Canonicity for Indexed Inductive-Recursive Types

András Kovács

University of Gothenburg & Chalmers University of Technology

15th Jan 2026, POPL, Rennes

Inductive-Recursive Types

Small Agda example.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Code} : \mathsf{Set}\ \textbf{where} \\
&\quad \mathbb{N}' : \mathsf{Code} \\
&\quad \Pi' : (A : \mathsf{Code}) \to (\mathsf{El}\, A \to \mathsf{Code}) \to \mathsf{Code}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{El} : \mathsf{Code} \to \mathsf{Set} \\
&\mathsf{El}\, \mathbb{N}' \qquad\quad = \mathbb{N} \\
&\mathsf{El}\, (\Pi'\, A\, B) = (a : \mathsf{El}\, A) \to \mathsf{El}\, (B\, a)
\end{aligned}
$$

Early and informal use by Per Martin-Löf [ML75, ML84].

Formal syntax & semantics developed by Dybjer & Setzer [Dyb00, DS99, DS03, DS06].

Inductive-Recursive Types

Let's parameterize Code and El with $\boldsymbol{u} : \textbf{Set}$ and $\textbf{el} : \boldsymbol{u} \to \textbf{Set}$, and add extra rules.

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Code} : \mathsf{Set}\ \textbf{where} \\
\quad \ldots \\
\quad \mathsf{u}' \quad : \mathsf{Code} \\
\quad \mathsf{el}' \quad : u \to \mathsf{Code} \\
\\
\mathsf{El} : \mathsf{Code} \to \mathsf{Set} \\
\ldots \\
\mathsf{El}\,\mathsf{u}' \qquad = u \\
\mathsf{El}\,(\mathsf{el}'\ a) \ = el\ a
\end{array}
$$

Inductive-Recursive Types

Let's parameterize Code and El with $\boldsymbol{u} : \mathbf{Set}$ and $\mathbf{el} : \boldsymbol{u} \to \mathbf{Set}$, and add extra rules.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Code} : \mathsf{Set}\ \textbf{where} \\
&\quad \ldots \\
&\quad \mathsf{u}' \quad : \mathsf{Code} \\
&\quad \mathsf{el}' \quad : u \to \mathsf{Code} \\
\\
&\mathsf{El} : \mathsf{Code} \to \mathsf{Set} \\
&\ldots \\
&\mathsf{El}\,\mathsf{u}' \qquad = u \\
&\mathsf{El}\,(\mathsf{el}'\,a) = el\ a
\end{aligned}
$$

Let $\mathsf{FamSet} = \Sigma(A : \mathsf{Set}).(A \to \mathsf{Set})$.
Code and El yield a function $\mathbf{F} : \mathbf{FamSet} \to \mathbf{FamSet}$

We get countable universes by iteration:

$$
\begin{aligned}
&\mathsf{U} : \mathbb{N} \to \mathsf{FamSet} \\
&\mathsf{U}\,\mathsf{zero} \qquad = (\bot, \mathsf{exfalso}) \\
&\mathsf{U}\,(\mathsf{suc}\ n) = \mathsf{F}\,(\mathsf{U}\ n)
\end{aligned}
$$

More generally: any well-founded universe hierarchy can be defined by induction-recursion inside Set.

Inductive-Recursive Types

IR is convenient for doing metatheory of universe features.

Recent works on various notions of first-class universe levels, all using IR in Agda formalizations:

- Consistency by AK [Kov22]
- Canonicity by Chan & Weirich [CW25]
- Here at POPL: normalization by Danielsson, Favier & Kubánek [DFK26]

Inductive-Recursive Types

IR is convenient for doing metatheory of universe features.

Recent works on various notions of first-class universe levels, all using IR in Agda formalizations:

- Consistency by AK [Kov22]
- Canonicity by Chan & Weirich [CW25]
- Here at POPL: normalization by Danielsson, Favier & Kubánek [DFK26]

Some other uses: partial functions [BC01], generic programming [BDJ03, Die17].

Inductive-Recursive Types

IR is convenient for doing metatheory of universe features.

Recent works on various notions of first-class universe levels, all using IR in Agda formalizations:

- Consistency by AK [Kov22]
- Canonicity by Chan & Weirich [CW25]
- Here at POPL: normalization by Danielsson, Favier & Kubánek [DFK26]

Some other uses: partial functions [BC01], generic programming [BDJ03, Die17].

Impredicative Prop is technically stronger than IR, but

- We may prefer weaker predicative foundations.
- IR formalization may be more direct & convenient.

IR implementation & syntactic metatheory

IR has been supported in Agda for 15+ years. Also available in Idris-es.

Informally the implementation is quite straightforward.

IR implementation & syntactic metatheory

IR has been supported in Agda for 15+ years. Also available in Idris-es.

Informally the implementation is quite straightforward.

Desired syntactic properties:

1. *Canonicity:* every closed term can be computed to a constructor.
2. *Normalization:* every open term can be computed to a normal form.

IR implementation & syntactic metatheory

IR has been supported in Agda for 15+ years. Also available in Idris-es.

Informally the implementation is quite straightforward.

Desired syntactic properties:

1. *Canonicity:* every closed term can be computed to a constructor.
2. *Normalization:* every open term can be computed to a normal form.

We show **canonicity**.

The object theory

Quotient inductive-inductive definition with these sorts:[1]

$$\begin{aligned}
\mathsf{Con} &: \mathsf{Set} \\
\mathsf{Ty} &: \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Sub} &: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Tm} &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}
\end{aligned}$$

We have

- An explicit substitution calculus, given as a category with family.
- Countable universes in Russell style.
- Basic type formers: Π, Σ, $\top$, Bool, intensional identity.
- IR types.

[1]We omit universe levels here and in the rest of the talk

Canonicity: basic setup

Artin gluing is an established proof-relevant version of **logical predicate interpretation** which works nicely for our syntax.[2]

We define a family of functions by induction:

$$
\begin{aligned}
&-^{\circ} : (\Gamma : \mathsf{Con}) &&\to \mathsf{Sub}\,\bullet\,\Gamma \to \mathsf{Set} \\
&-^{\circ} : (A : \mathsf{Ty}\,\Gamma) &&\to \{\gamma : \mathsf{Sub}\,\bullet\,\Gamma\}(\gamma^{\circ} : \Gamma^{\circ}\,\gamma) \to \mathsf{Tm}\,\bullet\,A[\gamma] \to \mathsf{Set} \\
&-^{\circ} : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) &&\to \{\gamma : \mathsf{Sub}\,\bullet\,\Gamma\}(\gamma^{\circ} : \Gamma^{\circ}\,\gamma) \to \Delta^{\circ}\,(\sigma \circ \gamma) \\
&-^{\circ} : (t : \mathsf{Tm}\,\Gamma\,A) &&\to \{\gamma : \mathsf{Sub}\,\bullet\,\Gamma\}(\gamma^{\circ} : \Gamma^{\circ}\,\gamma) \to A^{\circ}\,\gamma^{\circ}\,t[\gamma]
\end{aligned}
$$

$-^{\circ}$ acts on every type and term former and respects definitional equality.

[2]See e.g. [Coq19, KHS19]

Canonicity for inductive types

The main task is to specify what it means for a closed term to be canonical.

Canonicity for inductive types

The main task is to specify what it means for a closed term to be canonical.

Example: natural numbers.

In the syntax, assume $\mathbb{N} : \mathsf{Ty}\,\Gamma$ with $\mathsf{zero} : \mathsf{Tm}\,\Gamma\,\mathbb{N}$ and $\mathsf{suc} : \mathsf{Tm}\,\Gamma\,\mathbb{N} \to \mathsf{Tm}\,\Gamma\,\mathbb{N}$.

Canonicity for inductive types

The main task is to specify what it means for a closed term to be canonical.

Example: natural numbers.

In the syntax, assume $\mathbb{N} : \mathsf{Ty}\,\Gamma$ with $\mathsf{zero} : \mathsf{Tm}\,\Gamma\,\mathbb{N}$ and $\mathsf{suc} : \mathsf{Tm}\,\Gamma\,\mathbb{N} \to \mathsf{Tm}\,\Gamma\,\mathbb{N}$.

Closed canonical terms are specified by a metatheoretic indexed inductive type.

$$\begin{aligned}
&\mathbf{data}\,\mathbb{N}^\circ : \mathsf{Tm}\bullet\mathbb{N} \to \mathsf{Set}\,\mathbf{where}\\
&\quad \mathsf{zero}^\circ : \mathbb{N}^\circ\,\mathsf{zero}\\
&\quad \mathsf{suc}^\circ \;\, : (n : \mathsf{Tm}\bullet\mathbb{N}) \to \mathbb{N}^\circ\,n \to \mathbb{N}^\circ\,(\mathsf{suc}\,n)
\end{aligned}$$

Canonicity for IR types

Example: the canonicity predicate for the basic Code type is an indexed IR type in the metatheory.

Canonicity for IR types

Example: the canonicity predicate for the basic Code type is an indexed IR type in the metatheory.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Code}^\circ : \mathsf{Tm}\bullet\mathsf{Code} \to \mathsf{Set}\\
&\quad \mathbb{N}'^\circ : \mathsf{Code}^\circ\,\mathbb{N}'\\
&\quad \Pi'^\circ : \{A : \mathsf{Tm}\bullet\mathsf{Code}\}(A^\circ : \mathsf{Code}^\circ\,A)\\
&\qquad\qquad \{B : \mathsf{Tm}\bullet(\mathsf{El}\,A \to \mathsf{Code})\}(B^\circ : \{a : \mathsf{Tm}\bullet(\mathsf{El}\,a)\} \to \mathsf{El}^\circ\,A^\circ\,a \to \mathsf{Code}^\circ\,(B\,a))\\
&\qquad\qquad \to \mathsf{Code}^\circ\,(\Pi'\,A\,B)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{El}^\circ : \{t : \mathsf{Tm}\bullet\mathsf{Code}\} \to \mathsf{Code}^\circ\,t \to (\mathsf{Tm}\bullet(\mathsf{El}\,t) \to \mathsf{Set})\\
&\mathsf{El}^\circ\,\mathbb{N}'^\circ \qquad\qquad\quad = \mathbb{N}^\circ\\
&\mathsf{El}^\circ\,(\Pi'^\circ\,A^\circ\,B^\circ) = \lambda\,f.\{a : \mathsf{Tm}\bullet(\mathsf{El}\,A)\}(a^\circ : \mathsf{El}^\circ\,A^\circ\,a) \to \mathsf{El}^\circ\,(B^\circ\,a^\circ)\,(f\,a)
\end{aligned}
$$

Canonicity for IR types

Example: the canonicity predicate for the basic Code type is an indexed IR type in the metatheory.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Code} : \mathsf{Set}\ \textbf{where}\\
&\quad \mathbb{N}' : \mathsf{Code}\\
&\quad \Pi' : (A : \mathsf{Code}) \to (\mathsf{El}\,A \to \mathsf{Code}) \to \mathsf{Code}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{El} : \mathsf{Code} \to \mathsf{Set}\\
&\mathsf{El}\,\mathbb{N}' \qquad\quad = \mathbb{N}\\
&\mathsf{El}\,(\Pi'\,A\,B) = (a : \mathsf{El}\,A) \to \mathsf{El}\,(B\,a)
\end{aligned}
$$

Canonicity for all IR types

Code is a *single* IR type, but we want to show canonicity for *all* IR types.

We need to

1. Specificy and assume all IR types in the object theory.
2. Generalize the canonicity construction across IR type descriptions.

Specifying IR types

Following Dybjer & Setzer: the type of **IR signatures** is an **internal** inductive type:

$$
\begin{aligned}
&\textbf{data}\,\mathsf{Sig}\,(O : \mathsf{Set}_1) : \mathsf{Set}_1\ \textbf{where}\\
&\quad \iota\ : O \to \mathsf{Sig}\,O\\
&\quad \sigma : (A : \mathsf{Set}) \to (A \to \mathsf{Sig}\,O) \to \mathsf{Sig}\,O\\
&\quad \delta : (A : \mathsf{Set}) \to ((A \to O) \to \mathsf{Sig}\,O) \to \mathsf{Sig}\,O
\end{aligned}
$$

Example: the signature for Code:

$$
\begin{aligned}
&\mathsf{SigCode} : \mathsf{Sig}\,\mathsf{Set}\\
&\mathsf{SigCode} :\equiv \sigma\,\mathsf{Bool}\,\lambda\,t.\,\textbf{case}\,t\,\textbf{of}\\
&\quad \mathsf{true}\ \to \iota\,\mathbb{N}\\
&\quad \mathsf{false} \to \delta\,\top\,\lambda\,ElA.\,\delta\,(ElA\,\mathsf{tt})\,\lambda\,ElB.\,\iota\,((x : ElA\,\mathsf{tt}) \to ElB\,x)
\end{aligned}
$$

For each signature, we assume type formation, term formation, elimination and computation rules.

General canonicity

Somewhat technical. I try to give computational intuition.

General canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of $-^{\circ}$ is type-directed closed evaluation.
- Canonicity witnesses correspond to runtime values.

General canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of $-^{\circ}$ is type-directed closed evaluation.
- Canonicity witnesses correspond to runtime values.

How to evaluate an **IR type constructor** $\mathrm{IR}\,S$?

1. Evaluate the signature S to get a value (canonicity witness).
2. Do induction on the value to compute a *metatheoretic indexed IR signature*.
3. We take the indexed IR type for this signature to get the *canonicity predicate* for $\mathrm{IR}\,S$, i.e. the corresponding type of runtime values.

General canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of $-^{\circ}$ is type-directed closed evaluation.
- Canonicity witnesses correspond to runtime values.

How to evaluate an **IR type constructor** $\mathrm{IR}\,S$?

1. Evaluate the signature S to get a value (canonicity witness).
2. Do induction on the value to compute a *metatheoretic indexed IR signature*.
3. We take the indexed IR type for this signature to get the *canonicity predicate* for $\mathrm{IR}\,S$, i.e. the corresponding type of runtime values.

To evaluate **IR term constructors** and **IR elimination**, we use the corresponding meta-level indexed IR constructors and eliminators.

What about indexed IR?

We only show canonicity for plain IR types, but also show that indexed IR types are constructible in MLTT+IR.

- We use “fording”, i.e. converting indices to parameters and propositional identities.
- Since we work in MLTT without UIP, we have to use a modest amount of HoTT.

What about indexed IR?

We only show canonicity for plain IR types, but also show that indexed IR types are constructible in MLTT+IR.

- We use "fording", i.e. converting indices to parameters and propositional identities.
- Since we work in MLTT without UIP, we have to use a modest amount of HoTT.

Constructed IIR types have the same definitional computation rules as native IIR types in Agda.

However, we can represent some IIR types which are

1. not representable in Agda
2. don't support definitional β-rules

This is because we have *first-class* signatures and Agda doesn't.

Overview

1. Indexed IR types can be constructed from IR types and basic type formers.

2. Canonicity:

	MLTT with ω universes with IR
is proved canonical by	ETT with ω universes with IR + Set_ω and $\mathrm{Set}_{\omega+1}$
is proved consistent by	ZF with ω Mahlo cardinals + two inaccessible cardinals above

Future work

Normalization for IR types?

- We want to work internally to presheaves over variable renamings.
- *To use IR reducibility predicates, we first have to show that IR types exists in presheaves.*

Future work

Normalization for IR types?

- We want to work internally to presheaves over variable renamings.
- *To use IR reducibility predicates, we first have to show that IR types exists in presheaves.*

Canonicity for other kinds of inductive types?

- Quotient induction and QW-types are probably OK.
- Induction-induction seems harder.

Future work

Normalization for IR types?

- We want to work internally to presheaves over variable renamings.
- *To use IR reducibility predicates, we first have to show that IR types exists in presheaves.*

Canonicity for other kinds of inductive types?

- Quotient induction and QW-types are probably OK.
- Induction-induction seems harder.

Generalizing the proof?

- To a *syntactic parametricity translation*.
- To a *model construction*.

Future work

Normalization for IR types?

- We want to work internally to presheaves over variable renamings.
- *To use IR reducibility predicates, we first have to show that IR types exists in presheaves.*

Canonicity for other kinds of inductive types?

- Quotient induction and QW-types are probably OK.
- Induction-induction seems harder.

Generalizing the proof?

- To a *syntactic parametricity translation*.
- To a *model construction*.

Thank you!

Ana Bove and Venanzio Capretta.
Nested general recursion and partiality in type theory.
In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2001.

Marcin Benke, Peter Dybjer, and Patrik Jansson.
Universes for generic programs and proofs in dependent type theory.
Nord. J. Comput., 10(4):265–289, 2003.

Thierry Coquand.
Canonicity and normalization for dependent type theory.
Theor. Comput. Sci., 777:184–191, 2019.

Jonathan Chan and Stephanie Weirich.
Bounded first-class universe levels in dependent type theory.
CoRR, abs/2502.20485, 2025.

Nils Anders Danielsson, Naïm Camille Favier, and Ondřej Kubánek.
Normalisation for first-class universe levels.
Proc. ACM Program. Lang., 10(POPL), January 2026.

Larry Diehl.
Fully Generic Programming over Closed Universes of Inductive-Recursive Types.

PhD thesis, Portland State University, 2017.

Peter Dybjer and Anton Setzer.
A finite axiomatization of inductive-recursive definitions.
In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 1999.

Peter Dybjer and Anton Setzer.
Induction-recursion and initial algebras.
Ann. Pure Appl. Log., 124(1-3):1–47, 2003.

Peter Dybjer and Anton Setzer.
Indexed induction-recursion.
J. Log. Algebraic Methods Program., 66(1):1–49, 2006.

Peter Dybjer.
A general formulation of simultaneous inductive-recursive definitions in type theory.
Journal of Symbolic Logic, 65:525–549, 2000.

Ambrus Kaposi, Simon Huber, and Christian Sattler.
Gluing for type theory.

In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

András Kovács.
Generalized universe hierarchies and first-class universe levels.
In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

Per Martin-Löf.
An intuitionistic theory of types: Predicative part.
In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 73–118. Elsevier, 1975.

Per Martin-Löf.
Intuitionistic type theory, volume 1 of *Studies in Proof Theory*.
Bibliopolis, 1984.