Towards homotopy canonicity for propositional type theory

Benno van den Berg
ILLC, University of Amsterdam

Workshop on the Strength of Weak Type Theory
Amsterdam, 12 May 2023

Traditional picture

Traditional view

Computation = normalisation = what the machine does

From this point of view, the two forms of equality reflect a division of labour.

- Definitional equality: something the machine checks
- Propositional equality: something the human has to prove

Traditional picture

Traditional view

Computation = normalisation = what the machine does

From this point of view, the two forms of equality reflect a division of labour.

- Definitional equality: something the machine checks
- Propositional equality: something the human has to prove

There are a number of issues with this:

- How do we decide who does what?
- Type checking and checking definitional equality can be terribly expensive.

Is this really a proof?

We usually think of proofs as explaining why something is the true (giving the reason why something is true). But does this happen in type theory?

Is this really a proof?

We usually think of proofs as explaining why something is the true (giving the reason why something is true). But does this happen in type theory?

Let A be the Ackermann function.

Is this really a proof?

We usually think of proofs as explaining why something is the true (giving the reason why something is true). But does this happen in type theory?

Let A be the Ackermann function.

Then:

$$\vdash \mathsf{refl}(A(3, 2^{65536} - 3)) \in \mathrm{Id}(\mathbb{N}, A(3, 2^{65536} - 3), A(4, 3)).$$

Is this really a proof?

We usually think of proofs as explaining why something is the true (giving the reason why something is true). But does this happen in type theory?

Let A be the Ackermann function.

Then:

$$\vdash \mathsf{refl}(A(3, 2^{65536} - 3)) \in \mathrm{Id}(\mathbb{N}, A(3, 2^{65536} - 3), A(4, 3)).$$

Compare:

$$\vdash \mathsf{refl}(A(3, 2^{65535} - 3)) \in \mathrm{Id}(\mathbb{N}, A(3, 2^{65535} - 3), A(4, 3)).$$

Decidability

The question whether the following judgments are derivable is decidable:

$$\Gamma \vdash \sigma = \tau$$
$$\Gamma \vdash a = b \in \sigma$$
$$\Gamma \vdash a \in \sigma$$

Decidability

The question whether the following judgments are derivable is decidable:

$$\Gamma \vdash \sigma = \tau$$
$$\Gamma \vdash a = b \in \sigma$$
$$\Gamma \vdash a \in \sigma$$

Indeed, the first two are decidable by normalisation and the third by backwards proof search.

Decidability

The question whether the following judgments are derivable is decidable:

$$\Gamma \vdash \sigma = \tau$$
$$\Gamma \vdash a = b \in \sigma$$
$$\Gamma \vdash a \in \sigma$$

Indeed, the first two are decidable by normalisation and the third by backwards proof search.

So these questions are decidable *in theory*. But are they also decidable *in practice*?

Decidability

The question whether the following judgments are derivable is decidable:

$$\Gamma \vdash \sigma = \tau$$
$$\Gamma \vdash a = b \in \sigma$$
$$\Gamma \vdash a \in \sigma$$

Indeed, the first two are decidable by normalisation and the third by backwards proof search.

So these questions are decidable *in theory*. But are they also decidable *in practice*?

Theorem (Statman, 1979)

Equality in the typed lambda calculus is not elementary recursive.

Since type checking relies on checking definitional equalities, type checking is also not elementary recursive.

Quote from Geuvers and Wiedijk

Quote

In theorem provers based on type theory the main performance bottleneck is the convertibility check: if the calculated type of a term M is A, but it used in a context where the type should be B, then the system needs to verify that $A =_{\beta\iota\delta} B$, where δ is the equality arising from definitional expansion (unfolding definitions) and ι is the equality arising from functions defined by (higher order primitive) recursion. In fact, the inefficiency of the convertibility check means that type correctness is in practice only semi-decidable. Although in theory it is decidable whether a term M has type A, in practice when it is not correct the system could be endlessly reducing and would not terminate in an acceptable time anymore.

From: Herman Geuvers and Freek Wiedijk. *A logical framework with explicit conversions*. Electronic Notes in Theoretical Computer Science 199 (2008), 33-47.

Towards propositional type theory

It is not clear that . . .

- the machine should normalise.
- normalisation is the only thing the machine should do.

This also does not seem to be what happens in reality.

More realistic picture: the default option is that the human does everything and then we let pragmatic considerations decide what to leave to the machine.

Propositional type theory

Propositional type theory

Propositional type theory is a version of type theory without definitional equality and in which all computation rules are stated in propositional form. (Other names: *homotopy type theory with explicit conversions* or *objective type theory*.)

Propositional type theory

Propositional type theory

Propositional type theory is a version of type theory without definitional equality and in which all computation rules are stated in propositional form. (Other names: *homotopy type theory with explicit conversions* or *objective type theory*.)

Theorem (den Besten & BvdB)

The question whether $\Gamma \vdash a \in \sigma$ is derivable in propositional type theory or not can be decided in quadratic time.

Propositional type theory

Propositional type theory

Propositional type theory is a version of type theory without definitional equality and in which all computation rules are stated in propositional form. (Other names: *homotopy type theory with explicit conversions* or *objective type theory*.)

Theorem (den Besten & BvdB)

The question whether $\Gamma \vdash a \in \sigma$ is derivable in propositional type theory or not can be decided in quadratic time.

Claim

All the basic stuff of HoTT (first so many chapters of the HoTT book) can be formalised in HoTT with explicit conversions.

Propositional type theory

Propositional type theory

Propositional type theory is a version of type theory without definitional equality and in which all computation rules are stated in propositional form. (Other names: *homotopy type theory with explicit conversions* or *objective type theory*.)

Theorem (den Besten & BvdB)

The question whether $\Gamma \vdash a \in \sigma$ is derivable in propositional type theory or not can be decided in quadratic time.

Claim

All the basic stuff of HoTT (first so many chapters of the HoTT book) can be formalised in HoTT with explicit conversions.

Question

To what extent is standard homotopy type theory conservative over homotopy type theory with explicit conversions?

Propositional type theory

Arguments in favour of propositional type theory (or homotopy type theory with explicit conversions):

- Type-checking efficiently decidable
- Proofs become explanatory
- Sometimes arbitrary which equalities are propositional and which definitional in standard type theory (two forms of plus)
- Natural from homotopy-theoretic point of view

Propositional type theory

Arguments in favour of propositional type theory (or homotopy type theory with explicit conversions):

- Type-checking efficiently decidable
- Proofs become explanatory
- Sometimes arbitrary which equalities are propositional and which definitional in standard type theory (two forms of plus)
- Natural from homotopy-theoretic point of view

Drawbacks:

- Proof terms become very long!

Is propositional type theory still a framework for computation?

Type-theoretic paradigm

Computation = normalisation

What remains of this without definitional equality?

Is propositional type theory still a framework for computation?

Type-theoretic paradigm

Computation $=$ normalisation

What remains of this without definitional equality?

Conjecture

Propositional type theory still enjoys *homotopy canonicity*.

Homotopy canonicity

Given a closed term t of type $\mathbb{N}$ there exist a numeral $S^k(0)$ and a proof term p such that $\vdash p \in \mathrm{Id}(\mathbb{N}, t, S^k(0))$.

Of course, we want an *effective* proof: we want to be able to effectively obtain p and k from t.

Glueing

A nice categorical method for proving canonicity is *glueing* (aka *sconing* or the *Freyd cover*).

This is also what has been used by Kapulkin & Sattler in their proof of homotopy canonicity of (standard) homotopy type theory.

The classical source for this method is Lambek & Scott's book on higher order categorical logic. Let's revisit that proof.

Glueing

A nice categorical method for proving canonicity is *glueing* (aka *sconing* or the *Freyd cover*).

This is also what has been used by Kapulkin & Sattler in their proof of homotopy canonicity of (standard) homotopy type theory.

The classical source for this method is Lambek & Scott's book on higher order categorical logic. Let's revisit that proof.

By a *topos* we mean an elementary topos with a natural numbers object. The initial topos (the initial object in the category whose objects are toposes and whose morphisms are logical functors) is built from the syntax of higher-order arithmetic.

Topos-theoretic glueing

Glueing

Let $F : \mathcal{E} \to \mathcal{F}$ be a functor between toposes preserving finite limits. Then we can build a new topos $\mathrm{Gl}(F)$ as follows:

Objects: Triple (X, A, α) consisting of an object X in $\mathcal{E}$, an object A in $\mathcal{F}$ and a morphism $\alpha : X \to FA$.

Morphisms: A morphism $(X, A, \alpha) \to (Y, B, \beta)$ is a pair of morphisms $f : X \to Y$ and $g : A \to B$ such that

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} \\ FA & \xrightarrow[Fg]{} & FB \end{array}$$

commutes.

Moreover, the forgetful functor $\mathrm{Gl}(F) \to \mathcal{F}$ is logical.

Freyd cover

Let $\mathcal{I}$ be the initial topos and consider the functor

$$\Gamma : \mathcal{I} \to \mathsf{Sets}$$

sending X to $\mathrm{Hom}_{\mathcal{E}}(1, X)$ ("the global sections functor").

Freyd cover

Let $\mathscr{I}$ be the initial topos and consider the functor

$$\Gamma : \mathscr{I} \to \mathsf{Sets}$$

sending X to $\mathrm{Hom}_{\mathscr{E}}(1, X)$ ("the global sections functor"). This functor preserves limits, hence we obtain a logical functor

$$U : \mathrm{Gl}(\Gamma) \to \mathscr{I}.$$

Freyd cover

Let $\mathcal{I}$ be the initial topos and consider the functor

$$\Gamma : \mathcal{I} \to \mathrm{Sets}$$

sending X to $\mathrm{Hom}_{\mathcal{E}}(1, X)$ ("the global sections functor"). This functor preserves limits, hence we obtain a logical functor

$$U : \mathrm{Gl}(\Gamma) \to \mathcal{I}.$$

Since $\mathcal{I}$ is initial, we have a logical functor $V : \mathcal{I} \to \mathrm{Gl}(\Gamma)$ such that $U \circ V = 1$.

Freyd cover

Let $\mathcal{I}$ be the initial topos and consider the functor

$$\Gamma : \mathcal{I} \to \mathrm{Sets}$$

sending X to $\mathrm{Hom}_{\mathcal{E}}(1, X)$ ("the global sections functor"). This functor preserves limits, hence we obtain a logical functor

$$U : \mathrm{Gl}(\Gamma) \to \mathcal{I}.$$

Since $\mathcal{I}$ is initial, we have a logical functor $V : \mathcal{I} \to \mathrm{Gl}(\Gamma)$ such that $U \circ V = 1$.

Now consider a term $t : 1 \to \mathbb{N}$ in $\mathcal{I}$ and its image under V:

$$\begin{array}{ccc} 1 & \xrightarrow{k} & \mathbb{N} \\ \downarrow & & \downarrow \\ \Gamma 1 & \xrightarrow[\Gamma t]{} & \Gamma \mathbb{N} \end{array}$$

Freyd cover

Let $\mathcal{I}$ be the initial topos and consider the functor

$$\Gamma : \mathcal{I} \to \mathrm{Sets}$$

sending X to $\mathrm{Hom}_{\mathcal{E}}(1, X)$ ("the global sections functor"). This functor preserves limits, hence we obtain a logical functor

$$U : \mathrm{Gl}(\Gamma) \to \mathcal{I}.$$

Since $\mathcal{I}$ is initial, we have a logical functor $V : \mathcal{I} \to \mathrm{Gl}(\Gamma)$ such that $U \circ V = 1$.
Now consider a term $t : 1 \to \mathbb{N}$ in $\mathcal{I}$ and its image under V:

$$\begin{array}{ccc} 1 & \xrightarrow{k} & \mathbb{N} \\ \downarrow & & \downarrow \\ \Gamma 1 & \xrightarrow[\Gamma t]{} & \Gamma \mathbb{N} \end{array}$$

The top map is some $k \in \mathbb{N}$ and the commutativity of the square tells us that $S^k 0 = t$ in $\mathcal{I}$.

A variation for propositional type theory

The idea would be to find a similar proof for propositional type theory.

A variation for propositional type theory

The idea would be to find a similar proof for propositional type theory.

Let's just pretend that:

- path categories are an unproblematic semantics for propositional type theory (no coherence issues),
- and that there is an initial object $\mathcal{I}$ in the category of path categories and exact functors and it is built from the syntax of propositional type theory.

A variation for propositional type theory

The idea would be to find a similar proof for propositional type theory.

Let's just pretend that:

- path categories are an unproblematic semantics for propositional type theory (no coherence issues),
- and that there is an initial object $\mathcal{I}$ in the category of path categories and exact functors and it is built from the syntax of propositional type theory.

Exact functor

If $\mathcal{E}$ and $\mathcal{F}$ are path categories, then a functor $F : \mathcal{E} \to \mathcal{F}$ is *exact* if it preserves fibrations, weak equivalences, the terminal object and pullbacks of fibrations.

(To make it really interesting we need to consider path categories with homotopy Π-types, a homotopy natural numbers object and a univalent fibration, among other things.)

Glueing for path categories

Glueing for path categories (De Boer)

Let $F : \mathcal{E} \to \mathcal{F}$ be an exact functor between path categories. Then we can build a new path category as follows:

Objects: Triple (X, A, α) consisting of an object X in $\mathcal{E}$, an object A in $\mathcal{F}$ and a *fibration* $\alpha : X \to FA$.

Morphisms: A morphism $(X, A, \alpha) \to (Y, B, \beta)$ is a pair of morphisms $f : X \to Y$ and $g : A \to B$ such that

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} \\ FA & \xrightarrow[Fg]{} & FB \end{array}$$

commutes.

Equivalences are pointwise and the fibrations are the Reedy fibrations. Moreover, the forgetful functor $\mathrm{Gl}(F) \to \mathcal{F}$ is exact.

Freyd cover?

Now one would like to have an exact functor

$$\Gamma : \mathcal{I} \to \mathrm{Sets}.$$

How to turn Sets into a path category? Probably every map should be a fibration and the equivalences should be the isomorphism. But then the global sections functor is not exact.

Freyd cover?

Now one would like to have an exact functor

$$\Gamma : \mathcal{I} \to \mathrm{Sets}.$$

How to turn Sets into a path category? Probably every map should be a fibration and the equivalences should be the isomorphism. But then the global sections functor is not exact.

But shouldn't really glue to Sets, but to the category of ∞-groupoids. But how to construct an exact functor from $\mathcal{I}$ to the category of ∞-groupoids?

Freyd cover?

Now one would like to have an exact functor

$$\Gamma : \mathcal{I} \to \text{Sets}.$$

How to turn Sets into a path category? Probably every map should be a fibration and the equivalences should be the isomorphism. But then the global sections functor is not exact.

But shouldn't really glue to Sets, but to the category of ∞-groupoids. But how to construct an exact functor from $\mathcal{I}$ to the category of ∞-groupoids?

This is really hard, if not impossible!

Following Kapulkin & Sattler

Idea

We replace $\mathcal{I}$ by frames on $\mathcal{I}$ and then take the global sections functor to semisimplicial Kan complexes.

Roughly speaking, frames on $\mathcal{C}$ is the path category of semisimplicial objects in $\mathcal{C}$ with a Reedy-type structure.

Theorem (Paauw)

If $\mathcal{C}$ is a path category, then so is the category of frames on $\mathcal{C}$ and there is a functor $\mathrm{ev}_0 : \mathrm{Fr}(\mathcal{C}) \to \mathcal{C}$ which is an acyclic fibration of path categories.

Theorem (Paauw)

There exists a global sections functor $\Gamma : \mathrm{Fr}(\mathcal{C}) \to \mathrm{ssKan}$ which is exact.

As Kapulkin & Sattler observe, this is enough to perform the glueing construction and obtain homotopy canonicity.

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).
- Kapulkin & Sattler then use the right Kan extension ssSets $\to$ sSets to get to a genuine model of homotopy type theory.

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).
- Kapulkin & Sattler then use the right Kan extension ssSets $\to$ sSets to get to a genuine model of homotopy type theory.
- But since the model in simplicial sets is non-constructive, this only gives us an ineffective form of homotopy canonicity.

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).
- Kapulkin & Sattler then use the right Kan extension ssSets $\to$ sSets to get to a genuine model of homotopy type theory.
- But since the model in simplicial sets is non-constructive, this only gives us an ineffective form of homotopy canonicity.
- Can we use a constructive version of simplicial sets? (Unclear to me if the work of Gambino, Henry, Sattler and Szumilo is sufficient.)

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).
- Kapulkin & Sattler then use the right Kan extension ssSets $\to$ sSets to get to a genuine model of homotopy type theory.
- But since the model in simplicial sets is non-constructive, this only gives us an ineffective form of homotopy canonicity.
- Can we use a constructive version of simplicial sets? (Unclear to me if the work of Gambino, Henry, Sattler and Szumilo is sufficient.)
- Kapulkin & Sattler mention something about a suitable structure in some version of cubical sets; but the details of that have never appeared, as far as I am aware.

Issues

- It is unclear whether semisimplicial sets carry a model of full propositional type theory (how about homotopy Π-types?).
- Kapulkin & Sattler then use the right Kan extension ssSets $\to$ sSets to get to a genuine model of homotopy type theory.
- But since the model in simplicial sets is non-constructive, this only gives us an ineffective form of homotopy canonicity.
- Can we use a constructive version of simplicial sets? (Unclear to me if the work of Gambino, Henry, Sattler and Szumilo is sufficient.)
- Kapulkin & Sattler mention something about a suitable structure in some version of cubical sets; but the details of that have never appeared, as far as I am aware.
- In any case, we should switch to genuine models of propositional type theory and see if the frames and glueing construction still works for this genuine notion of model (say, comprehension categories with propositional identity types as in Daniël's talk).

Issues

In short: there is still a lot that needs to be sorted out!

Thank you!