

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.

data Code : Set **where**

\mathbb{N}' : Code

$\Pi' : (A : \text{Code}) \rightarrow (\text{El } A \rightarrow \text{Code}) \rightarrow \text{Code}$

$\text{El} : \text{Code} \rightarrow \text{Set}$

$\text{El } \mathbb{N}' = \mathbb{N}$

$\text{El } (\Pi' A B) = (a : \text{El } A) \rightarrow \text{El } (B a)$

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 $u : \mathbf{Set}$ and $el : u \rightarrow \mathbf{Set}$, and add extra rules.

```
data Code : Set where
```

```
...
```

```
u'   : Code
```

```
el'   : u → Code
```

```
El : Code → Set
```

```
...
```

```
El u'      = u
```

```
El (el' a) = el a
```

Inductive-Recursive Types

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

data Code : Set **where**

...

$u' : \mathbf{Code}$

$el' : u \rightarrow \mathbf{Code}$

El : Code \rightarrow Set

...

El $u' = u$

El ($el' a$) = $el a$

Let FamSet = $\Sigma(A : \mathbf{Set}).(A \rightarrow \mathbf{Set})$.

Code and El yield a function **F** : FamSet \rightarrow FamSet

We get countable universes by iteration:

$U : \mathbb{N} \rightarrow \mathbf{FamSet}$

$U \text{ zero} = (\perp, \text{exfalse})$

$U (\text{suc } n) = F (U n)$

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 & metatheory

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

The implementation is quite straightforward. *Intuitively*, there's nothing that could go wrong.

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

The implementation is quite straightforward. *Intuitively*, there's nothing that could go wrong.

However, syntactic properties had not been shown:

- 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

The syntax is a quotient inductive-inductive type with these sorts:¹

$$\text{Con} : \text{Set}$$
$$\text{Ty} : \text{Con} \rightarrow \text{Set}$$
$$\text{Sub} : \text{Con} \rightarrow \text{Con} \rightarrow \text{Set}$$
$$\text{Tm} : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Set}$$

We have

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

¹We omit universe levels here and elsewhere

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

We define a family of functions by induction:

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

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

²See 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 $\mathbb{N} : \text{Ty } \Gamma$ with `zero` and `suc`.

Closed canonical terms of type \mathbb{N} are specified by an **indexed inductive type** in the metatheory.

```
data  $\mathbb{N}^\circ : \text{Tm} \bullet \mathbb{N} \rightarrow \text{Set}$  where  
   $\text{zero}^\circ : \mathbb{N}^\circ \text{ zero}$   
   $\text{suc}^\circ : (n : \text{Tm} \bullet \mathbb{N}) \rightarrow \mathbb{N}^\circ n \rightarrow \mathbb{N}^\circ (\text{suc } n)$ 
```

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.

data $\text{Code}^\circ : \text{Tm} \bullet \text{Code} \rightarrow \text{Set}$

$\mathbb{N}'^\circ : \text{Code}^\circ \mathbb{N}'$

$\Pi'^\circ : \{A : \text{Tm} \bullet \text{Code}\} (A^\circ : \text{Code}^\circ A)$

$\{B : \text{Tm} \bullet (\text{El } A \rightarrow \text{Code})\} (B^\circ : \{a : \text{Tm} \bullet (\text{El } a)\} \rightarrow \text{El}^\circ A^\circ a \rightarrow \text{Code}^\circ (B a))$
 $\rightarrow \text{Code}^\circ (\Pi' A B)$

$\text{El}^\circ : \{t : \text{Tm} \bullet \text{Code}\} \rightarrow \text{Code}^\circ t \rightarrow (\text{Tm} \bullet (\text{El } t) \rightarrow \text{Set})$

$\text{El}^\circ \mathbb{N}'^\circ = \mathbb{N}^\circ$

$\text{El}^\circ (\Pi'^\circ A^\circ B^\circ) = \lambda f. \{a : \text{Tm} \bullet (\text{El } A)\} (a^\circ : \text{El}^\circ A^\circ a) \rightarrow \text{El}^\circ (B^\circ a^\circ) (f a)$

Canonicity for IR types

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

data Code : Set **where**

\mathbb{N}' : Code

$\Pi' : (A : \text{Code}) \rightarrow (\text{El } A \rightarrow \text{Code}) \rightarrow \text{Code}$

$\text{El} : \text{Code} \rightarrow \text{Set}$

$\text{El } \mathbb{N}' = \mathbb{N}$

$\text{El } (\Pi' A B) = (a : \text{El } A) \rightarrow \text{El } (B a)$

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 Specify 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:

```
data Sig ( $O : \text{Set}_1$ ) :  $\text{Set}_1$  where  
   $\iota : O \rightarrow \text{Sig } O$   
   $\sigma : (A : \text{Set}) \rightarrow (A \rightarrow \text{Sig } O) \rightarrow \text{Sig } O$   
   $\delta : (A : \text{Set}) \rightarrow ((A \rightarrow O) \rightarrow \text{Sig } O) \rightarrow \text{Sig } O$ 
```

For each signature, we postulate type formation, term formation, elimination and computation rules, for the described IR type.

Specifying IR types

Let's write out the signature for Code.

data Code : Set **where**

$\mathbb{N}' : \text{Code}$

$\Pi' : (A : \text{Code}) \rightarrow (\text{El } A \rightarrow \text{Code}) \rightarrow \text{Code}$

$\text{El} : \text{Code} \rightarrow \text{Set}$

$\text{El } \mathbb{N}' = \mathbb{N}$

$\text{El } (\Pi' A B) = (a : \text{El } A) \rightarrow \text{El } (B a)$

SigCode : Sig Set

SigCode $\equiv \sigma \text{ Bool } \lambda t. \text{case } t \text{ of}$

 true $\rightarrow \iota \mathbb{N}$

 false $\rightarrow \delta \top \lambda EIA. \delta (EIA \text{ tt}) \lambda EIB. \iota ((x : EIA \text{ tt}) \rightarrow EIB x)$

Generic canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of the proof is *type-directed closed evaluation*.
- Canonicity witnesses correspond to *runtime values*.

Generic canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of the proof is *type-directed closed evaluation*.
- Canonicity witnesses correspond to *runtime values*.

How to evaluate an **IR type constructor** $IR\ S$?

- 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 $IR\ S$, i.e. the corresponding type of runtime values.

Generic canonicity

Somewhat technical. I try to give computational intuition.

- The computational content of the proof is *type-directed closed evaluation*.
- Canonicity witnesses correspond to *runtime values*.

How to evaluate an **IR type constructor** $IR\ S$?

- 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 $IR\ S$, 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 $\text{MLTT} + \text{IR}$.

- 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 $\text{MLTT} + \text{IR}$.

- 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.

1. Indexed IR types can be constructed from IR types and basic type formers (omitted in this talk).

2. Canonicity:

MLTT with ω universes with IR

is proved canonical by ETT with ω universes with IR, plus Set_ω and $\text{Set}_{\omega+1}$

is proved consistent by ZF with ω Mahlo cardinals, plus 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.

Gluings 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.