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), 7 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:

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

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 build on 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 use Agda-style implicit function types for convenience, as $\{x : A\} \to B\,x$, to mark that a function argument should be inferred from context. 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 $t : A : \mathsf{U}_i$, with a transport operation $\mathsf{tr} : \{A : \mathsf{U}_i\}(P : A \to \mathsf{U}_j)\{x\,y : A\} \to x = y \to P\,x \to P\,y$. 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 $\mathsf{U}_{\max(i,\,j)}$. Finally, we have binary sum types, written as $A + B$ with constructors inj_1 and inj_2.

definitions, defn equality

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. Also, we internally define some functions on signatures which are required in the specification of other rules.
(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.

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

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.

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

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 $(\mathit{ElA}\,\mathsf{tt})$-many inductive fields, and bind $\mathit{ElB} : \mathit{ElA}\,\mathsf{tt} \to \mathsf{U}_0$ to represent the corresponding recursive output. Finally, $\iota\,((x : \mathit{ElA}\,\mathsf{tt}) \to \mathit{ElB}\,x)$ specifies the output of the recursive function for a Π' constructor.

Our signatures are identical to Dybjer and Setzer's [?], except for one difference. We have countable universe levels, 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.

2.1.2 Type and term formation. In this section we also follow Dybjer and Setzer [?], with minor differences of notation, and also accounting for the refinement of universe levels.

First, assuming i and $O : \mathsf{U}_j$, a signature $S : \mathsf{Sig}_i\, O$ can be interpreted as a function from $(A : \mathsf{U}_i) \times (A \to O)$ to $(A : \mathsf{U}_i) \times (A \to O)$. This can be extended to an endofunctor on the slice category U_i/O, but in the following we only need the action on objects. We split this action to two functions, to aid readability:

$$
\begin{aligned}
&-_0 : \mathsf{Sig}_i\, O \to (ir : \mathsf{U}_i) \to (ir \to O) \to \mathsf{U}_i \\
&S_0\,(\iota\, o) \quad\;\; ir\, el \;:\equiv \mathsf{Lift}\,\top \\
&S_0\,(\sigma\, A\, S)\; ir\, el \;:\equiv (a : A) \times (S\, a)_0\; ir\, el \\
&S_0\,(\delta\, A\, S)\; ir\, el \;:\equiv (f : A \to ir) \times (S\,(el \circ f))_0\; ir\, el
\end{aligned}
$$

$$
\begin{aligned}
&-_1 : (S : \mathsf{Sig}_i\, O) \to S_0\; ir\, el \to O \\
&S_1\,(\iota\, o) \quad\;\; x \quad\;\; :\equiv o \\
&S_1\,(\sigma\, A\, S)\,(a,\, x) :\equiv (S\, i)_1\, x \\
&S_1\,(\delta\, A\, S)\,(f,\, x) :\equiv (S\,(el \circ f))_1\, x
\end{aligned}
$$

Although we use Agda-like pattern matching notation above, these functions are formally defined by the elimination principle of Sig. Also note the quantification of the i and j universe levels. The object theory does not support universe polymorphism, so this quantification is understood to happen in the metatheory. The introduction rules are the following.

$$
\begin{aligned}
&\mathsf{IR} \qquad\;\; : (S : \mathsf{Sig}_i\, O) \to \mathsf{U}_i \\
&\mathsf{El} \qquad\;\; : \mathsf{IR}\, S \to O \\
&\mathsf{intro} \quad\;\, : S_0\,(\mathsf{IR}\, S)\, \mathsf{El} \to \mathsf{IR}\, S \\
&\mathsf{El}\text{–}\mathsf{intro} : \mathsf{El}\,(\mathsf{intro}\, x) \equiv S_1\, x
\end{aligned}
$$

Above, we leave some rule arguments implicit, like S in El, intro and El–intro. The rule El–intro specifies a definitional equality. Note that these rules are not internal definitions but part of the specification of the object theory. Hence, they are also assumed to be stable under object-theoretic substitution, formally speaking. On a high level, the introduction rules express the existence of an S-algebra where we view S as an endofunctor on U_i/O.

2.1.3 Elimination. Here we follow the specification in [?]. We assume another universe level k that specifies the size of the type into which we eliminate. We define two additional functions on signatures:

$$
\begin{aligned}
&-_{\mathsf{IH}} : (S : \mathsf{Sig}_i\, O)(P : ir \to \mathsf{U}_k) \to S_0\; ir\, el \to \mathsf{U}_{\max(i,\,k)} \\
&(\iota\, o)_{\mathsf{IH}} \quad\; P\, x \quad\;\; :\equiv \mathsf{Lift}\,\top \\
&(\sigma\, A\, S)_{\mathsf{IH}}\, P\,(a,\, x) :\equiv (S\, a)_{\mathsf{IH}}\, P\, x \\
&(\delta\, A\, S)_{\mathsf{IH}}\, P\,(f,\, x) :\equiv ((a : A) \to P\,(f\, a)) \times (S\,(el \circ f))_{\mathsf{IH}}\, P\, x
\end{aligned}
$$

$$
\begin{aligned}
&-_{\mathsf{map}} : (S : \mathsf{Sig}_i\, O)(P : ir \to \mathsf{U}_k) \to ((x : ir) \to P\, x) \to (x : S_0\; ir\, el) \to S_{\mathsf{IH}}\, P\, x \\
&(\iota\, o)_{\mathsf{map}} \quad\; P\, h\, x \quad\;\; :\equiv \uparrow\!\mathsf{tt} \\
&(\sigma\, A\, S)_{\mathsf{map}}\, P\, h\,(a,\, x) :\equiv (S\, a)_{\mathsf{map}}\, P\, h\, x \\
&(\delta\, A\, S)_{\mathsf{map}}\, P\, h\,(f,\, x) :\equiv (h \circ f,\; (S\,(el \circ f))_{\mathsf{map}}\, P\, h\, x)
\end{aligned}
$$

$-_{\mathsf{IH}}$ stands for “induction hypothesis”: it specifies having a witness of a predicate P for each inductive field in a value of $S_0\; ir\, el$. S_{map} maps over $S_0\; ir\, el$, applying the section $h : (x : ir) \to P\, x$

to each inductive field. Elimination is specified as follows.

$$\begin{aligned}
&\mathsf{elim} \quad : (P : \mathsf{IR}\,S \to \mathsf{U}_k) \to ((x : S_0\,(\mathsf{IR}\,S)\,\mathsf{El}) \to S_{\mathsf{IH}}\,P\,x \to P\,(\mathsf{intro}\,x)) \to (x : \mathsf{IR}\,S) \to P\,x \\
&\mathsf{elim}{-}\beta : \mathsf{elim}\,P\,f\,(\mathsf{intro}\,x) \equiv f\,x\,(S_{\mathsf{map}}\,P\,(\mathsf{elim}\,P\,f)\,x)
\end{aligned}$$

If we have function extensionality, this specification of elimination can be shown to be equivalent to the initiality of $(\mathsf{IR}\,S,\ \mathsf{El})$ as an S-algebra [?].

2.2 IIR Types

We extend Dybjer and Setzer's general IIR signatures [?] with refined universe levels. Since IIR is quite similar to IR, we give a more terse presentation of the rules and definitions in the following.

2.2.1 Signatures. We assume levels i, j, k, an indexing type $I : \mathsf{U}_k$ and a type family for the recursive output as $O : I \to \mathsf{U}_j$. Signatures are as follows.

$$\begin{aligned}
&\mathbf{data}\ \mathsf{Sig}_i\,I\,O : \mathsf{U}_{\max(i+1,\,j,\,k)}\ \mathbf{where} \\
&\quad \iota\ : (i : I) \to O\,i \to \mathsf{Sig}_i\,I\,O \\
&\quad \sigma : (A : \mathsf{U}_i) \to (A \to \mathsf{Sig}_i\,I\,O) \to \mathsf{Sig}_i\,I\,O \\
&\quad \delta\ : (A : \mathsf{U}_i)(ix : A \to I) \to (((a : A) \to O\,(ix\,a)) \to \mathsf{Sig}_i\,I\,O) \to \mathsf{Sig}_i\,I\,O
\end{aligned}$$

Example 2.2. We reproduce length-indexed vectors as an IIR type. We assume $A : \mathsf{U}_0$ for a type of elements in the vector, and a type $\mathsf{Tag} : \mathsf{U}_0$ with inhabitants Nil' and Cons'.

$$\begin{aligned}
&S : \mathsf{Sig}_0\,\mathsf{Nat}\,(\lambda\,_.\,\top) \\
&S :\equiv \sigma\,\mathsf{Tag}\,\$\,\lambda\,t.\,\mathbf{case}\ t\ \mathbf{of} \\
&\quad \mathsf{Nil}' \quad\ \to \iota\,\mathsf{zero}\,\mathsf{tt} \\
&\quad \mathsf{Cons}' \to \sigma\,\mathsf{Nat}\,\$\,\lambda\,n.\,\sigma\,A\,\$\,\lambda\,_.\,\delta\,\top\,(\lambda\,_.\,n)\,\$\,\lambda\,_.\,\iota\,(\mathsf{suc}\,n)\,\mathsf{tt}
\end{aligned}$$

We set O to be constant $\top$ because vectors do not have an associated recursive function. In the Nil' case, we simply set the constructor index to zero. In the Cons' case, we introduce a non-inductive field, binding n for the length of the tail of the vector. Then, when we introduce the inductive field using δ, we use $(\lambda\,\_.\,n)$ to specify that the length of the (single) inductive field is indeed n. Finally, the length of the Cons' constructor is $\mathsf{suc}\,n$.

2.2.2 Type and term formation. The signature actions $-_0$ and $-_1$ are similar to before:

$$\begin{aligned}
&-_0 : \mathsf{Sig}_i\,I\,O \to (ir : I \to \mathsf{U}_{\max(i,k)}) \to (\{i : I\} \to ir\,i \to O\,i) \to I \to \mathsf{U}_{\max(i,k)} \\
&S_0\,(\iota\,i'\,o) \quad\ \ ir\,el\,i :\equiv \mathsf{Lift}\,(i' = i) \\
&S_0\,(\sigma\,A\,S) \quad ir\,el\,i :\equiv (a : A) \times (S\,a)_0\,ir\,el\,i \\
&S_0\,(\delta\,A\,ix\,S)\,ir\,el\,i :\equiv (f : (a : A) \to ir\,(ix\,a)) \times (S\,(el \circ f))_0\,ir\,el\,i
\end{aligned}$$

$$\begin{aligned}
&-_1 : (S : \mathsf{Sig}_i\,I\,O) \to S_0\,ir\,el\,i \to O\,i \\
&S_1\,(\iota\,i'\,o) \quad (\uparrow x) \ :\equiv \mathsf{tr}\,O\,x\,o \\
&S_1\,(\sigma\,A\,S)\ (a,\ x) :\equiv (S\,i)_1\,x \\
&S_1\,(\delta\,A\,S)\ (f,\ x) :\equiv (S\,(el \circ f))_1\,x
\end{aligned}$$

Note the transport in $\mathsf{tr}\,O\,x\,o$: this is necessary, since o has type $O\,i'$ while the required type is $O\,i$. The type and term formation rules are the following.

$$\begin{aligned}
&\mathsf{IIR} &&: (S : \mathsf{Sig}_i\, I\, O) \to I \to \mathsf{U}_{\max(i,k)}\\
&\mathsf{El} &&: \mathsf{IIR}\, S\, i \to O\, i\\
&\mathsf{intro} &&: S_0\, (\mathsf{IIR}\, S)\, \mathsf{El}\, i \to \mathsf{IIR}\, S\, i\\
&\mathsf{El\text{-}intro} &&: \mathsf{El}\, (\mathsf{intro}\, x) \equiv S_1\, x
\end{aligned}$$

2.2.3 Elimination. $-_{\mathsf{IH}}$, $-_{\mathsf{map}}$ and elimination are as follows. We assume a level l for the target type of elimination.

$$\begin{aligned}
&-_{\mathsf{IH}} : (S : \mathsf{Sig}_i\, I\, O)(P : \{i : I\} \to ir\, i \to \mathsf{U}_l) \to S_0\, ir\, el\, i \to \mathsf{U}_{\max(i,l)}\\
&(\iota\, i\, o)_{\mathsf{IH}} \quad P\, x \quad :\equiv \mathsf{Lift}\, \top\\
&(\sigma\, A\, S)_{\mathsf{IH}} \quad P\, (a,\, x) :\equiv (S\, a)_{\mathsf{IH}}\, P\, x\\
&(\delta\, A\, ix\, S)_{\mathsf{IH}}\, P\, (f,\, x) :\equiv ((a : A) \to P\, (f\, a)) \times (S\, (el \circ f))_{\mathsf{IH}}\, P\, x
\end{aligned}$$

$$\begin{aligned}
&-_{\mathsf{map}} : (S : \mathsf{Sig}_i\, I\, O)(P : \{i : I\} \to ir\, i \to \mathsf{U}_l)\\
&\qquad \to (\{i : I\}(x : ir\, i) \to P\, x) \to (x : S_0\, ir\, el\, i) \to S_{\mathsf{IH}}\, P\, x\\
&(\iota\, o)_{\mathsf{map}} \quad P\, h\, x \quad :\equiv {\uparrow}\mathsf{tt}\\
&(\sigma\, A\, S)_{\mathsf{map}}\, P\, h\, (a,\, x) :\equiv (S\, a)_{\mathsf{map}}\, P\, h\, x\\
&(\delta\, A\, S)_{\mathsf{map}}\, P\, h\, (f,\, x) :\equiv (h \circ f,\, (S\, (el \circ f))_{\mathsf{map}}\, P\, h\, x)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{elim} \quad : (P : \{i : I\} \to \mathsf{IIR}\, S\, i \to \mathsf{U}_l) \to (\{i : I\}(x : S_0\, (\mathsf{IIR}\, S)\, \mathsf{El}\, i) \to S_{\mathsf{IH}}\, P\, x \to P\, (\mathsf{intro}\, x))\\
&\qquad \to (x : \mathsf{IIR}\, S\, i) \to P\, x\\
&\mathsf{elim}\text{-}\beta : \mathsf{elim}\, P\, f\, (\mathsf{intro}\, x) \equiv f\, x\, (S_{\mathsf{map}}\, P\, (\mathsf{elim}\, P\, f)\, x)
\end{aligned}$$

Examples, fording (if there's space)

3 Construction of IIR Types

small IR reference

We proceed to construct IIR types from IR types and other basic type formers. We assume i, j, k, $I : \mathsf{U}_k$ and $O : I \to \mathsf{U}_j$, and also assume definitions for IIR signatures and the four operations ($-_0$, $-_1$, $-_{\mathsf{IH}}$, $-_{\mathsf{map}}$). The task is to define IR, El, elim and elim–β. We use some abbreviations in the following:

- $\mathsf{Sig}_{\mathsf{IIR}}$ abbreviates the IIR signature type $\mathsf{Sig}_i\, I\, O$.
- $\mathsf{Sig}_{\mathsf{IR}}$ abbreviates the IR signature type $\mathsf{Sig}_{\max(i,k)}\, ((i : I) \times O\, i)$.

In a nutshell, the main idea in this section is to represent IIR signatures as IR signatures together with a well-indexing predicate on algebras. First, we define the encoding function for signatures:

$$\begin{aligned}
&\lfloor - \rfloor : \mathsf{Sig}_{\mathsf{IIR}} \to \mathsf{Sig}_{\mathsf{IR}}\\
&\lfloor \iota\, i\, o \rfloor \quad :\equiv \iota\, (i,\, o)\\
&\lfloor \sigma\, A\, S \rfloor \quad :\equiv \sigma\, (\mathsf{Lift}\, A)\, (\lambda\, a.\, \lfloor S\, {\downarrow} a \rfloor)\\
&\lfloor \delta\, A\, ix\, S \rfloor :\equiv \delta\, (\mathsf{Lift}\, A)\, \$\, \lambda\, f.\\
&\qquad \sigma\, ((a : A) \to \mathsf{fst}\, (f\, ({\uparrow} a)) = ix\, a)\, \$\, \lambda\, p.\\
&\qquad \lfloor S\, (\lambda\, a.\, \mathsf{tr}\, O\, (p\, a)\, (\mathsf{snd}\, (f\, ({\uparrow} a)))) \rfloor
\end{aligned}$$

There are two points of interest. First, the encoded IR signature has the recursive output type $(i : I) \times O\,i$, which lets us interpret $\iota\,i\,o$ as $\iota\,(i,\,o)$. Second, in the interpretation of δ, we already need to enforce well-indexing for inductive fields, or else we cannot recursively proceed with the translation. We solve this by adding an *extra field* in the output signature, which contains a well-indexing witness of type $((a : A) \to \mathsf{fst}\,(f\,(\uparrow a)) = ix\,a)$. This lets us continue the translation for S, by fixing up the return type of f by a transport.

Note on related work. Hancock et al. described the same translation from small IIR signatures to small IR signatures [?]. However, they did not present anything more about the reduction of IIR types to IR types.

We can already define the IIR and El rules for IIR types. Since the encoding of signatures already ensures that inductive fields in algebras are well-indexed, it only remains to ensure that the "top-level" index matches the externally supplied index.

$$\begin{aligned} &\mathsf{IIR} : \mathsf{Sig}_{\mathsf{IIR}} \to I \to \mathsf{U}_{\max(i,\,k)} && \mathsf{El} : \mathsf{IIR}\,S\,i \to O\,i \\ &\mathsf{IIR}\,S\,i :\equiv (x : \mathsf{IR}\,\lfloor S \rfloor) \times \mathsf{fst}\,(\mathsf{El}\,x) = i && \mathsf{El}\,(x,\,p) :\equiv \mathsf{tr}\,O\,p\,(\mathsf{snd}\,(\mathsf{El}\,x)) \end{aligned}$$

The following shorthand describes the data that we get when we peel off an intro from an $\mathsf{IR}\,i$ value:

$$\begin{aligned} -_{\lfloor 0 \rfloor} &: (S : \mathsf{Sig}_{\mathsf{IR}}) \to I \to \mathsf{U}_{\max(i,\,k)} \\ S_{\lfloor 0 \rfloor}\,i &:\equiv (x : \lfloor S \rfloor_0\,(\mathsf{IR}\,S)\,\mathsf{El}) \times \mathsf{fst}\,(S_1\,x) = i \end{aligned}$$

Now, we can show that $S_{\lfloor 0 \rfloor}\,i$ is equivalent to $S_0\,(\mathsf{IIR}\,S)\,\mathsf{El}\,i$, by induction on S. The induction on S is straightforward and we omit it here. We name the components of the equivalence as follows:

$$\begin{aligned} \overrightarrow{S_0} &: S_0\,(\mathsf{IIR}\,S)\,\mathsf{El}\,i \to S_{\lfloor 0 \rfloor}\,i \\ \overleftarrow{S_0} &: S_{\lfloor 0 \rfloor}\,i \to S_0\,(\mathsf{IIR}\,S)\,\mathsf{El}\,i \\ \eta &: \forall x.\,\overleftarrow{S_0}\,(\overrightarrow{S_0}\,x) = x \\ \epsilon &: \forall x.\,\overrightarrow{S_0}\,(\overleftarrow{S_0}\,x) = x \\ \tau &: \forall x.\,\mathsf{ap}\,\overrightarrow{S_0}\,(\eta\,x) = \epsilon\,(\overleftarrow{S_0}\,x) \end{aligned}$$

This is a half adjoint equivalence, as known from homotopy type theory [?]. The half adjoint coherence witness τ will be necessary at a later step for rearranging some transports. In the Agda formalization, we compute all components of equivalence by induction on S, although τ could be also generically recovered from the other four components [?].

4 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 says it looks OK
- Rename wrap to intro in Agda

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