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 established method of gluing along the global sections functor. 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), 22 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Instances of inductive-recursive (IR) definitions were first used by Martin-Löf [1975, 1984]. General definitions of IR types were formalized by Dybjer [2000] and Dybjer and Setzer [1999, 2003, 2006a], who also developed set-theoretic, realizability 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:

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

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 [Dybjer and Setzer 2006a].

An important application of (I)IR has been to develop semantics for object theories that support universe hierarchies. It has been used in normalization proofs [Abel et al. 2023, 2018; Pujet and Tabareau 2023], in modeling first-class universe levels [Kovács 2022] and proving canonicity for them [Chan and Weirich 2025]. Other applications are in characterizing domains of partial functions [Bove and Capretta 2001] and in generic programming over type descriptions [Diehl 2017].

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

IIR has been supported in Agda 2 since the early days of the system [Bove et al. 2009], and it is also available in Idris 1 [Brady 2013] and Idris 2 [Brady 2021]. In these systems, IIR has been implemented in the "obvious" way, supporting 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 of closed terms. The outline of our development is as follows.

- In Section 2 we specify what it means to support IR and IIR, using Dybjer and Setzer's rules with minor modifications [Dybjer and Setzer 2003, 2006b]. We use first-class signatures, meaning that descriptions of (I)IR types are given as ordinary inductive types internally.
- In Section 3 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 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.
- In Section 4, we give a proof-relevant logical predicate interpretation of the type theory, from which canonicity follows. Our method is based on a type-theoretic flavor of gluing along the global sections functor [Coquand 2019; Kaposi et al. 2019a]. 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 shall mostly work with internal definitions in an Agda-like syntax. In Section 4.2 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 sometimes omit the type of an implicit argument and write $\{x\} \to B\,x$. Also, we may omit the implicit quantification entirely: if there are variables in a type which are not quantified anywhere, they are understood to be implicitly quantified with a Π-type.

Σ-types: for $A : \mathsf{U}_i$ and $B : A \to \mathsf{U}_j$, we have $((x : A) \times B\,x) : \mathsf{U}_{\max(i,\,j)}$. We write $-,-$ for pairing and fst and snd for projections. 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 true = 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 write definitions with $:\equiv$.

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 [Dybjer 1994] 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 reproduce the Agda example from Section 1. 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'. 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 $\top$ sets the number of introduced fields. The naming of the freshly bound variable ElA is meant to suggest that it represents 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

$ElB : ElA\,\mathsf{tt} \to \mathsf{U}_0$ 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 those in [Dybjer and Setzer 2003], 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. For the sake of readability, we split this action to two functions. $\mathbb{E}$ yields the "total type" part of the interpretation of a signature, and $\mathbb{F}$ yields the morphism part, i.e. a function into O.

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

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

Above we use an Agda-like notation for specifying the implicit S argument to $\mathbb{F}$. 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.

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

Note that the rule El–intro specifies a definitional equality. Also, 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. We again follow Dybjer and Setzer [2003]. We assume a universe level k that specifies the size of the type into which we eliminate. We define two additional functions on signatures:

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

$$
\begin{aligned}
&\mathsf{map} : \{S : \mathsf{Sig}_i\, O\} \to ((x : ir) \to P\, x) \to (x : \mathbb{E}\, S\, ir\, el) \to \mathsf{IH}\, P\, x\\
&\mathsf{map}\,\{\iota\, o\} \quad g\, x \quad :\equiv \mathsf{tt}\\
&\mathsf{map}\,\{\sigma\, A\, S\}\, g\,(a,\, x) :\equiv \mathsf{map}\,\{S\, a\}\, g\, x\\
&\mathsf{map}\,\{\delta\, A\, S\}\, g\,(f,\, x) :\equiv (g \circ f,\ \mathsf{map}\,\{S\,(el \circ f)\}\, g\, x)
\end{aligned}
$$

$-_{\mathsf{IH}}$ stands for "induction hypothesis": it specifies having a witness of a predicate P for each inductive field in a value of $\mathbb{E}\, S\, ir\, el$. map maps over $\mathbb{E}\, S\, ir\, el$, applying the section $g : (x : ir) \to P\, x$ to each inductive field. Elimination is specified as follows.

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

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 [Dybjer and Setzer 2003, Section 4.4].

2.3 IIR Types

In IIR signatures, the sole deviation from Dybjer and Setzer [2006a] is again our use of countable universe levels.

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 argument, binding n for the length of the tail of the vector. Then, when we introduce the inductive argument using δ, we use $(\lambda\,\_.\, n)$ to specify that index of the argument is indeed n. Finally, the length of the Cons' constructor is $\mathsf{suc}\, n$.

2.3.2 *Type and term formation.* $\mathbb{E}$ and $\mathbb{F}$ are similar to before:

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

$$
\begin{aligned}
&\mathbb{F} : \{S : \mathsf{Sig}_i\, I\, O\} \to \mathbb{E}\, S\, ir\, el\, i \to O\, i\\
&\mathbb{F}\,\{\iota\, i'\, o\} \qquad (\uparrow x) \;:\equiv \mathsf{tr}\, O\, x\, o\\
&\mathbb{F}\,\{\sigma\, A\, S\} \qquad (a,\, x) :\equiv \mathbb{F}\,\{S\, a\}\, x\\
&\mathbb{F}\,\{\delta\, A\, ix\, S)\}\,(f,\, x) :\equiv \mathbb{F}\,\{S\,(el \circ f)\}\, x
\end{aligned}
$$

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.

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

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

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

$$
\begin{aligned}
&\mathsf{map} : \{S : \mathsf{Sig}_i\, I\, O\} \to (\{i : I\}(x : ir\, i) \to P\, x) \to (x : \mathbb{E}\, S\, ir\, el\, i) \to S_{\mathsf{IH}}\, P\, x\\
&\mathsf{map}\,\{\iota\, o\} \qquad g\, x \qquad :\equiv\; \mathsf{tt}\\
&\mathsf{map}\,\{\sigma\, A\, S\}\, g\,(a,\, x) :\equiv \mathsf{map}\,\{S\, a\}\, g\, x\\
&\mathsf{map}\,\{\delta\, A\, S\}\, g\,(f,\, x) :\equiv (g \circ f,\, \mathsf{map}\,\{S\,(el \circ f)\}\, g\, x)
\end{aligned}
$$

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

Notation. We overload El, intro and elim for IR and IIR types, but we will sometimes disambiguate them with a subscript, e.g. as $\mathsf{intro}_{\mathsf{IR}}$ or $\mathsf{intro}_{\mathsf{IIR}}$.

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$-\beta$. 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 short, the main idea is to represent IIR signatures as IR signatures together with a well-indexing predicate on algebras. First, we define an encoding function for signatures:

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

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 [Hancock et al. 2013, Section 6]. However, they only presented the translation of signatures, without the rest of the construction. Also, constructions and results for small IR do not generally transfer to our case of "large" IR.

Example 3.1. We compute the translation of the length-indexed vector signature from Example 2.2.

$$
\begin{aligned}
&\lfloor S \rfloor : \mathsf{Sig}\,(\mathsf{Nat} \times \top) \\
&\lfloor S \rfloor \equiv \sigma\,(\mathsf{Lift\;Tag})\,\$\,\lambda\, t.\, \textbf{case} \downarrow t\; \textbf{of} \\
&\quad \mathsf{Nil}' \quad\; \to \iota\,(\mathsf{zero},\, \mathsf{tt}) \\
&\quad \mathsf{Cons}' \to \sigma\,(\mathsf{Lift\;Nat})\,\$\,\lambda\, n.\, \sigma\,(\mathsf{Lift}\, A)\,\$\,\lambda\, \_. \\
&\qquad\qquad\quad \delta\,(\mathsf{Lift}\, \top)\,\$\,\lambda\, f.\, \sigma\,((x : \top) \to \mathsf{fst}\,(f\,(\uparrow x)) = (\downarrow n))\,\$\,\lambda\, p. \\
&\qquad\qquad\quad \iota\,(\mathsf{suc}\,(\downarrow n),\, \mathsf{tt})
\end{aligned}
$$

The first Nat component of the recursive result serves as the index. In the Cons′ case we have a single inductive field whose length is enforced with the extra $\sigma\,((x : \top) \to \mathsf{fst}\,(f\,(\uparrow x)) = n)$.

3.1 Type and Term Formers

We assume $S^* : \mathsf{Sig}_{\mathsf{IIR}}$ as a parameter to the constructions in the following. This is a "fixed" signature that we aim to construct IIR rules for. We need to distinguish this signature from other "varying" signatures that will appear in definitions and we will do induction on.

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. We define the IIR and El rules as follows.

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

The following definition describes the data that we get when we peel off an intro from an $\mathsf{IIR}\,i$ value.

$$\begin{aligned}
&\lfloor\mathbb{E}\rfloor : \mathsf{Sig}_{\mathsf{IIR}} \to I \to \mathsf{U}_{\max(i,k)}\\
&\lfloor\mathbb{E}\rfloor\, S\, i :\equiv (x : \mathbb{E}\,\lfloor S\rfloor\,(\mathsf{IR}\,\lfloor S^*\rfloor)\,\mathsf{El}) \times \mathsf{fst}\,(\mathbb{F}\,x) = i
\end{aligned}$$

Now, we can show that $\lfloor\mathbb{E}\rfloor\, S\, i$ is equivalent to $\mathbb{E}\,S\,\mathsf{IIR}\,\mathsf{El}\,i$, by induction on S. The induction is straightforward and we omit it here. We name the components of the equivalence as follows:[1]

$$\begin{aligned}
\overrightarrow{\mathbb{E}} &: \mathbb{E}\,S\,\mathsf{IIR}\,\mathsf{El}\,i \to \lfloor\mathbb{E}\rfloor\,S\,i\\
\overleftarrow{\mathbb{E}} &: \lfloor\mathbb{E}\rfloor\,S\,i \to \mathbb{E}\,S\,\mathsf{IIR}\,\mathsf{El}\,i\\
\eta &: (x : \mathbb{E}\,S\,\mathsf{IIR}\,\mathsf{El}\,i) \to \overleftarrow{\mathbb{E}}\,(\overrightarrow{\mathbb{E}}\,x) = x\\
\epsilon &: (x : \lfloor\mathbb{E}\rfloor\,S\,i) \qquad\to \overrightarrow{\mathbb{E}}\,(\overleftarrow{\mathbb{E}}\,x) = x\\
\tau &: (x : \mathbb{E}\,S\,\mathsf{IIR}\,\mathsf{El}\,i) \to \mathsf{ap}\,\overrightarrow{\mathbb{E}}\,(\eta\,x) = \epsilon\,(\overrightarrow{\mathbb{E}}\,x)
\end{aligned}$$

This is a half adjoint equivalence [Univalent Foundations Program 2013, Section 4.2]. The coherence witness τ will be shorty required for rearranging some transports. Next, we show that the two $\mathbb{F}$ operations are the same, modulo the previous equivalence, again by induction on S.

$$\lfloor\mathbb{F}\rfloor : \{S : \mathsf{Sig}_{\mathsf{IIR}}\}\{x : S_0\,\mathsf{IIR}\,\mathsf{El}\,i\} \to \mathsf{tr}\,O\,(\mathsf{snd}\,(\overrightarrow{\mathbb{E}}\,x))\,(\mathsf{snd}\,(\mathbb{F}\,(\mathsf{fst}\,(\overrightarrow{\mathbb{E}}\,x)))) = \mathbb{F}\,x$$

This lets us define the other introduction rules as well.

$$\begin{aligned}
&\mathsf{intro} : \mathbb{E}\,S^*\,\mathsf{IIR}\,\mathsf{El}\,i \to \mathsf{IIR}\,i && \mathsf{El\text{-}intro} : \mathsf{El}\,(\mathsf{intro}\,x) = \mathbb{F}\,x\\
&\mathsf{intro}\,x :\equiv (\mathsf{intro}_{\mathsf{IR}}\,(\mathsf{fst}\,(\overrightarrow{\mathbb{E}}\,x)),\ \mathsf{snd}\,(\overrightarrow{\mathbb{E}}\,x)) && \mathsf{El\text{-}intro} :\equiv \lfloor\mathbb{F}\rfloor
\end{aligned}$$

3.2 Elimination

We assume a level l for the elimination target. We aim to define the following:

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

Recall that $x : \mathsf{IIR}\,i$ is given as a pair of some $x : \mathsf{IR}\,\lfloor S^*\rfloor$ and $p : \mathsf{fst}\,(\mathsf{El}\,x) = i$. The idea is to use IR elimination on $x : \mathsf{IR}\,\lfloor S^*\rfloor$ while adjusting both P and f to operate on the appropriate data. For P, we generalize the induction goal over a well-indexing 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

$$\mathsf{elim}_{\mathsf{IR}}\,\lfloor P\rfloor : ((x : \mathbb{E}\,\lfloor S^*\rfloor\,(\mathsf{IR}\,\lfloor S^*\rfloor)\,\mathsf{El}) \to \mathsf{IH}\,\lfloor P\rfloor\,x \to \lfloor P\rfloor\,(\mathsf{intro}\,x)) \to (x : \mathsf{IR}\,\lfloor S^*\rfloor) \to \lfloor P\rfloor\,x.$$

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

$$\overleftarrow{\mathsf{IH}} : \{x : \lfloor\mathbb{E}\rfloor\,S\,i\} \to \mathsf{IH}\,\lfloor P\rfloor\,(\mathsf{fst}\,x) \to \mathsf{IH}\,P\,(\overleftarrow{\mathbb{E}}\,x)$$

[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 [Univalent Foundations Program 2013, Section 4.2].

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

$$\lfloor f \rfloor : (x : \mathbb{E}\,\lfloor S^{*} \rfloor\,(\mathsf{IR}\,\lfloor S^{*} \rfloor)\,\mathsf{El}) \to \mathsf{IH}\,\lfloor P \rfloor\,x \to \lfloor P \rfloor\,(\mathsf{intro}\,x)$$
$$\lfloor f \rfloor\,x\,ih\,p :\equiv \mathsf{tr}\,\big(\lambda\,(x,\,p).\,P\,(\mathsf{intro}\,x,\,p)\big)\,\big(\epsilon\,(x,\,p)\big)\,\big(f\,(\overleftarrow{\mathbb{E}}\,(x,\,p))\,(\overleftarrow{\mathsf{IH}}\,ih)\big)$$

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:

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

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

$$\begin{aligned}
&\mathsf{tr}\,(\lambda\,(x,\,p).\,P\,(\mathsf{intro}\,x,\,p))\\
&\quad(\epsilon\,(\overrightarrow{\mathbb{E}}\,x))\\
&\quad(f\,(\overleftarrow{\mathbb{E}}\,(\overrightarrow{\mathbb{E}}\,x))\,(\overleftarrow{\mathsf{IH}}\,(\mathsf{map}\,(\lambda\,x\,p.\,\mathsf{elim}\,P\,f\,(x,\,p))\,(\mathsf{fst}\,(\overrightarrow{\mathbb{E}}\,x)))))
\end{aligned}$$

Next, we prove by induction on S that map commutes with $\overrightarrow{\mathbb{E}}$:

$$\lfloor \mathsf{map} \rfloor : \{f\}\{x\} \to \mathsf{tr}\,(\mathsf{IH}\,P)\,(\eta\,x)\,\big(\overleftarrow{\mathsf{IH}}\,(\mathsf{map}\,f\,(\mathsf{fst}\,(\overrightarrow{\mathbb{E}}\,x)))\big) = \mathsf{map}\,(\lambda\,(x,\,p).\,f\,x\,p)\,x$$

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

$$f\,x\,\Big(\mathsf{tr}\,(\mathsf{IH}\,P)\,(\eta\,x)\,\big(\overleftarrow{\mathsf{IH}}\,(\mathsf{map}\,(\lambda\,x\,p.\,\mathsf{elim}\,P\,f\,(x,\,p))\,(\mathsf{fst}\,(\overrightarrow{\mathbb{E}}\,x)))\big)\Big)$$

This is 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 $\epsilon\,(\overrightarrow{\mathbb{E}}\,x)$, while we have $\eta\,x$ on the right. However, we have $\tau\,x : \mathsf{ap}\,\overrightarrow{\mathbb{E}}\,(\eta\,x) = \epsilon\,(\overrightarrow{\mathbb{E}}\,x)$, 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 3 in roughly 250 lines of Agda, using the same definitions as in this section, and the same notation, as far as Agda's syntax allows. For the assumption of IR, we verbatim reproduced the specification in Section 2.2, 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 [Cockx et al. 2021]; the two versions are the same except that rewrite rules have a performance cost in type checking and evaluation that we prefer to avoid.

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 2.1. 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, and we also write definitions with :=.
- Universe levels ω and $\omega + 1$, where $\mathrm{Set}_\omega : \mathrm{Set}_{\omega+1}$ and $\mathrm{Set}_{\omega+1}$ is a "proper type" that is not contained in any universe. Set_ω and $\mathrm{Set}_{\omega+1}$ are also closed under basic type formers.
- IR types (thus IIR types as well) in Set_i when i is finite.
- An internal type of finite universe levels. This is similar to Agda's internal type of finite levels, called Level [The Agda Team 2025]. The reason for this feature is the following. The object theory has countable levels represented as natural numbers, and 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 [Altenkirch and Kaposi 2016; Kovács 2023], to be described in Section 4.2.

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 2.1 and IR types as described in Section 2.2. More formally, the object theory is given as a quotient inductive-inductive type. The sets, operations and equations that we give in the following constitute the inductive signature.

4.2.1 Core substitution calculus. The basic judgmental structure is given as a category with families (CwF) [Castellan et al. 2019; Dybjer 1995] 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 bind 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:

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

  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, i.e. 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, i.e. 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 CwF combinators and De Bruijn indices get very hard to read when we get to more complicated rules like those in the specification of IR types. So we develop notations and conventions that are more tailored to our situation.

- 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 inter-derivable, 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 context). For example, if $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, then $\mathsf{Id}[\sigma] = \mathsf{Id}$ specifies stability under substitution for the identity type former. Hence, we shall omit substitution rules in the following.

4.2.4 Sigma types. We have

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

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

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.

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

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

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

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

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

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

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. As to the rest of the specification, we already saw it in Section 2.2 so we only give a short summary.

- $\mathbb{E}$, $\mathbb{F}$, IH and map are defined by SigElim and they satisfy the same definitional equations as in Section 2.2.
- IR, El, intro and 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 a 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 [Kovács 2023, Chapter 4]. 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 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.

We aim to define the following functions by induction on object syntax.

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

It is a proof-relevant logical predicate interpretation, an instance of a construction called *Artin gluing* [Artin et al. 1971, Exposé 4, Section 9.5] or *categorical gluing*. The concrete formulation that we use is the type-theoretic gluing by Kaposi et al. [2019a]. This is parameterized by a weak morphism between models of the object type theory. In our case, we take this morphism to be the global sections functor between the syntax and the standard Set model. This means that we build predicates over closed substitutions and closed terms. Coquand's canonicity proof also uses the same definitions as ours [Coquand 2019]. The type-theoretic gluing is a variation of gluing 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.

In the interpretation for types, note that i appears in $\mathsf{Ty}\,\Gamma\,i$ as a natural number, but it gets implicitly converted to a metatheoretic universe level in Set_i. We will keep omitting the conversion.

4.3.1 Interpretation of the CwF and the basic type formers. This was described in previously mentioned works [Coquand 2019; Kaposi et al. 2019a]. Additionally, the code supplement [Kaposi et al. 2019b] to [Kaposi et al. 2019c] has an Agda formalization of the canonicity model with the exact same universe setup and basic type formers that we use. Therefore we only present the parts here which are relevant to the interpretation of IR types. The interpretation of empty and extended contexts is as follows.

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

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 $(\gamma,\,\alpha)$: this is justified, since all values of $\mathsf{Sub}\,(\Gamma \triangleright A)$ are uniquely given as a pairing (similarly to pattern matching notation for plain Σ-types). The other CwF operations are as follows.

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

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 $f\,\alpha : \mathsf{Tm}\,B[\gamma,\,\alpha]$. We can also derive the interpretation of binary applications: $(t\,u)^\circ\,\gamma^\circ$ is computed to $t^\circ\,\gamma^\circ\,(u^\circ\,\gamma^\circ)$.

For **Σ-types**, we have

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

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

4.3.2 Interpretation of IR signatures. Signatures are given as an ordinary inductive family, 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.

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

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 definitions that only refer to closed terms.
(2) Then, we "contextualize" the definitions to get interpretations of object-theoretic rules.

In this section, the bulk of the work is step (1) and step (2) is fairly trivial. In step (1), we use $-^\circ$ to mark semantic definitions and we do not need to refer to $-^\circ$ as a family of interpretation functions on the object syntax. In step (2) we do overload $-^\circ$ but hope that this does not generate too much confusion.

4.3.3 Interpretation of IR types. The basic idea is that for each IR type, the corresponding canonicity predicate should be defined as a metatheoretic IIR type. This gets rather technical, so first let us look at an informal example for a concrete IR type.

Example 4.1. Consider the Agda code example in Section 1. We assume that it is a concrete "native" IR type in the object theory, and present the canonicity predicate for it as a metatheoretic

IIR type. First, the specification of the object-theoretic IR type, disregarding the eliminator:

$$
\begin{array}{llll}
\mathsf{Code} & : \mathsf{Tm}\,\Gamma\,\mathsf{U}_0 & \mathsf{El} : \mathsf{Tm}\,\Gamma\,(\mathsf{Code} \to \mathsf{U}_0) & \\
\mathsf{Nat}' & : \mathsf{Tm}\,\Gamma\,\mathsf{Code} & \mathsf{El}\,\mathsf{Nat}' & = \mathsf{Nat} \\
\Pi' & : \mathsf{Tm}\,\Gamma\,((A : \mathsf{Code}) \to (\mathsf{El}\,A \to \mathsf{Code}) \to \mathsf{Code}) & \mathsf{El}\,(\Pi'\,A\,B) & = (a : \mathsf{El}\,A) \to \mathsf{El}\,(B\,a)
\end{array}
$$

We assume $\mathsf{Nat}^\circ : \mathsf{Tm}\,\mathsf{Nat} \to \mathsf{Set}_0$. The canonicity predicate is the following IIR type.

$$
\begin{array}{l}
\textbf{data}\;\mathsf{Code}^\circ : \mathsf{Tm}\,\mathsf{Code} \to \mathsf{Set}_0 \\
\quad \mathsf{Nat}'^\circ : \mathsf{Code}^\circ\,\mathsf{Nat}' \\
\quad \Pi'^\circ \quad : \{A : \mathsf{Tm}\,\mathsf{Code}\}(A^\circ : \mathsf{Code}^\circ\,A) \\
\qquad\qquad \{B : \mathsf{Tm}\,(\mathsf{El}\,A \to \mathsf{Code})\}(B^\circ : \{a : \mathsf{Tm}\,(\mathsf{El}\,a)\} \to \mathsf{El}^\circ\,A^\circ\,a \to \mathsf{Code}^\circ\,(B\,a)) \\
\qquad\qquad \to \mathsf{Code}^\circ\,(\Pi'\,A\,B)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{El}^\circ : \{t : \mathsf{Tm}\,\mathsf{Code}\} \to \mathsf{Code}^\circ\,t \to (\mathsf{Tm}\,(\mathsf{El}\,t) \to \mathsf{Set}_0) \\
\mathsf{El}^\circ\,\mathsf{Nat}'^\circ\,t \qquad\quad = \mathsf{Nat}^\circ\,t \\
\mathsf{El}^\circ\,(\Pi'^\circ\,A^\circ\,B^\circ)\,f = \{a : \mathsf{Tm}\,A\} \to \mathsf{El}^\circ\,A^\circ\,a \to \mathsf{El}^\circ\,B^\circ\,(f\,a)
\end{array}
$$

This definition is sufficient to prove canonicity for Code as a concrete IR type. The task in this section is to do the same construction generically for all IR types.

We proceed to the semantic definitions. 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. Like in Section 3, we view S^* as a "fixed" top-level signature, in contrast to "varying" signatures that we will encounter in constructions. We will do induction on such varying signatures, but the induction must happen on *sub-signatures* of S^*, instead of arbitrary signatures.

Definition 4.2 (Paths to sub-signatures of S^*). We define an inductive family indexed over $S : \mathsf{Tm}\,(\mathsf{Sig}_i\,O)$ 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 S 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 with type $\mathsf{Tm}\,(A \to O)$.

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

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 $\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El}$:

$$
\begin{array}{l}
\mathsf{push} : \mathsf{Path}\,S^\circ \to \mathsf{Tm}\,(\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El}) \to \mathsf{Tm}\,(\mathbb{E}\,S^*\,(\mathsf{IR}\,S^*)\,\mathsf{El}) \\
\mathsf{push}\;\mathsf{here} \qquad\qquad\;\; t := t \\
\mathsf{push}\,(\mathsf{in\text{–}}\sigma\,p\,\{a\}\,a^\circ)\,t := \mathsf{push}\,p\,(a,\,t) \\
\mathsf{push}\,(\mathsf{in\text{–}}\delta\,p\,\{f\}\,f^\circ)\,t := \mathsf{push}\,p\,(f,\,t)
\end{array}
$$

We also show that this operation preserves $\mathbb{F}$, so we have $\mathbb{F}\,(\mathsf{push}\,p\,t) = \mathbb{F}\,t$.

Definition 4.3 (Encoding for signatures). This is only possible by induction on $\mathsf{Sig}^\circ\,S$, as we have no appropriate induction principle for $\mathsf{Tm}\,(\mathsf{Sig}_i\,O)$. As we recurse into a signature, we 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.

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

Some remarks:

- 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 $\mathsf{El} : \mathsf{Tm}\,(\mathsf{IR}\,S^* \to O)$ to the funtion type $\mathsf{Tm}\,(\mathsf{IR}\,S^*) \to \mathsf{Tm}\,O$.
- 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 $\mathsf{intro}\,(\mathsf{push}\,p\,\mathsf{tt}) : \mathsf{Tm}\,(\mathsf{IR}\,S^*)$. If we apply El to this term, it computes to $\mathbb{F}\,(\mathsf{push}\,p\,\mathsf{tt})$, which is the same as $\mathbb{F}\,\{\iota\,o\}\,\mathsf{tt}$, 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 $f : \mathsf{Tm}\,(A \to \mathsf{IR}\,S^*)$, then we use δ to specify inductive witnesses for all "subtrees" that are obtained by applying f to canonical terms.

Example 4.4. Let S^* be the signature from Example 2.1. It depends on the Tag and Nat types, so we assume evident $-^\circ$ interpretations for them. Now, $S^* : \mathsf{Tm}\,(\mathsf{Sig}_0\,\mathsf{U}_0)$ is a closed term that does not refer to any IR type or term former, so we can already fully compute the $-^\circ$ operation on it, obtaining $S^{*\circ} : \mathsf{Sig}^\circ\,(\lambda\,A.\,\mathsf{Tm}\,A \to \mathsf{Set}_0)\,S^*$. Then, we compute the following.

$$
\begin{aligned}
&\lfloor S^{*\circ} \rfloor\,\mathsf{here} : \mathsf{Sig}_{\mathsf{IIR}}\,(\mathsf{Tm}\,(\mathsf{IR}\,S^*))\,(\lambda\,t.\,\mathsf{Tm}\,(\mathsf{El}\,t) \to \mathsf{Set}_0)\\
&\lfloor S^{*\circ} \rfloor\,\mathsf{here} = \sigma\,(\mathsf{Tm}\,\mathsf{Tag})\,\$\,\lambda\,t.\,\sigma\,(\mathsf{Tag}^\circ\,t)\,\$\,\lambda\,t^\circ.\,\textbf{case}\;t^\circ\;\textbf{of}\\
&\quad \mathsf{Nat}'^\circ \to \iota\,(\mathsf{intro}\,(\mathsf{Nat}',\,\mathsf{tt}))\,\mathsf{Nat}^\circ\\
&\quad \Pi'^\circ \quad \to \sigma\,(\mathsf{Tm}\,(\top \to \mathsf{IR}\,S^*)) \qquad \$\,\lambda\,A.\,\delta\,((a : \mathsf{Tm}\,\top) \times \top) \qquad\qquad (\mathsf{fst} \circ A)\,\$\,\lambda\,\{ElA\}\,ElA^\circ.\\
&\qquad\qquad \sigma\,(\mathsf{Tm}\,(ElA\,\mathsf{tt} \to \mathsf{IR}\,S^*))\,\$\,\lambda\,B.\,\delta\,((a : \mathsf{Tm}\,(ElA\,\mathsf{tt})) \times ElA^\circ\,a)\,(\mathsf{fst} \circ B)\,\$\,\lambda\,\{ElB\}\,ElB^\circ.\\
&\qquad\qquad \iota\,\,(\mathsf{intro}\,(\Pi',\,A,\,B,\,\mathsf{tt}))\,(\lambda\,f.\,\{a : \mathsf{Tm}\,(ElA\,\mathsf{tt})\} \to ElA^\circ\,a \to ElB^\circ\,(f\,a))
\end{aligned}
$$

This specifies essentially the same IIR type that we had in Example 4.1, with some extra noise in the first argument of Π', which is now represented as a function with $\top$ domain.

Definition 4.5 (Interpretation of IR *and* El*).* This time around, encoded signatures get us precisely what we want:

$$
\begin{aligned}
&\mathsf{IR}^\circ : \mathsf{Tm}\,(\mathsf{IR}\,S^*) \to \mathsf{Set}_i &&\mathsf{El}^\circ : \{t : \mathsf{Tm}\,(\mathsf{IR}\,S^*)\} \to \mathsf{IR}^\circ\,t \to O^\circ\,(\mathsf{El}\,t)\\
&\mathsf{IR}^\circ := \mathsf{IIR}\,(\lfloor S^{*\circ} \rfloor\,\mathsf{here}) &&\mathsf{El}^\circ := \mathsf{El}_{\mathsf{IIR}}
\end{aligned}
$$

Definition 4.6 (Interpretation of intro*).* For this, we need to show an equivalence between two representations of IR°'s data, somewhat similarly to as in Section 3.1. For intro, we only need one component map of the equivalence, but later we will need all of it.

First, we define the predicate interpretations of $\mathbb{E}$ and $\mathbb{F}$. The general form states that $\mathbb{E}$ and $\mathbb{F}$ preserve predicates, but we will only need the special case when the *ir* and *el* arguments are $\mathsf{IR}\,S^*$ and El respectively.

$$\begin{aligned}
&\mathbb{E}^\circ : \mathsf{Sig}^\circ\,S \to \mathsf{Tm}\,(\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El}) \to \mathsf{Set}_i\\
&\mathbb{F}^\circ : \{S^\circ : \mathsf{Sig}^\circ\,S\}\{t : \mathsf{Tm}\,(\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El})\} \to \mathbb{E}^\circ\,S^\circ\,t \to O^\circ\,(\mathbb{F}\,t)
\end{aligned}$$

Second, we define

$$\begin{aligned}
&\lfloor\mathbb{E}\rfloor : \{S^\circ : \mathsf{Sig}^\circ\,S\} \to \mathsf{Path}\,S^\circ \to \mathsf{Tm}\,(\mathsf{IR}\,S^*) \to \mathsf{Set}_i\\
&\lfloor\mathbb{E}\rfloor\,p\,t := (t' : \mathsf{Tm}\,(\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El})) \times ((\mathsf{intro}\,(\mathsf{push}\,p\,t') = t) \times \mathbb{E}^\circ\,S^\circ\,t').
\end{aligned}$$

Next, we show the following equivalence by induction on S°:

$$\lfloor\mathbb{E}\rfloor\,\{S^\circ\}\,p\,t \simeq \mathbb{E}\,(\lfloor S^\circ\rfloor\,p)\,\mathsf{IR}^\circ\,\mathsf{El}^\circ\,t$$

We write $\overrightarrow{\mathbb{E}}\,p$ for the map with type $\lfloor\mathbb{E}\rfloor\,\{S^\circ\}\,p\,t \to \mathbb{E}\,(\lfloor S^\circ\rfloor\,p)\,\mathsf{IR}^\circ\,\mathsf{El}^\circ\,t$ and $\overleftarrow{\mathbb{E}}\,p$ for its inverse. This lets us interpret intro.

$$\begin{aligned}
&\mathsf{intro}^\circ : \{t : \mathsf{Tm}\,(\mathbb{E}\,S^*\,(\mathsf{IR}\,S^*)\,\mathsf{El})\} \to \mathbb{E}^\circ\,S^{*\circ}\,t \to \mathsf{IR}^\circ\,(\mathsf{intro}\,t)\\
&\mathsf{intro}^\circ\,\{t\}\,t^\circ := \mathsf{intro}_{\mathsf{IIR}}\,(\overrightarrow{\mathbb{E}}\,\mathsf{here}\,(t,\,\mathsf{refl},\,t^\circ))
\end{aligned}$$

Definition 4.7 (Interpretation of El–intro*).* Similarly as in Section 3.1, we need to show that $\mathbb{F}$ is preserved by signature encoding. For this, we need to annotate Path with additional information. Recall that the current definition of Path is not quite the most general notion of paths in signatures, because the in–δ constructor restricts the stored syntactic functions to the form $\mathsf{El}\circ f : \mathsf{Tm}\,(A \to O)$, only storing $f : \mathsf{Tm}\,(A \to \mathsf{IR}\,S^*)$. This restriction is required for the definition of push, where we need to produce $\mathsf{Tm}\,(\mathbb{E}\,S^*\,(\mathsf{IR}\,S^*)\,\mathsf{El})$ as output.

Now we also need to restrict the f° witnesses in in–δ to the form $f^\circ \circ \mathsf{El}^\circ$, where $f^\circ : \{a : \mathsf{Tm}\,A\} \to A^\circ\,a \to \mathsf{IR}^\circ\,(f\,a)$. We define a predicate over Path that expresses this:

$$\mathsf{restrict} : \mathsf{Path}\,S^\circ \to \mathsf{Set}_{\max(i+1,\,j+1)}$$

This is required for the predicate interpretation of push, which is defined by induction on Path:

$$\mathsf{push}^\circ : (p : \mathsf{Path}\,S^\circ) \to \mathsf{restrict}\,p \to \mathbb{E}^\circ\,S^\circ\,t \to E^\circ\,S^{*\circ}\,(\mathsf{push}\,p\,t)$$

This operation is preserved by $\mathbb{F}^\circ$:

$$\mathbb{F}^\circ\,(\mathsf{push}^\circ\,p\,q\,t^\circ) = \mathbb{F}^\circ\,t^\circ$$

We use push° in the statement of $\lfloor\mathbb{F}\rfloor$, which we prove by induction on S°:

$$\lfloor\mathbb{F}\rfloor : \mathbb{F}\,(\overrightarrow{\mathbb{E}}\,p\,t^\circ) = \mathbb{F}^\circ\,\{S^\circ\}\,(\mathsf{push}^\circ\,p\,q\,t^\circ)$$

Finally, we define $\mathsf{El\text{–}intro}^\circ$:

$$\begin{aligned}
&\mathsf{El\text{–}intro}^\circ : \{t\}(t^\circ : \mathbb{E}^\circ\,S^{*\circ}\,t) \to \mathsf{El}^\circ\,(\mathsf{intro}^\circ\,t^\circ) = \mathbb{F}^\circ\,t^\circ\\
&\mathsf{El\text{–}intro}^\circ := \lfloor\mathbb{F}\rfloor\,\mathsf{here}\,\mathsf{tt}
\end{aligned}$$

Above, tt witnesses the restriction of "here", which is trivial (since "here" does not contain in–δ).

Definition 4.8 (Interpretation of elim*).* We assume the following parameters to elimination:

$$\begin{aligned} k &: \mathsf{Nat} \\ P &: \mathsf{Tm}\,(\mathsf{IR}\,S^* \to \mathsf{U}_k) \\ P^\circ &: \{t\} \to \mathsf{IR}^\circ\,t \to \mathsf{Tm}\,(P\,t) \to \mathsf{Set}_k \end{aligned}$$

We define the predicate interpretations for IH and map first, specializing the *ir* and *el* arguments to $\mathsf{IR}\,S^*$ and El and the target level to k.

$$\begin{aligned} &\mathsf{IH}^\circ \quad : \mathbb{E}^\circ\,S^\circ\,t \to \mathsf{Tm}\,(\mathsf{IH}\,P\,t) \to \mathsf{Set}_{\max(i,\,k)} \\ &\mathsf{map}^\circ : \{f : \mathsf{Tm}\,((x : \mathsf{IR}\,S^*) \to P\,x)\}(f^\circ : \{t\}(t^\circ : \mathsf{IR}^\circ\,t) \to P^\circ\,t^\circ\,(f\,t))\{t\}(t^\circ : \mathbb{E}^\circ\,S^\circ\,t) \\ &\qquad\quad \to \mathsf{IH}^\circ\,t^\circ\,(\mathsf{map}\,f\,t) \end{aligned}$$

We also assume the induction method and its canonicity witness as parameters:

$$\begin{aligned} f &: \mathsf{Tm}\,((x : \mathbb{E}\,S^*\,(\mathsf{IR}\,S^*)\,\mathsf{El}) \to \mathsf{IH}\,P\,x \to P\,(\mathsf{intro}\,x)) \\ f^\circ &: \{t\}(t^\circ : \mathbb{E}^\circ\,S^{*\circ}\,t)\{ih\} \to \mathsf{IH}^\circ\,t^\circ\,ih \to P^\circ\,(\mathsf{intro}^\circ\,t^\circ)\,(f\,t\,ih) \end{aligned}$$

The goal is the following:

$$\mathsf{elim}^\circ : \{t : \mathsf{Tm}\,(\mathsf{IR}\,S^*)\}(t^\circ : \mathsf{IR}^\circ\,t) \to P^\circ\,t^\circ\,(\mathsf{elim}\,P\,f\,t)$$

We shall use IIR elimination on t° to give the definition. Again like in Section 3.2, we have to massage P° and f° to be able to pass them to $\mathsf{elim}_{\mathsf{IIR}}$. For the former, we have

$$\begin{aligned} &\lfloor P^\circ \rfloor : \{t\} \to \mathsf{IR}^\circ\,t \to \mathsf{Set}_k \\ &\lfloor P^\circ \rfloor\,\{t\}\,t^\circ := P^\circ\,t^\circ\,(\mathsf{elim}\,P\,f\,t). \end{aligned}$$

For the latter, we first define decoding for induction hypotheses, by induction on S°:

$$\overleftarrow{\mathsf{IH}} : \mathsf{IH}\,\lfloor P^\circ \rfloor\,t^\circ \to \mathsf{IH}^\circ\,\big(\mathsf{snd}\,(\mathsf{snd}\,(\overleftarrow{\mathbb{E}}\,p\,t^\circ))\big)\,\big(\mathsf{map}\,(\mathsf{elim}\,P\,f)\,(\mathsf{fst}\,(\overleftarrow{\mathbb{E}}\,p\,t^\circ))\big)$$

And define

$$\begin{aligned} &\lfloor f^\circ \rfloor : \{t\}(t^\circ : \mathbb{E}\,(\lfloor S^* \rfloor\,\mathsf{here})\,\mathsf{IR}^\circ\,\mathsf{El}^\circ\,t) \to \mathsf{IH}\,\lfloor P^\circ \rfloor\,t^\circ \to \lfloor P^\circ \rfloor\,(\mathsf{intro}\,t^\circ) \\ &\lfloor f^\circ \rfloor\,t^\circ\,ih^\circ := f^\circ\,\big(\mathsf{snd}\,(\mathsf{snd}\,(\overleftarrow{\mathbb{E}}\,\mathsf{here}\,t^\circ))\big)\,\big(\overleftarrow{\mathsf{IH}}\,\mathsf{here}\,ih^\circ\big). \end{aligned}$$

Hence, elimination is interpreted as follows:

$$\mathsf{elim}^\circ := \mathsf{elim}_{\mathsf{IIR}}\,\lfloor P^\circ \rfloor\,\lfloor f^\circ \rfloor$$

Definition 4.9 (Interpretation of elim–β*).* The goal is the following:

$$\mathsf{elim}\text{–}\beta^\circ : \{t\}(t^\circ : \mathbb{E}^\circ\,S^{*\circ}\,t) \to \mathsf{elim}^\circ\,(\mathsf{intro}^\circ\,t^\circ) = f^\circ\,t^\circ\,(\mathsf{map}^\circ\,\mathsf{elim}^\circ\,t^\circ)$$

The left hand side computes to the following:

$$f^\circ\,\Big(\mathsf{snd}\,\big(\mathsf{snd}\,(\overleftarrow{\mathbb{E}}\,\mathsf{here}\,(\overrightarrow{\mathbb{E}}\,\mathsf{here}\,(t,\,\mathsf{refl},\,t^\circ)))\big)\Big)\,\Big(\overleftarrow{\mathsf{IH}}\,\mathsf{here}\,\big(\mathsf{map}\,\mathsf{elim}^\circ\,(\overrightarrow{\mathbb{E}}\,\mathsf{here}\,(t,\,\mathsf{refl},\,t^\circ))\big)\Big)$$

The first argument to f° simplifies to t° by canceling $\overleftarrow{\mathbb{E}}$ and $\overrightarrow{\mathbb{E}}$. For the second argument, like in Section 3.2, we show that map appropriately commutes with encoding and decoding.

Remark. In the Agda formalization of elim–β, we use uniqueness of identity proofs (UIP) instead of trying to shuffle transports by homotopical reasoning. We do this mainly because it is easier. However, we conjecture that is possible skip UIP, and that might be useful in future applications and variations of our construction; see Section **??** for more discussion of this.

Definition 4.10 (Interpretation of object-theoretic IR rules). At this point we have semantic definitions that only mention closed terms. The final step is to generalize them to arbitrary contexts, thereby interpreting object-theoretic IR rules.

In the object theory we have $\mathsf{IIR} : \mathsf{Tm}\,\Gamma\,(\{O : \mathsf{U}_j\} \to \mathsf{Sig}_i\,O \to \mathsf{U}_i)$, hence by the specification of $-^\circ$ in Section 4.3, we need

$$\begin{gathered}\mathsf{IIR}^\circ : \{\gamma : \mathsf{Sub}\,\Gamma\}(\gamma^\circ : \Gamma^\circ\,\gamma)\{O : \mathsf{Tm}\,\mathsf{U}_j\}(O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_j)\\ \{S : \mathsf{Tm}\,(\mathsf{Sig}_i\,O)\}(S^\circ : \mathsf{Sig}^\circ\,O^\circ\,S) \to \mathsf{Tm}\,(\mathsf{IIR}\,S) \to \mathsf{Set}_i.\end{gathered}$$

We have previously defined a semantic IIR° with the following type (including all parameters):

$$\mathsf{IIR}^\circ : \{O : \mathsf{Tm}\,\mathsf{U}_j\}(O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_j)\{S : \mathsf{Tm}\,(\mathsf{Sig}_i\,O)\}(S^\circ : \mathsf{Sig}^\circ\,O^\circ\,S) \to \mathsf{Tm}\,(\mathsf{IIR}\,S) \to \mathsf{Set}_i$$

Hence, the contextual definition is just a constant function, i.e. $\mathsf{IIR}^\circ\,\{\gamma\}\,\gamma^\circ := \mathsf{IIR}^\circ$.

We also need to interpret stability under substitution. In the object theory we have $(\mathsf{IIR}\,\{\Gamma\})[\sigma] = \mathsf{IIR}\,\{\Delta\}$ when $\sigma : \mathsf{Sub}\,\Delta\,\Gamma$. Hence, we need to show $((\mathsf{IIR}\,\{\Gamma\})[\sigma])^\circ = (\mathsf{IIR}\,\{\Delta\})^\circ$ here. Computing this type further and applying function extensionality, we have the goal $\mathsf{IIR}^\circ\,(\sigma^\circ\,\gamma^\circ) = \mathsf{IIR}^\circ\,\gamma^\circ$, which is by definition $\mathsf{IIR}^\circ = \mathsf{IIR}^\circ$ for the non-contextual IIR° definition, and thus holds trivially. Every other IR rule and substitution rule is interpreted similarly, as constant functions and trivial equations. El–intro and elim–β are dispatched by the equations that we have already proved.

For an example, we look at El–intro. In the object theory, we have El–intro : $\mathsf{El}\,(\mathsf{intro}\,x) = \mathbb{F}\,x$, so we need to show $(\mathsf{El}\,(\mathsf{intro}\,x))^\circ\,\gamma^\circ = (\mathbb{F}\,x)^\circ\,\gamma^\circ$. Using the definition of $-^\circ$ for functions and the definitions of $\mathsf{El}^\circ\,\gamma^\circ$, $\mathsf{intro}^\circ\,\gamma^\circ$ and $\mathbb{F}^\circ\,\gamma^\circ$, this is computed to

$$\mathsf{El}^\circ\,(\mathsf{intro}^\circ\,(x^\circ\,\gamma^\circ)) = \mathbb{F}^\circ\,(x^\circ\,\gamma^\circ)$$

which is an instance of Definition 4.7. We omit describing the other rules. This concludes the canonicity interpretation of the object theory.

Example 4.11. Now that we have fully defined the $-^\circ$ functions that act on contexts, types, subtitutions and terms, we can look at an example for a canonicity statement. Recall S from Example 2.1 and also the corresponding IIR predicate from Example 4.4. Assuming $t : \mathsf{Tm}\,\bullet\,(\mathsf{IR}\,S)$, we have

$$\begin{aligned}&t^\circ\,\{\mathsf{id}\}\,\mathsf{tt} : (\mathsf{IR}\,S)^\circ\,\mathsf{tt}\ t[\mathsf{id}]\\ &t^\circ\,\{\mathsf{id}\}\,\mathsf{tt} : \mathsf{IR}^\circ\,(S^\circ\,\mathsf{tt})\,t\\ &t^\circ\,\{\mathsf{id}\}\,\mathsf{tt} : \mathsf{IIR}\,(\lfloor S^\circ\,\mathsf{tt}\rfloor\,\mathsf{here})\,t.\end{aligned}$$

Now, $\mathsf{IIR}\,(\lfloor S^\circ\,\mathsf{tt}\rfloor\,\mathsf{here})\,t$ expresses that t is either definitionally equal to $\mathsf{intro}\,(\mathsf{Nat}', \mathsf{tt})$ or to $\mathsf{intro}\,(\Pi', A, B, \mathsf{tt})$ for some A and B.

4.4 Mechanization

We formalized the semantic constructions in Sections 4.3.2 and 4.3.3 in Agda, but we did not formalize the object theory nor the contextual interpretations for its rules. It amounts to roughly 250 lines.

We used a *shallow embedding* for closed terms, where we work with the standard Set model of the object theory instead of an exact specification of the object syntax.

- We represent $\mathsf{Tm}\,U_i$ as Set_i.
- Given $A : \mathsf{Tm}\,U_i$, which is represented simply as $A : \mathsf{Set}_i$, we represent $\mathsf{Tm}\,A$ as A.

In other words, instead of working with sets of closed terms, we work with arbitrary metatheoretical sets. Hence, the formalization looks like an "internal" logical predicate interpretation of IR types using IIR types, where both IR and IIR types are assumed inside Agda. For example, instead of having

$\mathsf{Sig}_i\,O : \mathsf{Tm}\,\mathsf{U}_{\max(i+1,\,j)}$ and $\mathsf{Sig}^\circ\,O^\circ : \mathsf{Tm}\,(\mathsf{Sig}_i\,O) \rightarrow \mathsf{Set}_{\max(i+1,\,j)}$, we have $\mathsf{Sig}_i\,O : \mathsf{Set}_{\max(i+1,\,j)}$ and $\mathsf{Sig}^\circ\,O^\circ : \mathsf{Sig}_i\,O \rightarrow \mathsf{Set}_{\max(i+1,\,j)}$. Clearly, this is not a precise embedding and it is possible to do “illegal” constructions.

However, checking the legality of constructions is not difficult, and we only have to check a meager 250 lines. We could have used a more principled representation of closed terms, for example, following Kaposi et al. [2019c] where Agda's module system is used to prevent illegal constructions. We did not do so, again because of the small size of the project and because the additional safety features degrade the ergonomics of working in Agda. Concretely, we need to check the following in the formalization.

(1) “Stage separation”: we cannot eliminate from object types to meta-level types, e.g. cannot make a case distinction on an object-theoretic term with type Bool to compute something in the metatheory. Also, we cannot index object types by meta-types, or store meta-level data in object-level type or term formers.
(2) We can convert from $\mathsf{Tm}\,((a : A) \rightarrow B\,a)$ to $(a : \mathsf{Tm}\,A) \rightarrow \mathsf{Tm}\,(B\,a)$ but not the other way around.

Additionally, there is a difference in sizing: if $A : \mathsf{Tm}\,\mathsf{U}_i$ then $\mathsf{Tm}\,A : \mathsf{Set}_0$, while using shallow embedding we have $A : \mathsf{Set}_i$. This does not actually impact the formalization, because the sizes of $-^\circ$ components are computed from syntactic levels in any case.

5 Related Work

6 Conclusion and Future Work

7 TODO

- Cite specific pages of the Agda docs

References

Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. 2023. A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized. *Proc. ACM Program. Lang.* 7, ICFP (2023), 920–954. doi:10.1145/3607862

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2018. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.* 2, POPL (2018), 23:1–23:29. doi:10.1145/3158111

Thorsten Altenkirch and Ambrus Kaposi. 2016. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodik and Rupak Majumdar (Eds.). ACM, 18–29. doi:10.1145/2837614.2837638

Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. 1971. *Theorie de Topos et Cohomologie Etale des Schemas I, II, III.* Lecture Notes in Mathematics, Vol. 269, 270, 305. Springer.

Ana Bove and Venanzio Capretta. 2001. Nested General Recursion and Partiality in Type Theory. In *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2152)*, Richard J. Boulton and Paul B. Jackson (Eds.). Springer, 121–135. doi:10.1007/3-540-44755-5_10

Ana Bove, Peter Dybjer, and Ulf Norell. 2009. A Brief Overview of Agda - A Functional Language with Dependent Types. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 73–78. doi:10.1007/978-3-642-03359-9_6

Edwin C. Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *J. Funct. Program.* 23, 5 (2013), 552–593. doi:10.1017/S095679681300018X

Edwin C. Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference) (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:26. doi:10.4230/LIPICS.ECOOP.2021.9

Simon Castellan, Pierre Clairambault, and Peter Dybjer. 2019. Categories with Families: Unityped, Simply Typed, and Dependently Typed. *CoRR* abs/1904.00827 (2019). arXiv:1904.00827 http://arxiv.org/abs/1904.00827

Jonathan Chan and Stephanie Weirich. 2025. Bounded First-Class Universe Levels in Dependent Type Theory. *CoRR* abs/2502.20485 (2025). arXiv:2502.20485 doi:10.48550/ARXIV.2502.20485

Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. 2021. The taming of the rew: a type theory with computational assumptions. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–29. doi:10.1145/3434341

Thierry Coquand. 2019. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.* 777 (2019), 184–191. doi:10.1016/j.tcs.2019.01.015

Larry Diehl. 2017. *Fully Generic Programming over Closed Universes of Inductive-Recursive Types.* Ph. D. Dissertation. Portland State University.

Peter Dybjer. 1994. Inductive Families. *Formal Aspects Comput.* 6, 4 (1994), 440–465. doi:10.1007/BF01211308

Peter Dybjer. 1995. Internal Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers (Lecture Notes in Computer Science, Vol. 1158)*, Stefano Berardi and Mario Coppo (Eds.). Springer, 120–134. doi:10.1007/3-540-61780-9_66

Peter Dybjer. 2000. A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory. *J. Symb. Log.* 65, 2 (2000), 525–549. doi:10.2307/2586554

Peter Dybjer and Anton Setzer. 1999. A Finite Axiomatization of Inductive-Recursive Definitions. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1581)*, Jean-Yves Girard (Ed.). Springer, 129–146. doi:10.1007/3-540-48959-2_11

Peter Dybjer and Anton Setzer. 2003. Induction-recursion and initial algebras. *Ann. Pure Appl. Log.* 124, 1-3 (2003), 1–47. doi:10.1016/S0168-0072(02)00096-9

Peter Dybjer and Anton Setzer. 2006a. Indexed induction-recursion. *J. Log. Algebraic Methods Program.* 66, 1 (2006), 1–49. doi:10.1016/J.JLAP.2005.07.001

Peter Dybjer and Anton Setzer. 2006b. Indexed induction-recursion. *J. Log. Algebraic Methods Program.* 66, 1 (2006), 1–49. doi:10.1016/j.jlap.2005.07.001

Peter G. Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. 2013. Small Induction Recursion. In *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7941)*, Masahito Hasegawa (Ed.). Springer, 156–172. doi:10.1007/978-3-642-38946-7_13

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

Ambrus Kaposi, Simon Huber, and Christian Sattler. 2019a. Gluing for Type Theory. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:19. doi:10.4230/LIPIcs.FSCD.2019.25

Ambrus Kaposi, András Kovács, and Nicolai Kraus. 2019b. Formalisations in Agda using a morally correct shallow embedding. https://bitbucket.org/akaposi/shallow/src/master/

Ambrus Kaposi, András Kovács, and Nicolai Kraus. 2019c. Shallow Embedding of Type Theory is Morally Correct. In *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11825)*, Graham Hutton (Ed.). Springer, 329–365. doi:10.1007/978-3-030-33636-3_12

András Kovács. 2022. Generalized Universe Hierarchies and First-Class Universe Levels. In *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference) (LIPIcs, Vol. 216)*, Florin Manea and Alex Simpson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:17. doi:10.4230/LIPIcs.CSL.2022.28

András Kovács. 2023. Type-Theoretic Signatures for Algebraic Theories and Inductive Types. *CoRR* abs/2302.08837 (2023). arXiv:2302.08837 doi:10.48550/ARXIV.2302.08837

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

Per Martin-Löf. 1984. *Intuitionistic type theory.* Studies in Proof Theory, Vol. 1. Bibliopolis. iv+91 pages.

Loïc Pujet and Nicolas Tabareau. 2023. Impredicative Observational Equality. *Proc. ACM Program. Lang.* 7, POPL (2023), 2171–2196. doi:10.1145/3571739

The Agda Team. 2025. Agda documentation. https://agda.readthedocs.io/en/v2.8.0-rc2/

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics.* https://homotopytypetheory.org/book, Institute for Advanced Study.