Canonicity for Indexed Inductive-Recursive Types

ANDRÁS KOVÁCS, University of Gothenburg, Sweden and Chalmers University of Technology, Sweden

We prove canonicity for a Martin-Löf type theory with 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.

CCS Concepts: • **Theory of computation** → **Type theory**.

Additional Key Words and Phrases: inductive-recursive types, canonicity

ACM Reference Format:
András Kovács. 2026. Canonicity for Indexed Inductive-Recursive Types. *Proc. ACM Program. Lang.* 10, POPL, Article 43 (January 2026), 29 pages. https://doi.org/10.1145/3776685

1 Introduction

Induction-recursion (IR) allows us to mutually define an inductive type and a function that acts on elements of the type. A common application of IR is to define custom universes or universe hierarchies inside a type theory. For an example, in the proof assistant Agda, we can use IR to define a universe that is closed under our choice of type formers:

```
mutual
  data Code : Set₀ where
    Nat′ : Code
    Π′   : (A : Code) → (El A → Code) → Code

  El : Code → Set₀
  El Nat′       = Nat
  El (Π′ A B) = (a : El A) → El (B a)
```

Here, Code is a type of codes of types which behaves as a custom Tarski-style universe. This universe, unlike the ambient universe Set_0, supports an induction principle and can be used to define type-generic functions. Induction-recursion also allows the definition of custom *universe hierarchies*, and greatly increases the proof-theoretic strength of predicative Martin-Löf type theory. Indexed induction-recursion (IIR) additionally allows indexing the inductive type over some type, which lets us define inductive-recursive predicates [Dybjer and Setzer 2006].

Author's Contact Information: András Kovács, University of Gothenburg, Gothenburg, Sweden and Chalmers University of Technology, Gothenburg, Sweden, andrask@chalmers.se.

ACM 2475-1421/2026/1-ART43
https://doi.org/10.1145/3776685

Indexed induction-recursion was first used in an implicit and informal way in Martin-Löf's proof of normalization for his intuitionistic type theory [Martin-Löf 1975]. Later, his specification of a Tarski-style universe was a more clear-cut example of induction-recursion [Martin-Löf 1984]. The general specification of IR was formalized by Dybjer [2000] and Dybjer and Setzer [1999, 2003, 2006], who also developed set-theoretic, realizability and categorical semantics.

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 [Benke et al. 2003; Diehl 2017].

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 formalization in the literature.

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. The meaning of canonicity for a term depends on its type. For inductive types, a canonical term is given by a constructor whose fields are also canonical (inductively). Canonical functions map canonical inputs to canonical outputs, and canonical terms of Σ-types have canonical projections. On a high level, canonicity for a type theory justifies evaluation of closed terms. The outline of our development is the following.

- 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, 2006]. 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.
- The Agda formalization is available in the paper's supplement [Kovács 2025].

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.

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 the definitionally unique inhabitant tt. We have $\mathsf{Bool} : \mathsf{U}_0$ for Booleans. We have intensional identity types, as $t = u : \mathsf{U}_i$ for $t : A : \mathsf{U}_i$ and $u : A : \mathsf{U}_i$. 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 defining $\mathsf{Lift}\,A$ to be $A \times \top_j$. 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 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.

Our specification is identical to what is termed $\mathsf{IR}_{\mathsf{elim}}$ in [Dybjer and Setzer 2003], except for changes of notation and a difference in the universe setup. We have countable universe levels, while Dybjer and Setzer use a logical framework presentation with three universes.[1]

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.

[1] In that setup, "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.

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.

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

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

2.2.2 Type and term formation. First, assuming $O : \mathsf{U}_j$, 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.

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

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

Above we use an Agda-like notation for specifying the implicit S argument to $\mathbb{F}$. Here, we marked the quantification of all implicit arguments to $\mathbb{E}$ and $\mathbb{F}$, but we shall omit more of them from now

on. As to the arguments i and j, the object theory does not support universe polymorphism, so this quantification formally happens in the metatheory. The introduction rules are the following.

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

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.

Example 2.2. We look at how Code, Nat′, Π′ and El from the Agda example in Section 1 are obtained from the general rules for IR.

- We take the signature S from Example 2.1.
- Code is defined as $\mathsf{IR}\, S$, and we immediately have $\mathsf{El} : \mathsf{Code} \to \mathsf{U}_0$.
- We have $\mathsf{intro} : \mathbb{E}\, S\, \mathsf{Code}\, \mathsf{El} \to \mathsf{Code}$. If we try to compute $\mathbb{E}\, S\, \mathsf{Code}\, \mathsf{El}$, we get $(t : \mathsf{Tag}) \times \mathbb{E}\, (S'\, t)\, \mathsf{Code}\, \mathsf{El}$, where S' is the function $\lambda\, t.\, \mathbf{case}\, t\, \mathbf{of}\, \ldots$ inside S. Hence, the computation of $\mathbb{E}$ gets stuck on the neutral t. However, as soon as we specify the tag, we can fully compute the types of the Code constructors. We define the constructors as mkNat′ and mkΠ′, in order to disambiguate them from their tags.

  $$\begin{aligned}
  &\mathsf{mkNat'} : \mathsf{Code} && \mathsf{mk\Pi'} : (A : \mathsf{Code}) \to (\mathsf{El}\, A \to \mathsf{Code}) \to \mathsf{Code} \\
  &\mathsf{mkNat'} :\equiv \mathsf{intro}\, (\mathsf{Nat'}, \mathsf{tt}) && \mathsf{mk\Pi'}\, A\, B :\equiv \mathsf{intro}\, (\Pi', (\lambda\, _.\, A),\, B,\, \mathsf{tt})
  \end{aligned}$$

  The constant function $(\lambda\, \_.\, A)$ is required because inductive branching is always specified with a function, and in this case we have unary branching as $\top \to \mathsf{Code}$.
- Let us look at El. We have the expected definitional equalities:

  $$\mathsf{El}\, \mathsf{mkNat'} \equiv \mathsf{Nat} \qquad \mathsf{El}\, (\mathsf{mk\Pi'}\, A\, B) \equiv ((a : \mathsf{El}\, A) \to \mathsf{El}\, (B\, a))$$

  These are obtained from El–intro and the definition of $\mathbb{F}$. Unfolding the mkNat′ case, we have

  $$\mathsf{El}\, \mathsf{mkNat'} \equiv \mathsf{El}\, (\mathsf{intro}\, (\mathsf{Nat'}, \mathsf{tt})) \equiv \mathbb{F}\, \{S\}\, (\mathsf{Nat'}, \mathsf{tt}) \equiv \mathbb{F}\, \{\iota\, \mathsf{Nat}\}\, \mathsf{tt} \equiv \mathsf{Nat}.$$

  For mkΠ′, we have the following:

  $$\begin{aligned}
  &\mathsf{El}\, (\mathsf{mk\Pi'}\, A\, B) \\
  &\equiv \mathsf{El}\, (\mathsf{intro}\, (\Pi', (\lambda\, _.\, A),\, B,\, \mathsf{tt})) \\
  &\equiv \mathbb{F}\, \{S\}\, (\Pi', (\lambda\, _.\, A),\, B,\, \mathsf{tt}) \\
  &\equiv \mathbb{F}\, \{\delta\, \top\, \$\, \lambda\, \mathit{ElA}.\, \delta\, (\mathit{ElA}\, \mathsf{tt})\, \$\, \lambda\, \mathit{ElB}.\, \iota\, ((a : \mathit{ElA}\, \mathsf{tt}) \to \mathit{ElB}\, a)\}\, ((\lambda\, _.\, A),\, B,\, \mathsf{tt}) \\
  &\equiv \mathbb{F}\, \{\delta\, (\mathsf{El}\, A)\, \$\, \lambda\, \mathit{ElB}.\, \iota\, ((a : \mathsf{El}\, A) \to \mathit{ElB}\, a)\}\, (B,\, \mathsf{tt}) \\
  &\equiv \mathbb{F}\, \{\iota\, ((a : \mathsf{El}\, A) \to \mathsf{El}\, (B\, a))\}\, \mathsf{tt} \\
  &\equiv (a : \mathsf{El}\, A) \to \mathsf{El}\, (B\, a)
  \end{aligned}$$

2.2.3 Elimination. We assume a universe level k for the size of the type into which we eliminate. We define two additional functions on signatures:

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

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

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.

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

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

For IIR types we follow the "general IIR"[2] specification by Dybjer and Setzer [2006], again differing only in 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.

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

The most important addition is the field $ixs : A \to I$ in δ: it stands for "indexes" and it specifies the index of each inductive subtree.

Example 2.3. 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 constantly $\top$ 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

[2] As opposed to the "restricted" specification, where the index of every constructor is determined by the first field.

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 $\mathsf{suc}\ n$.

Remark. Why not use a "proper" IIR example here with non-trivial recursive component? The reason is that the non-artifical examples that we know are fairly complicated. Dybjer and Setzer [2006] list three examples of IIR: Martin-Löf's computability predicates [Martin-Löf 1984], Bove and Capretta's termination predicates [Bove and Capretta 2001] and Palmgren's higher-order universes [Palmgren 1998]. None of these are simple, and other proper IIR examples that we know are similar to one of these. However, we will present the canonicity predicate for Code in Examples 4.1 and 4.4; this is a proper IIR type that can be viewed as a simple example of a computability predicate. We also expand on this example in the current paper's supplement, detailing the derivation of constructors and the elimination principle from the general IIR rules [Kovács 2025].

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

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

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

Note the transport in $\mathsf{tr}\ O\ x\ o$: this is necessary, since o has type $O\ ix'$ while the required type is $O\ ix$. Also note the pattern matching notation in $(\uparrow\ x)$: this makes sense since $\uparrow$ is a definitional isomorphism and we could unfold the pattern to $(x,\ \mathsf{tt})$. The type and term formation rules are the following.

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

Example 2.4. Let us obtain Agda-style Vec constructors from the IIR rules. We take S from Example 2.3. We define $\mathsf{Vec}\ A\ n : \mathsf{U}_0$ to be $\mathsf{IIR}\ S\ n$, where we appropriately instantiate A in S. The derived constructors are the following.

$$\begin{aligned}
&\mathsf{nil} : \mathsf{Vec}\ A\ \mathsf{zero} \qquad \mathsf{cons} : (n : \mathsf{Nat}) \to A \to \mathsf{Vec}\ A\ n \to \mathsf{Vec}\ A\ (\mathsf{suc}\ n)\\
&\mathsf{nil} :\equiv \mathsf{intro}\ (\mathsf{nil}',\ (\uparrow \mathsf{refl})) \qquad \mathsf{cons}\ n\ a\ as :\equiv \mathsf{intro}\ (\mathsf{cons}',\ n,\ a,\ (\lambda\ _.\ as),\ (\uparrow \mathsf{refl}))
\end{aligned}$$

In nil, the type of $(\uparrow \mathsf{refl})$ is $\mathsf{zero} = \mathsf{zero}$, and in cons its type is $\mathsf{suc}\ n = \mathsf{suc}\ n$. El and El–intro are trivial in this case.

Remark: this specification uses identity types to pin down the index for each constructor. This style has been popularized as "Fordism" or "fording" by Conor McBride, and it has been used to represent indexed inductive types, e.g. in observational type theories [Altenkirch and McBride 2006; Pujet et al. 2025] and in the Glasgow Haskell Compiler's implementation of generalized algebraic datatypes (GADTs) [Vytiniotis et al. 2011].

2.3.3 Elimination. IH, map and elimination are as follows. We assume a level l for the target type of elimination.

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

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

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

Notation. We overload Sig, $\mathbb{E}$, $\mathbb{F}$, El, intro, El–intro, IH, map, elim and elim–β 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 ($\mathbb{E}$, $\mathbb{F}$, IH, map). The task is to define IIR, $\mathsf{El}_{\mathsf{IIR}}$, $\mathsf{intro}_{\mathsf{IIR}}$, $\mathsf{El\text{–}intro}_{\mathsf{IIR}}$, $\mathsf{elim}_{\mathsf{IIR}}$ and $\mathsf{elim\text{–}}\beta_{\mathsf{IIR}}$. 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 $\mathsf{Sig}_{\max(i,k)}\, ((ix : I) \times O\, ix)$.

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:

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

There are two points of interest. First, the encoded IR signature has the recursive output type $(ix : I) \times O\, ix$, which lets us interpret $\iota\, ix\, o$ as $\iota\, (ix,\, o)$. 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 $((a : A) \to \mathsf{fst}\, (f\, (\uparrow a)) = ixs\, a)$. 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 2.3. We have a slight technical annoyance: similarly as in the computation of $\mathbb{E}$ in Example 2.2, the computation of $\lfloor - \rfloor$ gets stuck on an unknown Tag. Thus, $\lfloor S \rfloor$ is computed to $\sigma\,(\mathsf{Lift}\ \mathsf{Tag})\ \$\ \lambda\, t.\ \lfloor \mathbf{case}\ \downarrow t\ \mathbf{of}\ \ldots \rfloor$. Hugunin [2020] solved this issue by adding an extra rule to signatures which represents *binary* branching. This can be iterated to get finite branching, and it allows recursion to proceed into branches. We do not use the extra rule, because our results do not require it and we prefer to keep our formalization as simple as possible. For the sake of illustration, we present $\lfloor S \rfloor$ below in a form where we manually push $\lfloor - \rfloor$ into the branches of the Tag split.

$$
\begin{aligned}
&\lfloor S \rfloor : \mathsf{Sig}\ (\mathsf{Nat} \times \top)\\
&\lfloor S \rfloor \equiv \sigma\,(\mathsf{Lift}\ \mathsf{Tag})\ \$\ \lambda\, t.\ \mathbf{case}\ \downarrow t\ \mathbf{of}\\
&\quad \mathsf{Nil}' \quad \to \iota\,(\mathsf{zero},\ \mathsf{tt})\\
&\quad \mathsf{Cons}' \to \sigma\,(\mathsf{Lift}\ \mathsf{Nat})\ \$\ \lambda\, n.\ \sigma\,(\mathsf{Lift}\ A)\ \$\ \lambda\ \_.\\
&\qquad\qquad \delta\,(\mathsf{Lift}\ \top_0)\ \$\ \lambda\, f.\ \sigma\,((x : \top_0) \to \mathsf{fst}\,(f\,(\uparrow x)) = (\downarrow n))\ \$\ \lambda\, p.\\
&\qquad\qquad \iota\,(\mathsf{suc}\,(\downarrow n),\ \mathsf{tt})
\end{aligned}
$$

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 $\sigma\,((x : \top_0) \to \mathsf{fst}\,(f\,(\uparrow x)) = (\downarrow n))$. We also present the above signature as an Agda-style IR definition below. We omit type lifting for the sake of brevity.

$$
\begin{aligned}
&\mathbf{data}\ \mathsf{List} : \mathsf{U}_0\ \mathbf{where}\\
&\quad \mathsf{nil}\ \ : \mathsf{List}\\
&\quad \mathsf{cons} : (n : \mathsf{Nat}) \to A \to (as : \top \to \mathsf{List}) \to ((x : \top) \to \mathsf{fst}\,(\mathsf{El}\,(as\,x)) = n) \to \mathsf{List}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{El} : \mathsf{List} \to \mathsf{Nat} \times \top\\
&\mathsf{El}\,\mathsf{nil} \qquad\qquad :\equiv (\mathsf{zero},\ \mathsf{tt})\\
&\mathsf{El}\,(\mathsf{cons}\,n\,a\,as\,p) :\equiv (\mathsf{suc}\,n,\ \mathsf{tt})
\end{aligned}
$$

This definition is not length-indexed, so we use the name "List" for the inductive type.

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.

We give some intuition for the distinction of fixed and varying signatures. Consider the definition of $\mathbb{E}_{\mathsf{IIR}}$ in Section 2.3.2: it recurses on the signature argument while keeping the *ir* and *el* arguments unchanged on every recursive call. Also consider $\mathsf{intro}_{\mathsf{IIR}}$ which has type $\mathbb{E}\,S\,(\mathsf{IIR}\,S)\,\mathsf{El}\,ix \to \mathsf{IIR}\,S\,ix$. If we want to do constructions on $\mathbb{E}\,S\,(\mathsf{IIR}\,S)\,\mathsf{El}\,ix$ or prove properties about its values, if we do induction on S, the $(\mathsf{IIR}\,S)$ and El values remain unchanged in inductive steps. Hence, most constructions must be parameterized over two signatures, a "varying" S and a "fixed" S^*, and we can do induction on an S that appears in $\mathbb{E}\,S\,(\mathsf{IIR}\,S^*)\,\mathsf{El}\,ix$. From a different angle: as we iterate through the fields of a constructor, the signature that specifies the remaining fields changes, but the signature that specifies the type of inductive fields stays the same.

We proceed to the definition of IIR and El. 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.

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

Example 3.2. Continuing the vector example and instantiating the above definitions, we obtain the length-indexed definition:

$$\begin{aligned}&\mathsf{Vec} : \mathsf{Nat} \to \mathsf{U}_0 && \mathsf{El} : \mathsf{Vec}\, n \to \top\\ &\mathsf{Vec}\, n :\equiv (x : \mathsf{List}) \times \mathsf{fst}\, (\mathsf{El}\, x) = n && \mathsf{El}\, (x,\, p) :\equiv \mathsf{tr}\, (\lambda\, \_.\, \top)\, p\, (\mathsf{snd}\, (\mathsf{El}\, x))\end{aligned}$$

Let us continue towards the generic intro definition. First, the following definition describes the data that we get when we peel off an intro from an $\mathsf{IIR}\, ix$ value.

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

Now, we can show that $\lfloor \mathbb{E} \rfloor\, S\, ix$ is equivalent to $\mathbb{E}\, S\, \mathsf{IIR}\, \mathsf{El}\, ix$, by induction on S. The induction is straightforward and we omit it here. We name the components of the equivalence as follows:[3]

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

This is a half adjoint equivalence [Univalent Foundations Program 2013, Section 4.2]. The coherence witness τ will be shortly 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.

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

This lets us define the other introduction rules as well.

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

3.2 Elimination

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

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

Recall that $x : \mathsf{IIR}\, ix$ is given as a pair of some $x : \mathsf{IR}\, \lfloor S^* \rfloor$ and $p : \mathsf{fst}\, (\mathsf{El}\, x) = ix$. 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

[3] 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].

P, we generalize the induction goal over a well-indexing witness:

$$\lfloor P \rfloor : \mathsf{IR}\, \lfloor S^* \rfloor \to \mathsf{U}_{\max(k,l)}$$
$$\lfloor P \rfloor\, x :\equiv \{ix : I\}(p : \mathsf{fst}\,(\mathsf{El}\, x) = ix) \to P\,(x,\, p)$$

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:

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

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:

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

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:

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

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 $\mathsf{elim}{-}\beta$ 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 σ 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 [Kovács 2025], using the same definitions as in this section, and the same notation, as far as Agda's syntax allows. For the assumption of IR types, 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 termination 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. Second, we give a logical predicate interpretation which yields canonicity.

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 $\mathsf{Set}_\omega : \mathsf{Set}_{\omega+1}$ 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.
- 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 2025a]. 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 the sizes of logical 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 sensible, because we talk about strict computation and canonicity, so we want to be precise about definitional content. In the metatheory we can be more liberal.

4.1.2 Consistency of the metatheory. As it often happens in type theory, our specific mixture of features is not covered by any particular reference. However, it is reasonable to expect that previous works can be combined and generalized to obtain consistency for our metatheory.

- Dybjer and Setzer [2006] gave a set-theoretic model for IIR. Here, there is a universe of small types which supports IIR types, which is modeled using a Mahlo cardinal M, and a universe of large types that contains the type of signatures, which is modeled using an inaccessible cardinal above M. The construction is not committed to this specific setup, and it can be adapted to having a countable number of Mahlo cardinals and two inaccessible cardinals above the Mahlo cardinals, in order to model our universe hierarchy. Also, having an internal type for finite universe levels is naturally supported in the set-theoretic model.
- Regarding the object syntax as a quotient inductive-inductive type (QIIT), it is a small and finitary QIIT, which is constructible from fairly weak metatheoretic assumptions,[4] and the general construction was described with minor formal gaps by Kovács [2023]. A construction of a QIIT for a specific type theory was fully formalized in Agda by Brunerie and de Boer [2020]. Alternatively, we could view the object syntax as essentially algebraic [Palmgren and Vickers 2007] or generalized algebraic [Cartmell 1978]. We chose the QIIT because the QIIT literature explicitly describes the type-theoretic induction principle that we use.

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. This consists of the following.

- 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.
- Types indexed by external natural numbers as levels, 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:

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

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

[4] Concretely, two universes, UIP, function extensionality, proposition extensionality, quotients and natural numbers.

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 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 occurrences of the name are desugared to a De Bruijn index. We write Π-types with 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 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 definitionally 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.

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

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

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

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

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

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

Note the [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 The Interpretation 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, induction 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 use an Agda-like notation, where we specify the resulting *section* of the displayed model, which consists of some 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. In such cases we find it easier to use than diagrammatic reasoning.

In the interpretation of contexts, note that we return in $\mathsf{Sub}\,\Gamma \to \mathsf{Set}_\omega$. This is so that arbitrary semantic types (at finite levels) can fit inside semantic contexts. Additionally, using the formal induction principle for the QIIT, the type of the target of elimination would be $\mathsf{Set}_{\omega+1}$; hence the need for both Set_ω and $\mathsf{Set}_{\omega+1}$.

In the interpretation of types, note that i appears in $\mathsf{Ty}\,\Gamma\,i$ as a natural number, but it gets implicitly converted to a universe level in Set_i. This conversion requires that finite levels are given by an internal type in the metatheory.

Remark. In Agda it would be possible to have $-^\circ : (\Gamma : \mathsf{Con}) \to \mathsf{Sub}\,\Gamma \to \mathsf{Set}_{(\mathsf{size}\,\Gamma)}$, where we compute the size by recursion on Γ. In this case, the type of $-^\circ$ would live in Set_ω and we could drop the assumption of $\mathsf{Set}_{\omega+1}$. However, as far as we know, such "dependently sized" functions have not been described in type theory literature, so we stick to non-dependently sized ones.

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 which are relevant to the interpretation of IR types. The interpretation of empty and extended contexts is as follows.

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

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 inhabitants of $\mathsf{Sub}\,(\Gamma \triangleright A)$ are uniquely given as a pairing. The other CwF operations are as follows.

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

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 use first use the syntactic Russell equation to cast t to the type $\mathsf{Ty}\ \Gamma\ i$, then t° still has 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.

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

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

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

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 $\mathsf{Sig}_i : \mathsf{Tm}\ \Gamma\ ((O : \mathsf{U}_j) \to \mathsf{U}_{\mathsf{max}(i+1,\ j)})$, so we have

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

Hence, we define an inductive type in the metatheory that is parameterized by $O : \mathsf{Tm}\ \mathsf{U}_j$ and $O^\circ : \mathsf{Tm}\ O \to \mathsf{Set}_j$ 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.

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

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:

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

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:

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

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

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

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

The predicate has to be inductive-recursive, because we have to recursively compute predicates for types of the form $\mathsf{El}\,A$. Using this definition of canonicity for Code, we could proceed and prove canonicity for an object theory that supports Code. Our task in this section is to generalize this: we do the same construction generically for all IR types.

We proceed to the semantic definitions. We assume the following parameters: $O : \mathsf{Tm}\,\mathsf{U}_j$, $O^\circ : \mathsf{Tm}\,O \to \mathsf{Set}_j$, $S^* : \mathsf{Tm}\,(\mathsf{Sig}_i\,O)$ 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 now 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.[5] 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.

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

We give an informal explanation for why we need Path. Consider Code from Section 1 and its signature S in Example 2.1. We aim to produce the signature that corresponds to Code° in Example 4.1, by induction on S. Note the return type of Π'°, which is $\mathsf{Code}^\circ\,(\Pi'\,A\,B)$. This refers to A and B which are bound *previously* in the constructor type. Hence, as we recurse through S, we need to remember the constructor fields that we already passed, and we represent this data as a snoc-list, using Path.

We can push the terms contained in a path onto a term with type $\mathbb{E}\,S\,(\mathsf{IR}\,S^*)\,\mathsf{El}$, getting a term with type $\mathbb{E}\,S^*\,(\mathsf{IR}\,S^*)\,\mathsf{El}$:

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

Note that we have $f : \mathsf{Tm}\,(A \to \mathsf{IR}\,S^*)$ in $\mathsf{in}\text{–}\delta$, although we could have used a more general definition of sub-signatures with $f : \mathsf{Tm}\,(A \to O)$. The more specific version is necessary to make the definition of push well-typed. We also show that push preserves $\mathbb{F}$, so we have $\mathbb{F}\,(\mathsf{push}\,p\,t) = \mathbb{F}\,t$.

Definition 4.3 (Interpretation of signatures). We translate signatures by induction on $\mathsf{Sig}^\circ\,S$; this is the only viable way, since 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 push the path onto tt to build up the correct term for the constructor index.

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

Some remarks:

[5] Recall that signatures are given as an inductive type, and all inductive types can be viewed as well-founded trees (W-types) equipped with subtree relations.

- 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 $-^\circ$ interpretations for them. We also assume $-\$^\circ-$ as the evident predicate interpretation of $-\$-$. 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 action of $-^\circ$ on it:

$$\begin{aligned}
&S^{*\circ} : \mathsf{Sig}^\circ\,(\lambda\,A.\,\mathsf{Tm}\,A \to \mathsf{Set}_0)\,S^*\\
&S^{*\circ} = \sigma^\circ\,\mathsf{Tag}^\circ\,\$^\circ\,\lambda\,t^\circ.\,\mathbf{case}\,t^\circ\,\mathbf{of}\\
&\quad \mathsf{Nat}'^\circ \to \iota^\circ\,\mathsf{Nat}^\circ\\
&\quad \Pi'^\circ \quad \to \delta^\circ\,\top^\circ\,\$^\circ\,\lambda\,\{\mathit{ElA}\}\,\mathit{ElA}^\circ.\,\delta^\circ\,(\mathit{ElA}^\circ\,\mathsf{tt}^\circ)\,\$^\circ\,\lambda\,\{\mathit{ElB}\}\,\mathit{ElB}^\circ.\\
&\qquad\qquad \iota^\circ\,(\lambda\,f.\,\{a : \mathsf{Tm}\,(\mathit{ElA}\,\mathsf{tt})\}(a^\circ : \mathit{ElA}^\circ\,\mathsf{tt}^\circ\,a) \to \mathit{ElB}^\circ\,a^\circ\,(f\,a))
\end{aligned}$$

Note that the computation of $S^{*\circ}$ does not get stuck on neutral tags, because $-^\circ$ is a metatheoretic operation that computes on every construction in the object syntax. Next, we compute the following.

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

This specifies essentially the same IIR type that we have seen in Example 4.1. We get some extra noise in the first inductive argument of Π'°, which is now a function with domain $\mathsf{Tm}\,\top \times \top$, and some more noise in the second inductive argument, where we pack the inputs into a Σ-type. We do not get stuck on neutral tags here either, but for a different reason: we have function extensionality and equality reflection in the metatheory, so we are allowed to push computation under the t° splitting, up to meta-level definitional equality.

Definition 4.5 (Interpretation of IR *and* El*).* This time around, translated signatures directly yield the interpretation of IR and El, and we do not need to fix it up with additional constraints.

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

Definition 4.6 (Interpretation of $\mathbb{E}, \mathbb{F}$ and intro*).* To interpret intro, we need to show an equivalence between two representations of IR°'s data, somewhat similarly to 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.

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

Second, we define

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

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.

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

Definition 4.7 (Interpretation of El–intro*).* Similarly as in Section 3.1, we need to show that $\mathbb{F}$ is preserved by signature translation. 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:

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

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 El–intro°:

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

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.

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

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°:

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

And define

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

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$-\beta$*).* 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:

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

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 $\overrightarrow{\mathbb{E}}$.

Remark. In the Agda formalization of elim$-\beta$, we use uniqueness of identity proofs (UIP) instead of trying to shuffle transports by homotopical reasoning. UIP is available in the metatheory since we assume equality reflection. We use UIP mainly because it makes the formalization easier. However, we conjecture that it is possible skip UIP, and that might be useful in future applications and variations of our construction. See Section 6.1 for more discussion. We note that we only use UIP

in elim–β, and we make this point explicit in the formalization by parameterizing elim–β over UIP and otherwise using the `-without-K` option [The Agda Team 2025b].

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

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

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

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

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

We also need to interpret stability under substitution. In the object theory we have $(\mathsf{IR}\,\{\Gamma\})[\sigma] = \mathsf{IR}\,\{\Delta\}$ when $\sigma : \mathsf{Sub}\,\Delta\,\Gamma$. Hence, we need to show $((\mathsf{IR}\,\{\Gamma\})[\sigma])^\circ = (\mathsf{IR}\,\{\Delta\})^\circ$ here. Computing this type further and applying function extensionality, we have the goal $\mathsf{IR}^\circ\,(\sigma^\circ\,\gamma^\circ) = \mathsf{IR}^\circ\,\gamma^\circ$, which is by definition $\mathsf{IR}^\circ = \mathsf{IR}^\circ$ for the non-contextual IR° 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.

In a nutshell, contextualization is easy because every IR type former and term former is interpreted as a closed term in the semantics, which is then weakened to arbitrary contexts, and the action of substitution on closed terms is trivial. However, we note that this setup is made possible by having enough universes, that is, every type is contained in some universe, which allows us to use term constants with Π-types in the specification.

For an example, we look at El–intro. In the object theory, we have $\mathsf{El}\,(\mathsf{intro}\,x) = \mathbb{F}\,x$, 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.

4.4 The Canonicity Result

Now that we have fully defined the $-^\circ$ interpretation that acts on contexts, types, subtitutions and terms, we can take any closed term $t : \mathsf{Tm}\,\bullet\,A$ and get $t^\circ\,\{\mathsf{id}\}\,\mathsf{tt} : A^\circ\,\{\mathsf{id}\}\,\mathsf{tt}\,t$ as a witness of canonicity for t. The most interesting case is when A is of the form $\mathsf{IR}\,S$.

Example 4.11. Recall S from Example 2.1. Assuming $t : \mathsf{Tm}\,\bullet\,(\mathsf{IR}\,S)$, we have

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

Note that $(\lfloor S^\circ\,\mathsf{tt}\rfloor\,\mathsf{here})$ was previously shown in Example 4.4. Now, we can use the metatheoretic $\mathsf{elim}_{\mathsf{IIR}}$ on $t^\circ\,\{\mathsf{id}\}\,\mathsf{tt}$ to prove 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.

4.5 Algorithm Extraction

Is our proof constructive and can we extract an evaluation algorithm from it? We believe that it would not be difficult to write a program that follows the computation in our construction. Also, similar evaluators have been used in Agda and Idris for a while. However, we do not make this formally precise in this paper and leave it to potential future work. In short, we see two ways to establish constructivity more rigorously.

(1) We could exhibit a constructive model of the metatheory. As far as we know, this has not been done, since Dybjer and Setzer only developed classical set-theoretical models of IR.
(2) We could prove canonicity for the metatheory. To handle the IR types in the metatheory, we can reuse the canonicity proof in this paper. However, we would also need to handle the QIIT of the object theory, and currently there is no generic canonicity proof for QIITs, and there are also some formal gaps in the construction of finitary QIITs from more basic type formers [Kovács 2023].

4.6 Mechanization

We formalized the semantic constructions in Sections 4.3.2 and 4.3.3 in Agda [Kovács 2025], but we did not formalize the object theory nor the contextual interpretations for its rules. The formalization is roughly 280 lines. Overall, we have found the Agda formalization to be extremely useful and it was crucial for coming up with the right definitions in the first place.

We use 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 $\mathsf{Tm}\,\mathsf{U}_i$ as Set_i.
- Given $A : \mathsf{Tm}\,\mathsf{U}_i$, 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 $\mathsf{Sig}_i\,O : \mathsf{Tm}\,\mathsf{U}_{\mathrm{max}(i+1,\,j)}$ and $\mathsf{Sig}^\circ\,O^\circ : \mathsf{Tm}\,(\mathsf{Sig}_i\,O) \to \mathsf{Set}_{\mathrm{max}(i+1,\,j)}$, we have $\mathsf{Sig}_i\,O : \mathsf{Set}_{\mathrm{max}(i+1,\,j)}$ and $\mathsf{Sig}^\circ\,O^\circ : \mathsf{Sig}_i\,O \to \mathsf{Set}_{\mathrm{max}(i+1,\,j)}$. 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 280 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 make the ergonomics of Agda somewhat worse. 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 $\mathsf{Tm}\,((a : A) \to B\,a)$ to $(a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(B\,a)$ 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 $-^\circ$ results are computed from syntactic levels in any case.

5 Related Work & Discussion

5.1 Algebraic & Categorical Metatheory

In this paper we used a reduction-free algebraic specification of syntax. This style has become more widespread in recent years, with normalization proofs for cubical [Sterling and Angiuli 2021] and multimodal type theories [Gratzer 2022] and a canonicity proof for a type theory with internal parametricity [Altenkirch et al. 2024]. In all of these cases, an important motivation is to cut down on boilerplate by working in a setting where syntactic definitional equalities are respected by all metatheoretic constructions. This is achieved by identifying object-level conversion with metatheoretic equality. This is convenient as long as we only talk about properties that are stable under conversion, and canonicity and normalization are such properties. On the other hand, a) we need a different or more complex setup if we are interested in conversion-unstable semantics such as cost semantics b) in current proof assistants, it is very difficult to do complete machine-checked formalizations in the algebraic style, mainly because of the abundance of type dependencies and propositional equalities, and the resulting deluge of transports ("transport hell").

The latter issue is partially addressed by "shallow embedding", where we represent the syntax of the object theory with a chosen *strict model* of the theory, where all equations are metatheoretic definitional equalities [Kaposi et al. 2019c]. We use this in Section 4.6, pretending that the Set model of the object theory is the syntax. Additionally or alternatively, we can cut out more boilerplate by working in a logic where everything is stable under object-theoretic substitutions or variable renamings. Two frameworks for this are synthetic Tait computability [Sterling 2021] and internal sconing [Bocquet et al. 2023]. Either would be applicable in our canonicity proof, but our case is not very technically complicated so we do not think that it is worth to use the heavier machinery.

5.2 Canonicity (and Parametricity) for Inductive Types

In the reduction-free algebraic style, Kovács [2024] proved conservativity for a two-level type theory whose outer level supports countable predicative universes with W-types. This involves a logical relation model which implies canonicity of the outer level as a corollary. Taken together with Hugunin's construction of (non-indexed) inductive types from W-types [Hugunin 2020], this yields canonicity for inductive types. In turn, the construction of indexed inductive types from non-indexed ones seems plausible; it is just a special case of Section 3 in the current paper, since IIR is a clear-cut generalization of inductive families.

Using directed reductions, Goguen [1994] proved strong $\beta\eta$-normalization for a type theory that supports inductive families, one impredicative universe of propositions and one predicative universe. Canonicity evidently follows from strong normalization. Also, Werner [1994] proved strong $\beta\eta$-normalization for a calculus of inductive constructions with one impredicative and one predicative universe.

Parametric models and translations of inductive types are closely related to our work, which can be viewed as a specific parametric construction. Bernardy et al. [2010] described a translation from a type theory to itself which interprets types as relations. This is "binary" parametricity in contrast to our "unary" logical predicates, but switching the arity is fairly mechanical. They described the interpretation of inductive families as well, although without going into formal details of signatures. Pédrot and Tabareau [2018] described a parametric translation for inductive families as well, which was also implemented as a Rocq plugin. Univalent parametric translations have also been described for inductive families and implemented in Rocq [Cohen et al. 2024; Tabareau et al. 2018, 2021].

5.3 Induction-Recursion

Our two primary sources throughout the paper are [Dybjer and Setzer 2003] and [Dybjer and Setzer 2006] for IR and IIR. We include a bit more general discussion of IR in the following.

First, it is natural to ask if IR types are constructible from a particular "universal" IR type, similarly to how inductive families are constructible from W-types. We might hope that this would simplify the metatheory of IR. Unfortunately no such type is known. However, *Mahlo universes* have been studied as relatively simple specifications from which many examples of IR types are constructible [Setzer 2000]. An *external Mahlo universe* is specified by the existence of a particular IR type in the universe. Hence, external Mahlo-ness is implied by IR, but it is not known if all IR types are constructible in an external Mahlo universe. While working on this paper, we attempted such a construction but currently we can only conjecture that it is not possible. Kubánek [2025] proved canonicity for external Mahlo universes. This is essentially a special case of our proof, instantiated to Mahlo sub-universes as an IR type former.

Second, it is worth to note that the investigation of *predicative foundations of mathematics* has been an important motivation for IR. Indeed, IR is close to the current strength limit of constructive predicative mathematics. The notion of predicativity is somewhat subjective, but in one sense a theory is considered predicative if it is amenable to ordinal analysis. Setzer [2000] calculated the strength of Martin-Löf type theory with one external Mahlo universe. Dybjer and Setzer [2024] investigated the predicativity of Mahlo universes in the sense of Martin-Löf's "extended predicativity". For IR, the exact proof-theoretic strength is unknown, but it still seems to be in the realm of ordinal analysis, unlike impredicative foundational systems such as ZFC or Rocq with impredicative Prop.

6 Future Work

6.1 Generalizing the Logical Predicate Interpretation

We describe two ways in which our logical predicate interpretation could be reformulated.

First, as a *syntactic translation* from a type theory that supports IR types to another one (possibly the same theory); see e.g. Boulier et al. [2017] and Bernardy et al. [2010]. Syntactic translations require that definitional equations in the source language are mapped to definitional equations in the target language. In our current construction, this does not hold, because neutral signatures block El–intro and elim–β. We could fix this by switching to second-class signatures. That is, we would specify a *sort* of signatures instead of a type of signatures, disallowing computation of signatures by elimination and thereby ruling out neutral signatures. The resulting syntactic translation could be extended to a *univalent parametricity* translation. Univalent parametricity translations have numerous practical use cases, and are currently implemented in a Rocq plugin called Trocq [Cohen et al. 2024], but without IR types, since Rocq does not support IR.

Second, as a *model construction* which extends type-theoretic gluing with IR types. This generalizes the previously described syntactic translation. In this case, we need to further restrict signatures. In the previous syntactic translation, signatures are part of the object theory, and in the initial model (i.e. the syntax) we can do induction on them. But in an arbitrary model we cannot do such induction. Therefore, we modify the theory, so that each IR signature has an "arity" or "shape", as a piece of metatheoretical data which specifies the number of constructors and their fields (but not the types of fields). This lets us do induction on shapes of signatures in arbitrary models. This is a bit similar to *contextuality* of models of type theories, where we require that typing contexts are inductively generated by context extensions [Cartmell 1986; Castellan et al. 2019].

6.2 Normalization for IR Types

Showing normalization for IR types is the natural upgrade to our canonicity result. This entails computing *open* terms to normal forms. Normalization is one way to get decidability of conversion for the object theory, although we could aim directly for decidable conversion as well. Decidable conversion is required for decidable type checking, so its metatheoretical value is quite obvious.

For normalization, we expect that the easiest way is again to extend previous formalizations for Martin-Löf type theory, still in the algebraic style [Altenkirch and Kaposi 2017; Bocquet et al. 2023; Coquand 2019; Sterling 2021]. This is significantly more complicated than canonicity. The general strategy is to define logical predicates internally to presheaves over syntactic variable renamings, to abstract away the details of stability under renaming. Like in the current paper, we would need to compute an IIR predicate for each IR signature, but we would need to handle neutral signatures and terms in addition. Note that the IIR predicates would be *internal to presheaves*. So first we would need to construct presheaf models of IR types. This task already seems to be a bit more complicated than the canonicity construction in the current paper. Nevertheless, it seems to be manageable and we believe that it would be worth to pursue this line of research.

Acknowledgements

I thank Ondřej Kubánek for discussions and I thank the anonymous reviewers and Peter Dybjer for their feedback on draft versions of this paper. This work was supported by grant 2019.0116 of the Knut and Alice Wallenberg Foundation.

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, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. 2024. Internal parametricity, without an interval. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 2340–2369.

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

Thorsten Altenkirch and Ambrus Kaposi. 2017. Normalisation by Evaluation for Type Theory, in Type Theory. *Log. Methods Comput. Sci.* 13, 4 (2017). doi:10.23638/LMCS-13(4:1)2017

Thorsten Altenkirch and Conor McBride. 2006. Towards Observational Type Theory. http://www.strictlypositive.org/ott.pdf

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.

Marcin Benke, Peter Dybjer, and Patrik Jansson. 2003. Universes for Generic Programs and Proofs in Dependent Type Theory. *Nord. J. Comput.* 10, 4 (2003), 265–289.

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2010. Parametricity and dependent types. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, Paul Hudak and Stephanie Weirich (Eds.). ACM, 345–356. doi:10.1145/1863543.1863592

Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. 2023. For the Metatheory of Type Theory, Internal Sconing Is Enough. In *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy (LIPIcs, Vol. 260)*, Marco Gaboardi and Femke van Raamsdonk (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:23. doi:10.4230/LIPICS.FSCD.2023.18

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The Next 700 Syntactical Models of Type Theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs* (Paris, France) *(CPP 2017)*. ACM, New York, NY, USA, 182–194. doi:10.1145/3018610.3018620

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

Guillaume Brunerie and Menno de Boer. 2020. Formalization of the initiality conjecture. https://github.com/guillaumebrunerie/initiality

John Cartmell. 1978. *Generalised algebraic theories and contextual categories.* Ph. D. Dissertation. Oxford University.

John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic* 32 (1986), 209–243.

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

Cyril Cohen, Enzo Crance, and Assia Mahboubi. 2024. Trocq: proof transfer for free, with or without univalence. In *European Symposium on Programming*. Springer, 239–268.

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. 2006. 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. 2024. The extended predicative Mahlo universe in Martin-Löf type theory. *J. Log. Comput.* 34, 6 (2024), 1032–1063. doi:10.1093/LOGCOM/EXAD022

Healfdene Goguen. 1994. *A typed operational semantics for type theory.* Ph. D. Dissertation. University of Edinburgh, UK. https://hdl.handle.net/1842/405

Daniel Gratzer. 2022. Normalization for Multimodal Type Theory. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 2:1–2:13. doi:10.1145/3531130.3532398

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

András Kovács. 2024. Closure-Free Functional Programming in a Two-Level Type Theory. *Proc. ACM Program. Lang.* 8, ICFP (2024), 659–692. doi:10.1145/3674648

András Kovács. 2025. Supplement to the paper "Canonicity for Indexed Inductive-Recursive Types". doi:10.5281/zenodo.17429493

Ondřej Kubánek. 2025. *Canonicity of the Mahlo Universe.* Master's thesis. Chalmers University of Technology, Gothenburg. https://github.com/kubaneko/External-Mahlo-Canonicity/blob/main/Main.pdf

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.

Erik Palmgren. 1998. On universes in type theory. In *Twenty-five years of constructive type theory (Oxford Logic Guides, Vol. 36).* Oxford University Press, 191 – 204.

Erik Palmgren and Steven J. Vickers. 2007. Partial Horn logic and cartesian categories. *Ann. Pure Appl. Log.* 145, 3 (2007), 314–353. doi:10.1016/j.apal.2006.10.001

Pierre-Marie Pédrot and Nicolas Tabareau. 2018. Failure is Not an Option: An Exceptional Type Theory. In *Programming Languages and Systems: 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings 27.* Springer, 245–271.

Loïc Pujet, Yann Leray, and Nicolas Tabareau. 2025. Observational Equality Meets CIC. *ACM Trans. Program. Lang. Syst.* 47, 2 (2025), 6:1–6:35. doi:10.1145/3719342

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

Anton Setzer. 2000. Extending Martin-Löf Type Theory by one Mahlo-universe. *Arch. Math. Log.* 39, 3 (2000), 155–181. doi:10.1007/s001530050140

Jonathan Sterling. 2021. *First Steps in Synthetic Tait Computability.* Ph. D. Dissertation. Carnegie Mellon University Pittsburgh, PA.

Jonathan Sterling and Carlo Angiuli. 2021. Normalization for Cubical Type Theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021.* IEEE, 1–15. doi:10.1109/LICS52264.2021.9470719

Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2018. Equivalences for free: univalent parametricity for effective transport. *Proceedings of the ACM on Programming Languages* 2, ICFP (2018), 1–29.

Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. 2021. The marriage of univalence and parametricity. *Journal of the ACM (JACM)* 68, 1 (2021), 1–44.

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

The Agda Team. 2025b. Agda documentation on `-without-K`. https://agda.readthedocs.io/en/v2.8.0/language/without-k.html#without-k

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

Dimitrios Vytiniotis, Simon L. Peyton Jones, Tom Schrijvers, and Martin Sulzmann. 2011. OutsideIn(X) Modular type inference with local assumptions. *J. Funct. Program.* 21, 4-5 (2011), 333–412. doi:10.1017/S0956796811000098

Benjamin Werner. 1994. *Une Théorie des Constructions Inductives.* Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-00196524

Received 2025-07-08; accepted 2025-11-06