of π_0-finite types over connected, π_1-finite type A, $\sum_{(x\,:\,A)} B(x)$ is π_0-finite.

Read: if B family of types finite up to isomorphism over A type with one connected component s.t. its identity types are finite up to isomorphism, then $\sum_{(x\,:\,A)} B(x)$ is finite up to isomorphism.

Proof and complexity analysis: blackboard.

Cheatsheet: we should have found $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Application to Finite Semigroups

Definition (Finite Semigroup)

Finite Type G + associative multiplication $\mu : G \to G \to G$

Conditions of Key Thm. 3 fulfilled:

Application to Finite Semigroups

Definition (Finite Semigroup)

Finite Type G + associative multiplication $\mu : G \to G \to G$

Conditions of Key Thm. 3 fulfilled:

- Finite type is connected (by univalence, $\mathsf{Fin}_n \simeq X$ implies $\mathsf{Fin}_n = X$ + propositional truncation of equivalence).

Application to Finite Semigroups

Definition (Finite Semigroup)

Finite Type G + associative multiplication $\mu : G \to G \to G$

Conditions of Key Thm. 3 fulfilled:

- Finite type is connected (by univalence, $\mathsf{Fin}_n \simeq X$ implies $\mathsf{Fin}_n = X$ + propositional truncation of equivalence).
- Associative multiplication is finite: equality on a set is a proposition and finiteness is closed under Σ-types by Key Thm. 2.

Application to Finite Semigroups

Definition (Finite Semigroup)

Finite Type G + associative multiplication $\mu : G \to G \to G$

Conditions of Key Thm. 3 fulfilled:

- Finite type is connected (by univalence, $\mathsf{Fin}_n \simeq X$ implies $\mathsf{Fin}_n = X$ + propositional truncation of equivalence).
- Associative multiplication is finite: equality on a set is a proposition and finiteness is closed under Σ-types by Key Thm. 2.

Complexity added: $\mathcal{O}(|G \to G \to G|)$ negligible (why? in a minute).

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.
- $\|\,B(G)\,\|_0 \simeq B(G)$ as $B(G)$ is a set.

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.
- $\|\,B(G)\,\|_0 \simeq B(G)$ as $B(G)$ is a set.
- A: finite types, equality: $G = H$ is a set (as G, H sets).

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.
- $\|\,B(G)\,\|_0 \simeq B(G)$ as $B(G)$ is a set.
- A: finite types, equality: $G = H$ is a set (as G, H sets).
- $(G = H) \simeq (G \simeq H)$.

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.
- $\|\,B(G)\,\|_0 \simeq B(G)$ as $B(G)$ is a set.
- A: finite types, equality: $G = H$ is a set (as G, H sets).
- $(G = H) \simeq (G \simeq H)$.

For a type of order n:

- $|B(G)| = |G \to G \to G| = o(n^{n^2})$
- $|G \simeq H| = n!$

The Final Adversary

Complexity of Key Thm. 3: $\mathcal{O}(|\,\|\,B(a)\,\|_0\,|^3(|\,\|\,a = a\,\|_0\,|))$

Here:

- B: associative multiplications.
- $\|\,B(G)\,\|_0 \simeq B(G)$ as $B(G)$ is a set.
- A: finite types, equality: $G = H$ is a set (as G, H sets).
- $(G = H) \simeq (G \simeq H)$.

For a type of order n:

- $|B(G)| = |G \to G \to G| = o(n^{n^2})$
- $|G \simeq H| = n!$

Total complexity: $\mathcal{O}(n^{3n^2} n!)$

Numerical Application

Total complexity: $\mathcal{O}(n^{3n^2} n!)$

Numerical Application

Total complexity: $\mathcal{O}(n^{3n^2} n!)$

Recall our initial aimed complexity: $\mathcal{O}(n^{n^2} n!)$. Not so bad, eh?

Numerical Application

Total complexity: $\mathcal{O}(n^{3n^2}n!)$

Recall our initial aimed complexity: $\mathcal{O}(n^{n^2}n!)$. Not so bad, eh?

For $n = 2$? 8192α operations...Far from the $32\alpha'$ of the naive algorithm.

Numerical Application

Total complexity: $\mathcal{O}(n^{3n^2} n!)$

Recall our initial aimed complexity: $\mathcal{O}(n^{n^2} n!)$. Not so bad, eh?

For $n = 2$? 8192α operations...Far from the $32\alpha'$ of the naive algorithm.

But, computers are powerful, isn't it easy to do that many operations?

Numerical Application

Total complexity: $\mathcal{O}(n^{3n^2}n!)$

Recall our initial aimed complexity: $\mathcal{O}(n^{n^2}n!)$. Not so bad, eh?

For $n = 2$? 8192α operations...Far from the $32\alpha'$ of the naive algorithm.

But, computers are powerful, isn't it easy to do that many operations?

$\Longrightarrow$ yes and no. We compute λ-terms, so it all depends on the evaluation function. But proofs are big, term size explodes and quickly slows down the evaluation.

Conclusion[5] (1)

Were cubical type theories a mistake?

[5] Thanks to T. Coquand and J. Höfer for their precious advice

Conclusion[5] (1)

Were cubical type theories a mistake?

Yes: proof was very painful to write in a prototype

[5] Thanks to T. Coquand and J. Höfer for their precious advice

Conclusion[5] (1)

Were cubical type theories a mistake?

No: the proof is bad, not the type theory

[5] Thanks to T. Coquand and J. Höfer for their precious advice

Conclusion[5] (1)

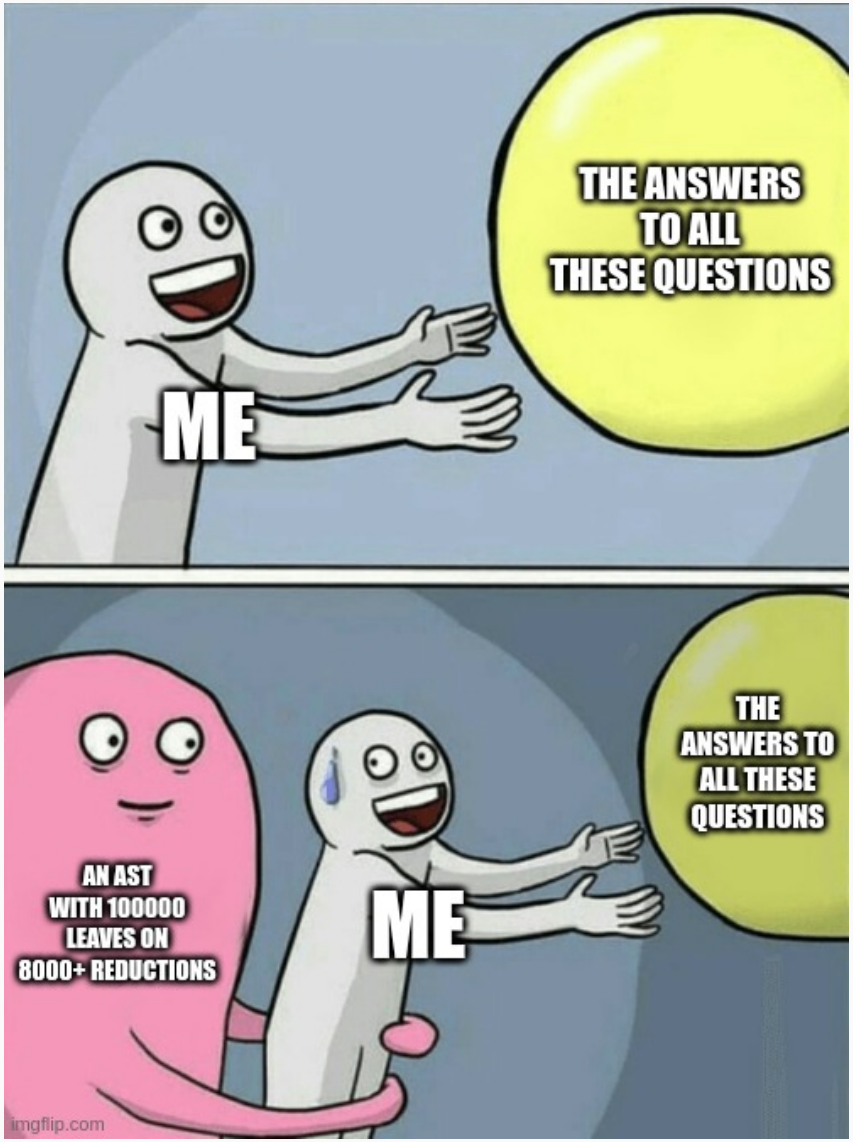
Were cubical type theories a mistake?

Bottom line: we don't know

- Large blow-up in the term size (is cubical involved? Or not?)
- Better proof? But what is *better*?
- Tradeoff between term size and theoretical complexity.

[5] Thanks to T. Coquand and J. Höfer for their precious advice

Conclusion (2)



Scribitur ad narrandum, non ad probandum.

Thanks for your attention!

A special thanks to Adrien M. for typesetting the xkcd.