A stratified approach to Löb induction

Daniel Gratzer, Lars Birkedal

FSCD 2022

Aarhus University

Guarded recursion

Big idea of guarded recursion

Use modalities to ensure recursive definitions are justified.

Two distinct motivations for this particular approach:

- A way to tame recursion in programming (and coinduction along the way)
- A foundation for version of domain theory suitable for denotational semantics

Both find a home in a guarded *dependent* type theory.

What is a guarded dependent type theory?

Guarded dependent type theory extends MLTT with at least the following:

1. An applicative functor $(\blacktriangleright, \mathsf{next}, \circledast)$ (the "later" modality)
2. A constant witnessing Löb induction $\mathsf{lob} : (\blacktriangleright A \to A) \to A$

This is the absolute minimum, we often require more!

1. A dependent version of the later modality: $\blacktriangleright \mathsf{U} \to \mathsf{U}$
2. Other modalities which complement $\blacktriangleright$ (its left adjoint $\blacktriangleleft$ or $\Box$)

These and many other extensions are explored in the literature.

Does this theory have canonicity?

How close is this to a well-behaved type theory?

- We have experience handling modalities like $\blacktriangleright$
- Type theories for $\blacktriangleright$ (with $\square$, $\blacktriangleleft$, ...) exist with canonicity/normalization/...

Does this theory have canonicity?

How close is this to a well-behaved type theory?

- We have experience handling modalities like $\blacktriangleright$
- Type theories for $\blacktriangleright$ (with $\Box$, $\blacktriangleleft$, ...) exist with canonicity/normalization/...

lob is a problem; in order to have a hope of canonicity, we must allow lob to unfold:

$$\mathrm{lob}(f) = f(\mathrm{next}(\mathrm{lob}(f)))$$

(Compare standard rule for fixed-point unfolding: $\mathtt{fix}(f) = f(\mathtt{fix}(f)))$

Is this minimal theory implementable?

Q Does this theory satisfy normalization? Can we implement it?

A Nope!

Adding the previous equation may save canonicity, but it destroys normalization.

Question

So, what do we do now?

Contributions

We offer a nuanced answer to the previous question:

1. We prove a no-go theorem with fairly minimal assumptions.
2. We argue that the correct approach is to split the theory into two halves.

This *stratified* guarded type theory is essentially two closely related halves:

- sGTT enjoys decidable type-checking, but not canonicity.
- dGTT has canonicity, but not decidable type-checking.
- Every model of dGTT is a model of sGTT (including syntax).

Slogan

Write sGTT, run dGTT

A no-go theorem for Löb induction

In addition to the primitives ($\blacktriangleright$, lob, the unfolding rule), we need guarded streams:

$$\mathtt{Str} \cong \mathsf{nat} \times \blacktriangleright \mathtt{Str}$$

With universes and dependent $\blacktriangleright$, we can obtain this through lob on U.

Using Löb induction, we can "tabulate" a function:

$$\mathtt{tabulate} : (\mathsf{nat} \to \mathsf{nat}) \to \mathtt{Str}$$

Lemma

The nth element of $\mathtt{tabulate}(f)$ *is definitionally equal to* $\mathsf{next}^n(f(n))$

A no-go theorem for Löb induction II

Question

When does $\texttt{tabulate}(f) = \texttt{tabulate}(g)$?

If next is injective on closed terms precisely when f and g are pointwise equal.

We can bundle this up into a no-go theorem:

Theorem

GTT satisfying these 2 additional requirements has undecidable conversion.

A no-go theorem for Löb induction III

Can we attack these two additional requirements?

- The addition of streams is fairly unobjectionable[1]
- Definitional injectivity for next on closed terms follows from adding $\square$ or $\blacktriangleleft$

In fact, these additional requirements seem validated by all prior systems.

[1] If we remove guarded streams I'll also have to think of new examples. Hardly seems worth it.

Lowering our expectations: guarded type theory sans unfolding

If we drop the requirement for canonicity, things become much simpler!

- We can take a modal type theory off-the-shelf.
- Adding lob without unfolding means that it does not impact conversion checking!
- We can even add a further constant: a propositional equality for unfolding.

MTT: an off-the-shelf modal type theory

We will base our type theory on MTT: a multimodal type theory.

- MTT is a framework: give a 2-category describing modalities, get a type theory.
- The original work on MTT included a description of a type theory with $\blacktriangleright$ and $\Box$
- We opt for including the left adjoint $\blacktriangleleft$ to $\blacktriangleright$ rather than $\Box$ but intuitively

$$\Box = \lim_n \blacktriangleleft^n$$

The result? A type theory equipped with a pair of modalities $\blacktriangleleft \dashv \blacktriangleright$.

MTT with ◄ and ►

To be more precise, modalities in MTT a paired with an adjoint action on contexts:

$$\frac{\Gamma.\{\ell\} \vdash A}{\Gamma \vdash \langle \ell \mid A \rangle = \blacktriangleright A} \qquad \frac{\Gamma.\{e\} \vdash A}{\Gamma \vdash \langle e \mid A \rangle = \blacktriangleleft A} \qquad \frac{\mu \in \{\ell, e\} \qquad \Gamma.\{\mu\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_\mu(M) : \langle \mu \mid A \rangle}$$

All MTT modalities preserve products and we also obtain the following combinators:

$$\mathsf{next} : A \to \blacktriangleright A \qquad \blacktriangleleft A \to A \qquad \eta : A \to \blacktriangleright \blacktriangleleft A \qquad \epsilon : \blacktriangleleft \blacktriangleright A \simeq A$$

sGTT

Consider MTT with the aforementioned mode theory plus the following two constants:

$$\frac{\Gamma \vdash M : \blacktriangleright A \to A}{\Gamma \vdash \mathsf{lob}(M) : A} \qquad \frac{\Gamma \vdash M : \blacktriangleright A \to A}{\Gamma \vdash \mathsf{unfold}(M) : \mathsf{Id}_A(\mathsf{lob}(M), M(\mathsf{next}(\mathsf{lob}(M))))}$$

We call this theory sGTT. While it doesn't support canonicity, we have the following:

Theorem

sGTT enjoys normalization and decidable type-checking.

We can also interpret it into $\mathbf{PSh}(\omega)$ and other standard models of guarded recursion.

Where do we stand?

In some sense, this is a natural stopping point!

- sGTT is perfectly workable and implementable.
- In fact, we have implemented it!

Two potential consequences of the lack of definitional equality:

1. Programs no longer "run" in any meaningful way.
2. This *might* make using sGTT as a proof assistant impossibly painful.

How far away are we from canonicity?

So sGTT lacks canonicity, but by how much? Consider the following:

$$\mathrm{lob}(M) = M(\mathrm{next}(\mathrm{lob}(M))) \qquad\qquad \mathrm{unfold}(M) = \mathrm{refl}(\mathrm{lob}(M))$$

These additions fall prey to our no-go theorem, so conversion is undecidable.

Question

Do we get canonicity back?

Canonicity in guarded type theory

In the presence of guarded recursive types, canonicity is a bit subtle.

Canonicity I

Every $\mathbf{1} \vdash M : \mathsf{nat}$ is equal to a numeral

We want to run programs "under a later". What can we say about $M : \blacktriangleright\mathsf{nat}$?

Canonicity in guarded type theory

In the presence of guarded recursive types, canonicity is a bit subtle.

Canonicity I

Every $\mathbf{1} \vdash M : \mathsf{nat}$ is equal to a numeral

We want to run programs "under a later". What can we say about $M : \blacktriangleright \mathsf{nat}$?

Canonicity II

Given $\mathbf{1} \vdash M : A$,

- if $A = \mathsf{nat}$ then $M = \bar{n}$ for some n.
- if $A = \blacktriangleright B$ then $M = \mathsf{mod}_{\ell}(N)$ for some $\mathbf{1}.\{\ell\} \vdash N : B$

We can't iterate this to analyze $\blacktriangleright \blacktriangleright \mathsf{nat}$; N doesn't have the right type.

Guarded canonicity

If we attempt to generalize to terms in $\mathbf{1}.\{\ell^n\}$, we must answer a question:

Question

How do we handle "infinite" values such as $\texttt{tabulate}(f)$?

Answer

We "gradualize" the algorithm, allowing one to extract finite prefixes.

Unlike prior work, we cannot use step-based counting to accomplish this.

Type-directed guarded canonicity

We introduce a novel construct to allow us to state guarded canonicity:

$$\frac{}{\mathbf{0}\ \mathsf{cx}}$$

- In the context $\mathbf{0}$, all judgments are true e.g. $\mathbf{0} \vdash 0 = 1 : \mathsf{nat}$
- We actually add $\mathbf{0}[\mu]$ such that $[\Gamma, \mathbf{0}[\mu]] \cong [\Gamma.\{\mu\}, \mathbf{0}]$

Lemma

$\mathbf{0}[\mu].\{\mu\} \vdash \mathcal{J}$ *always holds.*

Definition

dGTT is the system extending sGTT with the following two equalities and $\mathbf{0}[-]$:

$$\mathsf{lob}(M) = M(\mathsf{next}(\mathsf{lob}(M))) \qquad \mathsf{unfold}(M) = \mathsf{refl}(\mathsf{lob}(M))$$

Guarded canonicity II

Big Idea

Don't study canonicity in $\mathbf{1}$, study it in $\mathbf{0}[\mu].\{\nu\}$ for all (μ, ν).

- Intuitively: μ is the starting fuel and ν records how much we've spent.
- If ν exceeds μ, canonicity (a statement about judgments) trivializes.
- A purely type-directed way of recovering fuel

Guarded canonicity III

With this new idea, we can state guarded canonicity for dGTT.

Theorem

Given $\mathbf{0}[\mu].\{\nu\} \vdash M : A$

- *If* $A = \mathrm{nat}$ *then* $M = \bar{n}$ *for some* n
- *If* $A = \langle \xi \mid B \rangle$ *then* $M = \mathrm{mod}_{\xi}(N)$ *for some* $\mathbf{0}[\mu].\{\nu \circ \xi\} \vdash N : B$

We omit the other routine cases.

Proof.

By way of a novel extension of multimodal STC [Gra22] □

To run a program, choose an input fuel based on its type and apply the above theorem.

Let's return to our earlier issues with sGTT:

1. Programs no longer "run" in any meaningful way.
2. This might make using sGTT as a proof assistant impossibly painful.

We can't solve (1) directly, but we have an obvious compilation procedure to dGTT.

Theorem

Any reasonable model of sGTT is a model of dGTT.

We can use this compilation procedure to run programs!

Practicality?

The second point is much harder to answer (what does “practical” mean?)

- It’s an experimental question
- So we’ve done some experiments!

We revamp Boulmé and Hamon and formalize synchronous programming (SP).

Summary of the case study

We followed Boulmé and Hamon to present the semantics of key operations in SP.

- We interpret core types by a version of guarded streams.
- These guarded streams are highly dependent to allow variable rates of production.
- On top of this, we define various stream transformers and verify their laws.

These are the primitives needed to encode something like Lustre.

- We did the key proofs in sGTT
- It was surprisingly feasible!
- To mechanize, we'd need support for more syntactic sugar.
- "Compiled" to dGTT to run closed examples.

Highly related work

Cubical Clocked Type Theory (CloTT) presents another way around our no-go theorem

- Essentially, CloTT only allows lob to unfold at the top-level
- This prevents the uncontrolled unfolding and escapes the no-go theorem

It seems like CloTT may mirror the dGTT and sGTT

- Examples don't use equalities for lob, so evidence for sGTT usability!
- It is unclear that the definitional equality *could* help in formalization.

Question

Can we get these same metatheoretical results for CloTT?

Conclusions

We prove a no-go theorem for lob and contribute a pair of type theories skirting it.

- sGTT enjoys decidable type-checking, but not canonicity.
- dGTT has canonicity, but—by our no-go theorem—not decidable type-checking.
- Every model of dGTT is a model of sGTT (including syntax).

Questions?