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 other basic type formers, 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), 16 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 [?]. Other applications are in characterizing domains of partial functions [?] and in 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 the same computation rules that are available for Agda and Idris' IIR types. 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 more rigorous specification that is based on categories-with-families.

2.1 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 have Σ-types, written as $(x : A) \times B\,x$, which also has type $\mathsf{U}_{\max(i,\,j)}$. We have the unit type $\top : \mathsf{U}_i$ with unique inhabitant tt. We have $\mathsf{Bool} : \mathsf{U}_0$ for Booleans. We have intensional identity types, as $t = u : \mathsf{U}_i$ for $t : A : \mathsf{U}_i$. We define (by identity elimination) 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 derive some other type formers below.

- We define a universe lifting operation $\mathsf{Lift} : \mathsf{U}_i \to \mathsf{U}_{\max(i,\,j)}$ such that $\mathsf{Lift}\,A$ is definitionally isomorphic to A, by setting $\mathsf{Lift}\,A$ to $A \times \top_j$. We write the wrapping operation as $\uparrow : A \to \mathsf{Lift}\,A$ with inverse $\downarrow$.
- We define the empty type $\bot : \mathsf{U}_0$ as $\mathsf{true} = \mathsf{false}$.
- We can define finite sum types from $\bot$, Σ and Bool. These are useful as "constructor tags" in inductive types.

We write $-\equiv-$ for definitional equality, and give definitions using $:\equiv$.

2.2 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.

2.2.1 *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: it is either a primitive inductive family [?] or it is defined as a W-type [Hugunin 2020]. The choice is not crucial, but in this paper we treat Sig as a native inductive type, in order to avoid encoding overheads.

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.

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

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.

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.

2.2.2 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:

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

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

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.

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

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. 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.

2.2.3 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:

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

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

$-_{\mathsf{IH}}$ 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.

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

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 [?].

2.3 IIR Types

In IIR signatures, the sole deviation from Dybjer and Setzer is again our use of countable universe levels [?]. Since IIR is quite similar to IR, we present the rules without much commentary.

2.3.1 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.

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

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'.

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

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 $\mathsf{suc}\,n$.

2.3.2 Type and term formation. $-_0$ and $-_1$ are similar to before:

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

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

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.

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

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

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

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

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

3 Construction of IIR Types

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 an encoding function for signatures:

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

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 prior 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.

3.1 Type and Term Formers

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

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

The following shorthand describes the data that we get when we peel off an intro from an $\mathsf{IIR}\,S\,i$ 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 is straightforward and we omit it here. We name the components of the equivalence as follows:[1]

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

This is a half adjoint equivalence [?]. The half adjoint coherence witness τ will be necessary shortly for rearranging some transports. Next, we show that the two $-_1$ operations are the same, modulo the previous equivalence, again by induction on IIR signatures.

$$-_{\lfloor 1 \rfloor} : (S : \mathsf{Sig}_{\mathsf{IIR}})(x : S_0\,(\mathsf{IIR}\,S)\,\mathsf{El}\,i) \to \mathsf{tr}\,O\,(\mathsf{snd}\,(\overrightarrow{S_0}\,x))\,(\mathsf{snd}\,(\lfloor S \rfloor_1\,(\mathsf{fst}\,(\overrightarrow{S_0}\,x)))) = S_1\,x$$

This lets us define the other introduction rules as well.

$$\begin{aligned} &\mathsf{intro} : S_0\,(\mathsf{IIR}\,S)\,\mathsf{El}\,i \to \mathsf{IIR}\,S\,i &\qquad& \mathsf{El\text{–}intro} : \mathsf{El}\,(\mathsf{intro}\,x) \equiv S_1\,x \\ &\mathsf{intro}\,x :\equiv (\mathsf{intro}_{\mathsf{IR}}\,(\mathsf{fst}\,(\overrightarrow{S_0}\,x)),\,\mathsf{snd}\,(\overrightarrow{S_0}\,x)) && \mathsf{El\text{–}intro} :\equiv S_{\lfloor 1 \rfloor}\,x \end{aligned}$$

3.2 Elimination

We assume a level l for the elimination target. Recall the type of elim:

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

[1]In the Agda formalization, we compute τ by induction on S, although it could be generically recovered from the other four components as well [?].

Also recall that $x : \mathsf{IIR}\, S\, i$ is given as a pair of some $x : \mathsf{IR}\,\lfloor S \rfloor$ and $p : \mathsf{fst}\,(\mathsf{El}\, x) = i$. The idea here is to use IR elimination on $x : \mathsf{IR}\,\lfloor S \rfloor$, while adjusting both P and f to operate on the appropriate data. We will use the following induction motive. Note that we generalize the induction goal over the p witness.

$$\begin{aligned}&\lfloor P \rfloor : \mathsf{IR}\,\lfloor S \rfloor \to \mathsf{U}_{\max(k,\,l)}\\ &\lfloor P \rfloor\, x :\equiv \{i : I\}(p : \mathsf{fst}\,(\mathsf{El}\, x) = i) \to P\,(x,\, p)\end{aligned}$$

Now, we have

$$\mathsf{elim}_{\mathsf{IR}}\,\lfloor P \rfloor : ((x : \lfloor S \rfloor_0\,(\mathsf{IR}\,\lfloor S \rfloor)\,\mathsf{El}) \to \lfloor S \rfloor_{\mathsf{IH}}\,\lfloor P \rfloor\, x \to \lfloor P \rfloor\,(\mathsf{intro}\, x)) \to (x : \mathsf{IR}\,\lfloor S \rfloor) \to \lfloor P \rfloor\, x.$$

We adjust f to obtain the next argument to $\mathsf{elim}_{\mathsf{IR}}\,\lfloor P \rfloor$. f takes $S_{\mathsf{IH}}\, P\, x$ as input, so we need a "backwards" conversion:

$$\overleftarrow{S_{\mathsf{IH}}} : \{x : S_{\lfloor 0 \rfloor}\, i\} \to \lfloor S \rfloor_{\mathsf{IH}}\,\lfloor P \rfloor\,(\mathsf{fst}\, x) \to S_{\mathsf{IH}}\, P\,(\overleftarrow{S_0}\, x)$$

This is again defined by easy induction on S. The induction method $\lfloor f \rfloor$ is as follows.

$$\begin{aligned}&\lfloor f \rfloor : (x : \lfloor S \rfloor_0\,(\mathsf{IR}\,\lfloor S \rfloor)\,\mathsf{El}) \to \lfloor S \rfloor_{\mathsf{IH}}\,\lfloor P \rfloor\, x \to \lfloor P \rfloor\,(\mathsf{intro}\, x)\\ &\lfloor f \rfloor\, x\, ih\, p :\equiv \mathsf{tr}\,\big(\lambda\,(x,\, p).\, P\,(\mathsf{intro}\, x,\, p)\big)\,\big(\epsilon\,(x,\, p)\big)\,\big(f\,(\overleftarrow{S_0}\,(x,\, p))\,(\overleftarrow{S_{\mathsf{IH}}}\, ih)\big)\end{aligned}$$

Thus, the definition of elimination is:

$$\mathsf{elim}\, P\, f\,(x,\, p) :\equiv \mathsf{elim}_{\mathsf{IR}}\,\lfloor P \rfloor\,\lfloor f \rfloor\, x\, p$$

Only the β-rule remains to be constructed:

$$\mathsf{elim}{-}\beta : \mathsf{elim}\, P\, f\,(\mathsf{intro}\, x) \equiv f\, x\,(S_{\mathsf{map}}\, P\,(\mathsf{elim}\, P\, f)\, x)$$

Computing definitions on the **left hand side**, we get:

$$\begin{aligned}&\mathsf{tr}\,(\lambda\,(x,\, p).\, P\,(\mathsf{intro}\, x,\, p))\\ &\quad(\epsilon\,(\overrightarrow{S_0}\, x))\\ &\quad(f\,(\overleftarrow{S_0}\,(\overrightarrow{S_0}\, x))\,(\overleftarrow{S_{\mathsf{IH}}}\,(\lfloor S \rfloor_{\mathsf{map}}\,\lfloor P \rfloor\,(\lambda\, x\, p.\, \mathsf{elim}\, P\, f\,(x,\, p))\,(\mathsf{fst}\,(\overrightarrow{S_0}\, x)))))\end{aligned}$$

Next, we prove by induction on S that $-_{\mathsf{map}}$ commutes with $\overrightarrow{S_0}$:

$$S_{\lfloor \mathsf{map} \rfloor} : \forall\, f\, x.\ S_{\mathsf{map}}\, P\,(\lambda\,(x,\, p).\, f\, x\, p)\, x = \mathsf{tr}\,\big(S_{\mathsf{IH}}\, P\big)\,\big(\eta\, x\big)\,\big(\overleftarrow{S_{\mathsf{IH}}}\,(\lfloor S \rfloor_{\mathsf{map}}\,\lfloor P \rfloor\, f\,(\mathsf{fst}\,(\overrightarrow{S_0}\, x)))\big)$$

Using this equation to rewrite the **right hand side**, we get:

$$f\, x\,\Big(\mathsf{tr}\,\big(S_{\mathsf{IH}}\, P\big)\,\big(\eta\, x\big)\,\big(\overleftarrow{S_{\mathsf{IH}}}\,(\lfloor S \rfloor_{\mathsf{map}}\,\lfloor P \rfloor\,(\lambda\, x\, p.\, \mathsf{elim}\, P\, f\,(x,\, p))\,(\mathsf{fst}\,(\overrightarrow{S_0}\, x)))\big)\Big)$$

This is now promising; on the left hand side we transport the result of f, while on the right hand side we transport the argument of f. Now, the identification on the left is $\epsilon\,(\overrightarrow{S_0}\, x)$, while we have $\eta\, x$ on the right. However, we have $\tau\, x : \mathsf{ap}\,(\overrightarrow{S_0})\,(\eta\, x) = \epsilon\,(\overrightarrow{S_0}\, x)$, which can be used in conjunction with standard transport lemmas to match up the two sides. This concludes the construction of IIR types.

3.3 Strictness

We briefly analyze the strictness of computation for constructed IIR types. Clearly, since the construction is defined by induction on IIR signatures, we only have propositional El−intro and elim−β in the general case, where an IIR signature can be neutral.

However, we still support the same definitional IIR computation rules as Agda and Idris. That is because Agda and Idris only have second-class IIR signatures. There, signatures consist of constructors with fixed configurations of fields, where constructors are disambiguated by canonical name tags. El applied to a constructor computes definitionally, and so does the elimination principle when applied to a constructor. Using our IIR types, we encode Agda IIR types as follows:

- We have $\sigma\,\mathsf{Tag}\,S$ on the top to represent constructor tags.
- In S, we immediately pattern match on the tag.
- All other Sig subterms are canonical in the rest of the signature.

Thus, if we apply El or elim to a value with a canonical tag, we compute past the branching on the tag and then compute all the way on the rest of the signature. In the Agda supplement, we provide length-indexed vectors and the Code universe as examples for constructed IIR types with strict computation rules.

3.4 Mechanization

We formalized Section **??** in Agda. For the assumption of IR, we verbatim reproduced the specification in Section [?], turning IR into an inductive type and El and elim into recursive functions. The functions are not recognized as terminating by Agda, so we disable terminating checking for them. Alternatively, we could use rewrite rules [?]; the two versions are the same except that rewrite rules have a noticeable performance cost in evaluation.

One small difference is that our object theory does not have internal universe levels, so we understand level quantification to happen in a metatheory, while in Agda we use native universe polymorphism.

4 Canonicity

In this section we prove canonicity for the object theory extended with IR types. First, we specify the metatheory and the object theory in more detail.

4.1 Metatheory

4.1.1 Specification. The metatheory supports the following:

- A countable universe hierarchy and basic type formers as described in Section **??**. We write universes as Set_i instead of U_i, to avoid confusion with object-theoretic universes.
- Equality reflection. Hence, in the following we will only use $- = -$ to denote metatheoretic equality.
- Universe levels ω and $\omega + 1$, where $\mathsf{Set}_\omega : \mathsf{Set}_{\omega+1}$ and $\mathsf{Set}_{\omega+1}$ is a "proper type" that is not contained in any universe. Set_ω and $\mathsf{Set}_{\omega+1}$ are also closed under basic type formers.
- Each Set_i supports IR types when i is finite. Hence, Set_i also supports IIR types.
- A reflection principle which says that finite universe levels corresponds to elements of the internal type of natural numbers $\mathsf{Nat} : \mathsf{Set}_0$. This is similar to Agda's internal type of finite levels, called Level [?]. For a more formal specification of this internalization, see [?]. The reason for this feature is the following. The object theory has countable levels represented as natural numbers, so we have to interpret those numbers as metatheoretic levels in the canonicity model, to correctly specify sizes of reducibility predicates.

- The syntax of the object theory as a quotient inductive-inductive type [?], to be described in the following section.
- *Notation:* Lift is derivable the same way as we have seen, but we will make all lifting implicit in the metatheory. In the object theory, explicit lifting is advisable, because we talk about strict computation and canonicty, so we want to be precise about definitional content. In the metatheory, we have equality reflection, so we can be more loose.

4.1.2 *Consistency of the metatheory.*

TODO

4.2 The Object Theory

Informally, the object theory is a Martin-Löf type theory that supports basic type formers as described in Section **??** and IR types as described in Section **??**. More formally, the object theory is given as a quotient inductive-inductive type [?]. The sets, operations and equations that we give in the following together constitute the inductive signature.

4.2.1 Core substitution calculus. The basic judgmental structure is given as a category with families (CwF) [?] where types are additionally annotated with levels. Concretely, we have

- A category of contexts and substitutions. We have $\mathsf{Con} : \mathsf{Set}_0$ for contexts and $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}_0$ for substitutions. The empty context $\bullet$ is the terminal object with the unique substitution $\epsilon : \mathsf{Sub}\,\Gamma\,\bullet$. We write id for identity substitutions and $-\circ-$ for substitution composition.
- Level-indexed types, as $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Nat} \to \mathsf{Set}_0$, together with the functorial substitution operation $-[-] : \mathsf{Ty}\,\Delta\,i \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma\,i$.
- Terms as $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Set}_0$, with functorial substitution operation $-[-] : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma]$. *Notation:* both type and term substitution binds stronger than function application, so for example $\mathsf{Tm}\,\Gamma\,A[\sigma]$ means $\mathsf{Tm}\,\Gamma\,(A[\sigma])$.
- Context comprehension, consisting of a context extension operation $-\triangleright- : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Con}$, weakening morphism $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$, zero variable $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,A[\mathsf{p}]$ and substitution extension $-,- : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma] \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$, such that the following equations hold:

$$\begin{aligned} \mathsf{p} \circ (\sigma, t) &= \sigma \\ \mathsf{q}[\sigma, t] &= t \\ (\mathsf{p}, \mathsf{q}) &= \mathsf{id} \\ (\sigma, t) \circ \delta &= (\sigma \circ \delta, t[\delta]) \end{aligned}$$

  Note that a De Bruijn index N is represented as $\mathsf{q}[\mathsf{p}^N]$, where p^N is N-fold composition of weakening.

4.2.2 Universes. We have Russell-style universes, where sets of terms of universes are identified with sets of types. Concretely, we have $\mathsf{U} : (i : \mathsf{Nat}) \to \mathsf{Ty}\,\Gamma\,(i+1)$ such that $\mathsf{U}_i[\sigma] = \mathsf{U}_i$ and $\mathsf{Tm}\,\Gamma\,\mathsf{U}_i = \mathsf{Ty}\,\Gamma\,i$. This lets us implicitly convert between types and terms with universe types. Additionally, we specify that this casting operation commutes with substitution, so substituting $t : \mathsf{Tm}\,\Gamma\,\mathsf{U}_i$ as a term and then casting to a type is the same as first casting and then substituting as a type. Since we omit casts and overload $-[-]$, this rule looks trivial in our notation, but it still has to be assumed.

4.2.3 Functions. We have $\Pi : (A : \mathsf{Ty}\,\Gamma\,i) \to \mathsf{Ty}\,(\Gamma \triangleright A)\,j \to \mathsf{Ty}\,\Gamma\,\max(i,\,j)$ such that $(\Pi\,A\,B)[\sigma] = \Pi\,A[\sigma]\,B[\sigma \circ \mathsf{p},\,\mathsf{q}]$. Terms are specified by $\mathsf{app} : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,B$ and its definitional inverse $\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$. This isomorphism is natural in Γ, so we have a substitution rule $(\mathsf{lam}\,t)[\sigma \circ \mathsf{p},\,\mathsf{q}] = \mathsf{lam}\,t[\sigma]$.

Notation & conventions. So far we have used standard definitions, but now we develop some notations and conventions that are more tailored to our use case. CwF combinators and De Bruijn indices get very hard to read when get to more complicated rules like identity elimination or rules in the specification of IR types.

- Assuming $t : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$ and $u : \mathsf{Tm}\,\Gamma\,A$, traditional binary function application can be derived as $(\mathsf{app}\,t)[\mathsf{id},\,u] : \mathsf{Tm}\,\Gamma\,B[\mathsf{id},\,u]$. We overload the metatheoretic whitespace operator for this kind of object-level application.
- We may give a name to a binder (a binder can be a context extension or a Π, Σ or lam binder), and in the scope of the binder all occurrence of the name is desugared to a De Bruijn index. We write Π-types using the same notation as in the metatheory. For example, $(A : \mathsf{U}_i) \to A \to A$ is desugared to $\Pi\,\mathsf{U}_i\,(\Pi\,\mathsf{q}\,\mathsf{q}[\mathsf{p}])$. We also reuse the notation and behavior of implicit functions. We write object-level lambda abstraction as $\mathsf{lam}\,x.t$.
- In the following we specify all type and term formers as *term constants with an iterated Π-type*. For example, we will specify $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,((A : \mathsf{U}_i) \to A \to A \to \mathsf{U}_i)$, instead of abstracting over $A : \mathsf{Ty}\,\Gamma\,i$ and $t,\,u : \mathsf{Tm}\,\Gamma\,A$. In the general, the two flavors are interderivable, but sticking to object-level functions lets us consistently use the sugar for named binders. Also, specifying stability under substitution becomes very simple: a substituted term constant is computed to the same constant (but living in a possibly different implicit context). For example, if $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, then $\mathsf{Id}[\sigma] = \mathsf{Id}$ specifies stability under substitution. Hence, we shall omit substitution rules in the following.

4.2.4 Sigma types. We have

$$\begin{aligned}
\Sigma &: \mathsf{Tm}\,\Gamma\,((A : \mathsf{U}_i) \to (A \to \mathsf{U}_j) \to \mathsf{U}_{\max(i,\,j)})\\
-,- &: \mathsf{Tm}\,\Gamma\,(\{A : \mathsf{U}_i\}\{B : A \to \mathsf{U}_j\}(t : A) \to B\,t \to \Sigma\,A\,B)\\
\mathsf{fst} &: \mathsf{Tm}\,\Gamma\,(\{A : \mathsf{U}_i\}\{B : A \to \mathsf{U}_j\} \to \Sigma\,A\,B \to A)\\
\mathsf{snd} &: \mathsf{Tm}\,\Gamma\,(\{A : \mathsf{U}_i\}\{B : A \to \mathsf{U}_j\}(t : \Sigma\,A\,B) \to B\,(\mathsf{fst}\,t))
\end{aligned}$$

such that $\mathsf{fst}\,(t,\,u) = t$, $\mathsf{snd}\,(t,\,u) = u$ and $(\mathsf{fst}\,t,\,\mathsf{snd}\,u) = t$.

4.2.5 Unit. We have $\top_i : \mathsf{Tm}\,\Gamma\,\mathsf{U}_i$ with the unique inhabitant tt.

4.2.6 Booleans. Type formation is $\mathsf{Bool} : \mathsf{Tm}\,\Gamma\,\mathsf{U}_0$ with constructors true and false. Elimination is as follows.

$$\begin{aligned}
&\mathsf{BoolElim} : \mathsf{Tm}\,\Gamma\,((P : \mathsf{Bool} \to \mathsf{U}_i) \to P\,\mathsf{true} \to P\,\mathsf{false} \to (b : \mathsf{Bool}) \to P\,b)\\
&\mathsf{BoolElim}\,P\,t\,f\,\mathsf{true}\;\;= t\\
&\mathsf{BoolElim}\,P\,t\,f\,\mathsf{false} = f
\end{aligned}$$

4.2.7 Identity type.

$$\begin{aligned}
\mathsf{Id} &: \mathsf{Tm}\,\Gamma\,((A : \mathsf{U}_i) \to A \to A \to \mathsf{U}_i)\\
\mathsf{refl} &: \mathsf{Tm}\,\Gamma\,(\{A : \mathsf{U}_i\}(t : A) \to \mathsf{Id}\,A\,t\,t)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{J} : \mathsf{Tm}\,\Gamma\,(\{A : \mathsf{U}_i\}\{x : A\}(P : (y : A) \to \mathsf{Id}\,A\,x\,y \to \mathsf{U}_k) \\
&\qquad\qquad \to P\,(\mathsf{refl}\,x) \to \{y : A\}(p : \mathsf{Id}\,A\,x\,y) \to P\,y\,p) \\
&\mathsf{J}\,P\,r\,(\mathsf{refl}\,x) = r
\end{aligned}$$

4.2.8 IR types. First, we specify the type of signatures as an inductive type. We assume levels i and j.

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

$$\begin{aligned}
&\mathsf{SigElim} : \mathsf{Tm}\,\Gamma\,(\{O : \mathsf{U}_j\}(P : \mathsf{Sig}_i\,O \to \mathsf{U}_k) \\
&\qquad \to ((o : O) \to P\,(\iota\,o)) \\
&\qquad \to ((A : \mathsf{U}_i)(S : A \to \mathsf{Sig}_i\,O) \to ((a : A) \to P\,(S\,a)) \to P\,(\sigma\,A\,S)) \\
&\qquad \to ((A : \mathsf{U}_i)(S : (A \to O) \to \mathsf{Sig}_i\,O) \to ((f : A \to O) \to P\,(S\,f)) \to P\,(\delta\,A\,S)) \\
&\qquad \to (S : \mathsf{Sig}_i\,O) \to P\,S)
\end{aligned}$$

$$\begin{aligned}
\mathsf{SigElim}\,P\,i\,s\,d\,(\iota\,o) &= i \\
\mathsf{SigElim}\,P\,i\,s\,d\,(\sigma\,A\,S) &= s\,A\,S\,(\mathsf{lam}\,a.\,\mathsf{SigElim}\,P[\mathsf{p}]\,i[\mathsf{p}]\,s[\mathsf{p}]\,d[\mathsf{p}]\,(S[\mathsf{p}]\,a)) \\
\mathsf{SigElim}\,P\,i\,s\,d\,(\delta\,A\,S) &= d\,A\,S\,(\mathsf{lam}\,f.\,\mathsf{SigElim}\,P[\mathsf{p}]\,i[\mathsf{p}]\,s[\mathsf{p}]\,d[\mathsf{p}]\,(S[\mathsf{p}]\,f))
\end{aligned}$$

Note the $[\mathsf{p}]$ weakenings in the computation rules: P, i, s, d, and S are all terms quantified in some implicit context Γ, so when we mention them under an extra binder, we have to weaken them. Hence, we cannot fully avoid explicit substitution operations by using named binders. We already saw rest of the specification in Section **??** so we only give a short summary.

- $-_0$, $-_1$, $-_{\mathsf{map}}$ and $-_{\mathsf{IH}}$ are defined by SigElim and they satisfy the same definitional equations that we saw in Section **??**.
- IR, El, intro, elim are all specified as term constants that are only parameterized over contexts and some universe levels.

4.3 Canonicity of the Object Theory

On a high level, canonicity is proved by induction over the syntax of the object theory. Since the syntax is a quotient inductive-inductive type, it supports an induction principle, which we do not write out fully here, and only use one particular instance of it. Formally, the induction principle takes a *displayed model* as an argument, which corresponds to a bundle of induction motives and methods, and proofs that quotient equations are respected. We could present the current construction as a displayed model. However, we find it a bit more readable to instead use an Agda-like notation, where we specify the resulting *section* of the displayed model, which consists of a collection of mutual functions, mapping out from the syntax, which have action on constructors and respect all quotient equations.

Notation: In the following, we may write $\mathsf{Tm}\,A$ to mean $\mathsf{Tm}\bullet A$, and $\mathsf{Sub}\,\Gamma$ to mean $\mathsf{Sub}\bullet\Gamma$. This will reduce clutter since we will mostly work with closed terms and substitutions.

Using the above notation, we aim to define the following functions by induction on object syntax.

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

This has appeared in different flavors in previous literature. It is a proof-relevant logical predicate interpretation, which corresponds to *Artin gluing* [?], i.e. categorical gluing over the global sections functor [?]. The concrete formulation that we use is the type-theoretic gluing by Kaposi, Huber and Sattler [?]. This is a mild variation which uses dependent type families instead of the fibered families of the categorical flavor. The type-theoretic style becomes valuable when we get to the interpretation of more complicated type formers, where it is easier to use than diagrammatic reasoning.

4.3.1 Interpretation of the CwF and the basic type formers. This has been described in previous literature [?]; in particular the code supplement to [?] has an Agda formalization of the canonicity model with the same universe setup and basic type formers that we use. Therefore we only present parts here which are relevant to the IR type interpretation, which are the CwF, universes, sigma types, functions and the unit type. The interpretation of empty and extended contexts is as follows.

$$\begin{aligned}
\bullet^\circ\quad\ \gamma &:= \top \\
(\Gamma \rhd A)^\circ\,(\gamma, \alpha) &:= (\gamma^\circ : \Gamma^\circ\,\gamma) \times A^\circ\,\gamma^\circ\,\alpha
\end{aligned}$$

This says that the logical predicate holds for a closing substitution if it holds for each term in the substitution. Note the pattern matching notation in (γ, α): this is justified, since all values of $\mathsf{Sub}\,(\Gamma \rhd A)$ are uniquely given as a pairing (similarly to pattern matching notation for plain Σ-types). The other CwF operations are as follows.

$$\begin{aligned}
\mathsf{id}^\circ\,\gamma^\circ &:= \gamma^\circ & (\sigma, t)^\circ\,\gamma^\circ &:= (\sigma^\circ\,\gamma^\circ, t^\circ\,\gamma^\circ) \\
(\sigma \circ \delta)^\circ\,\gamma^\circ &:= \sigma^\circ\,(\delta^\circ\,\gamma^\circ) & \mathsf{p}^\circ\,(\gamma^\circ, \alpha^\circ) &:= \gamma^\circ \\
(A[\sigma])^\circ\,\gamma^\circ\,\alpha &:= A^\circ\,(\sigma^\circ\,\gamma^\circ)\,\alpha & \mathsf{q}^\circ\,(\gamma^\circ, \alpha^\circ) &:= \alpha^\circ \\
(t[\sigma])^\circ\,\gamma^\circ &:= t^\circ\,(\sigma^\circ\,\gamma^\circ) & \epsilon^\circ\,\gamma^\circ &:= \mathsf{tt}
\end{aligned}$$

The interpretation of **universes** is the following.

$$(\mathsf{U}_i)^\circ\,\gamma^\circ\,\alpha := \mathsf{Tm}\,\alpha \to \mathsf{Set}_i$$

This definition also supports the Russell universe rules. For illustration, assuming $t : \mathsf{Tm}\,\Gamma\,\mathsf{U}_i$, we have $t^\circ : \{\gamma : \mathsf{Sub}\,\Gamma\}(\gamma^\circ : \Gamma^\circ\,\gamma) \to \mathsf{Tm}\,t[\gamma] \to \mathsf{Set}_i$. If we first cast t to a type using the syntactic Russell universe equation, then t° has exactly the same type.

Note that we do not get a canonicity statement about types themselves, i.e. we do not get that every closed type is definitionally equal to one of the canonical type formers. This could be handled as well, but we skip it because it is orthogonal to the focus of this paper.

We interpret **functions** as follows.

$$\begin{aligned}
(\Pi\,A\,B)^\circ\,\{\gamma\}\,\gamma^\circ\,f &:= \{\alpha : \mathsf{Tm}\,A[\gamma]\}(\alpha^\circ : A^\circ\,\gamma^\circ\,\alpha) \to B^\circ\,(\gamma^\circ, \alpha^\circ)\,(f\,\alpha) \\
(\mathsf{lam}\,t)^\circ\,\gamma^\circ &:= \lambda\,\{\alpha\}\,\alpha^\circ .\, t^\circ\,(\gamma^\circ, \alpha^\circ) \\
(\mathsf{app}\,t)^\circ\,(\gamma^\circ, \alpha^\circ) &:= t^\circ\,\gamma^\circ\,\alpha^\circ
\end{aligned}$$

In the $\Pi\,A\,B$ case, note that $f : \mathsf{Tm}\,(\Pi\,A\,B)[\gamma]$, which means that we can apply it to α to get $f\,\alpha : \mathsf{Tm}\,B[\gamma, \alpha]$.

For Σ-**types**, we have

$$\begin{array}{ll} \Sigma^\circ\,\gamma^\circ\,A^\circ\,B^\circ\,(t,\,u) := (t^\circ : A^\circ\,t) \times B^\circ\,t^\circ\,u & \mathsf{fst}^\circ\,\gamma^\circ\,(t^\circ,\,u^\circ) \;:= t^\circ \\ (-,-)^\circ\,\gamma^\circ\,t^\circ\,u^\circ \quad := (t^\circ,\,u^\circ) & \mathsf{snd}^\circ\,\gamma^\circ\,(t^\circ,\,u^\circ) := u^\circ \end{array}$$

For the **unit type**, we have $\top^\circ\,\gamma^\circ\,t := \top$ and $\mathsf{tt}^\circ\,\gamma^\circ := \mathsf{tt}$.

4.3.2 Interpretation of IR signatures. Signatures are given as a particular parameterized inductive type, so in principle there should be nothing "new" in their logical predicate interpretation. We do detail it here because several later constructions depend on it. Recall that $\mathsf{Sig}_i : \mathsf{Tm}\,\Gamma\,((O : \mathsf{U}_j) \to \mathsf{U}_{\max(i+1,\,j)})$, so we have

$$\begin{aligned} &(\mathsf{Sig}_i)^\circ\,\gamma^\circ : ((O : \mathsf{U}_j) \to \mathsf{U}_{\max(i+1,\,j)})^\circ\,\gamma^\circ\,\mathsf{Sig}_i \\ &(\mathsf{Sig}_i)^\circ\,\gamma^\circ : \{O : \mathsf{Tm}\,\mathsf{U}_j\}(O^\circ : (\mathsf{U}_j)^\circ\,\gamma^\circ\,O) \to (\mathsf{U}_{\max(i+1,\,j)})^\circ\,\gamma^\circ\,(\mathsf{Sig}_i\,O) \\ &(\mathsf{Sig}_i)^\circ\,\gamma^\circ : \{O : \mathsf{Tm}\,\mathsf{U}_i\}(O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_i) \to \mathsf{Tm}\,(\mathsf{Sig}_i\,O) \to \mathsf{Set}_{\max(i+1,\,j)}. \end{aligned}$$

Hence, we define an inductive type in the metatheory that is parameterized by $O : \mathsf{Tm}\,U_i$ and $O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_i$ and indexed over $\mathsf{Tm}\,(\mathsf{Sig}_i\,O)$. We name this inductive type Sig°; the naming risks some confusion, but we shall take the risk and we will shortly explain the rationale.

$$\begin{aligned} &\mathbf{data}\,\mathsf{Sig}^\circ\,\{O : \mathsf{Tm}\,U_j\}\,(O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_i) : \mathsf{Tm}\,(\mathsf{Sig}_i\,O) \to \mathsf{Set}_{\max(i+1,\,j)} \\ &\quad \iota^\circ \;: \{o \;: \mathsf{Tm}\,O\}(o^\circ : O^\circ\,o) \to \mathsf{Sig}^\circ\,O^\circ\,(\iota\,o) \\ &\quad \sigma^\circ : \{A \;: \mathsf{Tm}\,\mathsf{U}_i\}(A^\circ : \mathsf{Tm}\,A \to \mathsf{Set}_i) \\ &\qquad\;\; \{S \;: \mathsf{Tm}\,(A \to \mathsf{Sig}_i\,O)\} \\ &\qquad\;\; (S^\circ : \{a : \mathsf{Tm}\,A\} \to A^\circ\,a \to \mathsf{Sig}^\circ\,O^\circ\,(S\,a)) \\ &\qquad\;\; \to \mathsf{Sig}^\circ\,O^\circ\,(\sigma\,A\,S) \\ &\quad \delta^\circ : \{A \;: \mathsf{Tm}\,\mathsf{U}_i\}(A^\circ : \mathsf{Tm}\,A \to \mathsf{Set}_i) \\ &\qquad\;\; \{S \;: \mathsf{Tm}\,((A \to O) \to \mathsf{Sig}_i\,O)\} \\ &\qquad\;\; (S^\circ : \{f : \mathsf{Tm}\,(A \to O)\} \to (\{a : \mathsf{Tm}\,A\} \to A^\circ\,a \to O^\circ\,(f\,a)) \to \mathsf{Sig}^\circ\,O^\circ\,(S\,f)) \\ &\qquad\;\; \to \mathsf{Sig}^\circ\,O^\circ\,(\delta\,S\,f) \end{aligned}$$

A witness of $\mathsf{Sig}^\circ\,O^\circ\,t$ tells us that t is a canonical constructor and it only contains canonical data, inductively. Now, we define $(\mathsf{Sig}_i)^\circ\,\gamma^\circ\,O^\circ\,t$ to be $\mathsf{Sig}^\circ\,O^\circ\,t$, and each syntactic Sig constructor is interpreted using the corresponding semantic constructor. For instance:

$$\begin{aligned} &\iota^\circ : \{\gamma : \mathsf{Sub}\,\Gamma\}(\gamma^\circ : \Gamma^\circ\,\gamma)\{O : \mathsf{Tm}\,\mathsf{U}_j\}\{O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_j\}\{o : \mathsf{Tm}\,O\} \to O^\circ\,o \to \mathsf{Sig}^\circ\,O^\circ\,(\iota\,o) \\ &\iota^\circ\,\gamma^\circ\,o^\circ := \iota^\circ\,o^\circ \end{aligned}$$

We skip the interpretation of the other constructors and the eliminator here. Above on the left side we use ι° for specifying the action of $-^\circ$ on the syntactic ι, while on the right side we use the metatheoretic Sig° constructor ι°. In general, the recipe is:

(1) We first give semantic interpretation of object constructions while only referring to *closed terms.*
(2) Then, we "contextualize" the definitions to get interpretations of object-theoretic rules.

In this section, the bulk of the work in interpreting IR types only needs to refer to closed terms, and the final step of contextualization is fairly trivial. Hence, we optimize our notation for the first phase.

4.3.3 *Interpretation of IR types.*

In the following, we follow the above recipe, first defining semantics with closed terms, then getting context-dependent interpretations for type and term formers. We assume the following parameters: $O : \mathsf{Tm}\, U_j$, $O^\circ : \mathsf{Tm}\, O \to \mathsf{Set}_j$, $S^* : \mathsf{Tm}\,(\mathsf{Sig}_i\, O)$ and $S^{*\circ} : \mathsf{Sig}^\circ\, O^\circ\, S^*$. *Abbreviation:* we write $\mathsf{Sig}^\circ\, S$ in the following, omitting the fixed O° parameter from the type. We view S^* as a "fixed" top-level signature, in contrast to "varying" S signatures that we will encounter in constructions. More concretely, we will do most constructions by induction on *canonical sub-signatures* of S^*.

Canonical sub-signatures of S^*. We define an inductive family indexed over S and $S^\circ : \mathsf{Sig}^\circ\, O^\circ\, S$, which represents paths into S^* that lead to S, viewing S as a subtree. Also, the subtree and all data in the path must be canonical (i.e. have $-^\circ$ witnesses). The path is represented as a left-associated *snoc-list* of data that can be plugged into σ and δ constructors. Moreover, we restrict the δ case, only allowing $f : \mathsf{Tm}\,(A \to \mathsf{IR}\, S^*)$ functions instead of functions of type $\mathsf{Tm}\,(A \to O)$.

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Path} : \{S : \mathsf{Tm}\,(\mathsf{Sig}_i\, O)\} \to \mathsf{Sig}^\circ\, S \to \mathsf{Set}_{\max(i+1,\, j+1)}\\
&\quad \mathsf{here} : \mathsf{Path}\, S^{*\circ}\\
&\quad \mathsf{in}\text{–}\sigma : \mathsf{Path}\,(\sigma^\circ\, A^\circ\, S^\circ) \to (a^\circ : A^\circ\, a) \to \mathsf{Path}\,(S^\circ\, a^\circ)\\
&\quad \mathsf{in}\text{–}\delta : \mathsf{Path}\,(\delta^\circ\, A^\circ\, S^\circ) \to \{f : \mathsf{Tm}\,(A \to \mathsf{IR}\, S^*)\}(f^\circ : \{a : \mathsf{Tm}\, A\} \to A^\circ\, a \to O^\circ\,(\mathsf{El}\,(f\, a)))\\
&\qquad\quad \to \mathsf{Path}\,(S^\circ\, f^\circ)
\end{aligned}
$$

If we have a path to $S^\circ : \mathsf{Sig}^\circ\, S$, we can push the terms contained in the path onto a term of $S_0\,(\mathsf{IR}\, S^*)\,\mathsf{El}$:

$$
\begin{aligned}
&\mathsf{push}_0 : \mathsf{Path}\, S^\circ \to \mathsf{Tm}\,(S_0\,(\mathsf{IR}\, S^*)\,\mathsf{El}) \to \mathsf{Tm}\,((S^*)_0\,(\mathsf{IR}\, S^*)\,\mathsf{El})\\
&\mathsf{push}_0\ \mathsf{here}\qquad\qquad t := t\\
&\mathsf{push}_0\ (\mathsf{in}\text{–}\sigma\, p\, \{a\}\, a^\circ)\ t := \mathsf{push}_0\, p\,(a,\, t)\\
&\mathsf{push}_0\ (\mathsf{in}\text{–}\delta\, p\, \{f\}\, f^\circ)\ t := \mathsf{push}_0\, p\,(f,\, t)
\end{aligned}
$$

This operation preserves $-_1$, so we have $S_1\, t = (S^*)_1\,(\mathsf{push}_0\, p\, t)$.

Encoding for signatures. Next, we define an encoding function for signatures, somewhat similarly to how we did in in Section ??. The idea is to compute a metatheoretical IIR signature which represents the logical predicate for a syntactic IR type. Note that this computation is only possible by induction on $\mathsf{Sig}^\circ\, S$ – we have no appropriate induction principle for $\mathsf{Tm}\,(\mathsf{Sig}_i\, O)$. As we recurse into a signature, we have to store the data that we have seen in a Path, and when we hit the base case ι°, we use the Path to build up the correct term for the constructor index.

$$
\begin{aligned}
&\lfloor - \rfloor : (S^\circ : \mathsf{Sig}^\circ\, S) \to \mathsf{Path}\, S^\circ \to \mathsf{Sig}_{\mathsf{IIR}}\,(\mathsf{Tm}\,(\mathsf{IR}\, S^*))\,(O^\circ \circ \mathsf{El})\\
&\lfloor \iota^\circ\, o^\circ \rfloor \qquad\quad p := \iota\,(\mathsf{intro}\,(\mathsf{push}_0\, p\,(\uparrow \mathsf{tt})))\, o^\circ\\
&\lfloor \sigma^\circ\, \{A\}\, A^\circ\, S^\circ \rfloor\, p := \sigma\,(\mathsf{Tm}\, A)\ \$\ \lambda\, a.\, \sigma\,(A^\circ\, a)\ \$\ \lambda\, a^\circ.\, \lfloor S^\circ\, a^\circ \rfloor\,(\mathsf{in}\text{–}\sigma\, p\, a^\circ)\\
&\lfloor \delta^\circ\, \{A\}\, A^\circ\, S^\circ \rfloor\, p := \sigma\,(\mathsf{Tm}\,(A \to \mathsf{IR}\, S^*))\ \$\ \lambda\, f.\, \delta\,((a : \mathsf{Tm}\, A) \times A^\circ\, a)(\lambda\,(a,\, \_).\, f\, a)\ \$\ \lambda\, f^\circ.\\
&\qquad\qquad\qquad\qquad \lfloor S^\circ\,(\lambda\,\{a\}\, a^\circ.\, f^\circ\,(a,\, a^\circ)) \rfloor\,(\mathsf{in}\text{–}\delta\, p\,(\lambda\,\{a\}\, a^\circ.\, f^\circ\,(a,\, a^\circ)))
\end{aligned}
$$

- The metatheoretic IIR type is indexed over $\mathsf{Tm}\,(\mathsf{IR}\, S^*)$, and the recursive output type is given by $O^\circ \circ \mathsf{El} : \mathsf{Tm}\,(\mathsf{IR}\, S^*) \to \mathsf{Set}_j$. Here, we implicitly cast the syntactic function $\mathsf{El} : \mathsf{Tm}\,(\mathsf{IR}\, S^* \to O)$ to $\mathsf{Tm}\,(\mathsf{IR}\, S^*) \to \mathsf{Tm}\, O$.
- In the ι° case, we have $o^\circ : O^\circ\, o$ and

$$\iota : (t : \mathsf{Tm}\,(\mathsf{IR}\, S^*)) \to O^\circ\,(\mathsf{El}\, t) \to \mathsf{Sig}_{\mathsf{IIR}}\,(\mathsf{Tm}\,(\mathsf{IR}\, S^*))\,(O^\circ \circ \mathsf{El}).$$

We have $\mathsf{intro}\,(\mathsf{push}_0\,p\,(\uparrow\mathsf{tt}))\ :\ \mathsf{Tm}\,(\mathsf{IR}\,S^*)$. If we apply El to this term, it computes to $(S^*)_1\,(\mathsf{push}_0\,p\,(\uparrow\mathsf{tt}))$, which is the same as $(\iota\,o)_1\,(\uparrow\mathsf{tt})$, which is the same as o, which makes $o^\circ : O^\circ\,o$ well-typed for the second argument.

- In the σ° case, we use two σ-s to abstract over a term and a canonicity witness for it.
- In the δ° case, we abstract over $f : \mathsf{Tm}\,(A \rightarrow \mathsf{IR}\,S^*)$, then we use δ to specify inductive witnesses for all "subtrees" that are obtained by applying f to canonical terms.

Example 4.1. We compute $\lfloor - \rfloor$ for the signature in Example **??**.

5 TODO

- Implicit lifting in metatheory in canonicity section
- Rename wrap to intro in Agda. Rename DeriveIndexed F0 equivalence in Agda.
- Canonicity metatheory: Loic, Anton says it looks OK

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