Cubical Syntax for Reflection-Free Extensional Equality

Jonathan Sterling
Carnegie Mellon University
http://cs.cmu.edu/~jmsterli
jmsterli@cs.cmu.edu

Carlo Angiuli
Carnegie Mellon University
http://cs.cmu.edu/~cangiuli
cangiuli@cs.cmu.edu

Daniel Gratzer
Aarhus University
http://jozefg.github.io
gratzer@cs.au.dk

Abstract

We contribute XTT, a cubical reconstruction of Observational Type Theory [7] which extends Martin-Löf's intensional type theory with a *dependent equality type* that enjoys function extensionality and a judgmental version of the *unicity of identity proofs* principle (UIP): any two elements of the same equality type are judgmentally equal. Moreover, we conjecture that the typing relation can be decided in a practical way. In this paper, we establish an algebraic canonicity theorem using a novel extension of the *logical families* or *categorical gluing* argument inspired by Coquand and Shulman [27, 48]: every closed element of boolean type is derivably equal to either true or false.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Algebraic semantics; Theory of computation → Denotational semantics

Keywords and phrases Dependent type theory, extensional equality, cubical type theory, categorical gluing, canonicity

Funding The authors gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. 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.

Acknowledgements We thank Lars Birkedal, Evan Cavallo, David Thrane Christiansen, Thierry Coquand, Kuen-Bang Hou (Favonia), Marcelo Fiore, Jonas Frey, Krzysztof Kapulkin, András Kovács, Dan Licata, Conor McBride, Darin Morrison, Anders Mörtberg, Michael Shulman, Bas Spitters, and Thomas Streicher for helpful conversations about extensional equality, algebraic type theory, and categorical gluing. We thank our anonymous reviewers for their insightful comments, and especially thank Robert Harper for valuable conversations throughout the development of this work. We also thank Paul Taylor for his `diagrams` package, which we have used to typeset the commutative diagrams in this paper.

1 Introduction

The past fifty years of constructive type theory can be summed up as the search for a scientific understanding of *equality*, punctuated by moments of qualitative change in our perception of the boundary between semantics (actual construction) and syntax (proof theory) from a type-theoretic point of view. Computation is critical to both the semantics and syntax of type theory—from Martin-Löf's meaning explanations [43], supplying type theory with its

direct semantics and intuitionistic grounding, to syntactic properties such as closed and open *canonicity* which establish computation as the indispensible method for deriving equations.

For too long, a limiting perspective on extensional type theory has prevailed, casting it as a particular syntactic artifact (for instance, the formalism obtained by stripping of their meaning the rules which incidentally appear in Martin-Löf's monograph [43]), a formal system which enjoys precious few desirable syntactic properties and is distinguished primarily by its *equality reflection* rule.

We insist on the contrary that the importance of extensional type theory lies not in the specific choice of syntactic presentation (historically, via equality reflection), but rather in the *semantic* characteristics of its equality connective, which are invariant under choice of syntax. The specifics of how such an equality construct is presented syntactically are entirely negotiable (the internal language of a doctrine is determined only up to equivalence), and therefore has an empirical component.

1.1 Internalizing equality: from judgments to types

Equality in type theory begins with a form of judgment $\Gamma \vdash A = B\ \mathit{type}$, which expresses that A and B are exactly the same type; because types can depend on terms, one also includes a form of judgment $\Gamma \vdash M = N : A$ to express that M and N are exactly the same element of A. This kind of equality, called *judgmental equality*, is silent in the sense that if $\Gamma \vdash A = B\ \mathit{type}$ holds and $\Gamma \vdash M : A$ holds, then $\Gamma \vdash M : B$ without further ado.

Judgmental equality in type theory is a completely top-level affair: it cannot be assumed or negated. On the other hand, both programming and mathematics require one to establish equations under the assumption of other equations (for instance, as part of an induction hypothesis). For this reason, it is necessary to *internalize* the judgmental equality $M = N : A$ as a type $\mathsf{Eq}_A(M, N)$ which can be assumed, negated, or inhabited by induction.

The simplest way to internalize judgmental equality as a type is to provide introduction and elimination rules which make the existence of a proof of $\mathsf{Eq}_A(M, N)$ equivalent to the judgment $M = N : A$:

$$\textsc{introduction}\ \frac{\Gamma \vdash M = N : A}{\Gamma \vdash \mathsf{refl} : \mathsf{Eq}_A(M, N)} \qquad \textsc{elimination}\ \frac{\Gamma \vdash P : \mathsf{Eq}_A(M, N)}{\Gamma \vdash M = N : A}$$

The ELIMINATION rule above is usually called *equality reflection*, and is characteristic of *extensional* versions of Martin-Löf's type theory. This presentation of the equality type is very strong, and broadens the reach of judgmental equality into assertions of higher-level.

A consequence of the equality reflection rule is that judgmental equality is no longer decidable, a pragmatic concern which affects implementation and usability. On the other hand, equality reflection implies numerous critical reasoning principles, including function extensionality (if two functions agree on all inputs, then they are equal), a judgmental version of the famous *unicity of identity proofs* (UIP) principle (any two elements of the equality type are equal), and perhaps the most crucial consequence of internalized equality, *coercion* (if $M : P(a_0)$ and $P : \mathsf{Eq}_A(a_0, a_1)$, then there is some term $P^*(M) : P(a_1)$; in this case, $P^*(M) = M$).

1.2 Extensional equality via equality reflection

The earliest type-theoretic proof assistants employed the equality reflection rule (or equivalent formulations) in order to internalize the judgmental equality, a method most famously

represented by Nuprl [25] and its descendents, including RedPRL [10]. The Nuprl-style formalisms act as a "window on the truth" for a *single* intended semantics inspired by Martin-Löf's computational meaning explanations [2]; semantic justification in the computational ontology is the *only* consideration when extending the Nuprl formalism with a new rule, in contrast to other traditions in which global properties (e.g. admissibility of structural rules, decidability of typing, interpretability in multiple models, etc.) are treated as definitive.

Rather than supporting *type checking*, proof assistants in this style rely heavily on interactive development of typing derivations using tactics and partial decision procedures. A notable aspect of the Nuprl family is that their formal sequents range not over typed terms (proofs), but over untyped raw terms (realizers); a consequence is that during the proof process, one must repeatedly establish numerous *type functionality* subgoals, which restore the information that is lost when passing from a proof to a realizer. To mitigate the corresponding blow-up in proof size, Nuprl relies heavily on untyped computational reasoning via pointwise functionality, a non-standard semantics for dependently typed sequents which has some surprising consequences, such as refuting the principle of *dependent cut* [38].

Another approach to implementing type theory with equality reflection is exemplified in the experimental Andromeda proof assistant [15], in which proofs are also built interactively using tactics, but judgments range over abstract proof derivations rather than realizers. This approach mitigates to some degree the practical problems caused by erasing information prematurely, and also enables interpretation into a broad class of semantic models.

Although Nuprl/RedPRL and Andromeda illustrate that techniques beyond mere type checking are profitable to explore, the authors' experiences building and using RedPRL for concrete formalization of mathematics underscored the benefits of having a practical algorithm to check types, particularly in the setting of cubical type theory (Section 1.6), whose higher-dimensional structure significantly reduces the applicability of Nuprl-style untyped reasoning.

In particular, whereas it is possible to treat *all* β-rules and many η-rules in non-cubical type theory as untyped rewrites, such an approach is unsound for the cubical account of higher inductive types and univalence [11]; consequently, in RedPRL many β/η rewrites must emit auxiliary proof obligations. Synthesizing these experiences and challenges led to the creation of the `redtt` proof assistant for Cartesian cubical type theory [9].

1.3 Equality in intensional type theory

Martin-Löf's Intensional Type Theory (ITT) [41, 46] represents another extremal point in the internalization of judgmental equality. ITT underapproximates the equality judgment via its *identity type*, characterized by rules like the following:

FORMATION

$$\frac{\Gamma \vdash A\ \mathit{type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{Id}_A(M, N)\ \mathit{type}}$$

INTRODUCTION

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{refl}_A(M) : \mathsf{Id}_A(M, M)}$$

ELIMINATION

$$\frac{\begin{array}{l}\Gamma, x : A, y : A, z : \mathsf{Id}_A(x, y) \vdash C(x, y, z)\ \mathsf{type} \\ \Gamma \vdash P : \mathsf{Id}_A(M, N) \qquad \Gamma, x : A \vdash Q : C(x, x, \mathsf{refl}_A(x))\end{array}}{\Gamma \vdash \mathsf{J}_{x,y,z.C}(P; x.Q) : C(M, N, P)} \quad \ldots$$

Symmetry, transitivity and coercion follow from the elimination rule of the identity type. Other properties which follow directly from equality reflection, such as the unicity of identity proofs and function extensionality, are not validated by ITT; indeed, there are

sufficiently intensional models of the identity type to refute both properties [51, 33]. While the desirability of the unicity principle is perhaps up for debate, especially in light of recent developments in Homotopy Type Theory [54], theorists and practitioners alike generally agree that function extensionality is desirable.

A significant selling-point for ITT is that, by avoiding equality reflection, it presents a theory which can be implemented using type checking and normalization. Consequently, β and η rules are totally automatic and never require intervention from the user—in contrast to systems like RedPRL, whose users are accustomed to establishing β/η equivalences by hand at times when heuristical tactics prove inadequate. The downsides of pure ITT, however, are manifold: function extensionality is absolutely critical in practice.

1.4 Setoids and internal model constructions

A standard technique for avoiding the deficiencies of the identity type in ITT is the *setoid construction* [32], an exact completion which glues an equivalence relation $=_A$ onto each type $|A|$ in the spirit of Bishop [16]. When using setoids, a function $A \to B$ consists of a type-theoretic function $f : |A| \to |B|$ together with a proof that it preserves the equivalence relation, $f_= : (x, y : |A|) \to x =_A y \to f(x) =_B f(y)$; a *dependent setoid* (family of setoids) is a type-theoretic family equipped with a coherent coercion operator.

Setoids are a discipline for expressing internally precisely the extrinsic properties required for constructions to be extensional (compatible with equality); these extra proof obligations must be satisfied in parallel with constructions at every turn. The state of affairs for setoids is essentially analogous to that of proof assistants with equality reflection, in which type functionality subgoals play a similar role to the auxiliary paperwork generated by setoids.

Paradoxically, however, every construction in ordinary ITT is automatically extensional in this sense. A solution to the problem of equality in type theory should, unlike setoids, take advantage of the fact that type theory is already restricted to extensional constructions, adding to it only enough language to refer to equality internally. This is the approach taken by both Observational Type Theory and XTT.

1.5 Observational Type Theory

The first systematic solution to the problem of syntax for extensional equality without equality reflection was Observational Type Theory (OTT) [6, 7], which built on early work by Altenkirch and McBride [3, 44]. The central idea of OTT is to work with a *closed* universe of types, defining by recursion for each pair of types A, B a type $\mathsf{Eq}(A, B)$ of proofs that A and B are equal, and for each pair of elements $M : A$ and $N : B$, a type of proofs $\mathsf{Eq}_{A,B}(M, N)$ that M and N are (heterogeneously) equal. Finally, one defines "generic programs" by recursion on type structure which calculate coercions and coherences along proofs of equality.

One can think of OTT as equipping the semantic setoid construction with a direct-style type-theoretic language, and adding to it closed, inductively defined universes of types. The heterogeneous equality of OTT, initially a simplifying measure adopted from McBride's thesis [44], is an early precursor of the *dependent paths* which appear in Homotopy Type Theory [54], Cubical Type Theory [24, 8, 11], and XTT.

Recently, McBride and his collaborators have made progress toward a cubical version of OTT, using a different cube category and coercion structure, in which one coerces only from 0 to 1, and obtains fillers using an affine rescaling operation [23].

1.6 Cubical Type Theory

In a rather different line of research, Voevodsky showed that Intensional Type Theory is compatible with a *univalence axiom* yielding an element of $\mathsf{Id}_{\mathcal{U}}(A, B)$ for every equivalence (coherent isomorphism) between types A, B [37, 54]. A univalent universe *classifies* types under a certain size cut-off in the sense of higher topos theory [40]. However, Intensional Type Theory extended with univalence lacks *canonicity*, because identity elimination computes only on refl and not on proofs constructed by univalence.

Since then, *cubical type theories* have been developed to validate univalence without disrupting canonicity [24, 11]. These type theories extend Martin-Löf's type theory with an abstract interval, maps out of which represent paths, a higher-dimensional analogue to equality; the interval has abstract elements, represented by a new sort of *dimension* variable i, and constant endpoints $0, 1$. Coercions arise as an instance of *Kan structure* governed directly by the structure of paths between types, which are nothing more than types dependent on an additional dimension variable.

There are currently two major formulations of cubical type theory. De Morgan cubical type theory [24] equips the interval with negation and binary connection (minimum and maximum) operations. Cartesian cubical type theory [8, 11], the closest relative of XTT, has no additional structure on the interval, but equips types with a much stronger notion of coercion generalizing the one described in Section 2.1.1.

1.7 Our contribution: XTT

We contribute XTT (Appendix A), a new type theory that supports extensional equality without equality reflection, using ideas from cubical type theory [24, 8, 11]. In particular, we obtain a compositional account of propositional equality satisfying function extensionality and a *judgmental* version of the unicity of identity proofs—when $P, Q : \mathsf{Eq}_A(M, N)$, we have $P = Q$ judgmentally—enabling us to substantially simplify our Kan operations (Section 2.1.2). Moreover, XTT is closed under a cumulative hierarchy[1] of closed universes à la Russell. We hope to integrate XTT into the `redtt` cubical proof assistant [9] as an implementation of extensional equality in the style of two-level type theory [11].

A common thread that runs through the XTT formalism is the decomposition of constructs from OTT into more modular, *judgmental* principles. For instance, rather than defining equality separately at every type and entangling the connectives, we define equality once and for all using the interval. Likewise, rather than ensuring that equality proofs are unique through brute force, we obtain unicity using a structural rule which does not mention the equality type.

By first developing the model theory of XTT in an algebraic way (Section 3), we then prove a canonicity theorem for the *initial* model of XTT (Section 3.2): any closed term of boolean type is equal to either true or false. This result is obtained using a novel extension of the categorical gluing technique described by Coquand and Shulman [27, 48]. Canonicity expresses a form of "computational adequacy"—in essence, that the equational theory of XTT suffices to derive any equation which ought to hold by (closed) computation—and is one of many syntactical considerations that experience has shown to be correlated to usability.

[1] As in previous work [49], we employ an *algebraic* version of cumulativity which does not require subtyping.

$$
\begin{array}{llll}
\textit{(cubes)} & \Psi, \Phi & ::= & \cdot \mid \Psi, i \mid \Psi, \xi \\
\textit{(contexts)} & \Gamma, \Delta & ::= & \cdot \mid \Gamma, x : A \\
\textit{(dimensions)} & r, s & ::= & i \mid \varepsilon \\
\textit{(constant dims.)} & \varepsilon & ::= & 0 \mid 1 \\
\textit{(constraints)} & \xi & ::= & r = r' \\
\textit{(universe levels)} & k, l & ::= & n \quad (n \in \mathbb{N}) \\
\textit{(types)} & A, B & ::= & M \mid (x : A) \to B \mid (x : A) \times B \mid \mathsf{Eq}_{i.A}(M, N) \mid \Uparrow_k^l A \mid \mathcal{U}_k \mid \mathsf{bool} \\
\textit{(terms)} & M, N & ::= & x \mid A \mid \lambda x.M \mid \mathsf{app}_{x:A.B}(M, N) \mid \langle M, N \rangle \mid \mathsf{fst}_{x:A.B}(M) \mid \mathsf{snd}_{x:A.B}(M) \mid \\
 & & & \lambda i.M \mid \mathsf{app}_{i.A}(M, r) \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}_{x.A}(M; N_0, N_1) \mid \\
 & & & [i.A] \downarrow^r_{r'} M \mid A \downarrow^r_{r'} M \ [s \ \mathsf{with} \ 0 \hookrightarrow j.N_0 \mid 1 \hookrightarrow j.N_1]
\end{array}
$$

Figure 1 A summary of the raw syntax of XTT. As a matter of top-level notation, we freely omit annotations that can be inferred from context, writing $M(N)$ for $\mathsf{app}_{x:A.B}(M, N)$. The annotations chosen in the raw syntax are the minimal ones required to establish a coherent interpretation into the initial XTT-algebra; for instance, it is unnecessary to include an annotation on the λ-abstraction.

2 Programming and proving in XTT

Like other cubical type theories, the XTT language extends Martin-Löf's type theory with a new sort of variable i ranging over an abstract interval with global elements 0 and 1; we call an element r of the interval a *dimension*, and we write ε to range over a constant dimension 0 or 1. Cubical type theories like XTT also use a special kind of hypothesis to constrain the values of dimensions: when r and s are dimensions, then $r = s$ is a *constraint*. In XTT, a single context Ψ accounts for both dimension variables (Ψ, i) and constraints $(\Psi, r = s)$. We will write $\Psi \mid r\ \mathit{dim}$ for when a dimension r is valid in a dimension context Ψ. The judgment $\Psi \mid r = s\ \mathit{dim}$ holds when r and s are equal as dimensions with respect to the constraints in Ψ. Dimensions can be substituted for dimension variables, an operation written $M\langle r/i \rangle$.

Finally, ordinary type-theoretic assumptions $x : A$ are kept in a context Γ that depends on Ψ. In XTT, a full context is therefore written $\Psi \mid \Gamma$. The meaning of a judgment at context $(\Psi, i = r)$ is completely determined by its instance under the substitution r/i. Under the false constraint $0 = 1$, all judgments hold; the resulting collapse of the typing judgment and the judgmental equality does not disrupt any important metatheoretic properties, because the theory of dimensions is decidable.

The general typehood judgment $\Psi \mid \Gamma \vdash A\ \mathit{type}_k$ means that A is a type of universe level k in context Γ over the cube Ψ; note that this judgment presupposes the well-formedness of Ψ, Γ. Likewise, the element typing judgment $\Psi \mid \Gamma \vdash M : A$ means that M is an element of the type A in Γ over Ψ as above; this form of judgment presupposes the well-formedness of A and thence Ψ, Γ. We also have typed judgmental equality $\Psi \mid \Gamma \vdash A = B\ \mathit{type}_k$ and $\Psi \mid \Gamma \vdash M = N : A$, which presuppose the well-formedness of *all* their constituents.

Dependent equality types

XTT extends Martin-Löf type theory with *dependent equality types* $\mathsf{Eq}_{i.A}(N_0, N_1)$ when $\Psi, i \mid \Gamma \vdash A\ \mathit{type}_k$ and $\Psi \mid \Gamma \vdash N_0 : A\langle 0/i \rangle$ and $\Psi \mid \Gamma \vdash N_1 : A\langle 1/i \rangle$. Geometrically, elements of this type are *lines* or *paths* in the type A ranging over dimension i, with left endpoint N_0 and right endpoint N_1.[2] This type captures internally the equality of N_0 and N_1; dependency

[2] Our dependent equality types are locally the same as dependent path types $\mathsf{Path}_{i.A}(N_0, N_1)$ from cubical type theories; however, we have arranged in XTT for them to satisfy a unicity principle by which

of A on the dimension i is in essence a cubical reconstruction of heterogeneous equality, albeit with different properties from the version invented by McBride in his thesis [44].

An element of the equality type $\mathsf{Eq}_{i.A}(N_0, N_1)$ is formed by the dimension λ-abstraction $\lambda i.M$, requiring that M is an element of A in the extended context, and that N_0, N_1 are the left and right sides of M respectively. Proofs P of equality are eliminated by dimension application, $P(r)$, and are subject to β, η, ξ rules analogous to those for function types. Finally, we have $P(\varepsilon) = N_\varepsilon$ always, extending Gentzen's principle of inversion to the side condition that we placed on M. More formally:

$$\frac{\Psi, i \mid \Gamma \vdash M : A}{\Psi \mid \Gamma \vdash \lambda i.M : \mathsf{Eq}_{i.A}(N_0, N_1)} \quad \frac{\Psi \mid r\ dim \quad \Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M(r) : A\langle r/i\rangle} \quad \frac{\Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M(\varepsilon) = N_\varepsilon : A\langle \varepsilon/i\rangle}$$

$$\frac{\Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\Psi \mid \Gamma \vdash M = \lambda i.M(i) : \mathsf{Eq}_{i.A}(N_0, N_1)} \quad \frac{\Psi, i \mid \Gamma \vdash M : A}{\Psi \mid \Gamma \vdash (\lambda i.M)(r) = M\langle r/i\rangle : A\langle r/i\rangle}$$

Function extensionality

A benefit of the cubical formulation of equality types is that the principle of function extensionality is trivially derivable in a computationally well-behaved way. Suppose that $f, g : (x : A) \to B$ and we have a family of equalities $h : (x : A) \to \mathsf{Eq}_{\_.B}(f(x), g(x))$; then, we obtain a proof that f equals g by abstraction and application:

$$\lambda i.\lambda x.h(x)(i) : \mathsf{Eq}_{\_.(x:A)\to B}(f, g)$$

In semantics of type theory, the structure of equality on a type usually mirrors the structure of the *elements* of that type in a straightforward way: for instance, a function of equations is used to equate two functions, and a pair of equations is used to equate two pairs. The benefit of the cubical approach is that this observation, at first purely empirical, is systematized by *defining* equality in every type in terms of the elements of that type in a context extended by a dimension.

Judgmental unicity of equality: boundary separation

In keeping with our desire to provide *convenient* syntax for working with extensional equality, we want proofs $P, Q : \mathsf{Eq}_{i.A}(N_0, N_1)$ of the same equation to be judgmentally equal. Rather than adding a rule to that effect, whose justification in the presence of the elimination rules for equality types would be unclear, we instead impose a more primitive *boundary separation* principle at the judgmental level: every term is completely determined by its boundary.[3]

$$\frac{\Psi \mid r\ dim \qquad \overbrace{\Psi, r = \varepsilon \mid \Gamma \vdash M = N : A}}{\Psi \mid \Gamma \vdash M = N : A}$$

In this rule we have abbreviated $\Psi, r = 0 \mid \Gamma \vdash M = N : A$ and $\Psi, r = 1 \mid \Gamma \vdash M = N : A$ as $\overbrace{\Psi, r = \varepsilon \mid \Gamma \vdash M = N : A}$. We shall make use of this notation throughout the paper.

---

they earn the name "equality" rather than "path".

[3] We call this principle "boundary separation" because it turns out to be exactly the fact that the collections of types and elements, when arranged into presheaves on the category of contexts, are *separated* with respect to a certain coverage on this category. We develop this perspective in the extended version of our paper.

We can now *derive* a rule that (judgmentally) equates all $P, Q : \mathsf{Eq}_{i.A}(N_0, N_1)$.

Proof. If $P, Q : \mathsf{Eq}_{i.A}(M, N)$, then to show that $P = Q$, it suffices to show that $\lambda i.P(i) = \lambda i.Q(i)$; by the congruence rule for equality abstraction, it suffices to show that $P(i) = Q(i)$ in the extended context. But by boundary separation, we may pivot on the boundary of i, and it suffices to show that $P(0) = Q(0)$ and $P(1) = Q(1)$. But these are automatic, because P and Q are both proofs of $\mathsf{Eq}_{i.A}(N_0, N_1)$, and therefore $P(\varepsilon) = Q(\varepsilon) = N_\varepsilon$. ◀

In an unpublished note from 2017, Thierry Coquand identifies a class of cubical sets equivalent to our separated types, calling them "Bishop sets" [26].

2.1 Kan operations: coercion and composition

How does one *use* a proof of equality? We must have at least a coercion operation which, given a proof $Q : \mathsf{Eq}_{\_.\mathcal{U}_k}(A, B)$, coherently transforms elements $M : A$ to elements of B.

2.1.1 Generalized coercion

In XTT, coercion and its coherence are obtained as instances of one general operation: for any two dimensions r, r' and a *line* of types $i.C$, if M is an element of $C\langle r/i\rangle$, then $[i.C]\downarrow^r_{r'} M$ is an element of $C\langle r'/i\rangle$.

$$\frac{\Psi \mid r, r' \ dim \qquad \Psi, i \mid \Gamma \vdash C \ type_k \qquad \Psi \mid \Gamma \vdash M : C\langle r/i\rangle}{\Psi \mid \Gamma \vdash [i.C]\downarrow^r_{r'} M : C\langle r'/i\rangle}$$

In the case of a proof $Q : \mathsf{Eq}_{\_.\mathcal{U}_k}(A, B)$ of equality between types, we coerce $M : A$ to the type B using the instance $[i.Q(i)]\downarrow^0_1 M$. But how does M relate to its coercion? *Coherence* of coercion demands their equality, although such an equation must relate terms of (formally) different types; this heterogeneous equality is stated in XTT using a *dependent* equality type $\mathsf{Eq}_{i.Q(i)}(M, [i.Q(i)]\downarrow^0_1 M)$. To construct an element of this equality type, we use the same coercion operator but with a different choice of r, r'; we construct this *filler* by coercing from 0 to a fresh dimension, obtaining $\lambda j.[i.Q(i)]\downarrow^0_j M : \mathsf{Eq}_{i.Q(i)}(M, [i.Q(i)]\downarrow^0_1 M)$:

$$j.Q(j) \quad \ni \quad M \ \frac{j.[i.Q(i)]\downarrow^0_j M}{}\ [i.Q(i)]\downarrow^0_1 M$$

To see that the filler $[i.Q(i)]\downarrow^0_j M$ has the correct boundary with respect to j, we inspect its instances under the substitutions $0/j, 1/j$. First, we observe that the right-hand side $([i.Q(i)]\downarrow^0_j M)\langle 1/j\rangle$ is exactly $[i.Q(i)]\downarrow^0_1 M$; second, we must see that $([i.Q(i)]\downarrow^0_j M)\langle 0/j\rangle$ is M, bringing us to an important equation that we must impose generally:

$$\Psi \mid \Gamma \vdash [i.C]\downarrow^r_r M = M : C\langle r/i\rangle$$

How do coercions compute?

In order to ensure that proofs in XTT can be computed to a canonical form, we need to explain generalized coercion in each type in terms of the elements of that type. To warm up, we explain how coercion must compute in a non-dependent function type:

$$[i.A \to B]\downarrow^r_{r'} M = \lambda x.[i.B]\downarrow^r_{r'} \left(M([i.A]\downarrow^{r'}_r x)\right)$$

That is, we abstract a variable $x : A\langle r'/i\rangle$ and need to obtain an element of type $B\langle r'/i\rangle$. By *reverse* coercion, we obtain $[i.A]\downarrow^{r'}_r x : A\langle r/i\rangle$; by applying M to this, we obtain an

element of type $B\langle r/i\rangle$. Finally, we coerce from r to r'. The version for dependent function types is not much harder, but requires a filler:

$$\frac{\widetilde{x} \triangleq \lambda j.[i.A] \downarrow_j^{r'} x}{[i.(x : A) \to B] \downarrow_{r'}^{r} M = \lambda x.[i.B[\widetilde{x}(i)/x]] \downarrow_{r'}^{r} M(\widetilde{x}(r))}$$

The case for dependent pair types is similar, but without the contravariance:

$$\frac{\widetilde{M_0} \triangleq \lambda j.[i.A] \downarrow_j^{r} \mathsf{fst}(M)}{[i.(x : A) \times B] \downarrow_{r'}^{r} M = \langle \widetilde{M_0}(r'), [i.B[\widetilde{M_0}(i)/x]] \downarrow_{r'}^{r} \mathsf{snd}(M)\rangle}$$

Coercions for base types (like bool) are uniformly determined by *regularity*, a rule of XTT stating that if A is a type which doesn't vary in the dimension i, then $[i.A] \downarrow_{r'}^{r} M$ is just M. Regularity makes type sense because $A\langle r/i\rangle = A = A\langle r'/i\rangle$; semantically, it is more difficult to justify in the presence of standard universes, and is not known to be compatible with principles like univalence.[4] But XTT is specifically designed to provide a theory of *equality* rather than *paths*, so we do not expect or desire to justify univalence at this level.[5]

The only difficult case is to define coercion for equality types; at first, we might try to define $[i.\mathsf{Eq}_{j.A}(N_0, N_1)] \downarrow_{r'}^{r} P$ as $\lambda j.[i.A] \downarrow_{r'}^{r} P(j)$, but this does not make type-sense: we need to see that $\big([i.A] \downarrow_{r'}^{r} P(j)\big)\langle \varepsilon/j\rangle = N_\varepsilon$, but we only obtain $\big([i.A] \downarrow_{r'}^{r} P(j)\big)\langle \varepsilon/j\rangle = [i.A] \downarrow_{r'}^{r} N_\varepsilon$, which is "off by" a coercion. Intuitively, we can solve this problem by specifying what values a coercion takes under certain substitutions: in this case, N_0 under $0/j$, and N_1 under $1/j$. We call the resulting operation *generalized composition*.

2.1.2 Generalized composition

For any dimensions r, r', s and a line of types $i.C$, if M is an element of $C\langle r/i\rangle$ and $i.N_0, i.N_1$ are lines of elements of C defined respectively on the subcubes $(s = 0), (s = 1)$ such that $N_\varepsilon\langle r/i\rangle = M$, then $[i.C] \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ 0 \hookrightarrow i.N_0 \mid 1 \hookrightarrow i.N_1]$ is an element of $C\langle r'/i\rangle$. This is called the *composite* of M with N_0, N_1 from r to r', schematically abbreviated $[i.C] \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow i.N_\varepsilon}]$. As with coercion, when $r = r'$, we have $[i.C] \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow i.N_\varepsilon}] = M$, and moreover, if $s = \varepsilon$, we have $[i.C] \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow i.N_\varepsilon}] = N_\varepsilon\langle r'/i\rangle$.

Returning to coercion for equality types, we now have exactly what we need:

$$[i.\mathsf{Eq}_{j.C}(N_0, N_1)] \downarrow_{r'}^{r} P = \lambda j.([i.C] \downarrow_{r'}^{r} P(j)\ [j\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow \_.N_\varepsilon}])$$

Next we must explain how the generalized composition operation computes at each type; in previous works [11], we have seen that it is simpler to instead *define* generalized composition in terms of a simpler *homogeneous* version, in which one composes in a type C rather than a line of types $i.C$; we write $C \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow j.N_\varepsilon}]$ for this homogeneous composition, defining the generalized composition in terms of it as follows:

$$[i.C] \downarrow_{r'}^{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow i.N_\varepsilon}] = C\langle r'/i\rangle \downarrow_{r'}^{r} ([i.C] \downarrow_{r'}^{r} M)\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow i.[i.C] \downarrow_{r'}^{i} N_\varepsilon}]$$

[4] Regularity is proved by Swan to be incompatible with univalent universes assuming that certain standard techniques are used [52]; however, it is still possible that there is a different way to model univalent universes with regularity. Awodey constructs a model of intensional type theory *without* universes in regular Kan cubical sets [13], using the term *normality* for what we have called regularity.

[5] Indeed, unicity of identity proofs is also incompatible with univalence. XTT is, however, compatible with a formulation in which it is just one level of a two-level type theory, along the lines of Voevodsky's Homotopy Type System, in which the other level would have a univalent notion of path that coexists in harmony with our notion of equality [55, 11].

Surprisingly, in XTT we do not need to build in any computation rules for homogeneous composition, because they are completely determined by judgmental boundary separation. For instance, we can derive a computation rule already for homogeneous composition in the dependent function type, by observing that the equands have the same boundary with respect to the dimension j:

$$(x : A) \to B \downarrow^{r}_{r'} M\ [j \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}] = \lambda x.B \downarrow^{r}_{r'} M(x)\ [j \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.N_\varepsilon}]$$

From homogeneous composition, we obtain *symmetry and transitivity* for the equality types. Given $P : \mathsf{Eq}_{\_.A}(M, N)$, we obtain an element of type $\mathsf{Eq}_{\_.A}(N, M)$ as follows:

$$\lambda i.A \downarrow^{0}_{1} P(0)\ [i \text{ with } 0 \hookrightarrow j.P(j) \mid 1 \hookrightarrow \_.P(0)]$$

Furthermore, given $Q : \mathsf{Eq}_{\_.A}(N, O)$, we obtain an element of type $\mathsf{Eq}_{\_.A}(M, O)$ as follows:

$$\lambda i.A \downarrow^{0}_{1} P(i)\ [i \text{ with } 0 \hookrightarrow \_.P(0) \mid 1 \hookrightarrow j.Q(j)]$$

▶ **Example 2.1** (Identity type)**.** It is possible to *define* Martin-Löf's identity type and its eliminator, albeit with a much stronger computation rule than is customary.

$$\mathsf{Id}_A(M, N) \triangleq \mathsf{Eq}_{\_.A}(M, N) \qquad \mathsf{refl}_A(M) \triangleq \lambda\_.M$$

$$\frac{\widetilde{P} \triangleq \lambda j.(A \downarrow^{0}_{j} P(0)\ [i \text{ with } 0 \hookrightarrow \_.P(0) \mid 1 \hookrightarrow k.P(k)])}{\mathsf{J}_{x,y,p.C(x,y,p)}(P; x.Q(x)) \triangleq [i.C(P(0), P(i), \widetilde{P})] \downarrow^{0}_{1} Q(P(0))}$$

This particular definition of J relies on XTT's *boundary separation* rule, but one could instead define it in a more complicated way without boundary separation. However, that this construction of the identity type models the computation rule relies crucially on *regularity*, which does not hold in other cubical type theories whose path types validate univalence. In the absence of regularity, one can define an operator with the same type as J but which satisfies its computation rule only up to a path.

2.2 Closed universes and type-case

In Section 2.1.1, we showed how to calculate coercions $[i.C] \downarrow^{r}_{r'} M$ in each type former C. In previous cubical type theories [24, 11], one could "uncover" all the things that a coercion must be equal to by reducing according to the rules which inspect the interior of the type line $i.C$. While this strategy can be used to establish canonicity for closed terms, it fails to uncover certain reductions for open terms, a prerequisite for algorithmic type checking.

Specifically, given a variable $q : \mathsf{Eq}_{\_.\mathcal{U}_k}(A_0 \to B_0, A_1 \to B_1)$, the coercion $[i.q(i)] \downarrow^{r}_{r'} M$ is *not* necessarily stuck, unlike in other cubical type theories. Suppose that we can find further proofs $Q_A : \mathsf{Eq}_{\_.\mathcal{U}_k}(A_0, A_1)$ and $Q_B : \mathsf{Eq}_{\_.\mathcal{U}_k}(B_0, B_1)$; in this case, $\lambda i.Q_A(i) \to Q_B(i)$ is *also* a proof of $\mathsf{Eq}_{\_.\mathcal{U}_k}(A_0 \to B_0, A_1 \to B_1)$, so by boundary separation it must be equal to q, and therefore $[i.q(i)] \downarrow^{r}_{r'} M$ must be equal to $[i.Q_A(i) \to Q_B(i)] \downarrow^{r}_{r'} M$. But the type-directed reduction rule for coercion applies only to the latter! Generally, to see how to reduce the first coercion, it seems that we need to be able to "dream up" proofs Q_A, Q_B out of thin air, or determine that they can't exist, an impossible task.

In XTT, we cut this Gordian knot by ensuring that Q_A, Q_B *always* exist, following the approach employed in OTT. To invert the equation q into Q_A and Q_B, we add an intensional *type-case* operator to XTT, committing to a closed and inductive notion of universe by allowing pattern-matching on types [46]. It is also possible to extend XTT with open and/or univalent universes which themselves lack boundary separation, as in two-level type theories.

For illustrative purposes, consider coercion along an equality between dependent function types. Given $q : \mathsf{Eq}_{\_.\mathcal{U}_k}((x : A_0) \to B_0, (x : A_1) \to B_1)$, we define by type-case the following:

$$Q_A \triangleq \lambda i.\mathsf{tycase}\ q(i)\ [\Pi_A B \mapsto A \mid \_ \mapsto \mathsf{bool}] : \mathsf{Eq}_{\_.\mathcal{U}_k}(A_0, A_1)$$
$$Q_B \triangleq \lambda i.\mathsf{tycase}\ q(i)\ [\Pi_A B \mapsto B \mid \_ \mapsto \lambda\_.\mathsf{bool}] : \mathsf{Eq}_{i.Q_A(i)\to\mathcal{U}_k}(\lambda x.B_0, \lambda x.B_1)$$

Because of q's boundary, we are concerned only with the Π branch of the above expressions, and are free to emit a "dummy" answer in other branches. With Q_A, Q_B in hand, we note that $q(i) = (x : Q_A(i)) \to Q_B(i)(x)$ using boundary separation; therefore, we are free to calculate $[i.q(i)] \downarrow^r_{r'} M$ as follows:

$$\frac{\widetilde{x} \triangleq \lambda j.[i.Q_A(i)] \downarrow^{r'}_j x}{[i.q(i)] \downarrow^r_{r'} M = \lambda x.[i.Q_B(i)(\widetilde{x}(i))] \downarrow^r_{r'} M(\widetilde{x}(r))}$$

This *lazy* style of computing with proofs of equality means, in particular, that coercing along an equation cannot tell the difference between a postulated axiom and a canonical proof of equality, making XTT compatible with extension by consistent equational axioms.

▶ Remark 2.2. One might wonder whether it is possible to tame the use of type-case above to something compatible with a *parametric* understanding of types, in which (as in OTT) one cannot branch on whether or not C is a function type or a pair type, etc. It is likely that this can be done, but we stress that the fundamental difficulty is not resolved: whether or not we allow general type-case, we have not escaped the need for type constructors to be disjoint and injective, which contradicts the role of universes in mathematics as (weak) classifiers of small families. Future work on XTT and its successors must focus on resolving this issue, quite apart from any considerations of parametricity.

2.3 Future extensions

Universe of propositions

XTT currently lacks one of the hallmarks of OTT, an extensional universe of proof-irrelevant propositions. In future work, we intend to extend XTT with a reflective subuniverse Ω of propositions closed under equality and universal and existential quantification over arbitrary types, satisfying:

- *Proof irrelevance.* For each proposition $\Psi \mid \Gamma \vdash p : \Omega$, we have $\Psi \mid \Gamma \vdash M = N : p$ for all $\Psi \mid \Gamma \vdash M, N : p$.
- *Extensionality (univalence).* For all $\Psi \mid \Gamma \vdash p, q : \Omega$, we have an element of $\mathsf{Eq}_{\_.\Omega}(p, q)$ whenever there are functions $p \to q$ and $q \to p$.

The *reflection* of the propositional subuniverse will take a type $\Psi \mid \Gamma \vdash A\ \mathit{type}_k$ to a proof-irrelevant proposition $\Psi \mid \Gamma \vdash \|A\| : \Omega$, acting as a strict truncation or squash type [25, 47, 14]. The addition of Ω will allow XTT to be used as a syntax for topos-theoretic constructions, with Ω playing the role of the subobject classifier.

(Indexed) Quotient Inductive Types

Another natural extension of XTT is the addition of *quotient types*; already considered as an extension to OTT by the Epigram Team [18] and more recently by Atkey [12], quotient types are essential when using type theory for either programming or mathematics. One of the ideas of Homotopy Type Theory and cubical type theories in particular is to reconstruct

the notion of quotienting by an equivalence relation as a special case of *higher inductive type* (HITs), a generalization of ordinary inductive types which allows constructors to target higher dimensions with a specified partial boundary. When working purely at the level of sets, as in XTT, these higher inductive types are called *quotient inductive types* (QITs) [5].

We intend to adapt the work of Cavallo and Harper [22] to a general schema for *indexed quotient inductive types* as an extension of XTT. The resulting system would support ordinary quotients by equivalence relations *en passant*, and when these equivalence relations are valued in Ω, one can show that they are effective. Quotient inductive types also enable the construction of free algebras for infinitary algebraic theories, usually obtained in classical set theory from the non-constructive axiom of choice [17, 39]. Another application of quotient inductive types is the definition of a *localization* functor with respect to a class of maps, enabling users of the extended XTT to work internally with sheaf subtoposes.

The extension of XTT with quotient inductive types means that we must account for *formal homogeneous composites* in QITs which are canonical forms [22, 28]. Ordinarily, this introduces a severe complicating factor to a canonicity proof, because the notion of canonical form ceases to be stable under all dimension substitutions [11, 34], but we expect the *proof-relevant* cubical logical families technique that we introduce in Section 3 to scale directly to the case of quotient inductive types without significant change, in contrast with classical approaches based on partial equivalence relations.

3 Algebraic model theory and canonicity

We have been careful to formulate the XTT language in a *(generalized) algebraic* way, obtaining automatically a category of algebras and homomorphisms which is equipped with an initial object [19, 20, 36]. That this initial object is isomorphic to the model of XTT obtained by constraining and quotienting its raw syntax under judgmental equality (i.e. the Lindenbaum–Tarski algebra) is an instance of Voevodsky's famous Initiality Conjecture [56], and we do not attempt to prove it here; we merely observe that this result has been established for several simpler type theories [50, 21].

Working within the category of XTT-algebras enables us to formulate and prove results like canonicity and normalization for the *initial* XTT-algebra in an economical manner, avoiding the usual bureaucratic overhead of reduction relations and partial equivalence relations, which were the state of the art for type-theoretic metatheory prior to the work of Shulman [48], Altenkirch and Kaposi [4], and Coquand [27].

Because our algebraic techniques involve defining families over *only* well-typed terms already quotiented by judgmental equality, we avoid many of the technical difficulties arising from working with the raw terms of cubical type theories, including the closure under "coherent expansion" which is critical to earlier cubical metatheories [11, 34]. Our abstract gluing-based approach therefore represents a methodological advance in metatheory for cubical type theories.

▶ **Theorem 3.1** (Canonicity). *In the initial XTT-algebra, if* $\cdot \mid \cdot \vdash M : \mathsf{bool}$*, then either* $\cdot \mid \cdot \vdash M = \mathsf{true} : \mathsf{bool}$ *or* $\cdot \mid \cdot \vdash M = \mathsf{false} : \mathsf{bool}$.

Following previous work [49], we employ for our semantics a variant of *categories with families* (cwf) [29] which supports a predicative hierarchy of universes à la Russell. A cwf in our sense begins with a *category of contexts* $\mathcal{C}$, and a presheaf of types $\mathsf{Ty}_{\mathcal{C}} : \widehat{\mathcal{C} \times \mathbb{L}}$; here $\mathbb{L}$ is the category of *universe levels*, with objects the natural numbers and unique arrows $l \longrightarrow k$

if and only if $k \leq l$.[6] The fiber of the presheaf of types $\mathsf{Ty}_{\mathcal{C}} : \widehat{\mathcal{C} \times \mathbb{L}}$ at (Γ, k) is written $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$, and contains the types in context Γ of universe level k. Reindexing implements simultaneous substitution $\gamma^* A$ and universe level shifting $\Uparrow^l_k A$. In our metatheory, we assume the Grothendieck Universe Axiom, and consequently obtain a transfinite ordinal-indexed hierarchy of meta-level universes $\mathcal{V}_k$. We impose the requirement that each collection of types $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ is k-small, i.e. $\mathsf{Ty}^k_{\mathcal{C}}(\Gamma) \in \mathcal{V}_k$.

Next, we require a *dependent* presheaf of elements $\mathsf{El}_{\mathcal{C}} : \widehat{\int \mathsf{Ty}_{\mathcal{C}}}$, whose fibers $\mathsf{El}_{\mathcal{C}}((\Gamma, k), A)$ we write $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$; to interpret the actions of level lifting on terms properly, we require the functorial actions $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A) \longrightarrow \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \Uparrow^l_k A)$ to be identities, strictly equating the fibers $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$ and $\mathsf{El}_{\mathcal{C}}(\Gamma \vdash \Uparrow^l_k A)$.

The remaining data of a basic cwf is a *context comprehension*, which for every context Γ and type $A \in \mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ determines an extended context $\Gamma.A$ with a weakening substitution $\Gamma.A \xrightarrow{\mathbf{p}} \Gamma$ and a variable term $\mathbf{q} \in \mathsf{El}_{\mathcal{C}}(\Gamma.A \vdash \mathbf{p}^* A)$.

Next, we specify what further structure is required to make such a cwf into an XTT-algebra. To represent contexts Ψ semantically, we use the *augmented Cartesian cube category* $\square_+$, which adjoins to the Cartesian cube category $\square$ an initial object; from this, we obtain equalizers $0 = 1$ in addition to the equalizers $i = r$ which exist in $\square$. We then require a *split fibration* $\mathcal{C} \xrightarrow{\mathfrak{u}} \square_+$ with a terminal object, which implements the dependency of contexts Γ on cubes Ψ and forces appropriate dimension restrictions to exist for contexts, types and elements. The split fibration induces all the structure necessary to implement dimension operations; we refer the reader to the extended version of our paper for details. In the following discussion, we limit ourselves to a few simpler consequences. First, we can apply dimension substitutions in terms and types, writing $\psi^\ddagger_\Gamma A$ to apply ψ in a type A in context Γ. We can also apply dimension substitutions to contexts, written $\psi^* \Gamma$. We write $\hat{\imath}$ for the dimension substitution which weakens by a dimension variable i. Finally, we write $\mathsf{Dim}_{\mathcal{C}}(\Gamma)$ for the set of valid dimensions expressions generated from $\mathfrak{u}(\Gamma)$.

▶ **Requirement** (Boundary separation in models)**.** In order to enforce boundary separation in XTT-algebras we require that types and elements over them satisfy a separation property. In the extended version of our paper we phrase the full condition as a separation requirement with respect to a particular Grothendieck topology on the category of contexts. A specific consequence is the familiar boundary separation principle for types: given two types $A, B \in \mathsf{Ty}_{\mathcal{C}}(\Gamma)$ and a dimension $i \in \mathfrak{u}(\Gamma)$, if $(\epsilon/i)^\ddagger A = (\epsilon/i)^\ddagger B$ for each $\epsilon \in \{0, 1\}$ then $A = B$.

▶ **Requirement** (Coercion in models)**.** An XTT-algebra must also come with a *coercion structure*, specifying how generalized coercion is interpreted in each type. For every type $A \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^* \Gamma)$ over Ψ, i, dimensions $r, r' \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)$, and element $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r/i)^\ddagger_{\hat{\imath}^* \Gamma} A)$, we require an element $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A} M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r'/i)^\ddagger_{\hat{\imath}^* \Gamma} A)$ with the following properties (in addition to naturality requirements):

- *Adjacency.* If $r = r'$ then $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A} M = M$.
- *Regularity.* If $A = \hat{\imath}^\ddagger_\Gamma A'$ for some $A' \in \mathsf{Ty}^n_{\mathcal{C}}(\Gamma)$, then $\mathbf{coe}^{r \rightsquigarrow r'}_{i.A} M = M$.

Additional equations in later requirements specify that generalized coercion computes properly in each connective. Similarly, a model must be equipped with a *composition structure* which specifies the interpretation of the composition operator.

Finally, we specify algebraically the data with which such a cwf must be equipped in order to model all the connectives of XTT (again, details are contained in the extended

[6] Observe that $\mathbb{L} = \omega^{\mathsf{op}}$; reversing arrows allows us to move types from smaller universes to larger ones.

version of our paper); to distinguish the abstract (De Bruijn) syntax of the cwf from the raw syntax of XTT we use boldface, writing $\mathbf{\Pi}(A, B)$, $\mathbf{papp}(i.A, M, r)$ and $\mathbf{U}_k$ to correspond to $(x : A) \to B$, $\mathsf{app}_{i.A}(M, r)$ and $\mathcal{U}_k$ respectively, etc. We take a moment to specify how some of the primitives of XTT are translated into requirements on a model.

▶ **Requirement** (Dependent equality types in models)**.** An XTT-algebra must model *dependent equality types*, which is to say that the following structure is exhibited:

- *Formation.* For each type $A \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^*\Gamma)$ and elements $\overrightarrow{N_\varepsilon \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (\varepsilon/i)^\ddagger A)}$, a type $\mathbf{Eq}(i.A, N_0, N_1) \in \mathsf{Ty}^n_{\mathcal{C}}(\Gamma)$.
- *Introduction.* For each $M \in \mathsf{El}_{\mathcal{C}}(\hat{\imath}^*\Gamma \vdash A)$, an element $\mathbf{plam}(i.A, M) \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, (0/i)^\ddagger M, (1/i)^\ddagger M))$.
- *Elimination.* For each $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, N_0, N_1))$ and $\overrightarrow{r \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)}$, an element $\mathbf{papp}(i.A, M, r) \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r/i)^\ddagger A)$ satisfying the equations $\overrightarrow{\mathbf{papp}(i.A, M, \epsilon) = N_\epsilon}$.
- *Computation.* For $M \in \mathsf{El}_{\mathcal{C}}(\hat{\imath}^*\Gamma \vdash A)$ and $r \in \mathsf{Dim}_{\mathcal{C}}(\Gamma)$, the equation:

$$\mathbf{papp}(i.A, \mathbf{plam}(i.A, i.M), r) = (r/i)^\ddagger M$$

- *Unicity.* For $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash \mathbf{Eq}(i.A, N_0, N_1))$, $M = \mathbf{plam}(i.A, j.\mathbf{papp}(i.\hat{\jmath}^\ddagger A, \hat{\jmath}^\ddagger M, j))$.
- *Level restriction.* The following equations:

$$\Uparrow^l_k \mathbf{Eq}(i.A, N_0, N_1) = \mathbf{Eq}(i.\Uparrow^l_k A, N_0, N_1) \qquad \mathbf{plam}(i.\Uparrow^l_k A, M)r = \mathbf{plam}(i.A, M)r$$

$$\mathbf{papp}(i.\Uparrow^l_k A, M, r) = \mathbf{papp}(i.A, M, r)$$

- *Naturality.* For $\Delta \xrightarrow{\gamma} \Gamma$, the following naturality equations:

$$\gamma^*\mathbf{Eq}(i.A, N_0, N_1) = \mathbf{Eq}(i.(\hat{\imath}^+\gamma)^*A, \gamma^*N_0, \gamma^*N_1)$$

$$\gamma^*\mathbf{plam}(i.A, i.M) = \mathbf{plam}(i.(\hat{\imath}^+\gamma)^*A, i.(\hat{\imath}^+\gamma)^*M)$$

$$\gamma^*\mathbf{papp}(i.A, M, r) = \mathbf{papp}(i.(\hat{\imath}^+\gamma)^*A, \gamma^*M, \gamma^*r)$$

- *Coercion.* When $\Gamma \overset{\mathsf{u}}{\longmapsto\!\triangleright} \Psi, j$ and $M \in \mathsf{El}_{\mathcal{C}}((r/j)^*\Gamma \vdash (r/j)^\ddagger\mathbf{Eq}(i.A, N_0, N_1))$ where $\Psi \mid r, r'$ *dim*, we require that $\mathbf{coe}^{r \rightsquigarrow r'}_{j.\mathbf{Eq}(i.A, N_0, N_1)} M$ equals the following abstraction:

$$\mathbf{plam}(i.(r'/j)^\ddagger A, i.\mathbf{com}^{r \rightsquigarrow r'}_{j.A}\ \mathbf{papp}(i.(r/j)^\ddagger A, \hat{\imath}^\ddagger M, i)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\hat{\jmath}^\ddagger N_\varepsilon}])$$

Any model of extensional type theory can be used to construct a model of XTT, so long as it is equipped with a cumulative, inductively defined hierarchy of universes closed under dependent function types, dependent pair types, extensional equality types and booleans. (Meaning explanations in the style of Martin-Löf [43] are one such model.) The interpretation of XTT into extensional models involves erasing dimensions, coercions, and compositions; the only subtlety, easily managed, is to ensure that all judgments under absurd constraints hold.

3.1 The cubical logical families construction

Any XTT-algebra $\mathcal{C}$ extends to a category $\mathcal{C}^\star$ of *proof-relevant logical predicates*, which we call *logical families* by analogy. The proof-relevant character of the construction enables a simpler proof of canonicity than is obtained with proof-irrelevant techniques, such as partial equivalence relations. Logical families are a type-theoretic version of the *categorical gluing* construction, in which a very rich semantic category (such as sets) is cut down to include

just the morphisms which track definable morphisms in $\mathcal{C}$;[7] one then uses the rich structure of the semantic category to obtain metatheoretic results about syntax (choosing $\mathcal{C}$ to be the initial model) without considering raw terms at any point in the process.

Usually, to prove canonicity one glues the initial model $\mathcal{C}$ together with **Set** along the global sections functor; this equips each context Γ with a family of sets $\Gamma^\bullet$ indexed in the *closing substitutions* for Γ. In order to prove canonicity for a cubical language like XTT, we will need a more sophisticated version of this construction, in which the global sections functor is replaced with something that determines substitutions which are closed with respect to term variables, but open with respect to dimension variables.

The split fibration $\mathcal{C} \xrightarrow{\mathsf{u}\triangleright} \square_+$ induces a functor $\square_+ \xrightarrow{\langle - \rangle} \mathcal{C}$ which takes every cube Ψ to the empty variable context over Ψ. This functor in turn induces a *nerve* construction $\mathcal{C} \xrightarrow{(\!|-|\!)} \widehat{\square}_+$, taking Γ to the cubical set $\mathcal{C}(\langle - \rangle, \Gamma)$.[8] Intuitively, this is the presheaf of substitutions which are closed with respect to term variables, but open with respect to dimension variables; when wearing $\widehat{\square}_+$-tinted glasses, these appear to be the closed substitutions.

This nerve construction extends to the presheaves of types and elements; we define the fiber of $(\!|\mathsf{Ty}_k|\!) : \widehat{\square}_+$ at Ψ to be the set $\mathsf{Ty}^k_{\mathcal{C}}(\langle \Psi \rangle)$; likewise, we define the fiber of $(\!|\mathsf{El}_k|\!) : \widehat{\int (\!|\mathsf{Ty}_k|\!)}$ at (Ψ, A) to be the set $\mathsf{El}_{\mathcal{C}}(\langle \Psi \rangle \vdash A)$. Internally to $\widehat{\square}_+$, we regard $(\!|\mathsf{El}_k|\!)$ as a dependent type over $(\!|\mathsf{Ty}_k|\!)$. We will then (abusively) write $(\!|A|\!)$ for the fiber of $(\!|\mathsf{El}_k|\!)$ determined by $A : (\!|\mathsf{Ty}_k|\!)$.

Category of cubical logical families

Gluing $\mathcal{C}$ together with $\widehat{\square}_+$ along $(\!|-|\!)$ gives us a category of *cubical logical families* $\mathcal{C}^\star$ whose objects are pairs $\overline{\Gamma} = (\Gamma, \Gamma^\bullet)$, with $\Gamma : \mathcal{C}$ and $\Gamma^\bullet$ a *dependent cubical set* over the cubical set $(\!|\Gamma|\!)$. In other words, $\Gamma^\bullet$ is a "Kripke logical family" on the substitutions $\langle \Psi \rangle \longrightarrow \Gamma$ which commutes with dimension substitutions $\Psi' \longrightarrow \Psi$. A morphism $\overline{\Delta} \longrightarrow \overline{\Gamma}$ is a substitution $\Delta \xrightarrow{\gamma} \Gamma$ together with a proof that γ preserves the logical family: that is, a closed element $\gamma^\bullet$ of the type $\prod_{\delta : (\!|\Delta|\!)} \Delta^\bullet(\delta) \to \Gamma^\bullet(\gamma^* \delta)$ in the internal type theory of $\widehat{\square}_+$. We write $\overline{\gamma}$ for the pair $(\gamma, \gamma^\bullet)$. We have a fibration $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}\triangleright} \mathcal{C}$ which merely projects Γ from $\overline{\Gamma} = (\Gamma, \Gamma^\bullet)$.

Glued type structure

Recall from Section 2.2 that we must model *closed* universes. Therefore, the standard presheaf universes which lift $\mathcal{V}_k$ to (weakly) classify all k-small presheaves are insufficient in our case; instead, we must equip each type with a *code* so that type-case is definable. Accordingly, we define for each $n \in \mathbb{N}$ an inductive cubical set $\mathfrak{U}^\bullet_n A : \mathcal{V}_{n+1}$ indexed over $A : (\!|\mathsf{Ty}_n|\!)$; internally to $\widehat{\square}_+$, the cubical set $\mathfrak{U}^\bullet_n A$ is the collection of *realizers* for the $\mathcal{C}$-type A. An imprecise but helpful analogy is to think of a realizer $\mathtt{A} : \mathfrak{U}^\bullet_n A$ as something like a whnf of A, with the caveat that $\mathtt{A}$ is an element of this inductively defined set, not a $\mathcal{C}$-type. Simultaneously, for each $\mathtt{A} : \mathfrak{U}^\bullet_n A$, we define a cubical family $\mathtt{A}^\circ : (\!|A|\!) \to \mathcal{V}_n$ of realizers of elements of A, with each $\mathtt{A}^\circ$ being the *logical family* of the $\mathcal{C}$-type A; finally, we also define realizers for coercion and composition by recursion on the realizers for types.[9] A fragment of

[7] The gluing construction is similar to realizability; the main difference is that in gluing, one considers collections of "realizers" which are *not* all drawn from a single computational domain.

[8] This construction is also called the *relative hom functor* by Fiore [30]; its use in logic originates in the study of definability for λ-calculus, characterizing the domains of discourse for Kripke logical predicates of varying arity [35]. We learned the connection to the abstract nerve construction in conversations with M. Fiore about his unpublished joint work with S. Awodey.

[9] It is important to note that we do *not* use large induction-recursion in $\widehat{\square}_+$ (to our knowledge, the construction of inductive-recursive definitions has not yet been lifted to presheaf toposes); instead, we

$$\frac{(j < n)}{\mathtt{univ}_j : \mathfrak{U}^\bullet_n \mathbf{U}_j} \qquad \frac{}{\mathtt{bool} : \mathfrak{U}^\bullet_n \mathbf{bool}}$$

$$\frac{\mathtt{A} : \mathfrak{U}^\bullet_n A \qquad \mathtt{B} : \prod_{M:\langle\!|A|\!\rangle} \mathtt{A}^\circ M \to \mathfrak{U}^\bullet_n(\langle \mathbf{id}, M\rangle^* B)}{\mathtt{pi}(\mathtt{A};\mathtt{B}) : \mathfrak{U}^\bullet_n \mathbf{\Pi}(A,B) \qquad \mathtt{sg}(\mathtt{A};\mathtt{B}) : \mathfrak{U}^\bullet_n \mathbf{\Sigma}(A,B)} \qquad \frac{\mathtt{A} : \prod_{i:\mathbb{I}} \mathfrak{U}^\bullet_n A_i \qquad \overrightarrow{N^\bullet_\varepsilon : \mathtt{A}(\varepsilon)^\circ N_\varepsilon}}{\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1) : \mathfrak{U}^\bullet_n \mathbf{Eq}(i.A_i, N_0, N_1)}$$

$$\begin{aligned}
\mathtt{univ}^\circ_n A &= \mathfrak{U}^\bullet_n A \\
\mathtt{bool}^\circ M &= (M = \mathbf{true}) + (M = \mathbf{false}) \\
\mathtt{pi}(\mathtt{A};\mathtt{B})^\circ M &= \textstyle\prod_{N:\langle\!|A|\!\rangle} \prod_{N^\bullet:\mathtt{A}^\bullet N} (\mathtt{B} N N^\bullet)^\circ \mathbf{app}(A,B,M,N) \\
\mathtt{sg}(\mathtt{A};\mathtt{B})^\circ M &= \textstyle\sum_{M^\bullet_0 : \mathtt{A}^\circ \mathbf{fst}(A,B,M)} (\mathtt{B}(\mathbf{fst}(A,B,M)) M^\bullet_0)^\circ \mathbf{snd}(A,B,M) \\
\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1)^\circ M &= \left\{ M^\bullet : \textstyle\prod_{i:\mathbb{I}} \mathtt{A}(i)^\circ \mathbf{papp}(i.A, M, i) \mid \overrightarrow{M^\bullet(\varepsilon) = N^\bullet_\varepsilon} \right\}
\end{aligned}$$

$$\begin{aligned}
[i.\mathtt{bool}]\downarrow^r_{r'} M^\bullet &= M^\bullet \\
[i.\mathtt{pi}(\mathtt{A};\mathtt{B})]\downarrow^r_{r'} M^\bullet &= \lambda N^\bullet.\, [i.\mathtt{B}([i.\mathtt{A}]\downarrow^{r'}_i N^\bullet)]\downarrow^r_{r'} M^\bullet\big([i.\mathtt{A}]\downarrow^{r'}_r N^\bullet\big) \\
[i.\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1)]\downarrow^r_{r'} M^\bullet &= \lambda k.\, [i.\mathtt{A}k]\downarrow^r_{r'} M^\bullet k\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow _.N^\bullet_\varepsilon}] \\
\mathtt{pi}(\mathtt{A};\mathtt{B})\downarrow^r_{r'} M^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i}] &= \lambda N^\bullet.\, \mathtt{B}N^\bullet \downarrow^r_{r'} M^\bullet N^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i N N^\bullet}] \\
\mathtt{eq}(\mathtt{A}; N^\bullet_0, N^\bullet_1)\downarrow^r_{r'} M^\bullet\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i}] &= \lambda j.\, \mathtt{A}j \downarrow^r_{r'} M^\bullet j\ [s \text{ with } \overrightarrow{\varepsilon \hookrightarrow i.M'^\bullet i j}] \\
&\vdots
\end{aligned}$$

Figure 2 The inductive definition of realizers $\mathfrak{U}^\bullet_n A : \mathcal{V}_{n+1}$ for types $A : \langle\!|\mathsf{Ty}_n|\!\rangle$ in $\widehat{\square}_+$; we also include a fragment of the realizers for Kan operations, which are also defined by recursion on the realizers for types. We write $\mathbb{I}$ for the representable presheaf $\mathbf{y}(i)$.

this definition is summarized in Figure 2. In the definition of $\mathtt{A}^\circ$ we freely make use of the internal type theory of $\widehat{\square}_+$. This not only exposes the underlying *logical relations* flavor of these definitions but simplifies a number of proofs (see the extended version of our paper).

From all this, we can define the cwf structure on $\mathcal{C}^\star$. We obtain a presheaf of types $\mathsf{Ty}_{\mathcal{C}^\star} : \widehat{\mathcal{C}^\star \times \mathbb{L}}$ by taking $\mathsf{Ty}^k_{\mathcal{C}^\star}(\overline{\Gamma})$ to be the set of pairs $\overline{A} = (A, A^\bullet)$ where $A \in \mathsf{Ty}^k_{\mathcal{C}}(\Gamma)$ and $A^\bullet$ is an element of the type $\prod_{\gamma:\langle\!|\Gamma|\!\rangle} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} \mathfrak{U}^\bullet_k(\gamma^* A)$ in the internal type theory of $\widehat{\square}_+$. To define the dependent presheaf of elements, we take $\mathsf{El}_{\mathcal{C}^\star}(\overline{\Gamma} \vdash \overline{A})$ to be the set of pairs $\overline{M} = (M, M^\bullet)$ where $M \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash A)$ and $M^\bullet$ is an element of the type $\prod_{\gamma:\langle\!|\Gamma|\!\rangle} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} (A^\bullet \gamma\gamma^\bullet)^\circ(\gamma^* M)$ in the internal type theory of $\widehat{\square}_+$. In this model, the context comprehension operation $\overline{\Gamma}.\overline{A}$ is defined as the pair $(\Gamma.A, (\overline{\Gamma}.\overline{A})^\bullet)$ where $(\overline{\Gamma}.\overline{A})^\bullet\langle\gamma, M\rangle$ is the cubical set $\sum_{\gamma^\bullet:\Gamma^\bullet(\gamma)} (A^\bullet \gamma\gamma^\bullet)^\circ(\gamma^* M)$; it is easy to see that we obtain realizers for the weakening substitution and the variable term.

▶ **Construction 3.2** (Dependent equality types in $\mathcal{C}^\star$)**.** Recall that we required a model of

model n object universes using the meta-universe $\mathcal{V}_{n+1}$. This is an instance of *small induction-recursion*, which can be translated into indexed inductive definitions which exist in every presheaf topos [31, 45].

XTT to have sufficient structure to interpret dependent equality types. Here, we discuss how to obtain the formation rule; the full construction can be found in the extended version of our paper. Suppose $\overline{A} \in \mathsf{Ty}^n_{\mathcal{C}^\star}(\hat{\imath}^*\overline{\Gamma})$ and elements $\overline{N_0}$ and $\overline{N_1}$ with $\overline{N}_\varepsilon \in \mathsf{El}_{\mathcal{C}^\star}(\overline{\Gamma} \vdash (\varepsilon/i)^\ddagger \overline{A})$. We wish to construct a type in $\mathsf{Ty}^n_{\mathcal{C}^\star}(\hat{\imath}^*\overline{\Gamma})$.

In $\mathcal{C}^\star$, such a type is a pair of a type $E \in \mathsf{Ty}^n_{\mathcal{C}}(\hat{\imath}^*\Gamma)$ from $\mathcal{C}$ with an element witnessing the logical family $\prod_{\gamma:(\!|\Gamma|\!)} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} \mathfrak{U}^\bullet_k(\gamma^* E)$. We will set the first component to the dependent equality type from $\mathcal{C}$ itself, namely $E = \mathbf{Eq}(i.A, N_0, N_1)$. For the second component, we wish to construct an element of $\prod_{\gamma:(\!|\Gamma|\!)} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} \mathfrak{U}^\bullet_k(\gamma^*\mathbf{Eq}(i.A, N_0, N_1))$. Inspecting the rules for $\mathfrak{U}^\bullet_k$ from Figure 2, there is only one choice: $E^\bullet = \lambda\gamma.\lambda\gamma^\bullet.\mathsf{eq}(A^\bullet\gamma\gamma^\bullet; N_0^\bullet\gamma\gamma^\bullet, N_1^\bullet\gamma\gamma^\bullet)$.

▶ **Construction 3.3** (Coercion in $\mathcal{C}^\star$)**.** The coercion structure on $\mathcal{C}^\star$ is constructed from the coercion structures on $\mathcal{C}$ and the coercion operator for codes from Figure 2.

Given a type $\overline{A} \in \mathsf{Ty}^n_{\overline{\Gamma}}(\hat{\imath}^*\overline{\Gamma})$ over Ψ, i, dimensions $r, r' \in \mathsf{Dim}_{\mathcal{C}}(\overline{\Gamma})$, and an element $\overline{M} \in \mathsf{El}_{\mathcal{C}}(\overline{\Gamma} \vdash (r/i)^\ddagger_{\hat{\imath}^*\overline{\Gamma}} \overline{A})$, we must construct an element of $\mathsf{El}_{\mathcal{C}}(\overline{\Gamma} \vdash (r'/i)^\ddagger_{\hat{\imath}^*\overline{\Gamma}} \overline{A})$. This element must be a pair of $N \in \mathsf{El}_{\mathcal{C}}(\Gamma \vdash (r'/i)^\ddagger_{\hat{\imath}^*\Gamma} A)$ and a term $N^\bullet : \prod_{\gamma:(\!|\Gamma|\!)} \prod_{\gamma^\bullet:\Gamma^\bullet(\gamma)} (A^\bullet\gamma\gamma^\bullet)^\circ(\gamma^* N)$. For the former, we rely on the coercion structure for $\mathcal{C}$ and pick $N = \mathbf{coe}^{r \rightsquigarrow r'}_{i.A} M$. For the latter, we use the coercion operation on codes defined in Figure 2 and choose $N^\bullet = \lambda\gamma.\lambda\gamma^\bullet.[i.A^\bullet\gamma\gamma^\bullet] \downarrow^r_{r'} M^\bullet\gamma\gamma^\bullet$.

It is routine to check that this coercion structure enjoys adjacency, regularity, and naturality once the corresponding properties are checked for the coercion operator on codes.

▶ **Theorem 3.4.** *$\mathcal{C}^\star$ is an XTT-algebra, and moreover, $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}} \mathcal{C}$ is a homomorphism of XTT-algebras.*

3.2 Canonicity theorem

Because $\mathcal{C}^\star$ is an XTT-algebra, we are now equipped to prove a canonicity theorem for the initial XTT-algebra $\mathcal{C}$: if M is an element of type **bool** in the empty context, then either $M = \mathbf{true}$ or $M = \mathbf{false}$, and not both.

Proof. We have $M \in \mathsf{El}_{\mathcal{C}}(\cdot \vdash \mathbf{bool})$, and therefore $[\![M]\!] \in \mathsf{El}_{\mathcal{C}^\star}(\cdot \vdash \overline{\mathbf{bool}})$. From this we obtain $N : \mathsf{El}_{\mathcal{C}}(\cdot \vdash \mathbf{bool})$ where $N = \pi_{\mathsf{syn}}[\![M]\!]$, and $N^\bullet \in \mathtt{bool}^\circ(N)$; by definition, $N^\bullet$ is either a proof that $N = \mathbf{true}$ or a proof that $N = \mathbf{false}$ (see Figure 2). Therefore, it suffices to observe that $\pi_{\mathsf{syn}}[\![M]\!] = M$; but this follows from the universal property of the initial XTT-algebra and the fact that $\mathcal{C}^\star \xrightarrow{\pi_{\mathsf{syn}}} \mathcal{C}$ is an XTT-homomorphism. Moreover, because the interpretation of **bool** in $\mathcal{C}^\star$ is disjoint, M cannot equal both **true** and **false**. ◀

References

1 Andreas Abel, Thierry Coquand, and Peter Dybjer. On the algebraic foundation of proof assistants for intuitionistic type theory. In Jacques Garrigue and Manuel V. Hermenegildo, editors, *Functional and Logic Programming*, pages 3–13. Springer Berlin Heidelberg, 2008.

2 Stuart Frazier Allen. A non-type-theoretic semantics for type-theoretic language, 1987.

3 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 412–420, July 1999. doi:10.1109/LICS.1999.782636.

4 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In *1st conference on Foundational Structures in Computation and Deduction (FSCD)*, 2016.

5 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles*

of Programming Languages, POPL '16, pages 18–29. ACM, 2016. URL: http://doi.acm.org/10.1145/2837614.2837638, doi:10.1145/2837614.2837638.

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

7 Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*, PLPV '07, pages 57–68. ACM, 2007.

8 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and models of cartesian cubical type theory. Preprint, February 2019. URL: https://github.com/dlicata335/cart-cube.

9 Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, Anders Mörtberg, and Jonathan Sterling. redtt: implementing Cartesian cubical type theory. Dagstuhl Seminar 18341: Formalization of Mathematics in Type Theory. URL: http://www.jonmsterling.com/pdfs/dagstuhl.pdf.

10 Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, and Jonathan Sterling. The RedPRL Proof Assistant (Invited Paper). In *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018.*, pages 1–10, 2018. doi:10.4204/EPTCS.274.1.

11 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9673, doi:10.4230/LIPIcs.CSL.2018.6.

12 Robert Atkey. Simplified observational type theory, 2018. URL: https://github.com/bobatkey/sott.

13 Steve Awodey. A cubical model of homotopy type theory. *Annals of Pure and Applied Logic*, 169(12):1270–1294, 2018. Logic Colloquium 2015. doi:10.1016/j.apal.2018.08.002.

14 Steven Awodey and Andrej Bauer. Propositions As [Types]. *J. Log. and Comput.*, 14(4):447–471, August 2004. URL: http://dx.doi.org/10.1093/logcom/14.4.447, doi:10.1093/logcom/14.4.447.

15 Andrej Bauer, Gaëtan Gilbert, Philipp Haselwarter, Matija Pretnar, and Christopher A. Stone. Design and implementation of the Andromeda proof assistant. TYPES, 2016. URL: http://www.types2016.uns.ac.rs/images/abstracts/bauer2.pdf.

16 Errett Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, 1967.

17 Andreas Blass. Words, free algebras, and coequalizers. *Fundamenta Mathematicae*, 117(2):117–160, 1983. URL: http://eudml.org/doc/211359.

18 Edwin Brady, James Chapman, Pierre-Évariste Dagand, Adam Gundry, Conor McBride, Peter Morris, Ulf Norell, and Nicolas Oury. An Epigram Implementation, February 2011.

19 John Cartmell. Generalised algebraic theories and contextual categories, January 1978.

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

21 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Undecidability of equality in the free locally cartesian closed category (extended version). *Logical Methods in Computer Science*, 13(4), 2017.

22 Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL):1:1–1:27, January 2019. URL: http://doi.acm.org/10.1145/3290314, doi:10.1145/3290314.

23 James Chapman, Fredrik Nordvall Forsberg, and Conor McBride. The Box of Delights (Cubical Observational Type Theory). Unpublished note, January 2018. URL: https://github.com/msp-strath/platypus/blob/master/January18/doc/CubicalOTT/CubicalOTT.pdf.

24 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. *IfCoLog Journal of Logics and their Applications*, 4(10):3127–3169, November 2017. URL: `http://www.collegepublications.co.uk/journals/ifcolog/?00019`.

25 R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., 1986.

26 Thierry Coquand. Universe of Bishop sets, February 2017. URL: `http://www.cse.chalmers.se/~coquand/bishop.pdf`.

27 Thierry Coquand. Canonicity and normalization for Dependent Type Theory. October 2018. URL: `https://arxiv.org/abs/1810.09367`, `arXiv:1810.09367`.

28 Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 255–264. ACM, 2018. URL: `http://doi.acm.org/10.1145/3209108.3209197`, `doi:10.1145/3209108.3209197`.

29 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs: International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers*, pages 120–134. Springer Berlin Heidelberg, 1996.

30 Marcelo Fiore. 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, pages 26–37. ACM, 2002. URL: `http://doi.acm.org/10.1145/571157.571161`, `doi:10.1145/571157.571161`.

31 Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications: 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, pages 156–172. Springer Berlin Heidelberg, 2013.

32 Martin Hofmann. Extensional concepts in intensional type theory, January 1995.

33 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, 1998.

34 Simon Huber. Canonicity for cubical type theory. *Journal of Automated Reasoning*, June 2018. URL: `https://doi.org/10.1007/s10817-018-9469-1`, `doi:10.1007/s10817-018-9469-1`.

35 Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 245–257. Springer Berlin Heidelberg, 1993.

36 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, January 2019. URL: `http://doi.acm.org/10.1145/3290315`, `doi:10.1145/3290315`.

37 Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of Univalent Foundations (after Voevodsky). Preprint, June 2016. URL: `https://arxiv.org/abs/1211.2851`, `arXiv:1211.2851`.

38 Alexei Kopylov. Type theoretical foundations for data structures, classes and objects, 2004.

39 Peter LeFanu Lumsdaine and Michael Shulman. Semantics of higher inductive types. 2017. URL: `https://arxiv.org/abs/1705.07088`, `arXiv:1705.07088`.

40 Jacob Lurie. *Higher Topos Theory*. Princeton University Press, 2009.

41 Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. Elsevier, 1975. `doi:10.1016/S0049-237X(08)71945-1`.

42 Per Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, August 1979. Published by North Holland, Amsterdam. 1982.

43 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

44 Conor McBride. Dependently typed functional programs and their proofs, 1999.

45 Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189–218, 2000.

46 Bengt Nordström, Kent Peterson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *International Series of Monographs on Computer Science*. Oxford University Press, 1990.

47 Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, LICS '01, pages 221–. IEEE Computer Society, 2001. URL: http://dl.acm.org/citation.cfm?id=871816.871845.

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

49 Jonathan Sterling. Algebraic type theory and universe hierarchies. December 2018. URL: http://www.jonmsterling.com/pdfs/algebraic-universes.pdf, arXiv:1902.08848.

50 Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results*. Birkhauser Boston Inc., 1991.

51 Thomas Streicher. Investigations into intensional type theory. Habilitationsschrift, Universität München, 1994.

52 Andrew Swan. Separating path and identity types in presheaf models of univalent type theory. 2018. URL: https://arxiv.org/abs/1808.00920, arXiv:https://arxiv.org/abs/1808.00920.

53 Paul Taylor. *Practical Foundations of Mathematics*. Cambridge studies in advanced mathematics. Cambridge University Press, 1999.

54 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, 2013.

55 Vladimir Voevodsky. A simple type system with two identity types. Talk at Andre Joyal's 70th birthday conference. (Slides available at https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS_slides.pdf), February 2013. URL: https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf.

56 Vladimir Voevodsky. Mathematical theory of type theories and the initiality conjecture, April 2016. Research proposal to the Templeton Foundation for 2016-2019, project description. URL: http://www.math.ias.edu/Voevodsky/other/Voevodsky%20Templeton%20proposal.pdf.

A The rules of XTT

In the following sections, we summarize the rules of XTT; we systematically omit obvious premises to equational rules and all congruence rules for judgmental equality, because these can be mechanically obtained from the typing rules. We will write $\mathcal{J}_=$ schematically to refer to an equality judgment $\Psi \mid \Gamma \vdash A = B\ \mathit{type}_k$ or $\Psi \mid \Gamma \vdash M = N : A$.

$$\textsc{cube/emp}\ \frac{}{\cdot\ \mathit{cube}_+} \qquad \textsc{cube/snoc/dim}\ \frac{\Psi\ \mathit{cube}_+}{\Psi, i\ \mathit{cube}_+} \qquad \textsc{cube/snoc/constr}\ \frac{\Psi\ \mathit{cube}_+ \quad \Psi \mid r, r'\ \mathit{dim}}{\Psi, r = r'\ \mathit{cube}_+}$$

$$\textsc{ctx/emp}\ \frac{}{\Psi \mid \cdot\ \mathit{ctx}} \qquad \textsc{ctx/snoc}\ \frac{\Psi \mid \Gamma\ \mathit{ctx} \quad \Psi \mid \Gamma \vdash A\ \mathit{type}_k}{\Psi \mid \Gamma, x : A\ \mathit{ctx}}$$

$$\textsc{constant}\ \frac{}{\Psi \mid \varepsilon\ \mathit{dim}} \qquad \textsc{variable}\ \frac{i \in \Psi}{\Psi \mid i\ \mathit{dim}} \qquad \textsc{reflexivity}\ \frac{}{\Psi \mid r = r\ \mathit{dim}}$$

$$\textsc{symm+trans}\ \frac{\Psi \mid r_0 = r_1\ \mathit{dim} \quad \Psi \mid r_1 = r_2\ \mathit{dim}}{\Psi \mid r_2 = r_0\ \mathit{dim}} \qquad \textsc{hyp}\ \frac{\Psi \ni r = r'}{\Psi \mid r = r'\ \mathit{dim}}$$

$$\textsc{variable}\ \frac{\Gamma \ni x : A}{\Psi \mid \Gamma \vdash x : A} \qquad \textsc{false constraint}\ \frac{\Psi \mid 0 = 1\ \mathit{dim}}{\Psi \mid \Gamma \vdash \mathcal{J}} \qquad \textsc{conversion}\ \frac{\Psi \mid \Gamma \vdash A_0 = A_1\ \mathit{type}_k \quad \Psi \mid \Gamma \vdash M : A_0}{\Psi \mid \Gamma \vdash M : A_1}$$

$$\textsc{boundary separation}\ \frac{\Psi \mid r\ \mathit{dim} \qquad \overbrace{\Psi, r = \varepsilon \mid \Gamma \vdash \mathcal{J}_=}}{\Psi \mid \Gamma \vdash \mathcal{J}_=}$$

$$\textsc{coercion}\ \frac{\begin{array}{c}\Psi \mid r, r'\ \mathit{dim} \\ \Psi, i \mid \Gamma \vdash A\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash M : A\langle r/i\rangle\end{array}}{\begin{array}{l}\Psi \mid \Gamma \vdash [i.A]\downarrow^r_{r'} M : A\langle r'/i\rangle \\ \Psi \mid \Gamma \vdash [i.A]\downarrow^r_{r} M = M : A\langle r/i\rangle\end{array}} \qquad \textsc{coercion regularity}\ \frac{\Psi, j, j' \mid \Gamma \vdash A\langle j/i\rangle = A\langle j'/i\rangle\ \mathit{type}_k}{\Psi \mid \Gamma \vdash [i.A]\downarrow^r_{r'} M = M : A\langle r'/i\rangle}$$

$$\textsc{composition}\ \frac{\Psi \mid r, r', s\ \mathit{dim} \quad \Psi \mid \Gamma \vdash M : A \quad \overbrace{\Psi, j, s = \varepsilon \mid \Gamma \vdash N_\varepsilon : A\ [j = r \hookrightarrow M]}}{\begin{array}{l}\Psi \mid \Gamma \vdash A\downarrow^r_{r'} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow j.N_\varepsilon}] : A \\ \Psi \mid \Gamma \vdash A\downarrow^r_{r} M\ [s\ \mathsf{with}\ \overbrace{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A \\ \Psi \mid \Gamma \vdash A\downarrow^r_{r'} M\ [\varepsilon\ \mathsf{with}\ \overbrace{\varepsilon' \hookrightarrow j.N_{\varepsilon'}}] = N_\varepsilon\langle r'/j\rangle : A\end{array}}$$

$$\textsc{lift formation}\ \frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \quad k \le l}{\begin{array}{l}\Psi \mid \Gamma \vdash \Uparrow^l_k A\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \Uparrow^k_k A = A\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^m_l \Uparrow^l_k A = \Uparrow^m_k A\ \mathit{type}_k\end{array}} \qquad \textsc{lift element}\ \frac{\Psi \mid \Gamma \vdash M : A}{\Psi \mid \Gamma \vdash M : \Uparrow^l_k A} \qquad \textsc{lift hypothesis}\ \frac{\Psi \mid \Gamma, x : A \vdash \mathcal{J}}{\Psi \mid \Gamma, x : \Uparrow^l_k A \vdash \mathcal{J}}$$

$$\textsc{lift coercion}\ \frac{}{\Psi \mid \Gamma \vdash [i.\Uparrow^l_k A]\downarrow^r_{r'} M = [i.A]\downarrow^r_{r'} M : \Uparrow^l_k A\langle r'/i\rangle}$$

PAIR FORMATION, LIFTING

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \quad \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k}{\begin{array}{l}\Psi \mid \Gamma \vdash (x : A) \times B\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k (x : A) \times B = (x : \Uparrow^l_k A) \times \Uparrow^l_k B\ \mathit{type}_l\end{array}}$$

PAIR INTRODUCTION

$$\frac{\begin{array}{l}\Psi \mid \Gamma \vdash A\ \mathit{type}_k \\ \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash M : A \\ \Psi \mid \Gamma \vdash N : B[M/x]\end{array}}{\Psi \mid \Gamma \vdash \langle M, N \rangle : (x : A) \times B}$$

PAIR ELIMINATION

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \quad \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k \quad \Psi \mid \Gamma \vdash M : (x : A) \times B}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{fst}_{x:A.B}(M) : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}_{x:A.B}(M) : B[\mathsf{fst}(M)/x] \\ \Psi \mid \Gamma \vdash \mathsf{fst}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M) = \mathsf{fst}_{x:A.B}(M) : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M) = \mathsf{snd}_{x:A.B}(M) : B[\mathsf{fst}(M)/x]\end{array}}$$

PAIR COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq [i.B[[i.A] \downarrow^r_i \mathsf{fst}(M)/x]] \downarrow^r_{r'} \mathsf{snd}(M)}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{fst}(\langle M, N \rangle) = M : A \\ \Psi \mid \Gamma \vdash \mathsf{snd}(\langle M, N \rangle) = N : B[M/x] \\ \Psi \mid \Gamma \vdash \mathsf{fst}([i.(x : A) \times B] \downarrow^r_{r'} M) = [i.A] \downarrow^r_{r'} \mathsf{fst}(M) : A\langle r'/i \rangle \\ \Psi \mid \Gamma \vdash \mathsf{snd}([i.(x : A) \times B] \downarrow^r_{r'} M) = H : B\langle r'/i \rangle[[i.A] \downarrow^r_{r'} \mathsf{fst}(M)/x]\end{array}}$$

PAIR UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \langle \mathsf{fst}(M), \mathsf{snd}(M) \rangle : (x : A) \times B}$$

FUNCTION FORMATION, LIFTING

$$\frac{\begin{array}{c}\Psi \mid \Gamma \vdash A\ \mathit{type}_k \\ \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k\end{array}}{\begin{array}{l}\Psi \mid \Gamma \vdash (x : A) \to B\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k (x : A) \to B = (x : \Uparrow^l_k A) \to \Uparrow^l_k B\ \mathit{type}_l\end{array}}$$

FUNCTION INTRODUCTION

$$\frac{\begin{array}{l}\Psi \mid \Gamma \vdash A\ \mathit{type}_k \\ \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k \\ \Psi \mid \Gamma, x : A \vdash M : B\end{array}}{\Psi \mid \Gamma \vdash \lambda x.M : (x : A) \to B}$$

FUNCTION ELIMINATION

$$\frac{\begin{array}{c}\Psi \mid \Gamma \vdash A\ \mathit{type}_k \quad \Psi \mid \Gamma, x : A \vdash B\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash M : (x : A) \to B \quad \Psi \mid \Gamma \vdash N : A\end{array}}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{app}_{x:A.B}(M, N) : B[N/x] \\ \Psi \mid \Gamma \vdash \mathsf{app}_{x:\Uparrow^l_k A.\Uparrow^l_k B}(M, N) = \mathsf{app}_{x:A.B}(M, N) : x[N/B]\end{array}}$$

FUNCTION COMPUTATION

$$\frac{\Psi, i \mid \Gamma \vdash \widetilde{N}[i] \triangleq [i.A] \downarrow^{r'}_i N}{\begin{array}{l}\Psi \mid \Gamma \vdash (\lambda x.M)(N) = M[N/x] : B[N/x] \\ \Psi \mid \Gamma \vdash ([i.(x : A) \to B] \downarrow^r_{r'} M)(N) = [i.B[\widetilde{N}[i]/x]] \downarrow^r_{r'} M(\widetilde{N}[r]) : C\end{array}}$$

FUNCTION UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \lambda x.M(x) : (x : A) \to B}$$

EQUALITY FORMATION,LIFTING

$$\frac{\Psi, i \mid \Gamma \vdash A\ \mathit{type}_k \quad \overrightarrow{\Psi, i, i = \varepsilon \mid \Gamma \vdash N_\varepsilon : A}}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{Eq}_{i.A}(N_0, N_1)\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k \mathsf{Eq}_{i.A}(N_0, N_1) = \mathsf{Eq}_{i.\Uparrow^l_k A}(N_0, N_1)\ \mathit{type}_l\end{array}}$$

EQUALITY INTRODUCTION

$$\frac{\Psi, i \mid \Gamma \vdash M : A\ [\overrightarrow{i = \varepsilon \hookrightarrow N_\varepsilon}]}{\Psi \mid \Gamma \vdash \lambda i.M : \mathsf{Eq}_{i.A}(N_0, N_1)}$$

EQUALITY ELIMINATION

$$\frac{\Psi \mid r\ \mathit{dim} \quad \Psi, i \mid \Gamma \vdash A\ \mathit{type}_k \quad \overrightarrow{\Psi, i, i = \varepsilon \mid \Gamma \vdash N_\varepsilon : A} \quad \Psi \mid \Gamma \vdash M : \mathsf{Eq}_{i.A}(N_0, N_1)}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{app}_{i.A}(M, r) : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash \mathsf{app}_{i.\Uparrow^l_k A}(M, r) = \mathsf{app}_{i.A}(M, r) : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash \mathsf{app}_{i.A}(M, \varepsilon) = N_\varepsilon : A\langle \varepsilon/i\rangle\end{array}}$$

EQUALITY COMPUTATION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash (\lambda i.M)(r) = M\langle r/i\rangle : A\langle r/i\rangle \\ \Psi \mid \Gamma \vdash ([j.\mathsf{Eq}_{i.A}(N_0, N_1)] \downarrow^r_{r'} P)(s) = [j.A\langle s/i\rangle] \downarrow^r_{r'} P(s)\ [s\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : A\langle r', s/j, i\rangle\end{array}}$$

EQUALITY UNICITY

$$\frac{}{\Psi \mid \Gamma \vdash M = \lambda i.M(i) : \mathsf{Eq}_{i.A}(N_0, N_1)}$$

BOOLEAN FORMATION, LIFTING, INTRODUCTION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{bool}\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash \Uparrow^l_k \mathsf{bool} = \mathsf{bool}\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \mathsf{true} : \mathsf{bool} \\ \Psi \mid \Gamma \vdash \mathsf{false} : \mathsf{bool}\end{array}}$$

BOOLEAN ELIMINATION

$$\frac{\begin{array}{l}\Psi \mid \Gamma, x : \mathsf{bool} \vdash C\ \mathit{type}_k \\ \Psi \mid \Gamma \vdash M : \mathsf{bool} \\ \Psi \mid \Gamma \vdash N_0 : C[\mathsf{true}/x] \\ \Psi \mid \Gamma \vdash N_1 : C[\mathsf{false}/x]\end{array}}{\Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(M; N_0, N_1) : C[M/x]}$$

BOOLEAN ELIMINATION LIFTING

$$\frac{}{\Psi \mid \Gamma \vdash \mathsf{if}_{x.\Uparrow^l_k C}(M; N_0, N_1) = \mathsf{if}_{x.C}(M; N_0, N_1) : \Uparrow^l_k C[M/x]}$$

BOOLEAN COMPUTATION

$$\frac{}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(\mathsf{true}; N_0, N_1) = N_0 : C[\mathsf{true}/x] \\ \Psi \mid \Gamma \vdash \mathsf{if}_{x.C}(\mathsf{false}; N_0, N_1) = N_1 : C[\mathsf{false}/x]\end{array}}$$

UNIVERSE FORMATION, LIFTING

$$\frac{k < l}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathcal{U}_k\ \mathit{type}_l \\ \Psi \mid \Gamma \vdash \Uparrow^m_l \mathcal{U}_k = \mathcal{U}_k\ \mathit{type}_m\end{array}}$$

UNIVERSE ELEMENTS

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k}{\Psi \mid \Gamma \vdash A : \mathcal{U}_k}$$

UNIVERSE EQUALITY

$$\frac{\Psi \mid \Gamma \vdash A_0 = A_1\ \mathit{type}_k}{\Psi \mid \Gamma \vdash A_0 = A_1 : \mathcal{U}_k}$$

TYPE-CASE

$$\frac{\begin{array}{l}\Psi \mid \Gamma \vdash C\ \mathit{type}_l \\ \Psi \mid \Gamma, x : \mathcal{U}_k, y : x \to \mathcal{U}_k \vdash M_\Pi : C \\ \Psi \mid \Gamma, x : \mathcal{U}_k, y : x \to \mathcal{U}_k \vdash M_\Sigma : C \\ \Psi \mid \Gamma, x_0 : \mathcal{U}_k, x_1 : \mathcal{U}_k, x^= : \mathsf{Eq}_{i.\mathcal{U}_k}(x_0, x_1), y_0 : x_0, y_1 : x_1 \vdash M_{\mathsf{Eq}} : C \\ \Psi \mid \Gamma \vdash M_{\mathsf{bool}} : C \\ \Psi \mid \Gamma \vdash M_{\mathcal{U}} : C\end{array}}{\Psi \mid \Gamma \vdash \mathsf{tycase}\ X\ [\Pi_x y \mapsto M_\Pi \mid \Sigma_x y \mapsto M_\Sigma \mid \mathsf{Eq}_{x_0, x_1, x^=}(y_0, y_1) \mapsto M_{\mathsf{Eq}} \mid \mathsf{bool} \mapsto M_{\mathsf{bool}} \mid \mathcal{U} \mapsto M_{\mathcal{U}}] : C}$$

TYPE-CASE COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H_{\mathbf{Eq}} \triangleq M[A\langle 0/i\rangle, A\langle 1/i\rangle, \lambda i.A, N_0, N_1/x_0, x_1, x^{=}, y_0, y_1]}{\begin{array}{l}\Psi \mid \Gamma \vdash \mathsf{tycase}\ ((z : A) \to B)\ [\Pi_x y \mapsto M \mid \ldots] = M[A, \lambda z.B/x, y] : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ ((z : A) \times B)\ [\ldots \mid \Sigma_x y \mapsto M \mid \ldots] = M[A, \lambda z.B/x, y] : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ \mathsf{bool}\ [\ldots \mid \mathsf{bool} \mapsto M \mid \ldots] = M : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ \mathcal{U}_{k'}\ [\ldots \mid \mathcal{U} \mapsto M] = M : C \\ \Psi \mid \Gamma \vdash \mathsf{tycase}\ (\mathsf{Eq}_{i.A}(N_0, N_1))\ [\ldots \mid \mathsf{Eq}_{x_0,x_1,x^{=}}(y_0, y_1) \mapsto M \mid \ldots] = H_{\mathbf{Eq}} : C\end{array}}$$

TYPE BOUNDARY

$$\frac{\Psi \mid \Gamma \vdash M : A \qquad \overrightarrow{\Psi, \xi \mid \Gamma \vdash M = N : A}}{\Psi \mid \Gamma \vdash M : A\ [\overrightarrow{\xi \hookrightarrow N}]}$$

TERM BOUNDARY

$$\frac{\Psi \mid \Gamma \vdash A\ \mathit{type}_k \qquad \overrightarrow{\Psi, \xi \mid \Gamma \vdash A = B\ \mathit{type}_k}}{\Psi \mid \Gamma \vdash A\ \mathit{type}_k\ [\overrightarrow{\xi \hookrightarrow B}]}$$

A.1 Derivable Rules

Numerous additional rules about compositions are *derivable* by exploiting boundary separation. In previous presentations of cubical type theory (which did not enjoy the unicity of equality proofs), it was necessary to include β-rules for compositions explicitly.

COMPOSITION REGULARITY

$$\frac{\overrightarrow{\Psi, j_0, j_1, i = \varepsilon \mid \Gamma \vdash N_\varepsilon\langle j_0/j\rangle = N_\varepsilon\langle j_1/j\rangle : A}}{\Psi \mid \Gamma \vdash A \downarrow^r_{r'} M\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A}$$

HETEROGENEOUS COMPOSITION

$$\frac{\Psi \mid r, r', s\ \mathit{dim} \quad \Psi \mid \Gamma \vdash M : A\langle r/j\rangle \quad \overrightarrow{\Psi, j, s = \varepsilon \mid \Gamma \vdash N_\varepsilon : A\ [j = r \hookrightarrow M]}}{\begin{array}{l}\Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r'} M\ [s\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : A\langle r'/j\rangle \\ \Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r} M\ [s\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = M : A\langle r/j\rangle \\ \Psi \mid \Gamma \vdash [j.A] \downarrow^r_{r'} M\ [\varepsilon\ \mathsf{with}\ \overrightarrow{\varepsilon' \hookrightarrow j.N_{\varepsilon'}}] = N_\varepsilon\langle r'/j\rangle : A\langle r'/j\rangle\end{array}}$$

LIFT COMPOSITION

$$\frac{}{\Psi \mid \Gamma \vdash \Uparrow^l_k A \downarrow^r_{r'} M\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] = A \downarrow^r_{r'} M\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}] : \Uparrow^l_k A}$$

LIFT TYPE COMPOSITION

$$\frac{}{\Psi \mid \Gamma \vdash \mathcal{U}_l \downarrow^r_{r'} \Uparrow^l_k A\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.\Uparrow^l_k B_\varepsilon}] = \Uparrow^l_k \mathcal{U}_k \downarrow^r_{r'} A\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon}]\ \mathit{type}_k}$$

PAIR COMPOSITION COMPUTATION (1)

$$\frac{\Psi \mid \Gamma \vdash H \triangleq A \downarrow^r_{r'} \mathsf{fst}(M)\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{fst}(N_\varepsilon)}]}{\Psi \mid \Gamma \vdash \mathsf{fst}((x : A) \times B \downarrow^r_{r'} M\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]) = H : A}$$

PAIR COMPOSITION COMPUTATION (2)

$$\frac{\begin{array}{l}\Psi, k \mid \Gamma \vdash \widetilde{M_1}[k] \triangleq A \downarrow^r_k \mathsf{fst}(M)\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{fst}(N_\varepsilon)}] \\ \Psi \mid \Gamma \vdash H \triangleq [k.B[\widetilde{M_1}[k]/x]] \downarrow^r_{r'} \mathsf{snd}(M)\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{snd}(N_\varepsilon)}]\end{array}}{\Psi \mid \Gamma \vdash \mathsf{snd}((x : A) \times B \downarrow^r_{r'} M\ [i\ \mathsf{with}\ \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]) = H : B[\widetilde{M_1}[r']/x]}$$

PAIR TYPE COMPOSITION

$$\frac{\begin{array}{c}\Psi, k \mid \Gamma \vdash \widetilde{A}[k] \triangleq \mathcal{U}_k \downarrow^r_k A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi, j \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{x}[j] \triangleq [k.\widetilde{A}[k]] \downarrow^{r'}_j x \\ \Psi \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{B} \triangleq \mathcal{U}_k \downarrow^r_{r'} B[\widetilde{x}[r]/x]\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon[\widetilde{x}[j]/x]}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow^r_{r'} ((x : A) \times B)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.(x : A_\varepsilon) \times B_\varepsilon}] = (x : \widetilde{A}[r']) \times \widetilde{B}\ \mathit{type}_l}$$

FUNCTION COMPOSITION COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq B[N/x] \downarrow^r_{r'} M(N)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon(N)}]}{\Psi \mid \Gamma \vdash ((x : A) \to B \downarrow^r_{r'} M\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon}])(N) = H : (x : A) \to B}$$

FUNCTION TYPE COMPOSITION

$$\frac{\begin{array}{c}\Psi, k \mid \Gamma \vdash \widetilde{A}[k] \triangleq \mathcal{U}_k \downarrow^r_k A\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi, j \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{x}[j] \triangleq [k.\widetilde{A}[k]] \downarrow^{r'}_j x \\ \Psi \mid \Gamma, x : \widetilde{A}[r'] \vdash \widetilde{B} \triangleq \mathcal{U}_k \downarrow^r_{r'} B[\widetilde{x}[r]/x]\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.B_\varepsilon[\widetilde{x}[j]/x]}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow^r_{r'} ((x : A) \to B)\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.(x : A_\varepsilon) \to B_\varepsilon}] = (x : \widetilde{A}[r']) \to \widetilde{B}\ \mathit{type}_l}$$

EQUALITY COMPOSITION COMPUTATION

$$\frac{\Psi \mid \Gamma \vdash H \triangleq A\langle s/i\rangle \downarrow^r_{r'} P(s)\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.Q_\varepsilon(s)}]}{\Psi \mid \Gamma \vdash (\mathsf{Eq}_{i.A}(N_0, N_1) \downarrow^r_{r'} P\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.Q_\varepsilon}])(s) = H : A\langle s/i\rangle}$$

EQUALITY TYPE COMPOSITION

$$\frac{\begin{array}{c}\Psi, j, i \mid \Gamma \vdash \widetilde{A}[j, i] \triangleq \mathcal{U}_k \downarrow^r_j A\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.A_\varepsilon}] \\ \Psi \mid \Gamma \vdash \widetilde{M} \triangleq [j.\widetilde{A}[j, r]] \downarrow^r_{r'} M\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.M_\varepsilon}] \\ \Psi \mid \Gamma \vdash \widetilde{N} \triangleq [j.\widetilde{A}[j, r']] \downarrow^r_{r'} N\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.N_\varepsilon}]\end{array}}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow^r_{r'} \mathsf{Eq}_{i.A}(M, N)\ [k \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathsf{Eq}_{i.A_\varepsilon}(M_\varepsilon, N_\varepsilon)}] = \mathsf{Eq}_{i.\widetilde{A}[r',i]}(\widetilde{M}, \widetilde{N})\ \mathit{type}_l}$$

BASE TYPE COMPOSITION

$$\frac{(\mathfrak{b} \in \{\mathsf{bool}, \mathcal{U}_{k'}\})}{\Psi \mid \Gamma \vdash \mathcal{U}_k \downarrow^r_{r'} \mathfrak{b}\ [i \text{ with } \overrightarrow{\varepsilon \hookrightarrow j.\mathfrak{b}}] = \mathfrak{b}\ \mathit{type}_l}$$