Directed proof-relevant logical relations in simplicial HoTT

RUNMING LI, Carnegie Mellon University, USA
HARRISON GRODIN, Carnegie Mellon University, USA
ROBERT HARPER, Carnegie Mellon University, USA

Intrinsically-typed presentations of type theory often use equality in the meta-language to represent object-language judgmental equality. In such equational syntax, proof-relevant logical relations define computability predicates on judgmental equivalence classes of types and terms. This approach, however, does not directly account for reduction, which is directed and plays a central role in many logical-relations arguments. This paper develops a directed version of proof-relevant logical relations in simplicial homotopy type theory, where reductions are internalized as *inequality types*. We construct object syntax as a directed quotient inductive type. The central observation is that contravariant families in simplicial type theory provide exactly the proof-relevant form of closure under expansion for logical relations: computability evidence can be transported backward along reductions, with the required functoriality and universal property built in. Using this observation, we construct a unary logical relations model with contravariant computability predicates and prove directed Boolean canonicity: every closed Boolean term reduces to either true or false. We then extend the construction to dependent types and universes, where a comonadic flat modality provides the discreteness needed for type conversion and universe predicates. Finally, we adapt the method to binary logical relations, separating vertical reduction from horizontal parametricity and obtaining a proof-relevant account of representation independence.

CCS Concepts: • **Theory of computation** → **Type theory**; **Constructive mathematics**; **Categorical semantics**.

1 INTRODUCTION

Logical relations begin with a simple idea: interpret each type by a family of computable terms, and interpret each type former by its action on such families. This is the pattern behind Tait's computability method [Tait 1967]. A product is computable when its projections are computable; a function is computable when it takes computable arguments to computable results. The fundamental theorem then states that every well-typed term is computable at its type. This method has become one of the standard tools of programming-language semantics, used to prove properties such as normalization, contextual equivalence, representation independence, and noninterference. The same idea also shapes foundational accounts of type theory itself. In the NuPRL tradition, a computational semantics of types—closely related to PER/logical-relations models—forms part of the basis on which the type system is justified [Allen 1987; Constable et al. 1986]. Related methods are also central to the separation-logic framework Iris [Jung et al. 2018; Timany et al. 2024].

In programming-language semantics, logical relations are often formulated relative to a reduction relation $\longrightarrow$ and its reflexive-transitive closure $\longrightarrow^*$. In that setting, computation has a direction: a term steps to, or reduces to, another term. A logical relation is then a family of predicates defined by induction on types; for each type A, the predicate $A^\bullet$ is the logical interpretation of A:

$$\begin{aligned}
(-)^\bullet\ (-) &: (A : \mathsf{Ty}) \to \mathsf{Tm}\ A \to \mathsf{Prop} \\
\mathsf{Bool}^\bullet\ (M) &:= (M \longrightarrow^* \mathsf{true}) \vee (M \longrightarrow^* \mathsf{false}) \\
(A \times B)^\bullet\ (P) &:= A^\bullet\ (\mathsf{fst}\ P) \wedge B^\bullet\ (\mathsf{snd}\ P) \\
(A \to B)^\bullet\ (F) &:= (M : A) \to A^\bullet\ (M) \to B^\bullet\ (\mathsf{app}\ F\ M).
\end{aligned}$$

Authors' addresses: Runming Li, runmingl@cs.cmu.edu, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA; Harrison Grodin, hgrodin@cs.cmu.edu, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA; Robert Harper, rwh@cs.cmu.edu, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA.

The Boolean predicate says that a closed Boolean is computable when it reduces to one of the canonical Booleans. The inductive argument commonly requires a closure under *expansion* lemma:

$$\text{if } M \longrightarrow M', \text{ then } A^\bullet\ (M') \Rightarrow A^\bullet\ (M).$$

For Booleans, the proof composes the step $M \longrightarrow M'$ with the reduction from M' to the chosen canonical Boolean. This lemma is needed to move computability backward along computation, keeping the fundamental theorem compatible with the transition.

A complementary route comes from category theory. Categorical gluing [Fiore 2002; Mitchell and Scedrov 1992] gives a *proof-relevant* account of logical relations in which the computability predicate is not merely a proposition, but a family of types whose inhabitants are computability witnesses carrying non-trivial structure. This form of logical relations is widely used in modern type theory semantics to prove canonicity, normalization, and parametricity for a range of type theories [Altenkirch and Kaposi 2017; Bocquet et al. 2023; Coquand 2018; Kaposi et al. 2019a].

In this approach, the syntactic component usually consists of judgmental equivalence classes of types and terms, rather than raw terms equipped with a transition system. For example, the product fragment contains constructors and equations of the following shape:

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{pair} & : & \mathsf{Tm}\ A \longrightarrow \mathsf{Tm}\ B \longrightarrow \mathsf{Tm}\ (A \times B) & \times_{\beta_1} & : & \mathsf{fst}\ (\mathsf{pair}\ M\ N) = M \\ \mathsf{fst} & : & \mathsf{Tm}\ (A \times B) \longrightarrow \mathsf{Tm}\ A & \times_{\beta_2} & : & \mathsf{snd}\ (\mathsf{pair}\ M\ N) = N \\ \mathsf{snd} & : & \mathsf{Tm}\ (A \times B) \longrightarrow \mathsf{Tm}\ B & \times_{\eta} & : & \mathsf{pair}\ (\mathsf{fst}\ P)\ (\mathsf{snd}\ P) = P. \end{array}$$

The logical relation assigns a proof-relevant predicate to each type. For example, the product predicate is defined by

$$\begin{array}{rcl} (-)^\bullet\ (-) & : & (A : \mathsf{Ty}) \longrightarrow \mathsf{Tm}\ A \longrightarrow \mathcal{U} \\ (A \times B)^\bullet\ (P) & \coloneqq & A^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ (\mathsf{snd}\ P). \end{array}$$

A term is computable when it inhabits the predicate at its type. For example, the computability evidence for a pair has the following type:

$$\begin{array}{rcl} (-)^\bullet & : & (M : \mathsf{Tm}\ A) \longrightarrow A^\bullet\ M \\ (\mathsf{pair}\ M\ N)^\bullet & : & (A \times B)^\bullet\ (\mathsf{pair}\ M\ N) \\ & = & A^\bullet\ (\mathsf{fst}\ (\mathsf{pair}\ M\ N)) \times B^\bullet\ (\mathsf{snd}\ (\mathsf{pair}\ M\ N)). \end{array}$$

Given $M : \mathsf{Tm}\ A$ and $N : \mathsf{Tm}\ B$, the induction hypotheses provide $M^\bullet : A^\bullet\ M$ and $N^\bullet : B^\bullet\ N$ as computability evidence for the components. The product β-equations identify fst and snd of $\mathsf{pair}\ M\ N$ with M and N. Thus the computability evidence for a pair is the pair of computability witnesses, modulo the two β-equations:

$$(\mathsf{pair}\ M\ N)^\bullet \coloneqq (M^\bullet,\ N^\bullet).$$

The projection evidence is obtained by extracting the relevant component:

$$(\mathsf{fst}\ P)^\bullet \coloneqq \pi_1(P^\bullet) \qquad (\mathsf{snd}\ P)^\bullet \coloneqq \pi_2(P^\bullet).$$

In particular, computability evidence respects equations. For example, in the case of $\times_{\beta_1}$, the computability evidence for $\mathsf{fst}\ (\mathsf{pair}\ M\ N)$ is equal to the computability evidence for M:

$$(\mathsf{fst}\ (\mathsf{pair}\ M\ N))^\bullet = \pi_1((\mathsf{pair}\ M\ N)^\bullet) = \pi_1(M^\bullet, N^\bullet) = M^\bullet.$$

Proof relevance matters here for two reasons. First, the computability of a universe should contain the computability predicate for each type in that universe. That extra structure is unavailable when computability predicates are proof-irrelevant. Second, the treatment of equations, such as the β-law above, requires a comparison of computability evidence. Gluing packages these obligations into one algebraic construction and the fundamental theorem becomes the construction of a model.

The two perspectives above emphasize different aspects of logical relations. Operational logical relations treat computation as directed structure: terms reduce, and computability must be closed backward along those reductions. Gluing, on the other hand, treats computability evidence as part of the semantics: equations must compare terms and the evidence. This paper asks whether these two aspects can coexist. The guiding question of this paper is therefore:

Can proof-relevant gluing for logical relations account for directed reductions?

1.1 From Equalities to Inequalities

The equational approach represents object-language judgmental equalities by meta-level equalities, in effect presenting the syntax as a quotient by those equations. This is convenient because equality in the meta-language already has the structural behavior expected of judgmental equality: reflexivity, transitivity, symmetry, and congruence. Compatibility with type and term formers is therefore inherited from meta-level congruence. For products, this looks as follows:

$$\frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash b \equiv b' : B}{\Gamma \vdash \mathsf{pair}\ a\ b \equiv \mathsf{pair}\ a'\ b' : A \times B} \qquad \begin{aligned}\mathsf{cong}_2\ \mathsf{pair} : a = a' \to b = b' \to \\ \mathsf{pair}\ a\ b = \mathsf{pair}\ a'\ b'\end{aligned}$$

Thus the left-hand compatibility rule is recovered from the right-hand meta-level congruence term.

The exact use of these equations depends on the chosen meta-language. In extensional type theory, equality reflection turns equality proofs into judgmental equalities. In intensional type theory and homotopy type theory, identity types or path types instead provide propositional equations, and constructions must transport along those equations.

Reduction does not fit in this equational story directly. Unlike judgmental equality, reduction is fundamentally directed. If representing judgmental equality calls for equality types in the meta-language, then representing reduction internally calls for an asymmetric analogue of equality. Directed type theories [Licata and Harper 2011] typically provide such structure in the form of homomorphism or inequality types. Among variants of directed type theories, this paper works in the simplicial homotopy type theory of Riehl and Shulman [2017], where inequality types $x \leq_A y$ serve as the directed counterpart of equality types. Their technical details are recalled later. For now, the important point is that inequalities are reflexive and monotone, like equalities, but lack symmetry.

In the equational case, the syntax is constructed intrinsically via quotient inductive-inductive presentation of type theory [Altenkirch et al. 2018; Altenkirch and Kaposi 2016; Kaposi et al. 2019b]. The present paper generalizes this idea by adding directed constructors to those quotients as well. A directed constructor generates an inequality rather than an equality. Thus, instead of representing product computation by quotienting with equations such as $\mathsf{fst}\ (\mathsf{pair}\ M\ N) = M$, an inequality type can *internalize* the reduction as:

$$\times_{\beta_1} \ : \ \mathsf{fst}\ (\mathsf{pair}\ M\ N) \leq M \qquad\qquad \times_{\beta_2} \ : \ \mathsf{snd}\ (\mathsf{pair}\ M\ N) \leq N.$$

With this syntax in place, the goal is a canonicity result by logical relations. For a closed Boolean term M, the desired theorem says that M reduces to a canonical Boolean:

$$M \leq \mathsf{true} \qquad \text{or} \qquad M \leq \mathsf{false}.$$

1.2 Contravariance as Proof-Relevant Expansion

Returning to the discussion of expansion, a computability predicate $A^\bullet$ should be able to transport evidence backward along reductions: each $f : M \leq M'$ should induce a map $f^* : A^\bullet\ (M') \to A^\bullet\ (M)$.

For a proof-irrelevant logical relation, such a map is sufficient. Proof relevance imposes a stronger requirement. As Section 3 will show, a bare map does not specify how transported evidence

behaves with respect to identities, composition, and the surrounding type structure. This is the familiar passage from proof-irrelevant to proof-relevant foundations: operations must come with coherence laws. In homotopy type theory, for example, path composition is not merely the map $- \bullet - : (x = y) \to (y = z) \to (x = z)$, but part of a coherent groupoid structure satisfying laws such as $\mathsf{refl} \bullet f = f$ and $f \bullet \mathsf{refl} = f$. The same phenomenon appears for logical relations: the expansion lemma is not merely a function $(M \le M') \to A^\bullet\ (M') \to A^\bullet\ (M)$; the map must be *universal* in the appropriate sense.

The key observation of this work is that simplicial type theory already provides this coherent form of expansion. Its notion of a *contravariant family* was developed to study synthetic fibrations, but it has exactly the structure needed for proof-relevant logical relations. At a high level, a contravariant family is a universal family equipped with backward transport along inequalities. In short:

Contravariance is proof-relevant closure under expansion.

The rest of the paper develops this observation into a directed version of proof-relevant logical relations. Each type is equipped with a contravariant computability predicate, and the resulting fundamental theorem transports computability evidence backwards along reductions.

1.3 Simplicial Homotopy Type Theory

The present work takes place in simplicial homotopy type theory [Riehl and Shulman 2017], a directed extension of homotopy type theory (HoTT) [Univalent Foundations Program 2013]. In HoTT, path type is proof-relevant equality: between two points, there may be many distinct paths.

1.3.1 Homotopy Levels. This higher path structure organizes types by homotopy level. A type is *contractible* when it has a distinguished point, called its center, to which every other point is equal; a type is a *proposition* when any two of its points are equal; and a type is a *set* when its path types are propositions, *i.e.* when any two paths between the same two points are equal:

$$\mathsf{isContr}\ A \ :\equiv \ \sum_{a:A} (x : A) \to a =_A x \qquad \mathsf{isProp}\ A \ :\equiv \ (x\ y : A) \to x =_A y$$
$$\mathsf{isSet}\ A \ :\equiv \ (x\ y : A) \to \mathsf{isProp}(x =_A y).$$

1.3.2 Equivalences. Equivalences between types can be expressed through contractible fibers. For a map $f : A \to B$, the fiber over $b : B$ records the preimages of b under f:

$$\mathsf{fib}_f(b) :\equiv \sum_{a:A} f(a) =_B b, \qquad \mathsf{isEquiv}(f) :\equiv (b : B) \to \mathsf{isContr}(\mathsf{fib}_f(b)).$$

Thus a map is an equivalence when every element of the codomain has a unique preimage, up to paths. This definition packages both inverse data and the coherence laws relating that inverse to the original map. The notation $A \simeq B$ means that such an equivalence $f : A \to B$ exists.

1.3.3 Higher Inductive Types. Homotopy type theory also provides higher inductive types (HITs). Unlike ordinary inductive types, a HIT may include both point constructors and path constructors. Two standard examples are the circle and set truncation:

$$\begin{array}{ll} \textbf{Inductive}\ S^1 : \mathcal{U}\ \textbf{where} & \textbf{Inductive}\ \|A\|_0 : \mathcal{U}\ \textbf{where} \\ \quad \mathsf{base} : S^1 & \quad |-| \quad\ : A \to \|A\|_0 \\ \quad \mathsf{loop} : \mathsf{base} =_{S^1} \mathsf{base} & \quad \mathsf{set}_{\|A\|_0} : \mathsf{isSet}\ \|A\|_0. \end{array}$$

The circle displays the path-constructor feature: in addition to the point base, the type contains a generated self-path loop at base. This path is part of the structure of the type, not merely $\mathsf{refl}_{\mathsf{base}}$. Set truncation uses higher constructors differently. The point constructor embeds each element of A into $\|A\|_0$, while $\mathsf{set}_{\|A\|_0}$ forces all path types of $\|A\|_0$ to be propositions. Thus $\|A\|_0$ retains the point-level information of A but forgets higher path information, leaving $\|A\|_0$ as a set. These path and

truncation constructors make HITs a natural way to present syntax modulo equations [Altenkirch and Kaposi 2016]: path constructors impose the quotient equations, and set truncation ensures that the quotient is a set rather than a higher type with additional path structure.

1.3.4 Simplicial HoTT. Simplicial homotopy type theory extends HoTT with a directed interval. Maps out of this interval behave as directed paths; the corresponding directed relation is written here as an inequality type $x \leq_A y$. These inequalities are asymmetric, because the directed interval has no reversal operation, but every function is guaranteed to be *monotone*, the directed analogue of congruent. They therefore provide the asymmetric analogue of path types needed to present judgmental reduction internally. The necessary definitions and theorems from simplicial type theory are recalled as needed throughout the paper; for a complete treatment, see Riehl and Shulman [2017]. The extension developed by Gratzer et al. [2026a] adds modalities from Gratzer [2023] to simplicial type theory. In particular, it includes the flat modality $\flat$ from crisp type theory [Licata et al. 2018; Shulman 2018], used in Section 4.

1.4 Contributions and Synopsis

The paper develops directed proof-relevant logical relations through the following contributions.

(1) **Directed quotient syntax.** Section 2 presents the syntax as a directed quotient inductive-inductive type, where reductions are modeled by directed inequalities.
(2) **Contravariance as proof-relevant expansion.** Section 3 identifies contravariance as the proof-relevant form of closure under expansion, requiring each computability predicate to carry a coherent contravariant structure.
(3) **Directed canonicity by gluing.** The unary logical relation of Section 3 proves directed Boolean canonicity: every closed Boolean term M satisfies either $M \leq \mathsf{true}$ or $M \leq \mathsf{false}$.
(4) **Mechanization.** As a proof of concept, Sections 2 and 3 are mechanized in Cubical Agda, including the directed syntax, the contravariance condition and its properties, and the logical relation model with products, functions, and Booleans. We mark definitions, lemmas, and constructions covered by the mechanization with 🤌 throughout the paper.
(5) **Universes and dependency.** Section 4 extends directed logical relations to universes and dependent types. The universe predicate requires the flat modality $\flat$ to establish contravariance.
(6) **Binary parametricity.** Section 5 adapts the construction to binary logical relations and parametricity, separating vertical reductions from horizontal parametricity witnesses and illustrating the result with a queue example.

2 MODELING REDUCTION IN SYNTAX

Typically the syntax of a type theory can be described as a signature in some logical framework. The judgmental structure and the type and term formers are constants in the signature, while judgmental equalities are represented using the equality notion supplied by the meta-language. One such presentation is the structure of a category with families (CwF) [Dybjer 1996].

The CwF part of the signature supplies the ambient judgments: contexts, substitutions, types in a context, and terms of a type in a context.

$$\begin{array}{lcllcl} \mathsf{Ctx} & : & \mathcal{U} & \mathsf{Ty} & : & \mathsf{Ctx} \to \mathcal{U} \\ \mathsf{Sub} & : & \mathsf{Ctx} \to \mathsf{Ctx} \to \mathcal{U} & \mathsf{Tm} & : & (\Gamma : \mathsf{Ctx}) \to \mathsf{Ty}\ \Gamma \to \mathcal{U} \end{array}$$

Here Ctx is the type of contexts, $\mathsf{Sub}\ \Delta\ \Gamma$ is the type of substitutions from context Γ to context Δ, $\mathsf{Ty}\ \Gamma$ is the type of types in context Γ, and $\mathsf{Tm}\ \Gamma\ A$ is the type of terms of type A in context Γ. Product types, for example, are then added as an extension of this CwF signature. The left half

below lists the type and term formers; the right half lists the judgmental equations.

$$\begin{array}{rcl} -\times- & : & \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \\ \mathsf{pair} & : & \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B \to \mathsf{Tm}\ \Gamma\ (A \times B) \\ \mathsf{fst} & : & \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ A \\ \mathsf{snd} & : & \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ B \end{array} \qquad \begin{array}{rcl} \times_{\beta_1} & : & \mathsf{fst}\ (\mathsf{pair}\ a\ b) = a \\ \times_{\beta_2} & : & \mathsf{snd}\ (\mathsf{pair}\ a\ b) = b \\ \times_{\eta} & : & \mathsf{pair}\ (\mathsf{fst}\ a)\ (\mathsf{snd}\ a) = a \end{array}$$

This is the equational presentation of the product fragment: the computational laws on the right are represented by equalities in the meta-language. The aim onward is to model judgmental reduction. Reduction is directed, so the equality constants above must be replaced by directed constants, represented internally by inequality types. The next step is therefore to recall the basic structure of simplicial type theory, which provides those inequalities.

2.0.1 Basic Structure of Simplicial Type Theory. Simplicial type theory extends HoTT [Univalent Foundations Program 2013] with one primitive object: a directed interval $\mathbb{2}$ with two endpoints $\mathsf{i0}$ and $\mathsf{i1}$. Its direction is part of the structure: $\mathbb{2}$ is a bounded order with $\mathsf{i0} \leq \mathsf{i1}$. A map out of $\mathbb{2}$ is a directed path in the target type.

Definition 2.1 (Inequality types). For $x, y : A$, we write

$$x \leq_A y \coloneqq \Sigma_{f:\mathbb{2}\to A}(f\ \mathsf{i0} = x) \times (f\ \mathsf{i1} = y)$$

for the inequality type of directed morphisms from x to y in A[1][2]. For every $x : A$, there is a reflexivity term $\mathsf{id}_x : x \leq_A x$ given by the constant path at x.

Based on this definition, inequalities propagate to type structures naturally; for example, inequality at dependent function types is an analog of the usual function extensionality.

Lemma 2.2 (Directed Function Extensionality - Riehl and Shulman 2017, Proposition 6.3). *For $f\ g : (x : A) \to B(x)$, the canonical map*

$$\begin{array}{rcl} (f \leq g) & \to & ((x : A) \to (f\ x \leq_{B(x)} g\ x)) \\ \alpha & \mapsto & \lambda x\ i.\ \alpha\ i\ x \end{array}$$

is an equivalence. The proof is identical to the proof of functional extensionality in cubical type theories [Angiuli et al. 2021; Cohen et al. 2018], although the interval here is different.

2.1 Judgmental Reduction as Directed Structure

Unlike identity types, inequality types are not symmetric. They nevertheless have the structural behavior needed to play the role of judgmental congruence. In particular, every function $f : A \to B$ is automatically monotone: it acts on inequalities functorially[3].

$$\begin{array}{l} \mathsf{mono}_f : (x \leq_A y) \to (f\ x \leq_B f\ y) \\ \mathsf{mono}_f\ h \coloneqq \lambda i.\ f\ (h\ i) \end{array}$$

Thus, to model judgmental reduction, we replace the equalities in the ordinary signature by inequalities. For product types, this turns the usual β rules into directed constructors. The left

[1] Riehl and Shulman [2017] defines inequality types using extension types, so that the endpoint conditions such as $f\ \mathsf{i0} = x$ are treated judgmentally. For this paper, it is enough to use the resulting inequality types and their expected structural principles, so we do not develop the machinery of extension types.

[2] We write $x \leq_A y$ rather than $\hom_A(x, y)$, which is common in accounts of simplicial type theory aimed at synthetic category theory [Gratzer et al. 2025b, 2026a,b; Riehl and Shulman 2017]. Here the notation emphasizes the role of these types as asymmetric replacements for equality in dependent type theories.

[3] Notationally we sometimes write mono_f simply as f for the functorial action.

column below writes these rules in the usual operational notation, while the right column gives their internal presentation as elements of inequality types:

$$\begin{array}{ll} \mathsf{fst}\ (\mathsf{pair}\ a\ b) \to_\beta a & \times_{\beta_1} : \mathsf{fst}\ (\mathsf{pair}\ a\ b) \le a \\ \mathsf{snd}\ (\mathsf{pair}\ a\ b) \to_\beta b & \times_{\beta_2} : \mathsf{snd}\ (\mathsf{pair}\ a\ b) \le b \end{array}$$

Congruence for reductions is then inherited from monotonicity. For instance, the congruence rule for fst is represented internally by applying mono to the projection function:

$$\frac{p \to_\beta p'}{\mathsf{fst}\ p \to_\beta \mathsf{fst}\ p'} \qquad \mathsf{mono}_{\mathsf{fst}} : p \le p' \to \mathsf{fst}\ p \le \mathsf{fst}\ p'$$

Apart from the computational rules represented as inequalities, the syntax retains the usual CwF structure; see, for example, Kaposi et al. [2019a]. In particular, the equations of the substitution calculus, together with the naturality equations for type and term formers, are still represented by equality in the meta-theory, not by inequalities. The ordinary CwF substitution structure, together with simple product types can be found below. We omit the standard CwF substitution equations such as $A[\tau \circ \sigma] = A[\tau][\sigma]$ for brevity. The boxed inequalities highlight the directed product reductions; the remaining laws are equalities. In the language of HoTT, these equations are generally paths rather than definitional equalities; strictly speaking, applying them requires transport along those paths. These transports are left implicit throughout the paper to avoid clutter.

CwF operations and context extension

$$\begin{array}{ll}
\mathsf{Ctx} & : \mathcal{U} \\
\mathsf{Ty} & : \mathsf{Ctx} \to \mathcal{U} \\
\mathsf{Sub} & : \mathsf{Ctx} \to \mathsf{Ctx} \to \mathcal{U} \\
\mathsf{Tm} & : (\Gamma : \mathsf{Ctx}) \to \mathsf{Ty}\ \Gamma \to \mathcal{U} \\
\mathsf{id} & : \mathsf{Sub}\ \Gamma\ \Gamma \\
- \circ - & : \mathsf{Sub}\ \Theta\ \Delta \to \mathsf{Sub}\ \Gamma\ \Theta \to \mathsf{Sub}\ \Gamma\ \Delta \\
\cdot & : \mathsf{Ctx} \\
\epsilon_\Gamma & : \mathsf{Sub}\ \Gamma\ \cdot \\
- \triangleright - & : (\Gamma : \mathsf{Ctx}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Ctx} \\
\mathsf{p} & : \mathsf{Sub}\ (\Gamma \triangleright A)\ \Gamma \\
\mathsf{q} & : \mathsf{Tm}\ (\Gamma \triangleright A)\ (A[\mathsf{p}]) \\
-[-] & : \mathsf{Ty}\ \Delta \to \mathsf{Sub}\ \Gamma\ \Delta \to \mathsf{Ty}\ \Gamma \\
-[-] & : \mathsf{Tm}\ \Delta\ A \to (\sigma : \mathsf{Sub}\ \Gamma\ \Delta) \to \mathsf{Tm}\ \Gamma\ (A[\sigma]) \\
(-, -) & : (\sigma : \mathsf{Sub}\ \Gamma\ \Delta) \to \mathsf{Tm}\ \Gamma\ (A[\sigma]) \to \mathsf{Sub}\ \Gamma\ (\Delta \triangleright A)
\end{array}$$

Product types and reductions

$$\begin{array}{ll}
- \times - & : \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \\
\mathsf{pair} & : \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B \to \mathsf{Tm}\ \Gamma\ (A \times B) \\
\mathsf{fst} & : \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ A \\
\mathsf{snd} & : \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ B \\
\times_{[]} & : (A \times B)[\sigma] = A[\sigma] \times B[\sigma] \\
\mathsf{pair}_{[]} & : (\mathsf{pair}\ a\ b)[\sigma] = \mathsf{pair}\ (a[\sigma])\ (b[\sigma]) \\
\mathsf{fst}_{[]} & : (\mathsf{fst}\ p)[\sigma] = \mathsf{fst}\ (p[\sigma]) \\
\mathsf{snd}_{[]} & : (\mathsf{snd}\ p)[\sigma] = \mathsf{snd}\ (p[\sigma]) \\
\times_{\beta_1} & : \boxed{\mathsf{fst}\ (\mathsf{pair}\ a\ b) \le a} \\
\times_{\beta_2} & : \boxed{\mathsf{snd}\ (\mathsf{pair}\ a\ b) \le b}
\end{array}$$

2.2 Initial Directed CwF via Directed Quotient Inductive-Inductive Types

Elements of the CwF signature are models of the type theory. In particular, the initial model gives the inductively generated syntax. To construct this syntax, the ordinary QIIT presentation of type theory [Altenkirch et al. 2018; Altenkirch and Kaposi 2016] is generalized to the directed setting. In HoTT, higher inductive types (HITs) generate not only points, but also paths between those points. A set-truncated HIT is a quotient inductive type (QIT). This is the mechanism used in the QIIT presentation of type theory to internalize judgmental equalities as paths in the syntax [Altenkirch and Kaposi 2016], where terms are quotiented by judgmental equalities. To internalize judgmental reductions, the inductive definition must also allow directed constructors. We call the resulting notion a *directed quotient inductive-inductive type* (directed QIIT).

A schematic directed-QIIT fragment for the product syntax is the following four mutually defined higher inductive types.

Inductive $\mathsf{Ctx} : \mathcal{U}$ **where**
- $\cdot \quad : \mathsf{Ctx}$
- $- \triangleright - \quad : (\Gamma : \mathsf{Ctx}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Ctx}$

Inductive $\mathsf{Sub} : \mathsf{Ctx} \to \mathsf{Ctx} \to \mathcal{U}$ **where**
- $\mathsf{id} \quad : \mathsf{Sub}\ \Gamma\ \Gamma$
- $- \circ - \quad : \mathsf{Sub}\ \Theta\ \Delta \to \mathsf{Sub}\ \Gamma\ \Theta \to \mathsf{Sub}\ \Gamma\ \Delta$
- $\dots$

Inductive $\mathsf{Ty} : \mathsf{Ctx} \to \mathcal{U}$ **where**
- $-[-] : \mathsf{Ty}\ \Delta \to \mathsf{Sub}\ \Gamma\ \Delta \to \mathsf{Ty}\ \Gamma$
- $\dots$
- $-\times- \quad : \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma$
- $\times_{[]} \quad : (A \times B)[\sigma] = A[\sigma] \times B[\sigma]$
- $\dots$

Inductive $\mathsf{Tm} : (\Gamma : \mathsf{Ctx}) \to \mathsf{Ty}\ \Gamma \to \mathcal{U}$ **where**
- $-[-] : \mathsf{Tm}\ \Delta\ A \to (\sigma : \mathsf{Sub}\ \Gamma\ \Delta) \to \mathsf{Tm}\ \Gamma\ (A[\sigma])$
- $\dots$
- $\mathsf{pair} \quad : \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B \to \mathsf{Tm}\ \Gamma\ (A \times B)$
- $\mathsf{fst} \quad : \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ A$
- $\mathsf{snd} \quad : \mathsf{Tm}\ \Gamma\ (A \times B) \to \mathsf{Tm}\ \Gamma\ B$
- $\dots$
- $\mathsf{pair}_{[]} : (\mathsf{pair}\ a\ b)[\sigma] = \mathsf{pair}\ (a[\sigma])\ (b[\sigma])$
- $\mathsf{fst}_{[]} : (\mathsf{fst}\ p)[\sigma] = \mathsf{fst}\ (p[\sigma])$
- $\mathsf{snd}_{[]} : (\mathsf{snd}\ p)[\sigma] = \mathsf{snd}\ (p[\sigma])$
- $\dots$
- $\times_{\beta_1} \quad : \boxed{\mathsf{fst}\ (\mathsf{pair}\ a\ b) \le a}$
- $\times_{\beta_2} \quad : \boxed{\mathsf{snd}\ (\mathsf{pair}\ a\ b) \le b}$
- $\dots$

Schematically, four aspects of this definition are worth noting.

2.2.1 Directed Higher Inductive Types. The resulting inductive definition is a mixture of point constructors, such as pair and fst, equality constructors, such as $\times_{[]}$ and $\mathsf{fst}_{[]}$, and inequality constructors, such as $\times_{\beta_1}$ and $\times_{\eta}$. The first two kinds are standard in higher inductive types, but directed constructors in the same inductive definition is worth justifying. Directed quotient inductive types are not new to this work. In a directed type theory setting, Grodin et al. [2024, Section 4.5.1] already introduce an instance of a directed HIT. The role of the present section is to explain how to use the same directed infrastructure for the initial syntax of the object theory.

The key point is that a directed constructor is syntactic sugar for ordinary higher constructors involving the directed interval. By Definition 2.1, a directed constructor $c : a \le_A b$ expands into a point constructor c_{rel} and two equality constructors c_{left} and c_{right} as follows:

Inductive $A : \mathcal{U}$ **where**	$\rightsquigarrow$	**Inductive** $A : \mathcal{U}$ **where**
$\dots$		$\dots$
$c : a \le_A b$		$c_{\mathsf{rel}} : \mathbb{2} \to A$
		$c_{\mathsf{left}} : c_{\mathsf{rel}}\ \mathsf{i0} = a$
		$c_{\mathsf{right}} : c_{\mathsf{rel}}\ \mathsf{i1} = b.$

Thus no new primitive HIT mechanism is needed. The boxed reductions in the inductive syntax, such as $\times_{\beta_1}$, should be read in this way.

2.2.2 Thin Truncation. In order for this higher inductive type to be a *quotient* inductive type, it needs to be set truncated [Univalent Foundations Program 2013, Section 6.10]. The most common way to do this is to add a constructor $\mathsf{set}_{\mathsf{Tm}} : \mathsf{isSet}(\mathsf{Tm}\ \Gamma\ A)$ in the inductive definition, which is a higher constructor that identifies all parallel paths in the type $\mathsf{Tm}\ \Gamma\ A$. The directed syntax uses the same idea for reductions. To keep reduction *thin*, we want to enforce each inequality type proof-irrelevant:

$$\mathsf{isThin}\ A \;=\; (x\ y : A) \to \mathsf{isProp}(x \le_A y).$$

Hence any two reductions between the same syntactic objects are identified. This is the directed analogue of set truncation used for ordinary judgmental equality, making the syntax a preorder

rather than more generally a category, which is important to model a *judgmental* reduction relation, in which there is at most one reduction between fixed endpoints.

One could instead keep reductions proof-relevant. That would lead to a richer syntax in which different reduction derivations between the same endpoints can carry higher-dimensional information, closer in spirit to bicategorical type theory [Ahrens et al. 2023]. The present paper does not use that extra structure; reductions are treated only up to proof-irrelevance.

2.2.3 Segal Condition for Syntax. In simplicial type theory, given $f : x \le y$ and $g : y \le z$ where $f\ \mathsf{i0} = x$, $f\ \mathsf{i1} = g\ \mathsf{i0} = y$, and $g\ \mathsf{i1} = z$, it is not automatic that there is a unique composite $h : x \le z$ with $h\ \mathsf{i0} = x$ and $h\ \mathsf{i1} = z$. The types for which directed paths do compose are called *Segal* types [Riehl and Shulman 2017, Section 5] or pre-categories [Gratzer et al. 2026a]. In the present setting this is exactly the property the syntax should have: reductions should compose. For Segal types, we write $f \cdot g$ for the composite of f and g.

Grodin et al. [2024, Section 4] make the same point in their construction of synthetic preorders: relevant types are restricted to a reflective subuniverse of Segal types, obtained by an *orthogonality* construction [Christensen et al. 2020; Fiore 1997; Rijke et al. 2020]. In the present setting, the inductively defined syntax should be built inside this subuniverse. In the Cubical Agda mechanization, the raw syntax is first constructed as a higher inductive type, as in Section 2.2.1, and then reflected into the subuniverse of thin Segal sets. The resulting syntax, such as $\mathsf{Tm}\ \Gamma\ A$, is set, thin, and Segal. The orthogonality construction is described in Appendix A; for the rest of this paper, however, it is sufficient to understand that reductions in the syntax compose.

2.2.4 Mapping Out of a Directed Quotient. To map out of a directed quotient type A into a family $P : A \to \mathcal{U}$, the map $f : (a : A) \to P\ a$ must interpret each point constructor and, for each directed constructor $h : x \le_A y$, provide the corresponding coherence between $f\ x$ and $f\ y$. Because these endpoints lie in different fibers, the coherence is not an ordinary inequality in a single type, but a dependent inequality over h. The usual non-dependent monotonicity condition is the constant-family special case: when P is constantly B, the obligation becomes $f\ x \le_B f\ y$.

Definition 2.3 (Dependent inequality types). For $h : x \le_A y$, $u : P\ x$, and $v : P\ y$, the notation

$$u \le_{P(h)} v \coloneqq \Sigma_{q:(i:\mathbb{2}) \to P\ (h\ i)} (q\ \mathsf{i0} = u) \times (q\ \mathsf{i1} = v)$$

denotes the type of dependent inequalities from u to v lying over h[4]. This is conceptually similar to PathP in cubical type theories [Angiuli et al. 2021; Cohen et al. 2018].

Thus the dependent eliminator sends each directed constructor $h : x \le_A y$ to a dependent inequality $f_h : f\ x \le_{P(h)} f\ y$.

3 LOGICAL RELATIONS MODEL

In this section we construct a logical relations model of an object type theory with directed reductions, and use it to prove canonicity. In an ordinary gluing style proof, such as that of Kaposi et al. [2019a], the goal is to prove a statement along the following lines: for every closed term $M : \mathsf{Tm} \cdot \mathsf{Bool}$, either $M = \mathsf{true}$ or $M = \mathsf{false}$, where equality is judgmental equality in the object theory. Here our goal is instead to prove that M reduces to either true or false: in the language of simplicial type theory, that $M \le \mathsf{true}$ or $M \le \mathsf{false}$.

The high-level strategy of proof-relevant logical relations is to equip each type with two pieces of data: a syntactic component A°, which is the underlying syntactic type, and a semantic component

[4]Here $h : x \le_A y$ is implicitly coerced to a map $h : \mathbb{2} \to A$; the transports induced by h in the definition of $u \le_{P(h)} v$ are also left implicit.

$A^\bullet$, which records evidence that terms of A° are well-behaved, or computable[5]. In particular, the semantics at Boolean $\mathsf{BOOL}^\bullet$ will imply the desired canonicity property. We will start with simple types to illustrate this construction as adapted to the directed setting.

3.1 Semantics of the Judgmental Structure

We build a unary gluing model over global sections of the syntactic CwF. Since canonicity is a statement about closed terms, the semantic predicate $\Gamma^\bullet$ for a syntactic context Γ° is indexed by closed substitutions $\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ$, namely global sections of Γ°. A substitution $\sigma^\circ : \mathsf{Sub}\ \Gamma^\circ\ \Delta^\circ$ is computable when, composed with computable global sections of Γ°, it gives computable global sections of Δ°.

$$\begin{array}{ll}\textbf{record}\ \mathsf{CTX} : \mathcal{U}\ \textbf{where} & \textbf{record}\ \mathsf{SUB}\ (\Gamma\ \Delta : \mathsf{CTX}) : \mathcal{U}\ \textbf{where}\\ \quad \Gamma^\circ : \mathsf{Ctx} & \quad \sigma^\circ : \mathsf{Sub}\ \Gamma^\circ\ \Delta^\circ\\ \quad \Gamma^\bullet : \mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathcal{U} & \quad \sigma^\bullet : (\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to \Gamma^\bullet\ \gamma^\circ \to \Delta^\bullet\ (\sigma^\circ \circ \gamma^\circ)\end{array}$$

For types and terms, we first recall the ordinary gluing definition, ignoring for the moment that the syntax has directed reductions. A glued type has an underlying syntactic type $A^\circ : \mathsf{Ty}\ \Gamma^\circ$, together with a computability predicate on closed terms of each closed instance of A°. A glued term is then a syntactic term M° together with a computability witness $M^\bullet$ that each closed instance of M° is computable at A°.

$$\begin{array}{ll}\textbf{record}\ \mathsf{TY}\ (\Gamma : \mathsf{CTX}) : \mathcal{U}\ \textbf{where} & \textbf{record}\ \mathsf{TM}\ (\Gamma : \mathsf{CTX})\ (A : \mathsf{TY}\ \Gamma) : \mathcal{U}\ \textbf{where}\\ \quad A^\circ : \mathsf{Ty}\ \Gamma^\circ & \quad M^\circ : \mathsf{Tm}\ \Gamma^\circ\ A^\circ\\ \quad A^\bullet : (\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to \Gamma^\bullet\ \gamma^\circ \to & \quad M^\bullet : \gamma^\circ\ \gamma^\bullet \to A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])\\ \qquad \mathsf{Tm}\ \cdot\ (A^\circ[\gamma^\circ]) \to \mathcal{U} & \end{array}$$

This is exactly the definition one would use for an ordinary proof-relevant gluing argument. The directed structure has not yet been used. To see what extra structure is needed, let us try to interpret the product type. The proof for products will identify the missing ingredient: the computability predicate of a type must be stable under directed expansion. After that, we will return to the definition of TY and strengthen it accordingly.

3.2 Semantics of Product

As in a typical logical-relations definition, a product is computable when both of its projections are computable at their respective types.

$$\begin{array}{l}\mathsf{PROD} : \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma\\ (\mathsf{PROD}\ A\ B)^\circ := A^\circ \times B^\circ\\ (\mathsf{PROD}\ A\ B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ P := A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{snd}\ P)\end{array}$$

The projections are unchanged from the equational story: the syntactic components are the syntactic projections, and the semantic components are the metatheoretic projections.

$$\begin{array}{ll}\mathsf{FST} : \mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B) \to \mathsf{TM}\ \Gamma\ A & \mathsf{SND} : \mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B) \to \mathsf{TM}\ \Gamma\ B\\ (\mathsf{FST}\ P)^\circ := \mathsf{fst}\ P^\circ & (\mathsf{SND}\ P)^\circ := \mathsf{snd}\ P^\circ\\ (\mathsf{FST}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet) & (\mathsf{SND}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet).\end{array}$$

Now suppose $M : \mathsf{TM}\ \Gamma\ A$ and $N : \mathsf{TM}\ \Gamma\ B$. The syntactic component of the pair is forced:

[5]The notation $\circ$ and $\bullet$ is inspired by Sterling and Harper [2021], where the authors use a pair of modalities $\bigcirc$ and $\bullet$ to isolate the syntax and the semantics in a gluing proof. Here we do not make this distinction explicit using modalities, but our story should be compatible with theirs.

$$\mathsf{PAIR} : \mathsf{TM}\ \Gamma\ A \to \mathsf{TM}\ \Gamma\ B \to \mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B)$$
$$(\mathsf{PAIR}\ M\ N)^\circ \coloneqq \mathsf{pair}\ M^\circ\ N^\circ.$$

For the semantic component, after fixing $\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ$ and $\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ$, the goal is to produce an element of $(\mathsf{PROD}\ A\ B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ])$, which is a metatheoretic product. The obvious definition would have the following shape:

$$(\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \Big(\underbrace{\qquad ? \qquad}_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ \left(\mathsf{fst}\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ])\right)} , \underbrace{\qquad ? \qquad}_{B^\bullet\ \gamma^\circ\ \gamma^\bullet\ \left(\mathsf{snd}\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ])\right)} \Big).$$

But the available witnesses are

$$M^\bullet\ \gamma^\circ\ \gamma^\bullet : A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ]) \qquad \text{and} \qquad N^\bullet\ \gamma^\circ\ \gamma^\bullet : B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (N^\circ[\gamma^\circ]).$$

In an equational proof-relevant logical relation, the two indices are identified by β-equalities for products. In the directed syntax, however, these are not equalities but merely reductions:

$$\times_{\beta_1}[\gamma^\circ] : \mathsf{fst}\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ]) \le M^\circ[\gamma^\circ],$$
$$\times_{\beta_2}[\gamma^\circ] : \mathsf{snd}\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ]) \le N^\circ[\gamma^\circ].$$

Logical relations practitioners are not unfamiliar with this situation. A logical relation is often defined by induction on syntax, after which one proves closure under expansion, or reverse reduction: if $M \to^*_\beta M'$ and M' is computable, then M is computable. In the directed setting, this suggests equipping each semantic type with an operation that transports computability witnesses backward along reductions:

$$\mathsf{expansion} : M \le M' \to A^\bullet\ \gamma^\circ\ \gamma^\bullet\ M' \to A^\bullet\ \gamma^\circ\ \gamma^\bullet\ M.$$

This operation solves the immediate typing problem above: it can turn $M^\bullet\ \gamma^\circ\ \gamma^\bullet$ into a witness at the reduct $\mathsf{fst}\ ((\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ])$. For proof-relevant predicates, however, the witness produced by expansion is itself meaningful data. It is therefore not enough to know that some witness can be transported backward along a reduction; we must know how the transported witness behave. At minimum, expansion should be functorial: expansion along the identity reduction should act as the identity on computability witnesses, and expansion along a composite reduction should agree with the composite of the corresponding expansion maps.

3.2.1 Contravariance as Proof-Relevant Expansion. *Contravariant family* in simplicial type theory serves as the right coherence condition for the expansion lemma. While the notion of contravariant families in simplicial type theory usually is used as a synthetic analogue of contravariant fibrations [Riehl and Shulman 2017] and in constructing directed univalence [Cavallo et al. 2026; Gratzer et al. 2026a; Weaver and Licata 2020], we identify the contravariance as exactly the right proof-relevant generalization of the expansion lemma in logical relations.

Definition 3.1 (Contravariant families - Riehl and Shulman 2017, Definition 8.2). A family $C : X \to \mathcal{U}$ is *contravariant* if for every $f : x \le_X y$ and $v : C(y)$, the type $\sum_{u:C(x)} u \le_{C(f)} v$ is contractible. In other words:

$$\mathsf{isContrav}\ C \coloneqq \quad (x\ y : X)\ (f : x \le_X y)\ (v : C(y)) \to \mathsf{isContr}\Big(\textstyle\sum_{u:C(x)} u \le_{C(f)} v\Big).$$

Every contravariant family C comes equipped with a backward transport operation.

Definition 3.2 (Contravariant transport). Suppose $c_X : \mathsf{isContrav}(C : X \to \mathcal{U})$. For $f : x \le_X y$, the *contravariant transport* along f is:

$$f^* : C(y) \to C(x)$$
$$f^*\ v \coloneqq \pi_1\big(\mathsf{center}(c_X\ x\ y\ f\ v)\big).$$

In particular, this contravariant transport is functorial and satisfies a universal property.

Lemma 3.3 (Functoriality of contravariant transport - Riehl and Shulman 2017, Proposition 8.16 ☞). *Contravariant transport preserves identities and composition: for $f : x \leq_X y$, $g : y \leq_X z$, and $w : C(z)$,*

$$(\mathrm{id}_x)^* v = v \qquad \text{and} \qquad (g \circ f)^* w = f^*(g^* w).$$

Lemma 3.4 (Universal property of contravariant transport - Riehl and Shulman 2017, Lemma 8.15 ☞). *For a contravariant family $C : X \to \mathcal{U}$ with $f : x \leq_X y$, $u : C(x)$, and $v : C(y)$, there is an equivalence*

$$(u \leq_{C(f)} v) \simeq (u = f^* v).$$

The move is to impose this condition on every computability predicate $A^\bullet$ carried by a glued type: isContrav $(A^\bullet\ \gamma^\circ\ \gamma^\bullet)$. The induced contravariant transport operation is the expansion operation needed above. Thus a glued type stores not only its computability predicate, but also the proof that this predicate is contravariant for each closed substitution and context witness:

record TY $(\Gamma : \mathrm{CTX}) : \mathcal{U}$ **where**
$A^\circ : \mathrm{Ty}\ \Gamma^\circ$
$A^\bullet : \gamma^\circ\ \gamma^\bullet \to \mathrm{Tm}\ \cdot\ (A^\circ[\gamma^\circ]) \to \mathcal{U}$
$c_A : \gamma^\circ\ \gamma^\bullet \to \mathrm{isContrav}\ (A^\bullet\ \gamma^\circ\ \gamma^\bullet)$

record TM $(\Gamma : \mathrm{CTX})(A : \mathrm{TY}\ \Gamma) : \mathcal{U}$ **where**
$M^\circ : \mathrm{Tm}\ \Gamma^\circ\ A^\circ$
$M^\bullet : \gamma^\circ\ \gamma^\bullet \to A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])$

With the refined definitions of TY and TM in place, inequalities between glued terms can be described explicitly. An inequality $M \leq N$ between glued terms is an inequality in a Σ-type: it relates both the underlying syntactic terms and the semantic witnesses. The following fact says that such inequalities split into an inequality in the base and a dependent inequality over it.

Lemma 3.5 (Inequalities at Σ-types ☞). *Given $B : A \to \mathcal{U}$, $x, y : A$, $u : B\ x$, and $v : B\ y$, there is an equivalence*

$$\left(\sum_{h : x \leq_A y} u \leq_{B(h)} v \right) \simeq (x, u) \leq_{\sum_{a:A} B\ a} (y, v).$$

Applying Lemma 3.5 to TM, an inequality $M \leq N$ consists of an underlying syntactic reduction $\rho^\circ : M^\circ \leq N^\circ$, together with, for each closed substitution γ° and computability witness $\gamma^\bullet$, a dependent inequality between the two computability witnesses over the closed reduction $\rho^\circ[\gamma^\circ]$:

$$M^\bullet\ \gamma^\circ\ \gamma^\bullet \leq_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\rho^\circ[\gamma^\circ])} N^\bullet\ \gamma^\circ\ \gamma^\bullet.$$

By Lemma 3.4, this dependent inequality is equivalently an equality with the contravariant transport:

$$M^\bullet\ \gamma^\circ\ \gamma^\bullet = (\rho^\circ[\gamma^\circ])^*(N^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

Thus the order on computability witnesses is forced by the contravariance condition on the glued type: whenever the syntax expands M° to N°, the witness for M is obtained by transporting the witness for N backward along that reduction.

3.2.2 Glued Semantics for Product and Pair ☞. The product definition can now be revisited, this time filling in the contravariance component. Suppose $f : P \leq P'$ is a reduction between closed product terms, and suppose (Φ', Ψ') is a computability witness for P', so that

$$\Phi' : A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathrm{fst}\ P') \qquad \text{and} \qquad \Psi' : B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathrm{snd}\ P').$$

Projecting f through the two eliminators gives reductions

$$\mathrm{fst}\ f : \mathrm{fst}\ P \leq \mathrm{fst}\ P' \qquad \mathrm{snd}\ f : \mathrm{snd}\ P \leq \mathrm{snd}\ P'.$$

The contravariant transports for A and B then give the required witness for P:

$$c_{\mathsf{PROD}\ A\ B}\ \gamma^\circ\ \gamma^\bullet\ (f : P \le P')\ (\Phi', \Psi') := \Big((\mathsf{fst}\ f)^*\ \Phi',\ (\mathsf{snd}\ f)^*\ \Psi'\Big).$$

This displayed term is the distinguished point of the lift type required by contravariance. The full contractibility proof follows by splitting the dependent inequality into its two components:

$$\sum_{(\Phi,\Psi)} (\Phi, \Psi) \le_{(\mathsf{PROD}\ A\ B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ f}\ (\Phi', \Psi') \simeq \Big(\sum_{\Phi} \Phi \le_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{fst}\ f)}\ \Phi'\Big) \times \Big(\sum_{\Psi} \Psi \le_{B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{snd}\ f)}\ \Psi'\Big).$$

The two factors are contractible by c_A and c_B, respectively, and hence so is their product.

Finally, the semantic component of the pair can be defined:

$$(\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet := \Big(\boxed{(\times_{\beta_1}[\gamma^\circ])^*}\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet),\ \boxed{(\times_{\beta_2}[\gamma^\circ])^*}\ (N^\bullet\ \gamma^\circ\ \gamma^\bullet)\Big).$$

The transports are necessary because the product predicate is indexed by the projections of the syntactic pair, while $M^\bullet$ and $N^\bullet$ live over the two components themselves. The β-reductions bridge precisely this gap. It remains to check that this interpretation respects the directed quotient constructors for product. This is the semantic content of mapping out of a directed quotient: the images of the point constructors pair, fst, and snd must respect inequalities $\times_{\beta_1}$ and $\times_{\beta_2}$.

3.2.3 Glued Terms Respect Directed Quotient. First consider inequality $\times_{\beta_1}$ in the syntax. As explained in Section 2.2.4, the mapping out must respect this inequality. This is menifested as the following proof obligation:

$$\mathsf{FST}\ (\mathsf{PAIR}\ M\ N) \le_{\mathsf{TM}\ \Gamma\ A} M.$$

Because TM is a Σ-type, by Lemma 3.5, this inequality consists of two parts: a syntactic reduction and a dependent inequality over it. The syntactic part is exactly the product β_1-reduction:

$$(\mathsf{FST}\ (\mathsf{PAIR}\ M\ N))^\circ = \mathsf{fst}\ (\mathsf{pair}\ M^\circ\ N^\circ) \le M^\circ.$$

For the semantic part, fix $\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ$ and $\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ$. The proof obligation is

$$(\mathsf{FST}\ (\mathsf{PAIR}\ M\ N))^\bullet\ \gamma^\circ\ \gamma^\bullet \le_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\times_{\beta_1}[\gamma^\circ])}\ M^\bullet\ \gamma^\circ\ \gamma^\bullet.$$

The left-hand side computes as follows:

$$\begin{aligned}
&(\mathsf{FST}\ (\mathsf{PAIR}\ M\ N))^\bullet\ \gamma^\circ\ \gamma^\bullet \\
&= \pi_1\big((\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet\big) && \text{by definition of } \mathsf{FST}^\bullet \\
&= \pi_1\Big((\times_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet),\ (\times_{\beta_2}[\gamma^\circ])^*(N^\bullet\ \gamma^\circ\ \gamma^\bullet)\Big) && \text{by definition of } \mathsf{PAIR}^\bullet \\
&= (\times_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet) && \text{by projection.}
\end{aligned}$$

Therefore the required semantic inequality is

$$(\times_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet) \le_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\times_{\beta_1}[\gamma^\circ])}\ M^\bullet\ \gamma^\circ\ \gamma^\bullet.$$

By Lemma 3.4, this is equivalent to the reflexive equality

$$(\times_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet) = (\times_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

This is where bare expansion in proof-irrelevant logical relations must be strengthened to universal contravariance. A bare map produces the transported witness, but validating the directed reduction in the logical relation requires the dependent inequality above. The universal property of contravariant transport turns that obligation into a reflexivity instance of equality. The $\times_{\beta_2}$ case is symmetric, using the second projection and the contravariance of $B^\bullet$.

3.2.4 What If We Have η-Reduction as Well? . The constructions above use only the product β-reductions; indeed for the canonicity result, any η-reduction would be optional. If the syntax does include an η-inequality, the present model validates the η-reduction: $\times_\eta : \mathsf{pair}\ (\mathsf{fst}\ P)\ (\mathsf{snd}\ P) \le P$. The reason is that projecting this η-reduction gives reductions with the same endpoints as the corresponding β-reductions, and the syntactic inequalities are thin per Section 2.2.2.

For $P : \mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B)$, the required glued reduction is

$$\mathsf{PAIR}\ (\mathsf{FST}\ P)\ (\mathsf{SND}\ P) \le_{\mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B)} P.$$

For the semantic part, again fix γ° and $\gamma^\bullet$. Unfolding the source gives

$$(\mathsf{PAIR}\ (\mathsf{FST}\ P)\ (\mathsf{SND}\ P))^\bullet\ \gamma^\circ\ \gamma^\bullet = \Big((\times_{\beta_1}[\gamma^\circ])^*\pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet),\ (\times_{\beta_2}[\gamma^\circ])^*\pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet)\Big).$$

Unfolding $(\mathsf{PROD}\ A\ B)^\bullet$, the required dependent inequality over $\times_\eta[\gamma^\circ]$ splits into two components:

$$(\times_{\beta_1}[\gamma^\circ])^*\pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet) \le_{A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{fst}\ (\times_\eta[\gamma^\circ]))} \pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet),$$
$$(\times_{\beta_2}[\gamma^\circ])^*\pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet) \le_{B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{snd}\ (\times_\eta[\gamma^\circ]))} \pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

By the universal property of contravariant transport, the first component is equivalent to

$$(\times_{\beta_1}[\gamma^\circ])^*\pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet) = (\mathsf{fst}\ (\times_\eta[\gamma^\circ]))^*\pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

Both $\times_{\beta_1}[\gamma^\circ]$ and $\mathsf{fst}\ (\times_\eta[\gamma^\circ])$ are reductions from $\mathsf{fst}\ (\mathsf{pair}\ (\mathsf{fst}\ P^\circ)\ (\mathsf{snd}\ P^\circ))[\gamma^\circ]$ to $\mathsf{fst}\ P^\circ[\gamma^\circ]$. The inequalities of the syntax are propositions, so these reductions are equal, and hence their contravariant transports agree. The second component is the same argument.

3.3 Semantics of Functions

The function type follows the same pattern as the product type. First recall the syntax:

$$-\Rightarrow- : \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathsf{Ty}\ \Gamma \qquad \mathsf{lam} : \mathsf{Tm}\ (\Gamma \triangleright A)\ (B[\mathsf{p}]) \to \mathsf{Tm}\ \Gamma\ (A \Rightarrow B)$$
$$\Rightarrow\beta\ : \boxed{\mathsf{app}\ (\mathsf{lam}\ N)\ M \le N[(\mathsf{id},M)]} \qquad \mathsf{app} : \mathsf{Tm}\ \Gamma\ (A \Rightarrow B) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B.$$

The logical relation for function types is standard: a computable function is one that sends computable inputs to computable outputs.

$$-\Rightarrow- : \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma$$
$$(A \Rightarrow B)^\circ := A^\circ \Rightarrow B^\circ$$
$$(A \Rightarrow B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ F := (M^\circ : \mathsf{Tm}\cdot(A^\circ[\gamma^\circ])) \to (M^\bullet : A^\bullet\ \gamma^\circ\ \gamma^\bullet\ M^\circ) \to B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{app}\ F\ M^\circ).$$

For the contravariance component, given $f : F \le F'$ and $\Phi' : (A \Rightarrow B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ F'$, define

$$\boxed{c_{A\Rightarrow B}\ \gamma^\circ\ \gamma^\bullet\ (f : F \le F')\ \Phi' := \lambda M^\circ\ M^\bullet.\ (\mathsf{app}\ f\ M^\circ)^*(\Phi'\ M^\circ\ M^\bullet).}$$

The reduction $\mathsf{app}\ f\ M^\circ : \mathsf{app}\ F\ M^\circ \le \mathsf{app}\ F'\ M^\circ$ is the functorial action of application on a reduction in the function position. The full contractibility proof is obtained from the contravariance of $B^\bullet$, followed by function extensionality.

3.3.1 Substitution Structure . The substitution and context-extension clauses used below are the standard gluing clauses; the full list is collected in Appendix B. The only change from the ordinary equational presentation occurs in clauses involving types. For example, type substitution must also provide the contravariance proof for $A[\sigma]$, inherited directly from the one for A. These additional components are straightforward.

3.3.2 Glued Semantics for Lambda and Application. The full constructor clauses are listed in Appendix B. Only the semantic components are needed here:

$$(\mathsf{LAM}\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet\ M^\circ\ M^\bullet := \boxed{(\Rightarrow_\beta[(\gamma^\circ,M^\circ)])^*}\big(N^\bullet\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet)\big)$$

$$(\mathsf{APP}\ F\ M)^\bullet\ \gamma^\circ\ \gamma^\bullet := F^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

The boxed transport is induced by the instantiated β-reduction

$$\Rightarrow_\beta[(\gamma^\circ,M^\circ)] : \mathsf{app}\ ((\mathsf{lam}\ N^\circ)[\gamma^\circ])\ M^\circ \le N^\circ[(\gamma^\circ,M^\circ)].$$

Thus the instantiated evidence $N^\bullet\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet) : B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (N^\circ[(\gamma^\circ,M^\circ)])$ is transported from evidence over the β-contractum to evidence over the application of the lambda.

3.3.3 Glued Functions Respect Directed Quotient. Consider $\Rightarrow_\beta$. For $N : \mathsf{TM}\ (\Gamma\triangleright A)\ (B[\mathsf{p}])$ and $M : \mathsf{TM}\ \Gamma\ A$, the required glued inequality is

$$\mathsf{APP}\ (\mathsf{LAM}\ N)\ M \le_{\mathsf{TM}\ \Gamma\ B} N[(\mathsf{ID},M)].$$

For the semantic part of this inequality, fix γ° and $\gamma^\bullet$. The left-hand side computes to

$$\begin{aligned}
&(\mathsf{APP}\ (\mathsf{LAM}\ N)\ M)^\bullet\ \gamma^\circ\ \gamma^\bullet \\
&= (\mathsf{LAM}\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet) && \text{by definition of } \mathsf{APP}^\bullet \\
&= (\Rightarrow_\beta[(\gamma^\circ,M^\circ[\gamma^\circ])])^*\big(N^\bullet\ (\gamma^\circ,M^\circ[\gamma^\circ])\ (\gamma^\bullet,M^\bullet\ \gamma^\circ\ \gamma^\bullet)\big) && \text{by definition of } \mathsf{LAM}^\bullet.
\end{aligned}$$

The witness inside the transport is exactly the semantic component of the substituted body:

$$(N[(\mathsf{ID},M)])^\bullet\ \gamma^\circ\ \gamma^\bullet = N^\bullet\ (\gamma^\circ,M^\circ[\gamma^\circ])\ (\gamma^\bullet,M^\bullet\ \gamma^\circ\ \gamma^\bullet).$$

By Lemma 3.4, the semantic inequality over $\Rightarrow_\beta[\gamma^\circ]$ is equivalent to the reflexive equality of this transported witness with itself, following the same pattern as for products.

3.4 Semantics of Booleans (and Canonicity)

The Boolean type is the point of the canonicity argument. Its computability predicate says exactly that a closed Boolean reduces to one of the two canonical Booleans. First recall the simple Boolean syntax, including the non-dependent eliminator.

$$\begin{array}{ll}
\mathsf{Bool} : \mathsf{Ty}\ \Gamma & \mathsf{if} : (C : \mathsf{Ty}\ \Gamma) \to \mathsf{Tm}\ \Gamma\ C \to \mathsf{Tm}\ \Gamma\ C \to \\
\mathsf{true} : \mathsf{Tm}\ \Gamma\ \mathsf{Bool} & \quad \mathsf{Tm}\ \Gamma\ \mathsf{Bool} \to \mathsf{Tm}\ \Gamma\ C \\
\mathsf{false} : \mathsf{Tm}\ \Gamma\ \mathsf{Bool} & \mathsf{Bool}_{\mathsf{true}} : \boxed{\mathsf{if}\ C\ U\ V\ \mathsf{true} \le U} \\
& \mathsf{Bool}_{\mathsf{false}} : \boxed{\mathsf{if}\ C\ U\ V\ \mathsf{false} \le V}.
\end{array}$$

Write $\ulcorner 0 \urcorner := \mathsf{true}$ and $\ulcorner 1 \urcorner := \mathsf{false}$. Then the glued Boolean type is:

$$\begin{aligned}
&\mathsf{BOOL} : \mathsf{TY}\ \Gamma \\
&\mathsf{BOOL}^\circ := \mathsf{Bool} \\
&\mathsf{BOOL}^\bullet\ \gamma^\circ\ \gamma^\bullet\ M := \boxed{\sum_{b:\{0,1\}} M \le_{\mathsf{Tm}\ \cdot\ \mathsf{Bool}} \ulcorner b \urcorner}.
\end{aligned}$$

The contravariance proof for $\mathsf{BOOL}^\bullet$ packages a familiar closure argument. In an ordinary logical-relations proof, expansion closure for this predicate would be proved by hand: from $f : M \le M'$ and $r : M' \le \ulcorner b \urcorner$, one composes reductions to obtain $f \cdot r : M \le \ulcorner b \urcorner$. This proof is elementary, but it is still an extra proof obligation. In the present setting, the same closure is available off the shelf from simplicial type theory: representable families are contravariant.

Lemma 3.6 (Representable Contravariant Families - Riehl and Shulman 2017, Proposition 8.13 ✋). *For any $a : X$, the representable family*

$$\lambda x.\, x \leq_X a : X \to \mathcal{U}$$

is contravariant if X is Segal. Its contravariant transport sends $r : y \leq_X a$ along $f : x \leq_X y$ to the composite $f^* r := f \cdot r : x \leq_X a$.

Indeed, for a fixed $b : \{0, 1\}$, the summand $M \mapsto M \leq \ulcorner b \urcorner$ is represented by $\ulcorner b \urcorner$. The finite sum over b is therefore contravariant by transporting inside the chosen summand. Explicitly, for $f : M \leq M'$ and $(b, r) : \mathsf{BOOL}^\bullet\ \gamma^\circ\ \gamma^\bullet\ M'$, define

$$\boxed{c_{\mathsf{BOOL}}\ \gamma^\circ\ \gamma^\bullet\ (f : M \leq M')\ (b, r) := (b,\ f \cdot r).}$$

The lift witness $(b, f \cdot r) \leq_{\mathsf{BOOL}^\bullet(f)} (b, r)$, and the contractibility of the corresponding lift type, are supplied by Lemma 3.6. Thus the usual composition proof has not disappeared; it has been isolated as a general simplicial type-theoretic fact and reused here rather than reproved specifically for Booleans. The constructors are immediate.

$$\mathsf{TRUE} : \mathsf{TM}\ \Gamma\ \mathsf{BOOL} \qquad \mathsf{FALSE} : \mathsf{TM}\ \Gamma\ \mathsf{BOOL}$$
$$\mathsf{TRUE}^\circ := \mathsf{true} \qquad \mathsf{FALSE}^\circ := \mathsf{false}$$
$$\mathsf{TRUE}^\bullet\ \gamma^\circ\ \gamma^\bullet := (0, \mathsf{id}_{\mathsf{true}}) \qquad \mathsf{FALSE}^\bullet\ \gamma^\circ\ \gamma^\bullet := (1, \mathsf{id}_{\mathsf{false}}).$$

The simple eliminator is handled by case analysis on the Boolean computability witness.

$$\mathsf{IF} : (C : \mathsf{TY}\ \Gamma) \to \mathsf{TM}\ \Gamma\ C \to \mathsf{TM}\ \Gamma\ C \to \mathsf{TM}\ \Gamma\ \mathsf{BOOL} \to \mathsf{TM}\ \Gamma\ C$$
$$(\mathsf{IF}\ C\ U\ V\ T)^\circ := \mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ T^\circ.$$

For the semantic component, suppose $T^\bullet\ \gamma^\circ\ \gamma^\bullet = (b, r)$. If $b = 0$, set

$$\rho_r^0 : (\mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ T^\circ)[\gamma^\circ] \leq (\mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ \mathsf{true})[\gamma^\circ] \leq U^\circ[\gamma^\circ],$$
$$\rho_r^0 := (\mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ r)[\gamma^\circ] \cdot \mathsf{Bool}_{\mathsf{true}}[\gamma^\circ].$$

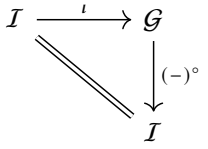
If $b = 1$, define ρ_r^1 in the same way, using $\mathsf{Bool}_{\mathsf{false}}$ and ending at $V^\circ[\gamma^\circ]$. Then

$$(\mathsf{IF}\ C\ U\ V\ T)^\bullet\ \gamma^\circ\ \gamma^\bullet := \begin{cases} \boxed{(\rho_r^0)^*}\ (U^\bullet\ \gamma^\circ\ \gamma^\bullet), & \text{if } T^\bullet\ \gamma^\circ\ \gamma^\bullet = (0, r), \\ \boxed{(\rho_r^1)^*}\ (V^\bullet\ \gamma^\circ\ \gamma^\bullet), & \text{if } T^\bullet\ \gamma^\circ\ \gamma^\bullet = (1, r). \end{cases}$$

The two β-laws $\mathsf{IF}\ C\ U\ V\ \mathsf{TRUE} \leq U$ and $\mathsf{IF}\ C\ U\ V\ \mathsf{FALSE} \leq V$ follow from the universal property of contravariant transport, exactly as in the product and function cases.

3.4.1 Canonicity via Fundamental Theorem of Logical Relations ✋. At this point there are two CwF models: the initial syntactic model $\mathcal{I}$ in *red* and the gluing model $\mathcal{G}$ in *blue*. The gluing model $\mathcal{G}$ has been constructed so that each object A° of $\mathcal{I}$ is sent to $(A^\circ, A^\bullet)$ in $\mathcal{G}$, with the syntactic component $\circ$ in the gluing model exactly the corresponding component of the initial model. This is the key point of the construction, and it is the fundamental theorem of logical relations as manifested in the diagram below.

For a syntactic closed term $M : \mathsf{Tm}\ \cdot\ \mathsf{Bool}$, the glued term $\iota(M)$ has syntactic projection M itself. Its semantic projection, instantiated at the closed context, gives the witness displayed on the right. The diagram on the left commutes by the initiality/induction principle of the syntactic model.



$$(\iota(M))^\bullet : \mathsf{BOOL}^\bullet\ \epsilon\ \cdot\ ()\ M = \sum_{b:\{0,1\}} M \leq \ulcorner b \urcorner$$

This says that M reduces to either true or false, which is the canonicity result for Booleans.

4 UNIVERSES AND DEPENDENCY

For simple type theory, adapting proof-relevant logical relations from terms quotiented by judgmental equality to terms quotiented by directed inequalities requires only a modest strengthening of the induction hypothesis: each computability predicate must be contravariant. Scaling this construction to dependent types and universes introduces a further constraint. Type conversion, universe predicates, and dependency must all remain compatible with this contravariance requirement.

4.1 Roadblocks to Naïve Dependent Attempts

Keeping the syntax from Section 2, where reductions are directed, together with the semantics from Section 3, where each type is assigned a contravariant computability predicate, leads to two related obstructions for a naïve dependent extension.

4.1.1 How to Model Type Conversion? In ordinary dependent type theory, type conversion is a symmetric rule:

$$\frac{M : A \qquad A \equiv B}{M : B}$$

where $A \equiv B$ is judgmental equality of types, induced by judgmental equality of the corresponding type codes. In an equational proof-relevant logical relation this rule is essentially invisible. Terms and types are quotiented by judgmental equality, so judgmentally equal types are identified by a path, and conversion becomes transport along that path in the metatheory:

$$\mathsf{transport}_{\mathsf{Tm}\ \Gamma} : A = B \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ B.$$

This mechanism is not available in the directed setting. The syntax records type reductions as directed inequalities rather than judgmental equalities. Quotienting first by judgmental equality would identify terms that the directed quotient only relates by directed inequalities, thereby erasing the directed structure. Thus the ordinary conversion rule has no premise of the required shape: a reduction $A \leq_{\mathsf{Ty}\ \Gamma} B$ is not a judgmental equality $A \equiv B$.

4.1.2 Failure of the Naïve Universe Predicate. Proof-relevant logical relations allow a "negative" formulation of the universe predicate: the computability content of a code may itself be a computability predicate for the decoded type. This formulation is unavailable in proof-irrelevant settings where computability predicates land in Prop, because Prop cannot retain the required predicate-level data. In those settings the universe predicate is usually formulated "positively", as an inductive lookup table of type codes and their predicates, restricting the universe to types specified in advance [Allen 1987; Angiuli 2019; Harper 1992; Martin-Löf 1998]. The proof-relevant approach is more flexible. Consider a Tarski/Coquand style universe, whose syntax is written as follows[6]:

$$\begin{array}{ll} \mathsf{U} : \mathsf{Ty}\ \Gamma & \mathsf{El} : \mathsf{Tm}\ \Gamma\ \mathsf{U} \to \mathsf{Ty}\ \Gamma \\ \mathsf{U}_\beta : \boxed{\mathsf{El}\ (\mathsf{Code}\ A) \leq A} & \mathsf{Code} : \mathsf{Ty}\ \Gamma \to \mathsf{Tm}\ \Gamma\ \mathsf{U}. \end{array}$$

The natural candidate for UNIV records, at each closed code, a computability predicate on the decoded type together with its contravariance requirement, effectively internalizing TY:

$$\begin{aligned} &\mathsf{UNIV}^\circ := \mathsf{U} \\ &\mathsf{UNIV}^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M : \mathsf{Tm} \cdot \mathsf{U}) := \textstyle\sum_{P : \mathsf{Tm} \cdot (\mathsf{El}\ M) \to \mathcal{U}} \mathsf{isContrav}\ P. \end{aligned}$$

[6] Girard's paradox is avoided by considering a cumulative hierarchy of universes and a cumulative hierarchy of judgments. This is achieved by adding levels to Ty and Tm and lifting levels for the universe: $\mathsf{U} : i \to \mathsf{Ty}_{i+1}\ \Gamma$.

This is the negative formulation: a code's computability content is itself a glued type's worth of structure. The definition is attractive, but its contravariant transport immediately runs into the same absence of type equality. Given a universe reduction $f : M \leq_{\mathsf{Tm}\ \cdot\ \cup} M'$ and a predicate $P' : \mathsf{Tm}\ \cdot\ (\mathsf{El}\ M') \to \mathcal{U}$, the decoded types $\mathsf{El}\ M$ and $\mathsf{El}\ M'$ are related by a directed inequality, not by a judgmental equality. Without equality between these decoded types, there is no canonical transport between the corresponding fibers of Tm on which the predicates are defined.

There is also a second problem. Even if such a transported predicate P were definable, the lift contractibility required by isContrav would still fail. A competing lift Q with $Q(N) \leq_{\mathcal{U}} P'(N)$ would have to be equal to P. In general, however, inequality in the metatheoretic universe $\mathcal{U}$ is not forced to collapse to equality, so two predicates below $P'(N)$ need not be equal.

In summary, the syntax calls for judgmental equality between types, while the semantics calls for equality between computability predicates. The rest of this section reconciles these requirements using *discretization* and the *flat modality*.

4.2 Discrete Types and Flat Modality

The main idea is to work with judgmentally equivalent types while retaining the directed reduction structure of terms. The critical observation is that reductions are contained in judgmental equalities, except for symmetry. Freely adding symmetry to directed structures in Ty effectively *discretizes* $\mathsf{Ty}\ \Gamma$, making judgmentally equivalent types available for conversion.

In simplicial type theory, a type is discrete when it has no non-trivial directed inequalities.

Definition 4.1 (Discrete types). A type X is discrete if the canonical map $x =_X y \to x \leq_X y$ is an equivalence:

$$\mathsf{isDiscrete}\ X \coloneqq \mathsf{isEquiv}\ (\_ : x =_X y \to x \leq_X y).$$

Example 4.2. Meta-language Booleans are discrete types. This is typically menifested as an axiom in simplicial type theory, *e.g.* Gratzer et al. [2026a, Axiom 7] and Gratzer et al. [2026b, Axiom 4]. ⌟

To the directed QIIT generating $\mathsf{Ty}\ \Gamma$ we add the constructor

$$\mathsf{disc}_{\mathsf{Ty}} : (A \leq_{\mathsf{Ty}\ \Gamma} B) \to (A =_{\mathsf{Ty}\ \Gamma} B).$$

Every reduction-induced inequality on types $r : A \leq_{\mathsf{Ty}\ \Gamma} B$ now yields a path $\mathsf{disc}_{\mathsf{Ty}}\ r : A =_{\mathsf{Ty}\ \Gamma} B$ and transport along this path defines the conversion equivalence

$$\mathsf{conv}_r : \mathsf{Tm}\ \Gamma\ A \nrightarrow \mathsf{Tm}\ \Gamma\ B.$$

We write conv when r is clear from context. In particular, because we have set truncation and thin truncation as in Section 2.2.2, immediately $\mathsf{disc}_{\mathsf{Ty}}$ is the inverse of canonical map $x =_{\mathsf{Ty}\ \Gamma} y \to x \leq_{\mathsf{Ty}\ \Gamma} y$, rendering $\mathsf{Ty}\ \Gamma$ discrete according to Definition 4.1.[7] The term judgment Tm is *not* discretized, and the canonicity statement depends on observing directed reductions of closed terms. Boolean canonicity asks that a closed Boolean term reduces (directedly) to a canonical Boolean; nothing in this statement is affected by collapsing the directed structure on Ty.

4.2.1 Contravariant Transport for Universe. Now with conv available, we can define the intended contravariant transport for the semantics of universe. We first use the following elementary closure property of contravariant families.

Lemma 4.3 (Contravariance under reindexing - Riehl and Shulman 2017, Remark 8.3). *Let $g : A \to B$ and $C : B \to \mathcal{U}$. If C is contravariant, then the reindexed family $C \circ g$ is also contravariant.*

[7] A more local way to discretize types is to ask for U_β to be an equality instead of an inequality, as El is the only way to lift terms to types. This approach will, however, introduce additional complexity in proving $\mathsf{Ty}\ \Gamma$ to be discrete.

Let $f : M \leq_{\mathsf{Tm}\,\cdot\,\cup} M'$ and let $P' : \mathsf{Tm}\,\cdot\,(\mathsf{El}\,M') \to \mathcal{U}$ be contravariant. Define

$$P := P' \circ \mathsf{conv}_{(\mathsf{El}\,f)} : \mathsf{Tm}\,\cdot\,(\mathsf{El}\,M) \to \mathcal{U}.$$

By Lemma 4.3, P is contravariant. It remains to exhibit the inequality $P \leq_{\mathsf{Tm}\,\cdot\,(\mathsf{El}\,f)\to\mathcal{U}} P'$ required of the lift. Pointwise, this means that for $N \leq_{\mathsf{Tm}\,\cdot\,(\mathsf{El}\,f)} N'$, there is an inequality $P\,N \leq_{\mathcal{U}} P'\,N'$. The dependent inequality over $\mathsf{El}\,f$ is exactly an ordinary inequality after conversion:

$$\mathsf{conv}_{\mathsf{El}\,f}\,N \leq_{\mathsf{Tm}\,\cdot\,(\mathsf{El}\,M')} N'.$$

Monotonicity of P' therefore gives $P'(\mathsf{conv}_{\mathsf{El}\,f}\,N) \leq_{\mathcal{U}} P'\,N'$, which is precisely $P\,N \leq_{\mathcal{U}} P'\,N'$ by definition of P. This proves existence of a lift. The remaining issue is uniqueness: the lift must be contractible. For another $Q : \mathsf{Tm}\,\cdot\,(\mathsf{El}\,M) \to \mathcal{U}$ with isContrav Q and $Q \leq_{\mathsf{Tm}\,\cdot\,(\mathsf{El}\,f)\to\mathcal{U}} P'$, contractibility would require $Q = P$. Pointwise, this would amount to identifying $Q\,N$ with $P\,N = P'(\mathsf{conv}\,N)$. The hypothesis on Q supplies only an inequality into P', and in a non-discrete universe such inequalities do not determine equalities.

4.2.2 Flat Modality. The repair, taking inspiration from crisp/modal type theory [Gratzer 2023; Licata et al. 2018; Shulman 2018], is to store predicate codes under a discrete modality so the inequality between computability predicates are forcibly discrete. As Gratzer et al. [2026a,b] have shown, simplicial type theory is consistent with the flat modality $\flat$ that takes the groupoid core of a type. For a type X, $\flat X$ is a discrete copy of X obtained by removing inequalities in X. The removal nature of this modality makes it *comonadic*.

Synthetically in type theory, this idempotent comonadic modality, analogous to a necessity modality in modal logic [Pfenning and Davies 2001], requires a validity context Δ of flat variables and normal context Γ. A term of type $\flat X$ can only be constructed using flat variables from the validity context [Shulman 2018, Figure 5]:

$$\frac{\Delta \mid \cdot \vdash X\ \mathsf{type}}{\Delta \mid \Gamma \vdash \flat X\ \mathsf{type}}\ \flat\text{-FORM} \qquad \frac{\Delta \mid \cdot \vdash M : X}{\Delta \mid \Gamma \vdash M^{\flat} : \flat X}\ \flat\text{-INTRO}$$

$$\frac{\Delta \mid \Gamma, x : \flat X \vdash C\ \mathsf{type} \qquad \Delta \mid \Gamma \vdash M : \flat X \qquad \Delta, u : X \mid \Gamma \vdash N : C[u^{\flat}/x]}{\Delta \mid \Gamma \vdash \mathsf{let}^{\flat}\ u = M\ \mathsf{in}\ N : C[M/x]}\ \flat\text{-ELIM}$$

The associated comonadic counit is

$$\varepsilon_X : \flat X \to X, \qquad \varepsilon_X(x) := \mathsf{let}^{\flat}\ u = x\ \mathsf{in}\ u.$$

It computes on introductions as $\varepsilon_X(M^{\flat}) = M$. The key property is discreteness: directed inequalities in $\flat X$ collapse to equalities,

$$M \leq_{\flat X} N \quad \Longrightarrow \quad M =_{\flat X} N.$$

The following axiom of simplicial type theory supplies the general principle:

Axiom 4.4 (Flat discreteness - Gratzer et al. 2026a, Axiom 6). *For type X, the constant-interval map $\lambda x.\,\lambda i.\,x : X \to X^{\mathbb{2}}$ is an equivalence if and only if the counit $\varepsilon_X : \flat X \to X$ is an equivalence.*

Here $X^{\mathbb{2}}$ denotes the type of directed paths in X. The premise therefore says that every directed path in X is constant, which is the synthetic expression of discreteness. This motivates the following terminology, following Rijke et al. [2020].

Definition 4.5. A type is *flat modal* when the comonadic counit is an equivalence:

$$\mathsf{isFlatModal}\ X := \mathsf{isEquiv}\ (\varepsilon_X : \flat X \to X).$$

For a flat modal type X, let κ denote the inverse of the comonadic counit; the inverse may be applied implicitly when the context is clear. In particular, $\flat X$ is flat modal by idempotency.

Lemma 4.6. *A type is flat modal if and only if it is discrete.*

In particular, if X is discrete, hence flat modal, an element $x : X$ may be used as a flat variable in the validity context. By construction, Ty Γ is discrete. The computability predicate for a type can therefore be stored in TY as a flat predicate code. For a closed syntactic type A° : Ty $\cdot$, define the type of flat predicate codes on closed terms of A° by[8]:

$$\mathsf{Pred}(A^\circ) \coloneqq \mathsf{let}^\flat\ T = A^\circ\ \mathsf{in}\ \flat(\mathsf{Tm}\ \cdot\ T \to \mathcal{U}).$$

The ordinary predicate used by terms is recovered fiberwise by the counit.

4.2.3 Revised CTX and TY with Flat Predicate Codes. The glued context and type records are refined to store flat codes alongside the ordinary data. Substitutions and terms remain as in Section 3.

record CTX : $\mathcal{U}$ **where**
  Γ° : Ctx
  $\Gamma^\bullet_\flat : \flat(\mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathcal{U})$
  $\Gamma^\bullet \coloneqq \varepsilon(\Gamma^\bullet_\flat)$

record TY (Γ : CTX) : $\mathcal{U}$ **where**
  A° : Ty Γ°
  $A^\bullet_\flat : (\gamma^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ) \to \mathsf{Pred}(A^\circ[\gamma^\circ])$
  $A^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \varepsilon(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet)$
  $c_A : \gamma^\circ\ \gamma^\bullet \to \mathsf{isContrav}\ (A^\bullet\ \gamma^\circ\ \gamma^\bullet)$

The fields $\Gamma^\bullet$ and $A^\bullet$ are now the counit-defined views of the underlying flat data.

4.2.4 Revisiting Product with Flat Predicate Codes. The construction from Section 3 can now be replayed with flat predicate codes. Products show the basic pattern. The syntactic component is unchanged, while the semantic component is stored as a flat code rather than an ordinary predicate. For a closed instance, the construction opens the flat fiber codes for A and B, forms the ordinary product predicate from the opened predicates, and introduces the result back into $\flat$. This is the functorial action of the flat modality on predicates.

$$(\mathsf{PROD}\ A\ B)^\circ \coloneqq A^\circ \times B^\circ$$
$$(\mathsf{PROD}\ A\ B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet \coloneqq \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ B^\bullet = B^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \big(\lambda P.\ A^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ (\mathsf{snd}\ P)\big)^\flat.$$

The ordinary predicate used in Section 3.2 is recovered by opening this flat code with the counit:

$$(\mathsf{PROD}\ A\ B)^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \varepsilon\big((\mathsf{PROD}\ A\ B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\big).$$

The counit computes through the flat eliminations:

$$\begin{aligned}\varepsilon\Big(\mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ B^\bullet = B^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ (\lambda P.\ A^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ (\mathsf{snd}\ P))^\flat\Big)\\ = \lambda P.\ (\varepsilon(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet))\ (\mathsf{fst}\ P) \times (\varepsilon(B^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet))\ (\mathsf{snd}\ P).\end{aligned}$$

Thus the recovered predicate is inductively exactly the product predicate from Section 3.2:

$$(\mathsf{PROD}\ A\ B)^\bullet\ \gamma^\circ\ \gamma^\bullet\ P = A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ \gamma^\circ\ \gamma^\bullet\ (\mathsf{snd}\ P).$$

Thus storing computability predicates flatly does not change the predicate seen by terms. Glued substitutions and terms keep the clauses of Section 3, as SUB and TM refer only to the counit-applied predicates $\Gamma^\bullet \coloneqq \varepsilon(\Gamma^\bullet_\flat)$ and $A^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \varepsilon(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet)$. Contravariant transports are handled through the same counit-defined views. The full set of clauses appears in Appendix B.

[8] Here A° is implicitly regarded as an element of $\flat(\mathsf{Ty}\ \cdot)$, using discreteness of Ty $\cdot$.

4.3 The Universe Glued Type

With the infrastructure of the flat modality, we can finally define UNIV. For each syntactic type $A^\circ : \mathsf{Ty}\ \cdot$, the candidate Cand is the collection of all computability predicates on that type.

$$\mathsf{Cand}(A^\circ) \coloneqq \sum_{A^\bullet_\flat : \mathsf{Pred}(A^\circ)} \mathsf{isContrav}\ (\varepsilon(A^\bullet_\flat))$$

The computability of a universe $M : \mathsf{Tm}\ \cdot\ \mathsf{U}$ is then candidates on its decoded type $\mathsf{El}\ M$.

$$\mathsf{UNIV}^\circ \coloneqq \mathsf{U},$$

$$\mathsf{UNIV}^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet \coloneqq \big(\lambda M.\ \mathsf{Cand}(\mathsf{El}\ M)\big)^\flat.$$

There are two uses of flatness here. Each candidate stores a flat predicate code $A^\bullet_\flat : \mathsf{Pred}(A^\circ)$, and the universe predicate itself is stored as a flat code. The contravariance proof for UNIV lifts candidates along a universe reduction $f : M \leq_{\mathsf{Tm}\ \cdot\ \mathsf{U}} M'$. Let

$$e_f \coloneqq \mathsf{disc}_{\mathsf{Ty}}(\mathsf{El}\ f) : \mathsf{El}\ M = \mathsf{El}\ M'.$$

Given a candidate $(A'^\bullet_\flat, c_{A'}) : \mathsf{Cand}(\mathsf{El}\ M')$, its contravariant transport back to $\mathsf{Cand}(\mathsf{El}\ M)$ is obtained by transporting the flat predicate code backwards along e_f:

$$f^*(A'^\bullet_\flat) \coloneqq \mathsf{transport}_{T \mapsto \mathsf{Pred}(T)}(e_f^{-1})\, A'^\bullet_\flat.$$

The contravariance proof is supplied by Lemma 4.3. After applying the counit, this is the ordinary conversion action from Section 4.2.1: $\varepsilon(f^*(A'^\bullet_\flat))\ N = \varepsilon(A'^\bullet_\flat)(\mathsf{conv}_{\mathsf{El}\ f}\ N)$.

The only point needing flatness is uniqueness of this lift. The crucial point is that the comparison of candidates is done at the level of flat predicate codes at $\mathsf{Pred}(\mathsf{El}\ M)$, which are discrete. Hence all inequalities at $\mathsf{Pred}(\mathsf{El}\ M)$ are equalities. Let $(A^\bullet_\flat, c_A)$ is any other candidate over $\mathsf{El}\ M$ together with a lift to $(A'^\bullet_\flat, c_{A'})$ over f. Projecting to predicate codes and converting the target back along e_f gives an inequality $A^\bullet_\flat \leq f^*(A'^\bullet_\flat)$ in $\mathsf{Pred}(\mathsf{El}\ M)$. This type is flat, hence discrete, so the inequality is an equality. After identifying the predicate codes, the contravariance witnesses also agree. Thus every lift is equal to the distinguished candidate, which gives the contractibility required by c_{UNIV}.

4.4 El and Code

The remaining Tarski operations are mostly bookkeeping. A term of UNIV already contains, at each glued substitution, a candidate for its decoded syntactic type. Thus, for $M : \mathsf{TM}\ \Gamma\ \mathsf{UNIV}$, write

$$M^\bullet\ \gamma^\circ\ \gamma^\bullet = (A^\bullet_{\flat,\gamma,\gamma^\bullet},\ c_{A,\gamma,\gamma^\bullet}) : \mathsf{Cand}\big(\mathsf{El}(M^\circ[\gamma^\circ])\big).$$

The interpretation of EL simply projects this candidate. In the other direction, if $A : \mathsf{TY}\ \Gamma$, then $\mathsf{CODE}\ A$ must produce a candidate at $\mathsf{Cand}\big(\mathsf{El}(\mathsf{Code}\ A^\circ[\gamma^\circ])\big)$, while A supplies $\big(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet,\ c_A\ \gamma^\circ\ \gamma^\bullet\big) : \mathsf{Cand}(A^\circ[\gamma^\circ])$. The reduction $\mathsf{U}_\beta[\gamma^\circ] : \mathsf{El}(\mathsf{Code}\ A^\circ[\gamma^\circ]) \leq A^\circ[\gamma^\circ]$ therefore lets the universe's contravariant structure pull the latter candidate back to the decoded code.

$$\begin{array}{lcl} (\mathsf{EL}\ M)^\circ & \coloneqq & \mathsf{El}\ M^\circ \\ (\mathsf{EL}\ M)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet & \coloneqq & A^\bullet_{\flat,\gamma,\gamma^\bullet} \\ c_{\mathsf{EL}\ M}\ \gamma^\circ\ \gamma^\bullet & \coloneqq & c_{A,\gamma,\gamma^\bullet}. \end{array} \qquad \begin{array}{lcl} (\mathsf{CODE}\ A)^\circ & \coloneqq & \mathsf{Code}\ A^\circ \\ (\mathsf{CODE}\ A)^\bullet\ \gamma^\circ\ \gamma^\bullet & \coloneqq & \boxed{\big(\mathsf{U}_\beta[\gamma^\circ]\big)^*}\big(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet,\ c_A\ \gamma^\circ\ \gamma^\bullet\big). \end{array}$$

4.5 Revisiting Dependency

The flat-code definition of TY supplies the coherence needed for dependency. Closed substitutions into Γ are compared by directed inequalities, not judgmental equalities. Once such a comparison is used in a type, it lands in the discretized type judgment. Thus, for $A : \mathsf{TY}\ \Gamma$ and $(f^\circ, f^\bullet) : (\gamma_0^\circ, \gamma_0^\bullet) \leq (\gamma_1^\circ, \gamma_1^\bullet)$, the syntactic component gives a type reduction and hence a judgmental equality; after opening the flat code, this is used as the substitution-invariance path substInv, which follows from monotonicity on $A_\flat^\bullet$ and flatness of predicate codes:

$$A^\circ[f^\circ] : A^\circ[\gamma_0^\circ] \leq_{\mathsf{Ty}} \cdot\ A^\circ[\gamma_1^\circ], \qquad \mathsf{substInv} : A^\bullet\ \gamma_0^\circ\ \gamma_0^\bullet\ (\mathsf{conv}^{-1}_{A^\circ[f^\circ]}\ M_1^\circ) = A^\bullet\ \gamma_1^\circ\ \gamma_1^\bullet\ M_1^\circ.$$

This is the only extra move needed for dependent type formers. For a dependent pair, as in the simple type case, the first component is obtained by ordinary contravariance:

$$\Phi \coloneqq (\Sigma_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet) : A^\bullet\ \gamma^\circ\ \gamma^\bullet\ \big(\mathsf{fst}\ (\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ]\big).$$

For the second component, the available evidence is $N^\bullet\ \gamma^\circ\ \gamma^\bullet$. However, after the first projection reduction, the second component must live over the first projection of the pair. Thus the desired witness is obtained by transporting this evidence along the substitution-invariance path substInv:

$$\begin{array}{lcl} N^\bullet\ \gamma^\circ\ \gamma^\bullet & : & B^\bullet\ (\gamma^\circ, M^\circ[\gamma^\circ])\ (\gamma^\bullet, M^\bullet\ \gamma^\circ\ \gamma^\bullet)\ (N^\circ[\gamma^\circ]), \\ \mathsf{transport}_{\mathsf{substInv}}(N^\bullet\ \gamma^\circ\ \gamma^\bullet) & : & B^\bullet\ (\gamma^\circ, \mathsf{fst}\ (\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ])\ (\gamma^\bullet, \Phi)\ \big(\mathsf{conv}^{-1}_{B^\circ[(\gamma^\circ, \Sigma_{\beta_1})]} N^\circ[\gamma^\circ]\big). \end{array}$$

Here substInv is obtained by applying substitution-invariance along the comparison induced by the same Σ_{β_1} reduction:

$$\big((\gamma^\circ, \Sigma_{\beta_1}), \_\big) : \big((\gamma^\circ, \mathsf{fst}\ (\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ]), (\gamma^\bullet, \Phi)\big) \leq \big((\gamma^\circ, M^\circ[\gamma^\circ]), (\gamma^\bullet, M^\bullet\ \gamma^\circ\ \gamma^\bullet)\big).$$

This converted endpoint is precisely the one appearing in the closed second projection reduction:

$$\Sigma_{\beta_2}[\gamma^\circ] : \mathsf{snd}\ (\mathsf{pair}\ M^\circ\ N^\circ)[\gamma^\circ] \leq \mathsf{conv}^{-1}_{B^\circ[(\gamma^\circ, \Sigma_{\beta_1})]} N^\circ[\gamma^\circ].$$

Contravariance of B can now be applied once more, giving the second witness and hence the semantic component of pairing:

$$(\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet \quad \coloneqq \quad ((\Sigma_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet), (\Sigma_{\beta_2}[\gamma^\circ])^*(\mathsf{transport}_{\mathsf{substInv}}(N^\bullet\ \gamma^\circ\ \gamma^\bullet))).$$

The point of this detour is modest but important: dependent syntactic reductions may change the type in which a term lives. Discreteness of Ty and flatness of predicate codes turn that type change into the exact predicate transport needed by the logical relation. Logical relations for other dependent type formers are handled similarly and can be found in Appendix B.

5 BINARY LOGICAL RELATIONS AND PARAMETRICITY

The unary model proves canonicity by attaching to each closed term a computability witness. Parametricity uses the same construction with one minor change: the semantic component relates two closed instances of the same syntactic object. The proof-relevant universe can therefore contain contravariant, heterogeneous relations, supporting representation-independence arguments.

Lemma 5.1 (Binary Contravariance - Riehl and Shulman 2017, Proposition 8.21). *A binary family $R : A_L \to A_R \to \mathcal{U}$ is contravariant if and only if $R\ M_L\ (-) : A_R \to \mathcal{U}$ and $R\ (-)\ M_R : A_L \to \mathcal{U}$ are both contravariant for each $M_L : A_L$ and $M_R : A_R$, respectively.*

Ordinary binary logical relations phrase closure under expansion one component at a time:

$$\text{if } M_L \longrightarrow_\beta M_L', \text{then } R\ M_L'\ M_R \Longrightarrow R\ M_L\ M_R, \qquad \text{if } M_R \longrightarrow_\beta M_R', \text{then } R\ M_L\ M_R' \Longrightarrow R\ M_L\ M_R.$$

Lemma 5.1 identifies the directed version of these two principles with contravariance of the uncurried relation. Thus reductions $f_L : M_L \le M'_L$ and $f_R : M_R \le M'_R$ determine a transport

$$(f_L, f_R)^* : R\ M'_L\ M'_R \to R\ M_L\ M_R.$$

This transport also records functoriality and its universal property. As in Section 4, the relation carried by a type must be stored as a flat predicate code. For closed types $A^\circ_L, A^\circ_R : \mathsf{Ty}\ \cdot$, write

$$\mathsf{Pred}_2(A^\circ_L, A^\circ_R) \coloneqq \mathsf{let}^\flat\ T_L = A^\circ_L\ \mathsf{in}\ \mathsf{let}^\flat\ T_R = A^\circ_R\ \mathsf{in}\ \flat(\mathsf{Tm}\ \cdot\ T_L \to \mathsf{Tm}\ \cdot\ T_R \to \mathcal{U}),$$
$$\mathsf{Cand}_2(A^\circ_L, A^\circ_R) \coloneqq \textstyle\sum_{R_\flat : \mathsf{Pred}_2(A^\circ_L, A^\circ_R)} \mathsf{isContrav}\ (\varepsilon(R_\flat)).$$

The binary glued records are then the pointwise binary versions of the unary ones:

record $\mathsf{CTX}_2 : \mathcal{U}$ **where**
$\Gamma^\circ : \mathsf{Ctx}$
$\Gamma^\bullet_\flat : \flat(\mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathcal{U})$
$\Gamma^\bullet \coloneqq \varepsilon(\Gamma^\bullet_\flat)$

record $\mathsf{TY}_2\ (\Gamma : \mathsf{CTX}_2) : \mathcal{U}$ **where**
$A^\circ : \mathsf{Ty}\ \Gamma^\circ$
$A^\bullet_\flat : (\gamma^\circ_L\ \gamma^\circ_R : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to \Gamma^\bullet\ \gamma^\circ_L\ \gamma^\circ_R$
$\quad \to \boxed{\mathsf{Pred}_2(A^\circ[\gamma^\circ_L], A^\circ[\gamma^\circ_R])}$
$A^\bullet\ \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet \coloneqq \varepsilon(A^\bullet_\flat\ \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet)$
$c_A : \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet \to \mathsf{isContrav}\ (A^\bullet\ \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet).$

A binary term has one syntactic component and relates its two closed instances:

record $\mathsf{TM}_2\ (\Gamma : \mathsf{CTX}_2)(A : \mathsf{TY}_2\ \Gamma) : \mathcal{U}$ **where**
$M^\circ : \mathsf{Tm}\ \Gamma^\circ\ A^\circ$
$M^\bullet : (\gamma^\circ_L\ \gamma^\circ_R : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ_L\ \gamma^\circ_R) \to \boxed{A^\bullet\ \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet\ (M^\circ[\gamma^\circ_L])\ (M^\circ[\gamma^\circ_R])}.$

The semantics universe is obtained by replacing unary candidates by binary candidates:

$$\mathsf{UNIV}_2{}^\bullet_\flat\ \gamma^\circ_L\ \gamma^\circ_R\ \gamma^\bullet \coloneqq \big(\lambda M_L\ M_R.\ \mathsf{Cand}_2(\mathsf{El}\ M_L, \mathsf{El}\ M_R)\big)^\flat.$$

As an example, consider the batched queue representation-independence argument of Sterling and Harper [2021]. The abstract package is a queue interface with a representation type, empty queue, enqueue, and dequeue operations:

$$\mathsf{Queue} \coloneqq \mathsf{Tm}\ \cdot\Big(\Sigma\ U\ \big(\mathsf{El}\ \alpha \times (\mathsf{nat} \to \mathsf{El}\ \alpha \to \mathsf{El}\ \alpha) \times (\mathsf{El}\ \alpha \to \mathsf{nat} \times \mathsf{El}\ \alpha)\big)\Big).$$
$$\mathsf{QUEUE} \coloneqq \mathsf{TM}_2\ \cdot\Big(\Sigma\ \mathsf{UNIV}_2\ \big(\mathsf{EL}\ \alpha \times (\mathsf{NAT} \to \mathsf{EL}\ \alpha \to \mathsf{EL}\ \alpha) \times (\mathsf{EL}\ \alpha \to \mathsf{NAT} \times \mathsf{EL}\ \alpha)\big)\Big).$$

Representation independence amounts to constructing an element of QUEUE, the glued version of this package. At the UNIV_2 component, choose α_L as the code of $\mathsf{List\ nat}$ for the simple list queue, and α_R as the code of $\mathsf{List\ nat} \times \mathsf{List\ nat}$. The abstraction relation has type

$$R : \mathsf{Tm}\ \cdot\ (\mathsf{El}\ \alpha_L) \to \mathsf{Tm}\ \cdot\ (\mathsf{El}\ \alpha_R) \to \mathcal{U}.$$

The relation of Sterling and Harper [2021] is the extensional queue invariant

$$R\ \ell\ (f, b) \coloneqq \ell = f +\!\!+ \mathsf{reverse}\ b.$$

This relation is, however, not contravariant in the present setting. Instead, we use the directed analogue below, which replaces equality by a common "upper bound". For each fixed xs, the two inequalities form a product of representable families, so contravariance is supplied by Lemma 3.6.

$$R\ \ell\ (f, b) \coloneqq \sum_{xs : \mathsf{List}\ \mathbb{N}} \big(\ell \le \ulcorner xs \urcorner\big) \times \big(f +\!\!+ \mathsf{reverse}\ b \le \ulcorner xs \urcorner\big).$$

6 CONCLUSION, RELATED AND FUTURE WORK

The result proved in this paper is, in one sense, unsurprising: logical relations should validate canonicity and parametricity even when syntax is presented by reductions rather than judgmental equalities. The contribution is the type-theoretic internalization of this idea. The choice of meta-language matters: inequality types of simplicial type theory provide a native account of reduction, while contravariant families supply the proof-relevant form of the expansion lemma needed by the logical relation. In another respect, this paper also gives an application of simplicial type theory beyond synthetic ∞-category theory. The remainder of this section situates the construction among related work and outlines directions for future development.

6.1 Mechanization

Mechanizing gluing and proof-relevant logical relations with intrinsic syntax, as in the present work, is difficult mainly because of *transport*. Many equations, such as naturality for pairs $(\mathsf{pair}\ a\ b)[\sigma] = \mathsf{pair}\ (a[\sigma])\ (b[\sigma])$, are not definitional; here they arise from path constructors in a higher inductive type. A mechanization must therefore transport along paths and reason about equalities, and in the present setting inequalities, between transported terms. Existing approaches either use extensional proof assistants with equality reflection such as those of NuPRL-style [Constable et al. 1986], as in Li et al. [2026], or encode the syntax in specific ways so that these equations become definitional [Kaposi and Pujet 2025; Xie and Bense 2026] using variants of observational type theory [Altenkirch et al. 2007; Pujet and Tabareau 2022]. The present work is committed to homotopy type theory as a meta-language, so neither move is available. Our experience therefore matches Chen et al. [2026]: formalizing intrinsic syntax in variants of Cubical Agda remains challenging.

We provide a Cubical Agda mechanization for the simply typed object language of Sections 2 and 3. It formalizes directed inequalities and contravariance, intrinsic syntax with products, functions, and Booleans, whose gluing model yields Boolean canonicity. Cubical Agda is used because it provides higher inductive types and a higher-dimensional meta-language; the simplicial layer is represented by postulating a directed interval and its bounded ordering. In the simply typed case, the transport burden is already somewhat unpleasant, though manageable with care. The mechanization should therefore be read as a proof of concept for the directed technique and for using simplicial type theory in logical relations. The dependently typed extension in Section 4 presents the transport-heavy intrinsic syntax problem emphasized by Chen et al. [2026], and its mechanization is left for future work. In particular, Agda has an experimental flat modality [Licata et al. 2018; The Agda Team 2026], matching out use in Section 4.2 which we conjecture to be be useful. Another relevant system is Rzk [Kudasov 2023], a proof assistant based on simplicial type theory for synthetic ∞-category theory [Kudasov et al. 2024]. Unlike *op. cit.*, the present mechanization does not develop the full simplicial layer, such as shapes, but it suffices to formalize directed gluing and prove canonicity for the simply typed object language.

6.2 Equational Proof-Relevant Logical Relations

Existing gluing arguments give proof-relevant logical-relations proofs of canonicity, normalization, and parametricity for many type theories: simply typed λ-theories [Sterling and Spitters 2018]; dependent type theory [Altenkirch and Kaposi 2017; Coquand 2018]; abstract canonicity and parametricity constructions [Bocquet et al. 2023; Kaposi et al. 2019a]; homotopy canonicity [Coquand et al. 2019; Shulman 2015]; cubical and multimodal normalization [Gratzer 2022; Sterling and Angiuli 2021]; canonicity with indexed inductive-recursive types [Kovács 2026]; and proof-relevant parametricity for ML-style modules [Sterling and Harper 2021].

These arguments are equational: the syntactic component is typically a CwF or a signature in a semantic logical framework [Harper 2021]. The logical predicate therefore ranges over judgmental equivalence classes, so object-language equations are used by equality transport. Compared to equality transport which always exists, directed transport is available only for contravariant families. Thus the present work replaces equality by a weaker directed structure while recovering the needed logical-relations argument. In Section 4 part of the equational story is recovered by discretizing syntactic types: judgmentally equal types must induce the same computability predicate. A more controlled way to recover judgmental equalities may come from synthetic phase distinctions. Introduced by Sterling and Harper [2021] and used for cost analysis, information flow, and compilation [Grodin et al. 2026, 2024; Niu et al. 2022; Sterling and Harper 2022; Theocharis and Brady 2026], phase distinctions add a type-theoretic proposition ϕ, inducing a two-world Kripke model containing an "in-the-phase" world where ϕ is available and an "out-of-the-phase" world where it is not. Judgmental equalities could live in the phase, while reductions by inequalities live out of it; monotonicity of Kripke worlds is preserved because reduction is contained in judgmental equality. Synthetically, this requires a modality that turns inequalities into equalities. In a similar directed setting, Grodin et al. [2024] proposes a monadic, localization-style [Christensen et al. 2020; Rijke et al. 2020] modality $\bigcirc$ satisfying $\bigcirc(x \leq y \to x = y)$ that could be useful in the present setting. A proper phase distinction between judgmental equality and reduction is left for future work, and may be the right way to handle identity types in the directed setting.

6.3 Synthetic Tait Computability

Synthetic Tait computability [Sterling 2021] gives an abstract formulation of gluing. It has been used for cubical and modal type theories [Gratzer 2022; Sterling and Angiuli 2021], ML-style modules and effect handlers [Sterling and Harper 2021; Yang 2024; Yang and Wu 2026], dependent call-by-push-value [Li and Harper 2025], controlling unfolding [Gratzer et al. 2025a], and simplicial type theory used here [Weinberger et al. 2022].

The synthetic presentation works in the internal language of the gluing category. Instead of specifying an external computability predicate $\mathsf{Tm}\,\cdot\,(A^\circ[\gamma^\circ]) \to \mathcal{U}$, one specifies an element of $\mathcal{U}$ whose syntactic part is $\mathsf{Tm}(A)$, using an extension type $\{\mathcal{U} \mid \mathsf{syn} \hookrightarrow \mathsf{Tm}(A)\}$. This gives a fibered view of the indexed construction used in the present work. The directed version should be obtainable by working in the internal language of the gluing category of simplicial sets and requiring the corresponding extension type to be contravariant, recovering the role played here by the proof-relevant expansion. A further advantage of this approach is its use of second-order syntax: contexts and substitutions need not be treated separately from types and terms, and the naturality conditions that appear in CwF-based gluing arguments are abstracted away. This is particularly useful in the setting of normalization [Gratzer 2022; Sterling 2021; Sterling and Angiuli 2021], where the construction becomes more complex. In present work canonicity is used as a compact illustration of the broader method, namely a proof-relevant logical-relations argument over syntax presented by directed reductions. It raises the further question of how to scale the method to richer logical relations, such as normalization, where a directed version of synthetic Tait computability may be useful.

6.4 Simplicial Type Theory and Directed Type Theory

Simplicial type theory is useful here because it combines directed inequalities and contravariant families with the usual infrastructure of homotopy type theory, including dependent types and HITs. Its main use so far in the literature has been synthetic ∞-category theory [Bardomiano-Martínez 2025; Buchholtz and Weinberger 2023; Gratzer et al. 2025b, 2026a,b; Riehl and Shulman 2017; Weinberger 2022, 2024]; the present work uses the same directed structure for the metatheory

of syntax and reduction. Indeed, Seely [1987] already considered modeling reduction in the lambda calculus in a categorical setting, albeit a slightly different one from ours. The beauty of the present work is that, because simplicial type theory is a synthetic theory of category theory, we can reason about categorically motivated presentations of reduction in a type-theoretic language.

Other directed type theories often equip inequality types with J-like eliminations or more fine-grained variance control [Laretto et al. 2026; Licata and Harper 2011; Neumann 2025a,b; Neumann and Altenkirch 2025; North 2019; Nuyts 2015]. These systems can substantially change ordinary type formers, such as Π-types, so the present construction may not transfer directly. Neumann [2025b] gives the closest point of contact: inequalities are used to represent reductions for a simple expression language in synthetic rewriting systems. The present work capitalizes on this idea in Section 2, albeit in a slightly different setting, to model reductions in type theory.

6.5 Binary Parametricity

Section 5 extends to proof-relevant parametricity. It can be seen as an analytical reconstruction of Sterling and Harper [2021]'s synthetic left and right syntactic modalities. The key point is to separate homogeneous vertical relations for judgmental equalities or reductions (M_L *vs.* M'_L and M_R *vs.* M'_R) from heterogeneous horizontal relations for parametricity (M_L *vs.* M_R and M'_L *vs.* M'_R).

$$\begin{array}{ccc} M_L & \overset{\Phi : A^\bullet\ M_L\ M_R}{\text{- - - - - - - - -}} & M_R \\ {\scriptstyle f_L : M_L \le M'_L}\ \downarrow & & \downarrow\ {\scriptstyle f_R : M_R \le M'_R} \\ M'_L & \underset{\Phi' : A^\bullet\ M'_L\ M'_R}{\text{- - - - - - - - -}} & M'_R \end{array}$$

This differs from the usual zig-zag closure of binary logical relations [Krishnaswami and Dreyer 2013], where one binary relation accounts for both judgmental equality and parametricity. The additional closure condition is then needed to move along the vertical dimension inside the horizontal relation. Categorically, the parametricity relation here, as in Sterling and Harper [2021], is a span $\mathsf{Tm} \cdot A_L \overset{s}{\leftarrow} A^\bullet \overset{t}{\rightarrow} \mathsf{Tm} \cdot A_R$; an element $M^\bullet : A^\bullet$ witnesses that $M_L = s(M^\bullet)$ and $M_R = t(M^\bullet)$ are parametrically related. Zig-zag closure instead makes the relation a quasi-partial equivalence relation, viewable as such a span equipped with heterogeneous transitivity. That extra structure is needed for the fundamental theorem only when the vertical and horizontal dimensions have been mixed. This separation clarifies that contravariance belongs to the vertical dimension: it transports computability along reductions, rather than expressing parametricity itself. It also suggests a double-categorical semantics for parametricity, left for future work.

DATA AVAILABILITY STATEMENT

The simply-typed variant of directed logical relation as described in Sections 2 and 3 is mechanized in Cubical Agda [Norell 2009; Vezzosi et al. 2019]. In particular, directed syntax is constructed via higher inductive types and reflective subuniverses according to Section 2.2.1 and Appendix A. The logical relations covers products, functions, and Booleans, with a fundamental theorem that concludes directed Boolean canonicity. The accompanying mechnization provides a README.agda that maps Agda definitions to definitions, theorems, and constructions presented in this paper.

ACKNOWLEDGMENTS

The authors thank Jonathan Sterling and Yue Yao for adjacent collaborations; and Daniel Gratzer and Ulrik Buchholtz for suggesting the use of the flat modality in simplicial type theory.

This material is based upon work supported by the United States Air Force Office of Scientific Research under grant numbers FA9550-21-0009 and FA9550-23-1-0434 (Tristan Nguyen, program manager) and the National Science Foundation under award number 2615896. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR, NSF. The authors are grateful for support from Jane Street Capital.

REFERENCES

Benedikt Ahrens, Paige Randall North, and Niels van der Weide. 2023. Bicategorical type theory: semantics and syntax. *Mathematical Structures in Computer Science* 33, 10 (2023), 868–912. https://doi.org/10.1017/S0960129523000312

Stuart Allen. 1987. *A Non-Type-Theoretic Semantics For Type-Theoretic Language.* Ph. D. Dissertation. Cornell University. https://hdl.handle.net/1813/6706

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. 2018. *Quotient Inductive-Inductive Types.* Springer International Publishing, 293–310. https://doi.org/10.1007/978-3-319-89366-2_16

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* (St. Petersburg, FL, USA) *(POPL '16)*. Association for Computing Machinery, New York, NY, USA, 18–29. https://doi.org/10.1145/2837614.2837638

Thorsten Altenkirch and Ambrus Kaposi. 2017. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science* 13, 4 (2017). https://doi.org/10.23638/LMCS-13(4:1)2017

Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational equality, now!. In *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification* (Freiburg, Germany) *(PLPV '07)*. Association for Computing Machinery, New York, NY, USA, 57–68. https://doi.org/10.1145/1292597.1292608

Carlo Angiuli. 2019. *Computational Semantics of Cartesian Cubical Type Theory.* Ph. D. Dissertation. Carnegie Mellon University. https://carloangiuli.com/papers/thesis.pdf

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. 2021. Syntax and models of Cartesian cubical type theory. *Mathematical Structures in Computer Science* 31, 4 (2021), 424–468. https://doi.org/10.1017/S0960129521000347

César Bardomiano-Martínez. 2025. Exponentiable functors between synthetic ∞-categories. *Mathematical Structures in Computer Science* (2025). https://doi.org/10.1017/S0960129525100339 arXiv:2407.18072 [math.CT]

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) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 260)*, Marco Gaboardi and Femke van Raamsdonk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 18:1–18:23. https://doi.org/10.4230/LIPIcs.FSCD.2023.18

Ulrik Buchholtz and Jonathan Weinberger. 2023. Synthetic fibered $(\infty, 1)$-category theory. *Higher Structures* 7, 1 (2023), 74–165. https://doi.org/10.21136/HS.2023.04

Evan Cavallo, Emily Riehl, and Christian Sattler. 2026. Directed univalence for simplicial objects in an ∞-topos. arXiv:2607.02420 [math.CT] https://arxiv.org/abs/2607.02420

Liang-Ting Chen, Fredrik Nordvall Forsberg, and Tzu-Chun Tsai. 2026. Can We Formalise Type Theory Intrinsically without Any Compromise? A Case Study in Cubical Agda. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Rennes, France) *(CPP '26)*. Association for Computing Machinery, New York, NY, USA, 201–215. https://doi.org/10.1145/3779031.3779090

J. Daniel Christensen, Morgan Opie, Egbert Rijke, and Luis Scoccola. 2020. Localization in Homotopy Type Theory. *Higher Structures* 4 (Feb. 2020), 1–32. Issue 1. https://doi.org/10.21136/HS.2020.01

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 69)*, Tarmo Uustalu (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 5:1–5:34. https://doi.org/10.4230/LIPIcs.TYPES.2015.5

Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. 1986. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, NJ.

Thierry Coquand. 2018. Canonicity and normalisation for Dependent Type Theory. arXiv:1810.09367 [cs.PL] https://arxiv.org/abs/1810.09367

Thierry Coquand, Simon Huber, and Christian Sattler. 2019. Homotopy Canonicity for Cubical Type Theory. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:23. https://doi.org/10.4230/LIPIcs.FSCD.2019.11

Peter Dybjer. 1996. Internal type theory. In *Types for Proofs and Programs*, Stefano Berardi and Mario Coppo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 120–134.

Marcelo Fiore. 2002. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP '02)*. Association for Computing Machinery, 26–37. https://doi.org/10.1145/571157.571161

Marcelo P. Fiore. 1997. An Enrichment Theorem for an Axiomatisation of Categories of Domains and Continuous Functions. *Mathematical Structures in Computer Science* 7, 5 (Oct. 1997), 591–618. https://doi.org/10.1017/S0960129597002429

Daniel Gratzer. 2022. Normalization for Multimodal Type Theory. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, Haifa Israel, 1–13. https://doi.org/10.1145/3531130.3532398

Daniel Gratzer. 2023. *Syntax and Semantics of Modal Type Theory*. Ph. D. Dissertation. Aarhus University. https://www.danielgratzer.com/papers/phd-thesis.pdf

Daniel Gratzer, Jonathan Sterling, Carlo Angiuli, Thierry Coquand, and Lars Birkedal. 2025a. Controlling unfolding in type theory. *Mathematical Structures in Computer Science* 35 (2025), e38. https://doi.org/10.1017/S0960129525100327

Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. 2025b. The Yoneda embedding in simplicial type theory. In *2025 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 127–142. https://doi.org/10.1109/LICS65433.2025.00017

Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. 2026a. Directed univalence in simplicial homotopy type theory. arXiv:2407.09146 [cs.LO] https://arxiv.org/abs/2407.09146

Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. 2026b. The ∞-category of ∞-categories in simplicial type theory. arXiv:2602.02218 [cs.LO] https://arxiv.org/abs/2602.02218

Harrison Grodin, Runming Li, and Robert Harper. 2026. Abstraction Functions as Types: Modular Verification of Cost and Behavior in Dependent Type Theory. *Proc. ACM Program. Lang.* 10, POPL, Article 31 (Jan. 2026), 28 pages. https://doi.org/10.1145/3776673

Harrison Grodin, Yue Niu, Jonathan Sterling, and Robert Harper. 2024. Decalf: A Directed, Effectful Cost-Aware Logical Framework. *Proc. ACM Program. Lang.* 8, POPL, Article 10 (Jan. 2024), 29 pages. https://doi.org/10.1145/3632852

Robert Harper. 1992. Constructing Type Systems over an Operational Semantics. *Journal of Symbolic Computation* 14, 1 (1992), 71–84. https://doi.org/10.1016/0747-7171(92)90026-Z

Robert Harper. 2021. An Equational Logical Framework for Type Theories. https://arxiv.org/abs/2106.01484

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (Nov. 2018), e20. https://doi.org/10.1017/S0956796818000151

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) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 25:1–25:19. https://doi.org/10.4230/LIPIcs.FSCD.2019.25

Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. 2019b. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.* 3, POPL, Article 2 (Jan. 2019), 24 pages. https://doi.org/10.1145/3290315

Ambrus Kaposi and Loïc Pujet. 2025. Type Theory in Type Theory using a Strictified Syntax. *Proc. ACM Program. Lang.* ICFP (Aug. 2025), 31 pages. https://doi.org/10.1145/3747535

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

Neelakantan R. Krishnaswami and Derek Dreyer. 2013. Internalizing Relational Parametricity in the Extensional Calculus of Constructions. In *Computer Science Logic 2013 (CSL 2013) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 23)*, Simona Ronchi Della Rocca (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 432–451. https://doi.org/10.4230/LIPIcs.CSL.2013.432

Nikolai Kudasov. 2023. Rzk: An experimental proof assistant based on a type theory for synthetic ∞-categories. https://github.com/rzk-lang/rzk

Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. 2024. Formalizing the ∞-Categorical Yoneda Lemma. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs* (London, UK) *(CPP 2024)*. Association for Computing Machinery, New York, NY, USA, 274–290. https://doi.org/10.1145/3636501.3636945

Andrea Laretto, Fosco Loregian, and Niccolò Veltri. 2026. Di- is for Directed: First-Order Directed Type Theory via Dinaturality. *Proc. ACM Program. Lang.* 10, POPL, Article 61 (Jan. 2026), 31 pages. https://doi.org/10.1145/3776703

Runming Li and Robert Harper. 2025. Canonicity for Cost-Aware Logical Framework via Synthetic Tait Computability. arXiv:2504.12464 (April 2025). https://doi.org/10.48550/arXiv.2504.12464 arXiv:2504.12464 [cs].

Runming Li, Yue Yao, and Robert Harper. 2026. Mechanizing Synthetic Tait Computability in Istari. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Rennes, France) *(CPP '26)*. Association for Computing Machinery, New York, NY, USA, 231–247. https://doi.org/10.1145/3779031.3779085

Daniel R. Licata and Robert Harper. 2011. 2-Dimensional Directed Type Theory. *Electronic Notes in Theoretical Computer Science* 276 (2011), 263–289. https://doi.org/10.1016/j.entcs.2011.09.026 Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVII).

Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. 2018. Internal Universes in Models of Homotopy Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 108)*, Hélène Kirchner (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 22:1–22:17. https://doi.org/10.4230/LIPIcs.FSCD.2018.22

Per Martin-Löf. 1998. An Intuitionistic Theory of Types. In *Twenty-Five Years of Constructive Type Theory*, Giovanni Sambin and Jan M. Smith (Eds.). Oxford Logic Guides, Vol. 36. Oxford University Press, New York, NY, USA, 127–172. Preliminary version 1972; reprinted version of an unpublished report from 1972. Proceedings of the Venice meeting, 1995.

John C. Mitchell and Andre Scedrov. 1992. Notes on Sconing and Relators. In *Selected Papers from the Workshop on Computer Science Logic (CSL '92)*. Springer-Verlag, Berlin, Heidelberg, 352–378.

Jacob Neumann. 2025a. *A Generalized Algebraic Theory of Directed Equality.* Ph. D. Dissertation. University of Nottingham.

Jacob Neumann. 2025b. A Judgmental Construction of Directed Type Theory. arXiv:2510.17494 [cs.LO] https://arxiv.org/abs/2510.17494

Jacob Neumann and Thorsten Altenkirch. 2025. Synthetic 1-Categories in Directed Type Theory. *LIPIcs, Volume 336, TYPES 2024* 336, 7:1–7:23. https://doi.org/10.4230/LIPICS.TYPES.2024.7

Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. 2022. A cost-aware logical framework. *Proc. ACM Program. Lang.* 6, POPL, Article 9 (Jan. 2022), 31 pages. https://doi.org/10.1145/3498670

Ulf Norell. 2009. Dependently Typed Programming in Agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation (TLDI '09)*. Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/1481861.1481862

Paige Randall North. 2019. Towards a Directed Homotopy Type Theory. *Electronic Notes in Theoretical Computer Science* 347 (2019), 223–239. https://doi.org/10.1016/j.entcs.2019.09.012 Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics.

Andreas Nuyts. 2015. *Towards a Directed Homotopy Type Theory based on 4 Kinds of Variance.* Master's thesis. KU Leuven, Leuven, Belgium. https://people.cs.kuleuven.be/~dominique.devriese/ThesisAndreasNuyts.pdf Promotor: Frank Piessens; Begeleiders: Dominique Devriese and Jesper Cockx.

Frank Pfenning and Rowan Davies. 2001. A Judgmental Reconstruction of Modal Logic. *Mathematical Structures in Computer Science* 11, 4 (Aug. 2001), 511–540. https://doi.org/10.1017/S0960129501003322

Loïc Pujet and Nicolas Tabareau. 2022. Observational equality: now for good. *Proc. ACM Program. Lang.* 6, POPL, Article 32 (Jan. 2022), 27 pages. https://doi.org/10.1145/3498693

Emily Riehl and Michael Shulman. 2017. A type theory for synthetic ∞-categories. *Higher Structures* 1 (2017), 147–224. Issue 1. arXiv:1705.07442 [math.CT] https://journals.mq.edu.au/index.php/higher_structures/article/view/36

Egbert Rijke, Michael Shulman, and Bas Spitters. 2020. Modalities in homotopy type theory. *Logical Methods in Computer Science* Volume 16, Issue 1, Article 2 (Jan 2020). https://doi.org/10.23638/LMCS-16(1:2)2020

Robert A. G. Seely. 1987. Modelling Computations: A 2-Categorical Framework. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science (LICS 1987)* (Ithaca, NY, USA). IEEE Computer Society Press, 65–71.

Michael Shulman. 2015. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science* 25, 5 (2015), 1203–1277. https://doi.org/10.1017/S0960129514000565

Michael Shulman. 2018. Brouwer's Fixed-Point Theorem in Real-Cohesive Homotopy Type Theory. *Mathematical Structures in Computer Science* 28, 6 (June 2018), 856–941. https://doi.org/10.1017/S0960129517000147

Jonathan Sterling. 2021. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.* Ph. D. Dissertation. Carnegie Mellon University. https://doi.org/10.5281/zenodo.6990769 Version 1.1, revised May 2022.

Jonathan Sterling and Carlo Angiuli. 2021. Normalization for Cubical Type Theory. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–15. https://doi.org/10.1109/LICS52264.2021.9470719

Jonathan Sterling and Robert Harper. 2021. Logical Relations as Types: Proof-Relevant Parametricity for Program Modules. *J. ACM* 68, 6 (Dec. 2021), 1–47. https://doi.org/10.1145/3474834

Jonathan Sterling and Robert Harper. 2022. Sheaf Semantics of Termination-Insensitive Noninterference. In *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 228)*, Amy P. Felty (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 5:1–5:19. https://doi.org/10.4230/LIPIcs.FSCD.2022.5

Jonathan Sterling and Bas Spitters. 2018. Normalization by gluing for free λ-theories. https://arxiv.org/abs/1809.08646

W. W. Tait. 1967. Intensional Interpretations of Functionals of Finite Type I. 32, 2 (1967), 198–212. http://www.jstor.org/stable/2271658

The Agda Team. 2026. *Flat Modality.* Agda. https://agda.readthedocs.io/en/latest/language/flat.html Agda 2.9.0 documentation.

Constantine Theocharis and Edwin Brady. 2026. Type Theory With Erasure. https://doi.org/10.48550/arXiv.2605.00655 arXiv:2605.00655 [cs.PL] Accepted to FSCD 2026.

Amin Timany, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. 2024. A Logical Approach to Type Soundness. *J. ACM* 71, 6, Article 40 (Nov. 2024), 75 pages. https://doi.org/10.1145/3676954

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

Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types. *Proceedings of the ACM on Programming Languages* 3, ICFP (July 2019), 87:1–87:29. https://doi.org/10.1145/3341691

Matthew Z. Weaver and Daniel R. Licata. 2020. A Constructive Model of Directed Univalence in Bicubical Sets. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science* (Saarbrücken, Germany) *(LICS '20)*. Association for Computing Machinery, New York, NY, USA, 915–928. https://doi.org/10.1145/3373718.3394794

Jonathan Weinberger. 2022. *A Synthetic Perspective on* $(\infty, 1)$*-Category Theory: Fibrational and Semantic Aspects*. Dissertation. Technische Universität Darmstadt, Darmstadt. https://doi.org/10.26083/tuprints-00020716

Jonathan Weinberger. 2024. Two-sided cartesian fibrations of synthetic $(\infty, 1)$-categories. *Journal of Homotopy and Related Structures* 19, 2 (June 2024). https://doi.org/10.1007/s40062-024-00348-3

Jonathan Weinberger, Benedikt Ahrens, Ulrik Buchholtz, and Paige North. 2022. Synthetic Tait Computability for Simplicial Type Theory. Extended abstract at the 28th International Conference on Types for Proofs and Programs (TYPES 2022). https://types22.inria.fr/files/2022/06/TYPES_2022_paper_17.pdf

Szumi Xie and Viktor Bense. 2026. Formalizing type theory through transport hell. Contributed talk at SSTT 2026, MFPS XLII & SSTT 2026. https://ul-fmf.github.io/mfps-sstt-2026/sstt-contributed-talks/

Zhixuan Yang. 2024. *Structure and Language of Higher-Order Algebraic Effects*. Ph. D. Dissertation. Imperial College London. https://yangzhixuan.github.io/pdf/yang-thesis.pdf

Zhixuan Yang and Nicolas Wu. 2026. Handling Higher-Order Effectful Operations with Judgemental Monadic Laws. *Proc. ACM Program. Lang.* 10, POPL, Article 36 (Jan. 2026), 30 pages. https://doi.org/10.1145/3776678

A THE ORTHOGONALITY CONSTRUCTION

This appendix records the orthogonality construction used in Section 2. It is included only for completeness. We do not claim originality for this construction: we follow the idea of Grodin et al. [2024] in constructing types as synthetic preorders. The general background is the standard theory of orthogonal reflective subuniverses and localizations in homotopy type theory [Christensen et al. 2020; Fiore 1997; Rijke et al. 2020]. This construction is used in our Cubical Agda mechanization, making use of the localization modality as a higher inductive type in the Cubical library.

A.1 Orthogonality

Definition A.1 (Orthogonality). Let $f : A \to B$ be a map of types. A type X is *orthogonal* to f, or *f-local*, when precomposition with f is an equivalence:

$$(- \circ f) : (B \to X) \to (A \to X).$$

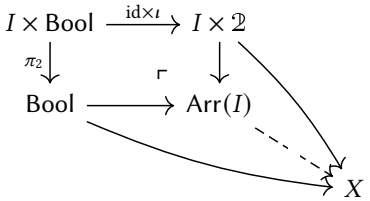
For a family of maps $\mathcal{F} = (f_i : A_i \to B_i)_{i:I}$, a type X is $\mathcal{F}$-local when it is local for each f_i. The point of the construction is that many familiar homotopical or directed properties can be expressed by choosing a suitable map f_i and asking for orthogonality to it.

A.2 The Three Localization Conditions

We use three localization conditions to ensure that a type is a set, thin, and Segal.

A.2.1 Setness. A type X is a set precisely when all of its loop spaces are propositions. In homotopy type theory this can be expressed by nullity with respect to the circle: X is set exactly when it is local for the terminal map $S^1 \longrightarrow \mathbb{1}$. Indeed, precomposition says that a point of X contains the same information as a map $S^1 \to X$, and maps out of the circle classify a point together with a loop. Thus localization forces all loops, and hence all paths, to be unique.

A.2.2 Thinness. Recall from Definition 2.1 that $x \leq_X y$ is the type of directed paths from x to y. Let $\iota : \mathsf{Bool} \to \mathbb{2}$ send the two Booleans to the two endpoints of the directed interval. For a type I, define $\mathsf{Arr}(I)$ by the following pushout square, whose dashed arrow displays the universal property:



Thus $\mathsf{Arr}(I)$ is obtained by gluing together I copies of the directed interval along their common source and target. The universal property says that maps out of $\mathsf{Arr}(I)$ are characterized by

$$(\mathsf{Arr}(I) \to X) \simeq \sum_{x_0, x_1 : X} (I \to x_0 \leq_X x_1).$$

Let $! : \mathsf{Bool} \to \mathbb{1}$ be the terminal map, and $\mathsf{Arr}(!) : \mathsf{Arr}(\mathsf{Bool}) \to \mathsf{Arr}(\mathbb{1})$ be the induced map. A type X is local for $\mathsf{Arr}(!)$ exactly when, for each $x_0, x_1 : X$, the diagonal map

$$(x_0 \leq_X x_1) \to (\mathsf{Bool} \to x_0 \leq_X x_1)$$

is an equivalence. This is precisely the statement that every hom type $x_0 \leq_X x_1$ is a proposition.

A.2.3 Segalness. Let Δ^2 be the directed 2-simplex and Λ^2_1 its inner spine:

$$\Delta^2 \coloneqq \{(s,t) : \mathbb{2} \times \mathbb{2} \mid t \leq_{\mathbb{2}} s\}, \qquad \Lambda^2_1 \coloneqq \{(s,t) : \mathbb{2} \times \mathbb{2} \mid s = \mathsf{i1} \vee t = \mathsf{i0}\}.$$

Write $\iota : \Lambda^2_1 \hookrightarrow \Delta^2$ for the inclusion. By Riehl and Shulman [2017, Theorem 5.5], a type X is Segal if and only if restriction along ι,

$$(- \circ \iota) : (\Delta^2 \to X) \to (\Lambda^2_1 \to X),$$

is an equivalence.

A.2.4 Putting It All Together. Putting the three maps together, define

$$\begin{array}{rcll} f_{\mathsf{set}} & \coloneqq & ! & : S^1 \to \mathbb{1}, \\ f_{\mathsf{thin}} & \coloneqq & \mathsf{Arr}(!) & : \mathsf{Arr}(\mathsf{Bool}) \to \mathsf{Arr}(\mathbb{1}), \qquad \mathcal{F} \coloneqq \{f_{\mathsf{set}}, f_{\mathsf{thin}}, f_{\mathsf{Segal}}\}. \\ f_{\mathsf{Segal}} & \coloneqq & \iota & : \Lambda^2_1 \to \Delta^2, \end{array}$$

An $\mathcal{F}$-local type is therefore simultaneously a set, thin, and Segal.

A.3 Localization

For any type A, let

$$\eta_A : A \to \mathsf{L}A$$

denote the localization of A at the family $\mathcal{F}$. This may be constructed as the standard higher inductive localization of a type [Christensen et al. 2020; Rijke et al. 2020]: the type $\mathsf{L}A$ is $\mathcal{F}$-local, and for every $\mathcal{F}$-local type X, precomposition with η_A is an equivalence

$$(\mathsf{L}A \to X) \simeq (A \to X).$$

Thus L is the reflector into the reflective subuniverse of $\mathcal{F}$-local types, with unit η. In particular, every type of the form $\mathsf{L}A$ is a set, thin, and Segal.

The universal property is the main feature of the localization used in this construction. To define a function out of $\mathsf{L}A$ into a local type X, it suffices to define a function $A \to X$.

A.4 Applying the Reflector to Syntax

Let $\mathsf{Sub}_0\ \Gamma\ \Delta$ and $\mathsf{Tm}_0\ \Gamma\ A$ denote the raw directed syntax inductively generated by the point, path, and directed constructors as in Section 2. The localized syntax used is obtained by reflecting the substitution and term carriers:

$$\mathsf{Sub}\ \Gamma\ \Delta \coloneqq \mathsf{L}(\mathsf{Sub}_0\ \Gamma\ \Delta), \qquad \mathsf{Tm}\ \Gamma\ A \coloneqq \mathsf{L}(\mathsf{Tm}_0\ \Gamma\ A).$$

We write the corresponding units as

$$\eta_{\mathsf{Sub}} : \mathsf{Sub}_0\ \Gamma\ \Delta \to \mathsf{Sub}\ \Gamma\ \Delta, \qquad \eta_{\mathsf{Tm}} : \mathsf{Tm}_0\ \Gamma\ A \to \mathsf{Tm}\ \Gamma\ A.$$

Because L lands in the $\mathcal{F}$-local subuniverse, the localized substitution and term types are automatically sets, thin types, and Segal types. Thus their directed paths compose, and parallel reductions between fixed endpoints are proof-irrelevant, without adding separate truncation constructors to the raw inductive syntax. This operation amounts to the free category on a graph.

B REFERENCE DEFINITIONS

This appendix is a reference sheet for the final glued structures and type-former clauses. The unary clauses use the flat-coded predicates of Section 4; they should be read as the flat refinement of the construction first presented in Section 3.

We do not repeat the β- and η-laws for the type formers. In each case, the proof has the same shape: contravariant transport of the semantic witness along the relevant syntactic reduction is compared with the original witness, and the universal property of contravariance (Lemma 3.4) turns this comparison into the same reflexivity argument used in Section 3. We write κ for the inverse of the counit ε on flat modal types, following Definition 4.5.

B.1 Unary Model

B.1.1 Predicate Codes. For a closed syntactic type $A^\circ : \mathrm{Ty}\ \cdot$, the predicate codes are:

$$\mathrm{Pred}(A^\circ) := \mathrm{let}^\flat\ T = \kappa\ A^\circ\ \mathrm{in}\ \flat(\mathrm{Tm}\ \cdot\ T \to \mathcal{U}).$$

B.1.2 Judgmental Structure. The final unary judgmental structures are:

record CTX : $\mathcal{U}$ **where**
$\Gamma^\circ : \mathrm{Ctx}$
$\Gamma^\bullet_\flat : \flat(\mathrm{Sub}\ \cdot\ \Gamma^\circ \to \mathcal{U})$
$\Gamma^\bullet := \varepsilon(\Gamma^\bullet_\flat)$

record SUB $(\Gamma\ \Delta : \mathrm{CTX}) : \mathcal{U}$ **where**
$\sigma^\circ : \mathrm{Sub}\ \Gamma^\circ\ \Delta^\circ$
$\sigma^\bullet : (\gamma^\circ : \mathrm{Sub}\ \cdot\ \Gamma^\circ) \to \Gamma^\bullet\ \gamma^\circ \to \Delta^\bullet\ (\sigma^\circ \circ \gamma^\circ)$

record TY $(\Gamma : \mathrm{CTX}) : \mathcal{U}$ **where**
$A^\circ : \mathrm{Ty}\ \Gamma^\circ$
$A^\bullet_\flat : (\gamma^\circ : \mathrm{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ) \to \mathrm{Pred}(A^\circ[\gamma^\circ])$
$A^\bullet\ \gamma^\circ\ \gamma^\bullet := \varepsilon(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet)$
$c_A : (\gamma^\circ : \mathrm{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ) \to \mathrm{isContrav}\ (A^\bullet\ \gamma^\circ\ \gamma^\bullet)$

record TM $(\Gamma : \mathrm{CTX})(A : \mathrm{TY}\ \Gamma) : \mathcal{U}$ **where**
$M^\circ : \mathrm{Tm}\ \Gamma^\circ\ A^\circ$
$M^\bullet : (\gamma^\circ : \mathrm{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma^\circ) \to A^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])$

B.1.3 Substitutions and Context Extension. The substitution clauses are:

$$\begin{aligned}
(\mathrm{ID}_\Gamma)^\circ &:= \mathrm{id}\\
(\mathrm{ID}_\Gamma)^\bullet\ \gamma^\circ\ \gamma^\bullet &:= \gamma^\bullet\\
(\tau\circ\sigma)^\circ &:= \tau^\circ \circ \sigma^\circ\\
(\tau\circ\sigma)^\bullet\ \gamma^\circ\ \gamma^\bullet &:= \tau^\bullet\ (\sigma^\circ \circ \gamma^\circ)\ (\sigma^\bullet\ \gamma^\circ\ \gamma^\bullet)\\
(A[\sigma])^\circ &:= A^\circ[\sigma^\circ]\\
(A[\sigma])^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet &:= A^\bullet_\flat\ (\sigma^\circ \circ \gamma^\circ)\ (\sigma^\bullet\ \gamma^\circ\ \gamma^\bullet)\\
c_{A[\sigma]}\ \gamma^\circ\ \gamma^\bullet\ f\ v &:= c_A\ (\sigma^\circ \circ \gamma^\circ)\ (\sigma^\bullet\ \gamma^\circ\ \gamma^\bullet)\ f\ v\\
(M[\sigma])^\circ &:= M^\circ[\sigma^\circ]\\
(M[\sigma])^\bullet\ \gamma^\circ\ \gamma^\bullet &:= M^\bullet\ (\sigma^\circ \circ \gamma^\circ)\ (\sigma^\bullet\ \gamma^\circ\ \gamma^\bullet)
\end{aligned}$$

$$\cdot^\circ := \cdot$$
$$\cdot^\bullet_\flat := \left(\lambda \gamma^\circ.\ 1\right)^\flat$$
$$(\Gamma \triangleright A)^\circ := \Gamma^\circ \triangleright A^\circ$$
$$(\Gamma \triangleright A)^\bullet_\flat := \mathsf{let}^\flat\ \Gamma^\bullet = \Gamma^\bullet_\flat\ \mathsf{in}\ \mathsf{let}^\flat\ \mathcal{A} = \kappa\left(\lambda \gamma^\circ\ \gamma^\bullet.\ A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\right)\ \mathsf{in}\ \left(\lambda \delta^\circ. \sum_{\delta^\bullet : \Gamma^\bullet\ (\mathsf{p} \circ \delta^\circ)} \mathsf{let}^\flat\ A^\bullet = \mathcal{A}\ (\mathsf{p} \circ \delta^\circ)\ \delta^\bullet\ \mathsf{in}\ A^\bullet\ (\mathsf{q}[\delta^\circ])\right)^\flat$$
$$(\sigma, M)^\circ := (\sigma^\circ, M^\circ)$$
$$(\sigma, M)^\bullet\ \gamma^\circ\ \gamma^\bullet := \left(\sigma^\bullet\ \gamma^\circ\ \gamma^\bullet,\ M^\bullet\ \gamma^\circ\ \gamma^\bullet\right)$$
$$(\mathsf{p})^\circ := \mathsf{p}$$
$$(\mathsf{p})^\bullet\ \delta^\circ\ (\delta^\bullet, a^\bullet) := \delta^\bullet$$
$$(\mathsf{q})^\circ := \mathsf{q}$$
$$(\mathsf{q})^\bullet\ \delta^\circ\ (\delta^\bullet, a^\bullet) := a^\bullet$$

Taking the counit of the context-extension predicate recovers the ordinary semantic predicate:

$$\varepsilon((\Gamma \triangleright A)^\bullet_\flat)\ \delta^\circ = \sum_{\delta^\bullet : \varepsilon(\Gamma^\bullet_\flat)\ (\mathsf{p} \circ \delta^\circ)} \varepsilon\left(A^\bullet_\flat\ (\mathsf{p} \circ \delta^\circ)\ \delta^\bullet\right)\ (\mathsf{q}[\delta^\circ]).$$

B.1.4 Non-Dependent Products. The non-dependent product type clauses are:

$$\mathsf{PROD} : \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma$$
$$(\mathsf{PROD}\ A\ B)^\circ := A^\circ \times B^\circ$$
$$(\mathsf{PROD}\ A\ B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet := \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ B^\bullet = B^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \left(\lambda P.\ A^\bullet\ (\mathsf{fst}\ P) \times B^\bullet\ (\mathsf{snd}\ P)\right)^\flat$$
$$c_{\mathsf{PROD}\ A\ B}\ \gamma^\circ\ \gamma^\bullet\ (f : P \le P')\ (\Phi', \Psi') := \left((\mathsf{fst}\ f)^*\ \Phi',\ (\mathsf{snd}\ f)^*\ \Psi'\right)$$

The uniqueness proof $c_{\mathsf{PROD}\ A\ B}$ is pointwise by c_A and c_B.

$$\mathsf{PAIR} : \mathsf{TM}\ \Gamma\ A \to \mathsf{TM}\ \Gamma\ B \to \mathsf{TM}\ \Gamma\ (\mathsf{PROD}\ A\ B)$$
$$(\mathsf{PAIR}\ M\ N)^\circ := \mathsf{pair}\ M^\circ\ N^\circ$$
$$(\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet := \left((\times_{\beta_1}[\gamma^\circ])^*\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet),\ (\times_{\beta_2}[\gamma^\circ])^*\ (N^\bullet\ \gamma^\circ\ \gamma^\bullet)\right)$$
$$(\mathsf{FST}\ P)^\circ := \mathsf{fst}\ P^\circ$$
$$(\mathsf{FST}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet)$$
$$(\mathsf{SND}\ P)^\circ := \mathsf{snd}\ P^\circ$$
$$(\mathsf{SND}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet)$$

B.1.5 Non-Dependent Functions. The non-dependent function type clauses are:

$$- \Rightarrow - : \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma \to \mathsf{TY}\ \Gamma$$
$$(A \Rightarrow B)^\circ := A^\circ \Rightarrow B^\circ$$
$$(A \Rightarrow B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet := \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ B^\bullet = B^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \left(\lambda F.\ (M^\circ : \mathsf{Tm}\ \cdot\ (A^\circ[\gamma^\circ])) \to A^\bullet\ M^\circ \to B^\bullet\ (\mathsf{app}\ F\ M^\circ)\right)^\flat$$
$$c_{A \Rightarrow B}\ \gamma^\circ\ \gamma^\bullet\ (f : F \le F')\ \Phi' := \lambda M^\circ\ M^\bullet.\ (\mathsf{app}\ f\ M^\circ)^*(\Phi'\ M^\circ\ M^\bullet)$$

The uniqueness proof $c_{A \Rightarrow B}$ is by c_B.

$$\mathsf{LAM} : \mathsf{TM}\ (\Gamma \triangleright A)\ (B[\mathsf{p}]) \to \mathsf{TM}\ \Gamma\ (A \Rightarrow B)$$
$$(\mathsf{LAM}\ N)^\circ := \mathsf{lam}\ N^\circ$$
$$(\mathsf{LAM}\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet\ M^\circ\ M^\bullet := (\Rightarrow_\beta[(\gamma^\circ,M^\circ)])^*\big(N^\bullet\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet)\big)$$

$$\mathsf{APP} : \mathsf{TM}\ \Gamma\ (A \Rightarrow B) \to \mathsf{TM}\ \Gamma\ A \to \mathsf{TM}\ \Gamma\ B$$
$$(\mathsf{APP}\ F\ M)^\circ := \mathsf{app}\ F^\circ\ M^\circ$$
$$(\mathsf{APP}\ F\ M)^\bullet\ \gamma^\circ\ \gamma^\bullet := F^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet)$$

B.1.6 Dependent Pairs. The dependent pair type clauses are:

$$\Sigma : (A : \mathsf{TY}\ \Gamma) \to \mathsf{TY}\ (\Gamma \triangleright A) \to \mathsf{TY}\ \Gamma$$
$$(\Sigma\ A\ B)^\circ := \Sigma\ A^\circ\ B^\circ$$
$$(\Sigma\ A\ B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet := \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ \mathcal{B} = \kappa\left(\lambda M^\circ\ M^\bullet.\ B^\bullet_\flat\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet)\right)\ \mathsf{in}$$
$$\left(\lambda P.\ \sum_{\Phi : A^\bullet\ (\mathsf{fst}\ P)} \mathsf{let}^\flat\ B^\bullet = \mathcal{B}\ (\mathsf{fst}\ P)\ \Phi\ \mathsf{in}\ B^\bullet\ (\mathsf{snd}\ P)\right)^\flat$$

The contravariance proof $c_{\Sigma\ A\ B}$ is pointwise by c_A and c_B.

$$\mathsf{PAIR} : (M : \mathsf{TM}\ \Gamma\ A) \to \mathsf{TM}\ \Gamma\ (B[(\mathsf{ID},M)]) \to \mathsf{TM}\ \Gamma\ (\Sigma\ A\ B)$$
$$(\mathsf{PAIR}\ M\ N)^\circ := \mathsf{pair}\ M^\circ\ N^\circ$$
$$(\mathsf{PAIR}\ M\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet := \begin{pmatrix}(\Sigma_{\beta_1}[\gamma^\circ])^*(M^\bullet\ \gamma^\circ\ \gamma^\bullet),\\ (\Sigma_{\beta_2}[\gamma^\circ])^*\big(\mathsf{transport}_{\mathsf{substInv}}(N^\bullet\ \gamma^\circ\ \gamma^\bullet)\big)\end{pmatrix}$$
$$\mathsf{FST} : \mathsf{TM}\ \Gamma\ (\Sigma\ A\ B) \to \mathsf{TM}\ \Gamma\ A$$
$$(\mathsf{FST}\ P)^\circ := \mathsf{fst}\ P^\circ$$
$$(\mathsf{FST}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_1(P^\bullet\ \gamma^\circ\ \gamma^\bullet)$$

$$\mathsf{SND} : (P : \mathsf{TM}\ \Gamma\ (\Sigma\ A\ B)) \to \mathsf{TM}\ \Gamma\ (B[(\mathsf{ID},\mathsf{FST}\ P)])$$
$$(\mathsf{SND}\ P)^\circ := \mathsf{snd}\ P^\circ$$
$$(\mathsf{SND}\ P)^\bullet\ \gamma^\circ\ \gamma^\bullet := \pi_2(P^\bullet\ \gamma^\circ\ \gamma^\bullet)$$

B.1.7 Dependent Functions. The dependent function type clauses are:

$$\Pi : (A : \mathsf{TY}\ \Gamma) \to \mathsf{TY}\ (\Gamma \triangleright A) \to \mathsf{TY}\ \Gamma$$
$$(\Pi\ A\ B)^\circ := \Pi\ A^\circ\ B^\circ$$
$$(\Pi\ A\ B)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet := \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ \mathcal{B} = \kappa\left(\lambda M^\circ\ M^\bullet.\ B^\bullet_\flat\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet)\right)\ \mathsf{in}$$
$$\left(\lambda F.\ (M^\circ : \mathsf{Tm}\ \cdot\ (A^\circ[\gamma^\circ])) \to \right.$$
$$\left.\quad (M^\bullet : A^\bullet\ M^\circ) \to \mathsf{let}^\flat\ B^\bullet = \mathcal{B}\ M^\circ\ M^\bullet\ \mathsf{in}\ B^\bullet\ (\mathsf{app}\ F\ M^\circ)\right)^\flat$$

The contravariance proof $c_{\Pi\ A\ B}$ is by c_B on the induced application reduction in the extended context.

$$\mathsf{LAM} : \mathsf{TM}\ (\Gamma \triangleright A)\ B \to \mathsf{TM}\ \Gamma\ (\Pi\ A\ B)$$
$$(\mathsf{LAM}\ N)^\circ := \mathsf{lam}\ N^\circ$$
$$(\mathsf{LAM}\ N)^\bullet\ \gamma^\circ\ \gamma^\bullet\ M^\circ\ M^\bullet := (\Pi_\beta[(\gamma^\circ,M^\circ)])^*\big(N^\bullet\ (\gamma^\circ,M^\circ)\ (\gamma^\bullet,M^\bullet)\big)$$

$$\mathsf{APP} : (F : \mathsf{TM}\ \Gamma\ (\Pi\ A\ B)) \to (M : \mathsf{TM}\ \Gamma\ A) \to \mathsf{TM}\ \Gamma\ (B[(\mathsf{ID},M)])$$
$$(\mathsf{APP}\ F\ M)^\circ := \mathsf{app}\ F^\circ\ M^\circ$$
$$(\mathsf{APP}\ F\ M)^\bullet\ \gamma^\circ\ \gamma^\bullet := F^\bullet\ \gamma^\circ\ \gamma^\bullet\ (M^\circ[\gamma^\circ])\ (M^\bullet\ \gamma^\circ\ \gamma^\bullet)$$

B.1.8 Booleans. The Boolean type clauses are:

$$\begin{aligned}
&\mathsf{BOOL} : \mathsf{TY}\ \Gamma\\
&\mathsf{BOOL}^\circ \coloneqq \mathsf{Bool}\\
&\mathsf{BOOL}^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet \coloneqq \left(\lambda M.\ \sum_{b:\{0,1\}} M \leq_{\mathsf{Tm}\ \cdot\ \mathsf{Bool}} \ulcorner b \urcorner\right)^\flat\\
&\mathsf{BOOL}^\bullet\ \gamma^\circ\ \gamma^\bullet\ M \coloneqq \sum_{b:\{0,1\}} M \leq_{\mathsf{Tm}\ \cdot\ \mathsf{Bool}} \ulcorner b \urcorner\\
&c_{\mathsf{BOOL}}\ \gamma^\circ\ \gamma^\bullet\ (f : M \leq M')\ (b, r) \coloneqq (b,\ f \cdot r)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{TRUE}^\circ \coloneqq \mathsf{true}\\
&\mathsf{TRUE}^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq (0, \mathsf{id}_{\mathsf{true}})\\
&\mathsf{FALSE}^\circ \coloneqq \mathsf{false}\\
&\mathsf{FALSE}^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq (1, \mathsf{id}_{\mathsf{false}})
\end{aligned}$$

The lift witness and uniqueness part of c_{BOOL} are supplied componentwise by Lemma 3.6. For the dependent Boolean eliminator, if $T^\bullet\ \gamma^\circ\ \gamma^\bullet = (b, r)$, the eliminator is:

$$\begin{aligned}
&C_{\mathsf{true}} \coloneqq C[(\mathsf{ID},\mathsf{TRUE})]\\
&C_{\mathsf{false}} \coloneqq C[(\mathsf{ID},\mathsf{FALSE})]\\
&C_T \coloneqq C[(\mathsf{ID},T)]
\end{aligned}$$

$$\begin{aligned}
&\mathsf{IF} : (C : \mathsf{TY}\ (\Gamma \triangleright \mathsf{BOOL})) \to \mathsf{TM}\ \Gamma\ C_{\mathsf{true}} \to \mathsf{TM}\ \Gamma\ C_{\mathsf{false}} \to (T : \mathsf{TM}\ \Gamma\ \mathsf{BOOL}) \to \mathsf{TM}\ \Gamma\ C_T\\
&(\mathsf{IF}\ C\ U\ V\ T)^\circ \coloneqq \mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ T^\circ
\end{aligned}$$

$$(\mathsf{IF}\ C\ U\ V\ T)^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \begin{cases} \left((\mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ r)[\gamma^\circ] \cdot \mathsf{Bool}_{\mathsf{true}}[\gamma^\circ]\right)^*(U^\bullet\ \gamma^\circ\ \gamma^\bullet) & \text{if } b = 0,\\ \left((\mathsf{if}\ C^\circ\ U^\circ\ V^\circ\ r)[\gamma^\circ] \cdot \mathsf{Bool}_{\mathsf{false}}[\gamma^\circ]\right)^*(V^\bullet\ \gamma^\circ\ \gamma^\bullet) & \text{if } b = 1. \end{cases}$$

As in the Σ case, this notation suppresses the preliminary $\mathsf{transport}_{\mathsf{substInv}}$ that moves the branch witness across the dependent fibre of C; the treatment is the same as in Σ.

B.1.9 Universe. The universe type uses candidates, *i.e.* predicate equipped with contravariance:

$$\mathsf{Cand}(A^\circ) \coloneqq \sum_{A^\bullet_\flat : \mathsf{Pred}(A^\circ)} \mathsf{isContrav}\ (\varepsilon(A^\bullet_\flat)).$$

The universe clauses are:

$$\begin{aligned}
&\mathsf{UNIV}^\circ \coloneqq \mathsf{U},\\
&\mathsf{UNIV}^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet \coloneqq \left(\lambda M.\ \mathsf{Cand}(\mathsf{El}\ M)\right)^\flat\\
&(\mathsf{EL}\ M)^\circ \coloneqq \mathsf{El}\ M^\circ,\\
&(\mathsf{EL}\ M)^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet \coloneqq A^\bullet_{\flat,\gamma,\gamma^\bullet},\\
&c_{\mathsf{EL}\ M}\ \gamma^\circ\ \gamma^\bullet \coloneqq c_{A,\gamma,\gamma^\bullet}\\
&(\mathsf{CODE}\ A)^\circ \coloneqq \mathsf{Code}\ A^\circ,\\
&(\mathsf{CODE}\ A)^\bullet\ \gamma^\circ\ \gamma^\bullet \coloneqq \left(\mathsf{U}_\beta[\gamma^\circ]\right)^*\left(A^\bullet_\flat\ \gamma^\circ\ \gamma^\bullet,\ c_A\ \gamma^\circ\ \gamma^\bullet\right)
\end{aligned}$$

The contravariance proof c_{UNIV} is the candidate-lifting construction along universe reductions explained in Section 4. In the EL clause, the notation means that $M^\bullet\ \gamma^\circ\ \gamma^\bullet = (A^\bullet_{\flat,\gamma,\gamma^\bullet}, c_{A,\gamma,\gamma^\bullet})$.

B.2 Binary Model

This part records the binary structures used in Section 5.

B.2.1 Binary Predicate Codes. For closed syntactic types $A_L^\circ, A_R^\circ : \mathsf{Ty}\ \cdot$, the binary predicates are:

$$\mathsf{Pred}_2(A_L^\circ, A_R^\circ) \coloneqq \mathsf{let}^\flat\ T_L = \kappa\ A_L^\circ\ \mathsf{in}\ \mathsf{let}^\flat\ T_R = \kappa\ A_R^\circ\ \mathsf{in}\ \flat(\mathsf{Tm}\ \cdot\ T_L \to \mathsf{Tm}\ \cdot\ T_R \to \mathcal{U}).$$

B.2.2 Binary Judgmental Records. The final binary judgmental records are:

record $\mathsf{CTX}_2 : \mathcal{U}$ **where**
$\Gamma^\circ : \mathsf{Ctx}$
$\Gamma^\bullet_\flat : \flat(\mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathsf{Sub}\ \cdot\ \Gamma^\circ \to \mathcal{U})$
$\Gamma^\bullet \coloneqq \varepsilon(\Gamma^\bullet_\flat)$

record $\mathsf{TY}_2\ (\Gamma : \mathsf{CTX}_2) : \mathcal{U}$ **where**
$A^\circ : \mathsf{Ty}\ \Gamma^\circ$
$A^\bullet_\flat : (\gamma_L^\circ\ \gamma_R^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to \Gamma^\bullet\ \gamma_L^\circ\ \gamma_R^\circ \to \mathsf{Pred}_2(A^\circ[\gamma_L^\circ], A^\circ[\gamma_R^\circ])$
$A^\bullet\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet \coloneqq \varepsilon(A^\bullet_\flat\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet)$
$c_A : \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet \to \mathsf{isContrav}\ (A^\bullet\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet)$

record $\mathsf{TM}_2\ (\Gamma : \mathsf{CTX}_2)(A : \mathsf{TY}_2\ \Gamma) : \mathcal{U}$ **where**
$M^\circ : \mathsf{Tm}\ \Gamma^\circ\ A^\circ$
$M^\bullet : (\gamma_L^\circ\ \gamma_R^\circ : \mathsf{Sub}\ \cdot\ \Gamma^\circ) \to (\gamma^\bullet : \Gamma^\bullet\ \gamma_L^\circ\ \gamma_R^\circ) \to A^\bullet\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet\ (M^\circ[\gamma_L^\circ])\ (M^\circ[\gamma_R^\circ])$

B.2.3 Binary Type-Former Examples. The binary Boolean predicate records a common Boolean value:

$\mathsf{BOOL}_2 : \mathsf{TY}_2\ \Gamma$
$\mathsf{BOOL}_2{}^\circ \coloneqq \mathsf{Bool}$
$$\mathsf{BOOL}_{2\flat}^\bullet\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet \coloneqq \left(\lambda M_L\ M_R.\ \sum_{b:\{0,1\}} (M_L \leq_{\mathsf{Tm}\ \cdot\ \mathsf{Bool}} \ulcorner b \urcorner) \times (M_R \leq_{\mathsf{Tm}\ \cdot\ \mathsf{Bool}} \ulcorner b \urcorner)\right)^\flat$$
$$c_{\mathsf{BOOL}_2}\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet\ (f_L : M_L \leq M_L')\ (f_R : M_R \leq M_R')\ (b, (r_L, r_R)) \coloneqq (b, (f_L \cdot r_L, f_R \cdot r_R))$$

The binary product predicate is pointwise on projections:

$\mathsf{PROD}_2 : \mathsf{TY}_2\ \Gamma \to \mathsf{TY}_2\ \Gamma \to \mathsf{TY}_2\ \Gamma$
$(\mathsf{PROD}_2\ A\ B)^\circ \coloneqq A^\circ \times B^\circ$
$(\mathsf{PROD}_2\ A\ B)^\bullet_\flat\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet \coloneqq \mathsf{let}^\flat\ A^\bullet = A^\bullet_\flat\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet\ \mathsf{in}\ \mathsf{let}^\flat\ B^\bullet = B^\bullet_\flat\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet\ \mathsf{in}$
$(\lambda P_L\ P_R.\ A^\bullet\ (\mathsf{fst}\ P_L)\ (\mathsf{fst}\ P_R) \times B^\bullet\ (\mathsf{snd}\ P_L)\ (\mathsf{snd}\ P_R))^\flat$

Everything else follows the same simple pattern to turn unary semantics to binary semantics.

B.2.4 Binary Universe. The binary universe uses binary candidates:

$$\mathsf{Cand}_2(A_L^\circ, A_R^\circ) \coloneqq \sum_{R_\flat : \mathsf{Pred}_2(A_L^\circ, A_R^\circ)} \mathsf{isContrav}\ (\varepsilon(R_\flat)).$$

The binary universe is:

$\mathsf{UNIV}_2{}^\circ \coloneqq \mathsf{U}$
$\mathsf{UNIV}_{2\flat}^\bullet\ \gamma_L^\circ\ \gamma_R^\circ\ \gamma^\bullet \coloneqq (\lambda M_L\ M_R.\ \mathsf{Cand}_2(\mathsf{El}\ M_L, \mathsf{El}\ M_R))^\flat$