Canonicity for Indexed Inductive-Recursive Types

ANONYMOUS AUTHOR(S)

We prove canonicity for a Martin-Löf type theory that supports a countable universe hierarchy where each universe supports indexed inductive-recursive (IIR) types. We proceed in two steps. First, we construct IIR types from inductive-recursive (IR) types and intensional identity types, in order to simplify the subsequent canonicity proof. The constructed IIR types support the same definitional computation rules that are available in Agda's native IIR implementation. Second, we give a canonicity proof for IR types, building on the well-known method of Artin gluing. The main idea is to encode the canonicity predicate for each IR type using a metatheoretic IIR type.

ACM Reference Format:
Anonymous Author(s). 2018. Canonicity for Indexed Inductive-Recursive Types. *J. ACM* 37, 4, Article 111 (August 2018), 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Induction-recursion (IR) was first used by Martin-Löf in an informal way [?], then made formal by Dybjer and Setzer [?], who also developed set-theoretic and categorical semantics [?]. A common application of IR is to define custom universe hierarchies inside a type theory. In the proof assistant Agda, we can use IR to define a universe that is closed under our choice of type formers:

$$
\begin{array}{l}
\textbf{mutual}\\
\quad \textbf{data}\,\mathsf{Code} : \mathsf{Set}_0\ \textbf{where}\\
\qquad \mathsf{Nat}' : \mathsf{U}\\
\qquad \Pi' \quad : (A : \mathsf{Code}) \to (\mathsf{El}\,A \to \mathsf{Code}) \to \mathsf{Code}\\
\\
\quad \mathsf{El} : \mathsf{Code} \to \mathsf{Set}_0\\
\quad \mathsf{El}\,\mathsf{Nat}' \qquad = \mathsf{Nat}\\
\quad \mathsf{El}\,(\Pi'\,A\,B) = (a : \mathsf{El}\,A) \to \mathsf{El}\,(B\,a)
\end{array}
$$

Here, Code is a type of codes of types which behaves as a custom Tarski-style universe. This universe, unlike the ambient Set_0 universe, supports an induction principle and can be used to define type-generic functions. *Indexed induction-recursion* (IIR) additionally allows indexing Code over some type, which lets us define inductive-recursive predicates [?].

One application of IR has been to develop semantics for object theories that support universe hierarchies. IR has been used in normalization proofs [?], in modeling first-class universe levels [?] and proving canonicity for them [?], and in characterizing domains of partial functions [?]. Another application is to do generic programming over universes of type descriptions [?] or data layout descriptions [?].

IIR has been supported in Agda 2 since the early days of the system [?], and it is also available in Idris 1 and Idris 2 [?]. In these systems, IR has been implemented in the "obvious" way, supporting

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

closed program execution in compiler backends and normalization during type checking, but without any formal justification.

Our **main contribution** is to **show canonicity** for a Martin-Löf type theory that supports a countable universe hierarchy, where each universe supports indexed inductive-recursive types. Canonicity means that every closed term is definitionally equal to a canonical term. Canonical terms are built only from constructors; for instance, a canonical natural number term is a numeral. Hence, canonicity justifies evaluation for closed terms. The outline of our development is as follows.

(1) In Section **??** we specify what it means to support IR and IIR, using Dybjer and Setzer's rules with minor modifications [?]. We use first-class signatures, meaning that descriptions of (I)IR types are given as ordinary inductive types internally.
(2) In Section **??** we construct IIR types from IR types and other basic type formers. This allows us to only consider IR types in the subsequent canonicity proof, which is a significant simplification. In the construction of IIR types, we lose some definitional equalities when IIR signatures are neutral, but we still get strict computation for canonical signatures. This matches the computational behavior of Agda and Idris, where IIR signatures are second-class and necessarily canonical. We formalize the construction in Agda.
(3) In Section **??**, we give a proof-relevant logical predicate interpretation of the type theory, from which canonicity follows. We follow the well-known method of Artin gluing [?]. The main challenge here is to give a logical predicate interpretation of IR types. We do this by using IIR in the metatheory: from each object-theoretic signature we compute a metatheoretic IIR signature which encodes the canonicity predicate for the corresponding IR type. We formalize the predicate interpretation of IR types in Agda, using a shallow embedding of the syntax of the object theory. Hence, there is a gap between the Agda version and the fully formal construction, but we argue that it is a modest gap.

2 Specification for (I)IR types

In this section we describe the object type theory, focusing on the specification of IR and IIR types. We do not yet go into the formal details; instead, we shall mostly work with internal definitions in an Agda-like syntax. In Section **??** we will give a rigorous specification that is based on categories-with-families.

Basic type formers. We have a countable hierarchy of Russell-style universes, written as U_i, where i is an external natural number. We have $\mathsf{U}_i : \mathsf{U}_{i+1}$. We have Π-types as $(x : A) \to B\,x$, which has type $\mathsf{U}_{\max(i,\,j)}$ when $A : \mathsf{U}_i$ and $B : A \to \mathsf{U}_j$. We also have a lifting operation $\mathsf{Lift}\; i\; j : \mathsf{U}_i \to \mathsf{U}_j$ together with $\uparrow : A \to \mathsf{Lift}\; i\; j\; A$ and $\downarrow : \mathsf{Lift}\; i\; j\; A \to A$ such that $\uparrow$ and $\downarrow$ are definitional inverses. We have intensional identity types, as $t = u : \mathsf{U}_i$ for $t : A : \mathsf{U}_i$. We have $\top : \mathsf{U}_0$ for the unit type with the unique inhabitant tt. We have Σ-types, written as $(x : A) \times B\,x$, where the type also lands in $\mathsf{U}_{\max(i,\,j)}$.

2.1 IR types

The object theory additionally supports inductive-recursive types. On a high level, the specification consists of the following.

(1) A type of signatures. Each signature describes an IR type. Since signatures are first-class internal values, we also internally define some *semantics of signatures*, interpreting them as endofunctors on slice categories. This interpretation is required in the rest of the specification.
(2) Rules for type formation, term formation and the recursive function, with a computation rule for the recursive function.

(3) The induction principle with a β rule.

2.1.1 IR signatures. Signatures are parameterized by the following data:

- The level i is the size of the IR type that is being specified.
- The level j is the size of the recursive output type.
- $O : \mathsf{U}_j$ is the output type.

IR signatures are specified by the following inductive type. We only mark i and O as parameters to Sig, since j is inferable from O.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Sig}_i\ O : \mathsf{U}_{\max(i+1,\,o)}\\
&\quad \iota\ : O \to \mathsf{Sig}\\
&\quad \sigma : (A : \mathsf{U}_i) \to (A \to \mathsf{Sig}) \to \mathsf{Sig}\\
&\quad \delta\ : (A : \mathsf{U}_i) \to ((A \to O) \to \mathsf{Sig}) \to \mathsf{Sig}
\end{aligned}
$$

Formally, we can view Sig in two ways. We can either view it as just a particular family of W-types, or as an inductive type that is primitively part of the object theory. The choice is not important, since inductive families are constructible from W-types [Hugunin 2020].

Example 2.1. We can reproduce the Agda example from Figure [?]. First, we need an enumeration type to represent the constructor labels of Code. We assume this as $\mathsf{Tag} : \mathsf{U}_0$ with constructors Nat' and Π', and we use an informal case splitting operation for it. We also assume $\mathsf{Nat} : \mathsf{U}_0$ for natural numbers and a right-associative $-\$-$ operator for function application.

$$
\begin{aligned}
&S : \mathsf{Sig}_0\ \mathsf{U}_0\\
&S := \sigma\,\mathsf{Tag}\ \$\ \lambda\, t.\ \textbf{case}\ t\ \textbf{of}\\
&\quad \mathsf{Nat}' \to \iota\,\mathsf{Nat}\\
&\quad \Pi' \quad \to \delta\,\top\ \$\ \lambda\, ElA.\ \delta\,(ElA\ \mathsf{tt})\ \$\ \lambda\, ElB.\ \iota\,((x : ElA\ \mathsf{tt}) \to ElB\ x)
\end{aligned}
$$

First, we introduce a choice between two constructors by $\sigma\,\mathsf{Tag}$. In the Nat' branch, we specify that the recursive function maps the constructor to Nat. In the Π' branch, we first introduce a single inductive constructor field by $\delta\,\top$, where $\top$ represents the number of introduced fields. The naming of the freshly bound variable ElA is meant to suggest that it represent the recursive function's output for the inductive field. It has type $\top \to \mathsf{U}_0$. Next, we introduce $(ElA\ \mathsf{tt})$-many inductive fields, and bind $ElB : ElA\ \mathsf{tt} \to \mathsf{U}_0$ to represent the corresponding recursive output. Finally, $\iota\,((x : ElA\ \mathsf{tt}) \to ElB\ x)$ specifies the output of the recursive function for a Π' constructor.

Comparison to prior specifications.

(1) Dybjer and Setzer's finite axiomatization [?]. There are two main differences. First, Dybjer and Setzer factor out the O field from the ι constructor into a separate annotation on signatures, called "Arg". However, the bundle of a signature with an Arg over it is clearly equivalent to our version. Second, we have a countable universe hierarchy, while Dybjer and Setzer use a logical framework presentation with only three universes, set, stype and type, where set contains the inductively specified type, stype contains the non-inductive constructor arguments and type contains the recursive output type and the type of signatures.
(2) The specification of small induction-recursion by Hancock et al. [?]. Small IR mentions only two universes, Set and Set_1, where the inductive type and the recursive output type are both in Set, and Set_1 hosts the type of signatures. Small IR signatures store an O value in the ι constructor, just like out signatures.

3 TODO

- Small IIR paper: reduction of small IIR to IR + identity, similar to mine
- Bove-Capretta: IIR is used to model domains of partial functions.
- Canonicity metatheory: Loic, Anton say it looks OK

a

References

Jasper Hugunin. 2020. Why Not W?. In *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy (LIPIcs, Vol. 188)*, Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:9. doi:10.4230/LIPICS.TYPES.2020.8