Canonicity for 2-Dimensional Type Theory

Daniel R. Licata* Robert Harper *

Carnegie Mellon University

{drl,rwh}@cs.cmu.edu

Abstract

Higher-dimensional dependent type theory enriches conventional one-dimensional dependent type theory with additional structure expressing equivalence of elements of a type. This structure may be employed in a variety of ways to capture rather coarse identifications of elements, such as a universe of sets considered modulo isomorphism. Equivalence must be respected by all families of types and terms, as witnessed computationally by a type-generic program. Higher-dimensional type theory has applications to code reuse for dependently typed programming, and to the formalization of mathematics. In this paper, we develop a novel judgemental formulation of a two-dimensional type theory, which enjoys a canonicity property: a closed term of boolean type is definitionally equal to true or false. Canonicity is a necessary condition for a computational interpretation of type theory as a programming language, and does not hold for existing axiomatic presentations of higher-dimensional type theory. The method of proof is a generalization of the NuPRL semantics, interpreting types as syntactic groupoids rather than equivalence relations.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages

General Terms Languages, Theory

1. Introduction

A growing body of work [4, 8, 9, 12, 14, 15, 22–24] on intensional dependent type theory [11, 16, 18] elucidates the latent higher-dimensional structure given by the Martin-Löf intensional identity type. The identity type, $\mathsf{Id}_A\ M\ N$, is the type of evidence for equivalence of the objects M and N of type A. The elimination rule for the identity type ensures that type families indexed by A respect equivalence at A: if F is an A-indexed family of types, and $\alpha : \mathsf{Id}_A\ M\ N$, then $F[M]$ and $F[N]$ are isomorphic types. Similarly, families of objects indexed by A must respect equivalence at A: mappings are functional in their domain, in that application to equivalent elements yields equivalent results. Objects of identity type are themselves subject to higher equivalences inhabiting the iterated identity type $\mathsf{Id}_{\mathsf{Id}_A\ M\ N}\ \alpha\ \beta$, and so on, indefinitely. This higher-dimensional structure is the key to establishing surprisingly close connections between type theory, category theory (interpreting types as higher-dimensional groupoids, and equivalences as morphisms), and homotopy theory (interpreting types as topological spaces, and equivalences as paths). The interplay between these viewpoints is a source of much current investigation.

However, experience with intensional type theory, both as a programming formalism and as a language for formalizing mathematics, suggests that the general principles of equivalence originally proposed by Martin-Löf are too restrictive. For example, equivalence at function types amounts to definitional equality, whereas in mathematical practice one tends to identify functions extensionally—two functions are equal iff they take equal arguments to equal results. Equivalence of types themselves is similarly restrictive. Formally, this is typically considered as equivalence for members of a *universe*, which is a type whose elements themselves determine types. The standard account of equivalence for universes amounts to equivalence of presentations: two types are equivalent if they are written the same way. However, it is common mathematical practice to identify *isomorphic* sets—to treat two sets as interchangable when there are functions back and forth that compose to the identity.

This motivates extensions of intensional type theory with coarser notions of equivalence: Altenkirch [2], Altenkirch et al. [3], Hofmann [10] investigate functional extensionality. Voevodsky's *univalence axiom* [23] goes further by treating elements of universes as equivalent whenever the spaces they determine are weakly equivalent in the sense of homotopy theory. A particular case of univalence, considered here, arises when considering the universe set of extensional sets: two sets are deemed equivalent iff they are isomorphic. This amounts to introducing a new canonical form, $\mathsf{iso}(f, g, \alpha, \beta)$, of the identity type $\mathsf{Id}_{\mathsf{set}}\ A\ B$, given by functions $f : A \to B$ and $g : B \to A$ and evidence α and β witnessing their inverse relationship to one another. However, this represents a more significant departure from the traditional view of equivalence than might at first be apparent. Since two sets can be isomorphic in many ways, isomorphism must be treated as a computationally relevant *structure* imposed on two sets, rather than as a computationally irrelevant *property* of them. For example, the set 2 of booleans is isomorphic to itself both by the identity function and by the negation function, which swaps the two booleans. Thus, the type $\mathsf{Id}_{\mathsf{set}}\ 2\ 2$ is not a proof-irrelevant proposition, but has real computational content. A *higher-dimensional type* is one, such as set, whose identity type has computational content.

Higher-dimensional type theory has several applications in programming and formal mathematics. In programming, postulating that equivalence of sets is isomorphism has the consequence that *all families of types respect isomorphism*. For example, consider

* This research was sponsored in part by the National Science Foundation under grant number CCF-0702381 and CCF-1116703 and by the Microsoft-Carnegie Mellon Center for Computational Thinking. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.
Copyright © 2012 ACM 978-1-4503-1083-3/12/01...$10.00

the family of types $\mathsf{Mon}[A]$, where $A : \mathsf{set}$, representing monoids on A:

$$\begin{array}{l}\mathsf{Mon}[A] = \\ \quad \Sigma\, m : A \to A \to A. \\ \quad \Sigma\, u : A. \\ \quad \Sigma\, assoc : (\Pi x, y, z : A. \mathsf{Id}\ (m\ (m\ x\ y)\ z)\ (m\ x\ (m\ y\ z))). \\ \quad \Sigma\, unitl : (\Pi x : A. \mathsf{Id}\ (m\ u\ x)\ x). \\ \quad \Sigma\, unitr : (\Pi x : A. \mathsf{Id}\ (m\ x\ u)\ x). 1\end{array}$$

A monoid consists of a multiplication operation m, a unit operation u, together with proofs that these satisfy the monoid laws. Monoid structures are useful for a parallel reduce operator, which schedules computation in an arbitrary way while producing a deterministic result. In current dependent type theories, a programmer must show that each individual type family, such as Mon, respects isomorphism—that $A \cong B$ entails $\mathsf{Mon}[A] \cong \mathsf{Mon}[B]$. This is particularly laborious for dependently typed programming, where isomorphisms arise frequently from type refinements; e.g. if the type $\mathsf{vec}[n : \mathbb{N}]$ classifies vectors of length n, then $\Sigma\, n{:}\mathbb{N}.\, \mathsf{vec}[n]$ is isomorphic to the unrefined type list. Moreover, it is unnecessary, as there is in fact no way to define a family that does *not* respect isomorphism. In contrast, in higher-dimensional type theory, the elimination rule for the identity type induces a monoid structure on B from a monoid structure on A, and shows that these structures are isomorphic: using substitutivity of Id-types $\mathsf{subst}_C : \mathsf{Id}_A\ M\ N \to C[M] \to C[N]$, given any $\mathsf{iso}(f, g, \alpha, \beta) : \mathsf{Id}_{\mathsf{set}}\ A\ B$, the term

$$\mathsf{subst}_{\mathsf{Mon}}(\mathsf{iso}(f, g, \alpha, \beta)) : \mathsf{Mon}[A] \to \mathsf{Mon}[B]$$

induces an isomorphism between monoid structures.

The same idea also has applications to the formalization of mathematics: a proof assistant based on higher-dimensional type theory supports the common informal mathematical practice of identifying isomorphic sets. Another application is to the formalization of homotopy theory in type theory [1] in which types are interpreted as topological spaces, and equivalences as paths in the space [4]. Under this correspondence a space may be inductively defined by giving generators for its points and for its paths. For example, the interval, I, may be specified by the endpoints $0, 1 : I$ and the path $seg : \mathsf{Id}_I\ 0\ 1$ between them. Similarly, the circle S^1 may be specified as having, say, two points, $0, 1 : S^1$, and two paths $a : \mathsf{Id}_{S^1}\ 0\ 1$ and $b : \mathsf{Id}_{S^1}\ 1\ 0$. The higher-dimensional structure of type theory ensures, for example, that a family of types indexed by a space induces an action that transports objects along paths in the space. These definitions have already been used to formalize some basic results in algebraic topology [1]—all higher homotopy groups are abelian; the fundamental group of the circle is isomorphic to $\mathbb{Z}$—and suggest a new, type-theoretic approach towards more sophisticated problems, like determining the homotopy groups of the spheres.

A fundamental question that arises in higher-dimensional type theory concerns the computational behavior of identity elimination. For example, when we use $\mathsf{subst}_{\mathsf{Mon}}(\mathsf{iso}(f, g, \alpha, \beta))$ to construct a function $\mathsf{Mon}[A] \to \mathsf{Mon}[B]$, what function does it construct? The answer lies in the functorial action of the dependent type constructors, as expressed by Hofmann and Streicher [12]. In this example, the action of Mon sends the unit to to its image under f, wraps multiplication with f and g to transfer back and forth between the two algebras, and uses the cancellation of inverses to validate the laws. Thus, from a programming standpoint, the main benefit of higher-dimensional type theory is to equip every type and term in the language with their functorial actions, as a generic program. This allows programmers to work up to isomorphism and other such structures, facilitating code reuse for dependently typed programs. Topologically, these programs witness the process of transporting points along paths in spaces.

However, this computational behavior is latent in current presentations of higher-dimensional type theory, which extend the canonical members of the identity type, without extending the computation rules for its elimination form. The standard computation rule for subst says that $\mathsf{subst}_\mathsf{C}(\mathsf{refl})$ reduces to the identity function. Higher dimensional types add new canonical forms of equivalence, such as $\mathsf{iso}(f, g, \alpha, \beta)$, but, in current presentations, no new computation rules. This runs afoul of the computational interpretation of type theory, which demands that the elimination forms for a type be post-inverse to the introductory forms. For example, the term $\mathsf{subst}_{\mathsf{Mon}}(\mathsf{iso}(f, g, \alpha, \beta))$ "gets stuck", even though it is well-typed! In logical terms, the property of *canonicity* [10] of observable types fails: there are closed terms of, say, boolean type that are neither equal to true nor equal to false. A weaker notion of canonicity-up-to-equivalence has been conjectured, but not yet proved [23].

In this paper, we define a *two*-dimensional univalent type theory (2TT), and prove that it does enjoy canonicity. 2TT is based on a *judgemental*, rather than *propositional*, account of its higher-dimensional structure. The key innovation is to consider a judgement of the form $\Gamma \vdash \alpha : M \simeq_A N$ stating that α is evidence for the equivalence of M and N as objects of type A. The computational content of such equivalences is made explicit by judgemental operations that ensure that families of types and objects respect equivalence. Function extensionality and univalence are naturally accommodated as rules of equivalence, and the induced action accounts for the computational content of the equivalence proofs. As we show in Section 4, the identity type is reformulated simply as an internalization of the equivalence judgement (that is, as a hom-type). The Martin-Löf elimination rule is derivable from this interpretation using the judgemental apparatus for equivalence.

The judgemental formulation, described in Section 2, is adapted from earlier work [14] on the directed (non-symmetric) case. We require that the forms of evidence include identity ($\mathsf{refl}_M : M \simeq_A M$), composition (if $\alpha : M \simeq_A N$ and $\beta : N \simeq_A P$ then $\beta \circ \alpha : M \simeq_A P$), and inverse (if $\alpha : M \simeq_A N$ then $\alpha^{-1} : N \simeq_A M$), corresponding to the reflexivity, symmetry, and transitivity properties of an equivalence relation. Moreover, these operations must satisfy the unit, associativity, and inverse laws (on the nose, in the two-dimensional case) for a *groupoid* [12], a category in which all maps are invertible. Preservation of equivalence amounts to the requirement that families of types be *functorial* in their indices: if $\alpha : M \simeq_A N$, and F is an A-indexed family of types, then $\mathsf{map}_{x:A.F}\ \alpha\ : F[M] \to F[N]$ is the *action* of F on evidence for the equivalence of indices. This action must preserve identities (the null action) and composition (the composite action), and hence inverses (the reverse action).

2TT has a simple interpretation in the category of groupoids: each type is interpreted as a groupoid, with objects modeling terms, and maps modeling equivalences—in essence, 2TT constructs a type-theoretic syntax out of the groupoid interpretation of type theory [12]. To justify that 2TT solves the aforementioned problems with the computational interpretation of higher-dimensional type theory, we prove canonicity in Section 3, showing that a closed term of boolean type is definitionally equal to either true or false. The proof may be seen as a generalization of the semantics of the NuPRL type theory [5] (in turn based on Tait's reducibility method): in our proof, types are interpreted as groupoids [12], rather than as equivalence relations.

2. Syntax

Judgemental Framework As discussed above, we achieve canonicity in 2TT using a judgemental presentation of equivalence. First, this means that, starting from the usual judgement forms of contexts Γ ctx, dependent types $\Gamma \vdash A$ type (where Γ ctx), and terms

$$\boxed{\text{Identity and composition for } \Gamma \vdash \theta : \Delta}$$

$$\frac{\Gamma \supseteq \Delta}{\Gamma \vdash \mathsf{id}^{\Gamma}_{\Delta} : \Delta} \qquad \frac{\Gamma_2 \vdash \theta_2 : \Gamma_3 \quad \Gamma_1 \vdash \theta_1 : \Gamma_2}{\Gamma_1 \vdash \theta_2[\theta_1] : \Gamma_3} \qquad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma_0 \vdash \delta : \theta_1 \simeq_\Gamma \theta_2}{\Gamma_0 \vdash \theta[\delta] : \theta[\theta_1] \simeq_\Delta \theta[\theta_2]}$$

$$\begin{array}{lcll}
\theta_0[\theta[\theta']] & \equiv & \theta_0[\theta][\theta'] & \textit{1-subst assoc/unit} \\
\theta_0[\mathsf{id}_\Gamma] & \equiv & \theta_0 & \\
\mathsf{id}^{\Gamma}_{\Gamma}[\theta] & \equiv & \theta & \\
\\
\theta[\delta[\delta']] & \equiv & \theta[\delta][\delta'] & \textit{1-resp assoc} \\
\theta[\mathsf{refl}_{\theta'}] & \equiv & \mathsf{refl}_{\theta[\theta']} & \textit{1-resp preserves refl.} \\
\theta[\theta'][\delta] & \equiv & \theta[\theta'[\delta]] & \textit{1-resp for 1-subst}
\end{array}$$

$$\boxed{\text{Identity, Inverses, and Composition for } \Gamma \vdash \delta : \theta \simeq_\Delta \theta'}$$

$$\frac{}{\Gamma \vdash \mathsf{refl}^{\Delta}_{\theta} : \theta \simeq_\Delta \theta} \qquad \frac{\Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2}{\Gamma \vdash \delta^{-1} : \theta_2 \simeq_\Delta \theta_1} \qquad \frac{\begin{array}{c}\Gamma \vdash \delta_1 : \theta_1 \simeq_\Delta \theta_2 \\ \Gamma \vdash \delta_2 : \theta_2 \simeq_\Delta \theta_3\end{array}}{\Gamma \vdash \delta_2 \circ \delta_1 : \theta_1 \simeq_\Delta \theta_3} \qquad \frac{\begin{array}{c}\Gamma \vdash \delta : \theta \simeq_\Delta \theta' \\ \Gamma_0 \vdash \delta_0 : \theta_0 \simeq_\Gamma \theta'_0\end{array}}{\Gamma_0 \vdash \delta[\delta_0] : \theta[\theta_0] \simeq_\Delta \theta'[\theta'_0]}$$

$$\begin{array}{lcll}
(\delta_3 \circ \delta_2) \circ \delta_1 & \equiv & \delta_3 \circ (\delta_2 \circ \delta_1) & \textit{trans assoc/unit} \\
(\delta \circ \mathsf{refl}) & \equiv & \delta & \\
(\mathsf{refl} \circ \delta) & \equiv & \delta & \\
(\delta \circ \delta^{-1}) & \equiv & \mathsf{refl} & \textit{inverse} \\
(\delta^{-1} \circ \delta) & \equiv & \mathsf{refl} &
\end{array}
\qquad
\begin{array}{lcll}
\delta_0[\delta[\delta']] & \equiv & \delta_0[\delta][\delta'] & \textit{2-resp assoc/unit} \\
\delta_0[\mathsf{refl}_{\mathsf{id}}] & \equiv & \delta_0 & \\
\mathsf{refl}_{\mathsf{id}^{\Gamma}_{\Gamma}}[\delta] & \equiv & \delta & \\
(\delta_1 \circ \delta_2)[\delta_3 \circ \delta_4] & \equiv & \delta_1[\delta_3] \circ \delta_2[\delta_4] & \textit{interchange} \\
\mathsf{refl}_\theta[\delta] & \equiv & \theta[\delta] & \textit{delegate}
\end{array}$$

$$\boxed{\text{Composition for } \Gamma \vdash A \ \mathsf{type}}$$

$$\frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash A \ \mathsf{type}}{\Gamma \vdash A[\theta] \ \mathsf{type}} \qquad \frac{\Delta \ \mathsf{ctx} \quad \Delta \vdash C \ \mathsf{type} \quad \Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2 \quad \Gamma \vdash M : C[\theta_1]}{\Gamma \vdash \mathsf{map}_{\Delta.C}\ \delta\ M : C[\theta_2]}$$

$$\begin{array}{rcll}
A[\theta[\theta']] & \equiv & A[\theta][\theta'] & \textit{0-subst assoc/unit} \\
A[\mathsf{id}_\Gamma] & \equiv & A & \\
\mathsf{map}_{\Delta.C}\ \mathsf{refl}_\theta\ M & \equiv & M & \textit{0-resp functoriality} \\
\mathsf{map}_{\Delta.C}\ (\delta_2 \circ \delta_1)\ M & \equiv & \mathsf{map}_{\Delta.C}\ \delta_2\ (\mathsf{map}_{\Delta.C}\ \delta_1\ M) & \\
(\mathsf{map}_{\Delta.C}\ \delta\ M)[\theta_0] & \equiv & \mathsf{map}_{\Delta.C}\ \delta[\mathsf{refl}_{\theta_0}]\ M[\theta_0] & \textit{1-subst for } \mathsf{map} \\
(\mathsf{map}_C\ (\delta : \theta_1 \simeq \theta_2)\ M)[\delta' : \theta'_1 \simeq \theta'_2] & \equiv & \mathsf{resp}\ (x.\mathsf{map}_C\ (\delta[\mathsf{refl}_{\theta'_2}])\ x)\ (M[\delta']) & \textit{1-resp for } \mathsf{map} \\
\mathsf{map}_{\Delta.C[\theta : \Delta']}\ \delta\ M & \equiv & \mathsf{map}_{\Delta'.C}\ \mathsf{refl}_\theta[\delta]\ M & \textit{def. } \mathsf{map} \textit{ for } A[\theta]
\end{array}$$

$$\boxed{\text{Composition for } \Gamma \vdash M : A}$$

$$\frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash M : A}{\Gamma \vdash M[\theta] : A[\theta]} \qquad \frac{\Delta \vdash M : A \quad \Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2}{\Gamma \vdash M[\delta] : (\mathsf{map}_{\Delta.A}\ \delta\ (M[\theta_1])) \simeq_{A[\theta_2]} M[\theta_2]}$$

$$\begin{array}{lcll}
M[\theta[\theta']] & \equiv & M[\theta][\theta'] & \textit{1-subst assoc/unit} \\
M[\mathsf{id}_\Gamma] & \equiv & M & \\
M[\delta[\delta']] & \equiv & M[\delta][\delta'] & \textit{1-resp assoc/unit} \\
M[\mathsf{refl}_\theta] & \equiv & \mathsf{refl}_{M[\theta]} & \textit{1-resp preserves refl.} \\
M[\theta][\delta] & \equiv & M[\theta[\delta]] & \textit{1-resp for 1-subst}
\end{array}$$

$$\boxed{\text{Identity, Inverses, and Composition for } \Gamma \vdash \alpha : M \simeq_A N}$$

$$\frac{}{\Gamma \vdash \mathsf{refl}^{A}_{M} : M \simeq_A M} \qquad \frac{\Gamma \vdash \alpha : M_1 \simeq_A M_2}{\Gamma \vdash \alpha^{-1} : M_2 \simeq_A M_1} \qquad \frac{\begin{array}{c}\Gamma \vdash \alpha_1 : M_1 \simeq_A M_2 \\ \Gamma \vdash \alpha_2 : M_2 \simeq_A M_3\end{array}}{\Gamma \vdash \alpha_2 \circ \alpha_1 : M_1 \simeq_A M_3} \qquad \frac{\begin{array}{c}\Gamma_0 \vdash \delta_0 : \theta_0 \simeq_\Gamma \theta'_0 \\ \Gamma \vdash \alpha : M \simeq_A N\end{array}}{\Gamma_0 \vdash \alpha[\delta_0] : (\mathsf{map}_{\Gamma.A}\ \delta_0\ (M[\theta_0])) \simeq_{A[\theta'_0]} N[\theta'_0]}$$

$$\begin{array}{lcll}
(\alpha_3 \circ \alpha_2) \circ \alpha_1 & \equiv & \alpha_3 \circ (\alpha_2 \circ \alpha_1) & \textit{trans assoc/unit} \\
(\alpha \circ \mathsf{refl}) & \equiv & \alpha & \\
(\mathsf{refl} \circ \alpha) & \equiv & \alpha & \\
(\alpha \circ \alpha^{-1}) & \equiv & \mathsf{refl} & \textit{inverse} \\
(\alpha^{-1} \circ \alpha) & \equiv & \mathsf{refl} & \\
\alpha[\delta[\delta']] & \equiv & \alpha[\delta][\delta'] & \textit{2-resp assoc/unit} \\
\alpha[\mathsf{refl}_{\mathsf{id}}] & \equiv & \alpha & \\
(\alpha_1 \circ \alpha_2)[\delta_3 \circ \delta_4] & \equiv & \alpha_1[\delta_3] \circ \mathsf{resp}\ (x.\mathsf{map}\ \delta_3\ x)\ (\alpha_2[\delta_4]) & \textit{interchange} \\
\mathsf{refl}_M[\delta] & \equiv & M[\delta] & \textit{delegate}
\end{array}$$

Omitted Rules: All judgements respect equality; all equality judgements are congruences.

Derived forms:

Derived forms:	$\mathsf{resp}\ (x.F)\ \alpha$	means	$(x.F)[\mathsf{refl}_{\mathsf{id}}, \alpha/x]$
	$\mathsf{map}^1_{x:A.B}\ \alpha\ M$	means	$\mathsf{map}_{\Gamma,x:A.B}\ (\mathsf{refl}_{\mathsf{id}}, \alpha/x)\ M$

Figure 1. Judgemental Presentation of Equivalence

Empty context:

$$\frac{}{\cdot\ \mathsf{ctx}} \quad (\mathsf{id}_{\cdot} \text{ is the only canonical substitution}) \quad (\mathsf{refl}_{\mathsf{id}_{\cdot}} \text{ is the only canonical equivalence}) \qquad \begin{array}{lcll} \theta : \cdot & \equiv & \mathsf{id}_{\cdot} & \textit{1-}\eta \\ \delta : \theta \simeq_{\cdot} \theta' & \equiv & \mathsf{refl}_{\cdot} & \textit{2-}\eta \end{array}$$

Term variables:

$$\frac{\Gamma\ \mathsf{ctx} \quad \Gamma \vdash A\ \mathsf{type}}{\Gamma\,,\, x{:}A\ \mathsf{ctx}} \quad \frac{x{:}A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma \vdash M : A[\theta]}{\Gamma \vdash \theta, M/x : \Delta, x : A} \quad \frac{\begin{array}{c}\Gamma \vdash \delta : \theta \simeq_{\Delta} \theta' \\ \Gamma \vdash \alpha : (\mathsf{map}_{\Delta.A}\ \delta\ M) \simeq_{A[\theta']} N\end{array}}{\Gamma \vdash (\delta, \alpha/x) : (\theta, M/x) \simeq_{\Delta, x:A} (\theta', N/x)}$$

$$\begin{array}{rcll} \mathsf{id}_{\Gamma}^{\Gamma, x:A}[\theta, M/x] & \equiv & \theta & \textit{1-}\beta \\ x[\theta, M/x] & \equiv & M & \textit{1-}\beta \\ \theta : (\Gamma\,,\, x{:}A) & \equiv & \mathsf{id}_{\Gamma}[\theta], x[\theta]/x & \textit{1-}\eta \\ \mathsf{id}_{\Gamma}^{\Gamma, x:A}[\delta, \alpha/x] & \equiv & \delta & \textit{2-}\beta \\ x[\delta, \alpha/x] & \equiv & \alpha & \textit{2-}\beta \\ \delta : \theta \simeq_{(\Gamma, x:A)} \theta' & \equiv & \mathsf{id}_{\Gamma}[\delta], x[\delta]/x & \textit{2-}\eta \end{array} \quad \Bigg| \quad \begin{array}{rcll} \mathsf{id}_{\Gamma, x:A} & \equiv & \mathsf{id}_{\Gamma}, x/x & \textit{1-id} \\ (\theta, M/x)[\theta_0] & \equiv & \theta[\theta_0], M[\theta_0]/x & \textit{1-subst} \\ (\theta, M/x)[\delta_0] & \equiv & \theta[\delta_0], M[\delta_0]/x & \textit{1-resp} \\ \mathsf{refl}_{\theta, M/x} & \equiv & \mathsf{refl}_{\theta}, \mathsf{refl}_{M}/x & \textit{refl} \\ (\delta, \alpha/x)^{-1} & \equiv & (\delta^{-1}, (\mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta^{-1}\ x)\ \alpha^{-1})/x) & \textit{sym} \\ (\delta_2, \alpha_2/x) \circ (\delta_1, \alpha_1/x) & \equiv & (\delta_2 \circ \delta_1), (\alpha_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta_2\ x)\ \alpha_1)/x & \textit{trans} \\ (\delta, \alpha/x)[\delta_0] & \equiv & \delta[\delta_0], \alpha[\delta_0]/x & \textit{2-resp} \end{array}$$

Figure 2. Contexts

$$\frac{\begin{array}{l}\Gamma \vdash A\ \mathsf{type} \\ \Gamma\,,\, x{:}A \vdash B\ \mathsf{type}\end{array}}{\Gamma \vdash \Pi\, x{:}A.\, B\ \mathsf{type}} \quad \frac{\Gamma\,,\, x{:}A \vdash M : B}{\Gamma \vdash \lambda\, x.\, M : \Pi\, x{:}A.\, B} \quad \frac{\begin{array}{l}\Gamma \vdash M_1 : \Pi\, x{:}A.\, B \\ \Gamma \vdash M_2 : A\end{array}}{\Gamma \vdash M_1\ M_2 : B[M_2/x]} \quad \frac{\Gamma\,,\, x{:}A \vdash \alpha : (M\ x) \simeq_B (N\ x)}{\Gamma \vdash \lambda\, x.\, \alpha : M \simeq_{\Pi\, x:A.\, B} N} \quad \frac{\Gamma \vdash \alpha : M \simeq_{\Pi\, x:A.\, B} N \quad \Gamma \vdash \beta : M_1 \simeq_A N_1}{\Gamma \vdash \alpha\, \beta : \mathsf{map}_B^1\ \beta\ (M M_1) \simeq_{B[N_1/x]} (N N_1)}$$

$$\begin{array}{lcll} (\lambda\, x.\, M)\ N & \equiv & M[N/x] & \textit{1-}\beta \\ M : \Pi\, x{:}A.\, B & \equiv & \lambda\, x.\, M\ x & \textit{1-}\eta \\ (\lambda\, x.\, \alpha_1)\ \alpha_2 & \equiv & \alpha_1[\mathsf{refl}, \alpha_2/x] & \textit{2-}\beta \\ \alpha : M \simeq_{\Pi\, x:A.\, B} N & \equiv & \lambda\, x.\, \alpha\ (\mathsf{refl}_x) & \textit{2-}\eta \\ \\ (\Pi\, x{:}A.\, B)[\theta_0] & \equiv & \Pi\, x{:}A[\theta_0].\, B[\theta_0, x/x] & \textit{0-subst} \\ \mathsf{map}_{\Delta.\Pi\, x:A.\, B}\ \delta\ M & \equiv & & \textit{0-resp} \\ \quad \lambda\, x.\, \mathsf{map}_{\Delta, x:A.B}\ (\delta, \mathsf{refl}/x)\ (M\ (\mathsf{map}_{\Delta.A}\ \delta^{-1}\ x)) & & & \end{array} \quad \Bigg| \quad \begin{array}{lcll} (\lambda\, x.\, M)[\theta_0] & \equiv & \lambda\, x.\, M[(\theta_0, x/x)] & \textit{1-subst} \\ (M_1\ M_2)[\theta_0] & \equiv & (M_1[\theta_0])\ (M_2[\theta_0]) & \textit{1-subst} \\ (\lambda\, x.\, M)[\delta] & \equiv & \lambda\, x.\, M[\delta, \mathsf{refl}/x] & \textit{1-resp} \\ (M\ N)[\delta] & \equiv & M[\delta]\ N[\delta] & \textit{1-resp} \\ \\ \mathsf{refl}_M & \equiv & \lambda\, x.\, \mathsf{refl}_{M\ x} & \textit{refl} \\ (\lambda\, x.\, \alpha)^{-1} & \equiv & \lambda\, x.\, \alpha^{-1} & \textit{sym} \\ (\lambda\, x.\, \alpha_2) \circ (\lambda\, x.\, \alpha_1) & \equiv & \lambda\, x.\, \alpha_2 \circ \alpha_1 & \textit{trans} \\ (\lambda\, x{:}A.\, \alpha)[\delta_0] & \equiv & \lambda\, x{:}A[\theta_0'].\, \alpha[\delta_0, \mathsf{refl}/x] & \textit{2-resp} \\ (\alpha_1\ \alpha_2)[\delta_0] & \equiv & (\alpha_1[\delta_0])\ (\alpha_2[\delta_0]) & \textit{2-resp} \end{array}$$

Figure 3. Π-Types

$$\frac{}{\Gamma \vdash 2\ \mathsf{type}} \quad \frac{}{\Gamma \vdash \mathsf{true} : 2} \quad \frac{}{\Gamma \vdash \mathsf{false} : 2} \quad \frac{\begin{array}{l}\Gamma \vdash M : 2 \\ \Gamma\,,\, x{:}2 \vdash C\ \mathsf{type} \\ \Gamma \vdash M_1 : C[\mathsf{true}/x] \\ \Gamma \vdash M_2 : C[\mathsf{false}/x]\end{array}}{\Gamma \vdash \mathsf{if}_{x.C}(M, M_1, M_2) : C[M/x]} \quad \frac{\begin{array}{l}\Gamma \vdash M : 2 \\ \Gamma\,,\, x{:}2 \vdash C\ \mathsf{type} \\ \Gamma\,,\, x{:}2 \vdash M_1, M_2 : C \\ \Gamma \vdash \alpha_1 : M_1[\mathsf{true}/x] \simeq_{C[\mathsf{true}/x]} M_2[\mathsf{true}/x] \\ \Gamma \vdash \alpha_2 : M_1[\mathsf{false}/x] \simeq_{C[\mathsf{false}/x]} M_2[\mathsf{false}/x]\end{array}}{\Gamma \vdash \mathsf{if}_{x.C}(M, \alpha_1, \alpha_2) : M_1[M/x] \simeq_{C[M/x]} M_2[M/x]}$$

$$(\mathsf{refl}_{\mathsf{true}} \text{ and } \mathsf{refl}_{\mathsf{false}} \text{ are canonical}) \quad \frac{\Gamma \vdash \alpha : M \simeq_2 N}{\Gamma \vdash M \equiv N}\ \textit{reflection} \quad \frac{\Gamma \vdash \alpha : M \simeq_2 N}{\Gamma \vdash \alpha \equiv \mathsf{refl}_M}\ \textit{uip} \quad \frac{\Gamma \vdash \alpha : \mathsf{true} \simeq_2 \mathsf{false}}{\Gamma \vdash \mathcal{J}} \ (\text{where } \mathcal{J} \text{ is any judgement})$$

$$\begin{array}{rcll} \mathsf{if}(\mathsf{true}, M_1, M_2) & \equiv & M_1 & \textit{1-}\beta \\ \mathsf{if}(\mathsf{false}, M_1, M_2) & \equiv & M_2 & \textit{1-}\beta \\ M[(N : El(2))/x] & \equiv & \mathsf{if}(N, M[\mathsf{true}/x], M[\mathsf{false}/x]) & \textit{1-}\eta \\ \mathsf{if}_{x.C}(\mathsf{true}, \alpha_1, \alpha_2) & \equiv & \alpha_1 & \textit{12-}\beta \\ \mathsf{if}_{x.C}(\mathsf{false}, \alpha_1, \alpha_2) & \equiv & \alpha_2 & \textit{12-}\beta \\ \alpha[(\mathsf{refl}_{M\,:\,2})/x] & \equiv & \mathsf{if}(M, \alpha[\mathsf{refl}_{\mathsf{true}}/x], \alpha[\mathsf{refl}_{\mathsf{false}}/x]) & \textit{12-}\eta \\ \\ 2[\theta] & \equiv & 2 & \textit{0-subst} \\ \mathsf{map}_{\Delta.2}\ \delta\ M & \equiv & M & \textit{0-resp} \end{array} \quad \Bigg| \quad \begin{array}{rcll} \mathsf{true}[\theta] & \equiv & \mathsf{true} & \textit{1-subst} \\ \mathsf{false}[\theta] & \equiv & \mathsf{false} & \textit{1-subst} \\ \mathsf{if}_{x\,:\,2.C}(M, M_1, M_2)[\theta] & \equiv & & \textit{1-subst} \\ \quad \mathsf{if}_{x\,:\,2.C[\theta, x/x]}(M[\theta], M_1[\theta], M_2[\theta]) & & & \\ \\ \mathsf{true}[\delta] & \equiv & \mathsf{refl}_{\mathsf{true}} & \textit{1-resp} \\ \mathsf{false}[\delta] & \equiv & \mathsf{refl}_{\mathsf{false}} & \textit{1-resp} \\ \mathsf{if}_{x\,:\,2.C}(M, M_1, M_2)[\delta : \theta_1 \simeq \theta_2] & \equiv & & \textit{1-resp} \\ \quad \mathsf{if}_{x\,:\,2.C[\theta_2, x/x]}(M[\theta_2], M_1[\delta], M_2[\delta]) & & & \\ \\ \mathsf{refl} & & \text{is canonical} & \\ -^{-1}, - \circ - & & \text{trivial by } \textit{uip} & \\ \mathsf{if}(M, \alpha_1, \alpha_2)[\delta : \theta_1 \simeq \theta_2] & \equiv & \mathsf{if}(M[\theta_2], \alpha_1[\delta], \alpha_2[\delta]) & \textit{2-resp} \end{array}$$

Figure 4. Booleans (as an Extensional Set)

$$\frac{}{\Gamma \vdash \mathsf{set}\ \mathsf{type}} \quad \frac{\Gamma \vdash S : \mathsf{set}}{\Gamma \vdash El(S)\ \mathsf{type}}$$

$$\frac{\Gamma \vdash \alpha : M \simeq_{El(S)} N}{\Gamma \vdash M \equiv N} \quad \frac{\Gamma \vdash \alpha : M \simeq_{El(S)} N}{\Gamma \vdash \alpha \equiv \mathsf{refl}_M}$$

$$\begin{array}{rcll}
\mathsf{set}[\theta_0] & \equiv & \mathsf{set} & \textit{0-subst} \\
(El(S))[\theta_0] & \equiv & El(S[\theta_0]) & \\
\mathsf{map}_{\Delta.\mathsf{set}}\ \delta\ M & \equiv & M & \textit{0-resp} \\
-^{-1}, - \circ - \text{ for set} & & \text{groupoid laws plus below} & \\
\mathsf{refl}_{M\,:\,El(S)} & & \text{is canonical} & \\
-^{-1}, - \circ -, \text{2-resp for } El(S) & & \text{trivial by } \textit{uip} &
\end{array}$$

$$\frac{}{\Gamma \vdash \mathsf{bool} : \mathsf{set}} \quad (\mathsf{refl}_{\mathsf{bool}} \text{ is canonical}) \quad \frac{}{\Gamma \vdash \mathsf{not} : \mathsf{bool} \simeq_{\mathsf{set}} \mathsf{bool}} \quad \frac{\Gamma \vdash M : 2}{\Gamma \vdash \mathsf{in}\ M : El(\mathsf{bool})} \quad \frac{\Gamma \vdash M : El(\mathsf{bool})}{\Gamma \vdash \mathsf{out}\ M : 2}$$

$$\begin{array}{rclll}
\mathsf{out}\ (\mathsf{in}\ M) & \equiv & M & & \textit{1-}\beta \\
\mathsf{in}\ (\mathsf{out}\ M) & \equiv & M & & \textit{1-}\eta \\
\mathsf{map}_{\Delta.El(S)}\ \delta\ \mathsf{true} & \equiv & \mathsf{false} & \text{if } S[\delta] \equiv \mathsf{not} & \textit{0-resp} \\
\mathsf{map}_{\Delta.El(S)}\ \delta\ \mathsf{false} & \equiv & \mathsf{true} & \text{if } S[\delta] \equiv \mathsf{not} & \textit{0-resp}
\end{array}$$

$$\begin{array}{rcll}
\mathsf{bool}[\theta] & \equiv & \mathsf{bool} & \textit{1-subst} \\
(\mathsf{in}\ M)[\theta] & \equiv & \mathsf{in}\ M[\theta] & \textit{1-subst} \\
(\mathsf{out}\ M)[\theta] & \equiv & \mathsf{out}\ M[\theta] & \textit{1-subst} \\
\mathsf{bool}[\delta] & \equiv & \mathsf{refl}_{\mathsf{bool}} & \textit{1-resp} \\
\mathsf{not}^{-1} & \equiv & \mathsf{not} & \textit{sym} \\
\mathsf{not}[\delta] & \equiv & \mathsf{not} & \textit{2-resp}
\end{array}$$

Figure 5. Universe

$\Gamma \vdash M : A$ (where Γ ctx and $\Gamma \vdash A$ type), we add a judgement of term equivalences $\Gamma \vdash \alpha : M \simeq_A M'$ (where Γ ctx and $\Gamma \vdash A$ type and $\Gamma \vdash M, M' : A$). Second, this equivalence judgement is defined by rules specific to each type A, including equations explaining the meaning of identity (refl), inverses (α^{-1}), and composition ($\alpha_2 \circ \alpha_1$) at each type. Third, all families of types and terms respect equivalence, as expressed by the following operations:

$$\frac{\Gamma, x{:}A \vdash B\ \mathsf{type} \quad \Gamma \vdash \alpha : M_1 \simeq_A M_2 \quad \Gamma \vdash M : B[M_1/x]}{\Gamma \vdash \mathsf{map}^1_{x:A.B}\ \alpha\ M : B[M_2/x]}$$

$$\frac{\Gamma \vdash \alpha : M \simeq_A N \quad \Gamma, x{:}A \vdash F : B}{\Gamma \vdash \mathsf{resp}\ (x.F)\ \alpha : F[M/x] \simeq_B F[N/x]}$$

map expresses that a family of types indexed by A respects equivalence at A: Given equivalent terms M_1 and M_2, map determines a function from $B[M_1/x]$ to $B[M_2/x]$. As we discuss below, this function has an inverse given by $\mathsf{map}^1_{x:A.B}\ \alpha^{-1}$, so $B[M_1/x]$ and $B[M_2/x]$ are isomorphic. Similarly, $\mathsf{resp}\ (x.F)\ \alpha$ expresses that a family of terms of type B (where x is not free in B) with a free variable $x{:}A$ respects equivalence at A. $\mathsf{map}^1_{x:A.B}\ \alpha\ M$ computes on the structure of the family B, while $\mathsf{resp}\ (x.F)\ \alpha$ computes on the structure of the family F. There is an additional resp-like operation which expresses that equivalences themselves respect equivalence.

To define these operations, it is helpful to consider n-ary families such as $x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n.B$. To this end, we present 2TT as an explicit substitution calculus, with additional judgements for substitutions $\Gamma \vdash \theta : \Delta$ (where Γ ctx and Δ ctx) and equivalences between them $\Gamma \vdash \delta : \theta \simeq_\Delta \theta'$ (where Γ ctx and Δ ctx and $\Gamma \vdash \theta, \theta' : \Delta$). The treatment of dependent types in Pitts [20]'s survey article provides an introduction to this style of syntax, with an explicit substitution judgement and internalized composition principles. In full generality, map and resp express respect for equivalence between substitutions for a whole context, and are written $\mathsf{map}_{\Delta.C}\ \delta\ M$ and $M[\delta]$ respectively. An analogous operation $\alpha[\delta]$ expresses that equivalence itself respects equivalence, while $\theta[\delta]$ and $\delta[\delta']$ extend these resp-principles to substitutions for an entire context, and equivalences between them.

The rules for this judgemental framework are presented in Figure 1. We present the definitional equality rules as equations (e.g. $\theta \equiv \theta'$), as a notational shorthand for inference rules defining typed equality judgements (e.g. $\Gamma \vdash \theta \equiv \theta' : \Delta$). By convention, each equation implicitly has premises asserting that each of the metavariables occuring in the equation are well-typed, and maintains the invariant that the subjects of the equation are well-typed given these assumptions. Showing that the two sides of the equation have the same type sometimes involves definitional equality reasoning, but the equation being defined is never necessary for showing that its subjects have the same type.

The first three rules define identity and composition for substitutions. To improve readability, we make weakening admissible (rather than using de Bruijn form), so the identity substitution id is really the composition of the identity substitution with projections that forget any number of variables. We write $\Gamma \supseteq \Delta$ to mean that Δ is obtained from Γ by dropping some number of variables. All judgements of the form $\Gamma \vdash J$ are weakenable: If $\Gamma \vdash J$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash J$. Composition of substitutions $\theta_2[\theta_2]$, which we refer to as 1-substitution, is standard in explicit substitution calculi. The additional composition operation, $\theta[\delta]$, forces substitutions to respect equivalence, analogously to resp for terms above: substitution instances by equivalent substitutions are equivalent. For this reason, we refer to it as *1-resp*. The first three equations say that 1-substitution is associative and unital. In the second equation, id_Γ can in fact be a weakening, in which case θ is tacitly weakened in the right-hand side. The third equation only makes sense when $\Gamma \vdash \mathsf{id} : \Gamma$, which we notate by $\mathsf{id}^\Gamma_\Gamma$. The next two rules say that *1-resp* associates with *2-resp* ($\delta[\delta']$), which is the analogous operation for equivalences (defined below), and preserves identities refl(defined below).

The next four rules define identity, inverses, and composition for equivalences. Equivalences are always reflexive (refl), symmetric (δ^{-1}), and transitive ($\delta_2 \circ \delta_1$). Additionally, equivalences themselves respect equivalence ($\delta[\delta_0]$), which we call *2-resp*. The equations say that: Transitivity is associative, with unit refl, and inverses given by $-^{-1}$. *2-resp* is also associative and unital, with unit $\Gamma \vdash \mathsf{refl}_{\mathsf{id}^\Gamma_\Gamma} : \mathsf{id}_\Gamma \simeq_\Gamma \mathsf{id}_\Gamma$ (by above, $\theta[\mathsf{id}_\Gamma]$ equals θ). The *interchange law* relates *2-resp* and transitivity: transitivity followed by *2-resp* is the same as *2-resp* followed by transitivity. This is a coherence requirement between the two forms of composition in a 2-category; we discuss some special cases in the extended version of this article [13].

The rule *delegate* delegates *2-resp* at reflexivity to *1-resp*. We do not define *2-subst*, $\delta[\theta]$, directly, as this composition is definable as $\delta[\mathsf{refl}_\theta]$.

Next, we define the corresponding operations for dependent types, terms, and term equivalences. A dependent type A can be pre-composed with a substitution, written $A[\theta]$; and has a functorial action $\mathsf{map}_{\Delta.A}\ \delta\ M$, which expresses that families of types respect equivalence. We refer to these as *0-subst* and *0-resp*. The equations say: Substitution into types (*0-subst*) is associative with unit refl. map is functorial, preserving reflexivity and transitivity. The next two rules define *1-subst* and *1-resp* for map, which reassociate the

1-subst/*1-resp* with the *0-resp*. The next rule defines map for a composition, again by reassociating.

Like all contextual judgements, terms are closed under substitution ($M[\theta]$) and respect equivalence ($M[\delta]$). Because terms are dependent on the context, the latter requires "adjusting" $M[\theta_1]$ by δ so that it lives in the same type as $M[\theta_2]$. The equality rules are analogous to those for substitutions: *1-subst* is associative and unital, and *1-resp* is associative and preserves reflexivities.

The rules for term equivalences are analogous to the rules for equivalences, specifying reflexivity, transitivity, and *2-resp*. The equations say that transitivity is associative, invertible, and unital, that *2-resp* is associative and unital, and that the order of trans and *2-resp* can be interchanged. The interchange rule uses the derived form resp, described above.

Contexts The general methodology for defining a context is to specify (1) A formation rule for Γ. (2) A substitution rule $\theta : \Gamma$, and a hypothesis rule for one of the other judgements (e.g. the term rule for x for the context former $\Gamma\ ,\, x{:}A$). These function as the introduction and elimination rules for the context, which are products of some sort, eliminated by first projections (which are implicit in id) and variables (representing projections). (3) An equivalence rule for $\delta : \theta \simeq_\Gamma \theta'$ (4) Equations defining *1-$\beta\eta$* ($\beta\eta$ for θ), *2-$\beta\eta$* ($\beta\eta$ for δ), *identity* id_Γ, *1-subst* $\theta[\theta']$, *1-resp* $\theta[\delta']$, *reflexivity* refl_θ, *symmetry* δ^{-1}, *transitivity* $\delta \circ \delta'$, and *2-resp* $\delta[\delta']$. In general, refl and δ^{-1} and $\delta \circ \delta'$ are defined in a type-directed manner, while the *subst/resp* principles are defined in a syntax-directed manner, giving one rule for each syntactic construct.

In Figure 2 we carry out this methodology for the basic contexts: The empty context has a trivial substitution into it, and a trivial equivalence from this substitution to itself. For the equations, it suffices to stipulate that these are unique.

For context extension, if A is a type well-formed in Γ, then Γ can be extended with a variable of type A. A variable can be used as a term; the typing rule checks that the variable is in the context. The substitution into an extended context $\Delta\ ,\, x{:}A$ is a pair of a substitution θ into Δ and a term of type A, adjusted by θ (this is analogous to the usual introduction rule for a Σ-type). An equivalence between such substitutions is a pair of equivalences, one between the substitutions, and the other between the terms (adjusted by the first component). As these substitutions and equivalences are pairs, the first set of rules gives the expected $\beta\eta$ rules, for the projections given by id and variables. The next rules define the identity, composition, and inverse operations componentwise.

Π-*types* In general, a type is specified by (1) A formation rule for A. (2) Introduction and elimination rules for terms, defining $M : A$. (3) Introduction and elimination rules for equivalences, defining $\alpha : M \simeq_A M'$. (4) Equations defining *1-$\beta\eta$* ($\beta\eta$ for M), *2-$\beta\eta$* ($\beta\eta$ for α), *0-substitution* $A[\theta]$, *0-resp* $\mathsf{map}_A\ \delta\ M$, *1-substitution* $M[\theta]$, *1-resp* $M[\delta]$, *reflexivity* refl_M, *transitivity* $\alpha \circ \alpha'$, and *2-resp* $\alpha[\delta]$.

In Figure 3, we give the rules for dependent function types. The formation and term rules are standard. The equivalence introduction rule says that an equivalence at Π can be introduced by giving a family of equivalences that work for each element—the function extensionality rule. An equivalence is eliminated by applying to equivalent arguments, yielding an equivalence between the results. These rules have been considered in categorically-motivated accounts of functionally extensional propositional equality [9].

The $\beta\eta$-rules are the expected rules for functions, both at the term and equivalence levels. We write $M[N/x]$ to abbreviate $M[\mathsf{id}, N/x]$. Substitution into a Π-type proceeds compositionally. $\mathsf{map}_{\Pi\, x:A.\, B}$ is given by pre- and post-composition. *1-subst* and *1-resp* are both defined compositionally, as is *2-resp*. The rule for refl defines the identity at function type in terms of the componentwise identity.

The rules for dependent pairs, which are analogous to the rules for $\Gamma\ ,\, x{:}A$, are presented in the extended version [13].

Booleans In Figure 4, we give rules for a base type of booleans, including the usual true, false, and if-then-else constructs. We additionally include an if-then-else that eliminates towards equivalences, permitting equivalences to be defined by case distinction. For simplicity, we specify booleans as an *extensional set*, corresponding semantically to the discrete groupoid with two objects. This means that (1) equivalent booleans are definitionally equal (*reflection*) and (2) any equivalence is equal to reflexivity (*uip*). Finally, we include a rule stipulating that true and false are not equivalent, which would otherwise require large eliminations to prove. These rules illustrate that 2TT is compatible with treating discrete types as in extensional type theory, though a more intensional treatment of booleans is also possible. The $\beta\eta$-rules are standard for sum types; *0-resp* is trivial because 2 is a constant family; the *0/1-subst* rules are standard. The *1-resp* rule for if uses the equivalence-level if-then-else, and is well-typed because of *reflection*; the *2-resp* rule for if-then-else is analogous. No specific equations need to be given for $-^{-1}$ and $- \circ -$ because of *uip*.

Sets and elements Finally, we need to seed the type theory with a base type with non-trivial equivalences, and a family of types dependent on it, to ensure that we have an example where map really has computational content. A simple example is to consider a universe set that contains discrete types. That is, each term $S : \mathsf{set}$ will determine a type $El(S)$ whose elements have no non-identity equivalences between them. However, set itself is *not* a discrete type, because equivalence between sets S and T may be given by an isomorphism between $El(S)$ and $El(T)$:

$$\frac{\begin{array}{ll} f : S \to T & \alpha : \lambda\, x. f(g(x)) \simeq \lambda\, x.\, x \\ g : T \to S & \beta : \lambda\, x.\, g(f(x)) \simeq \lambda\, x.\, x \end{array}}{\mathsf{iso}(f, g, \alpha, \beta) : S \simeq_{\mathsf{set}} T}$$

Then $\mathsf{map}^1_{a:\mathsf{set}.C}\ (\mathsf{iso}(f, g, \alpha, \beta))$ asserts that all families of types respect isomorphism of sets.

In Figure 5, we define a simple universe of sets. We make two simplifications: first, the universe contains a code for exactly one set, booleans; second, equivalences are given explicitly by the two automorphisms on 2, refl (the identity function) and not. The approach readily scales to a richer universe closed under Π and Σ, following our previous work [14], and to programmer-defined isomorphisms given by $\mathsf{iso}(f, g, \alpha, \beta)$, as we show in the extended version of this article [13]. The first four rules define the type set and the family $El(-)$, and give *reflection* and *uip* for $El(-)$, expressing discreteness of sets in the universe. The equations for *0-subst* are compositional; *0-resp* for set is trivial because it is a constant family. For $M : El(S)$, *1-resp* generates an equation $\mathsf{map}_{El(S)}\ \delta\ M[\theta_1] \equiv M[\theta_2]$ by *reflection*, so no equations for $M[\delta]$ are necessary.

The next four inference rules specify the type 2: $El(\mathsf{bool})$ is isomorphic to 2 by in and out and their $\beta\eta$ rules. The two equivalences $\mathsf{bool} \simeq_{\mathsf{set}} \mathsf{bool}$ are $\mathsf{refl}_{\mathsf{bool}}$ (identity) and not (negation); $\mathsf{map\ not}$ *is* computationally relevant, as it interchanges true and false. The symmetry rule says that not is involutive.

3. Properties

3.1 Consistency

To show that the calculus is consistent, we can adapt Hofmann and Streicher's groupoid interpretation [12] to our formulation of 2TT. We have given a proof for a similar formalism for directed type theory in previous work [14]. We interpret each judgement as follows:

$[\![\Gamma]\!]$ is a category. $[\![\Gamma \vdash \theta : \Delta]\!]$ is a functor $[\![\theta]\!] : [\![\Gamma]\!] \longrightarrow [\![\Delta]\!]$. $[\![\Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2]\!]$ is a natural transformation $[\![\delta]\!] : [\![\theta_1]\!] \simeq [\![\theta_2]\!] : [\![\Gamma]\!] \longrightarrow [\![\Delta]\!]$. $[\![A]\!]$ is a functor $[\![A]\!] : [\![\Gamma]\!] \longrightarrow$ *GPD*, where *GPD* is the (large) category of groupoids and functors. Terms and term equivalences are interpreted as "dependent" functors and natural transformations. Because the details of this proof have been covered in previous work, we do not review them here. As a corollary of the interpretation, we get various consistency results: $[\![2]\!]$ is the discrete category on two objects, so the interpretation shows that it is not the case that $\cdot \vdash \mathsf{true} \equiv \mathsf{false} : 2$, or that $\cdot \vdash \alpha : \mathsf{true} \simeq_2 \mathsf{false}$. Moreover, if the calculus is extended with an empty type 0 interpreted as the empty category, the interpretation implies that not all types are inhabited. We do not attempt a proof-theoretic proof of consistency of definitional equality, e.g. using reduction or normalization-by-evaluation [7]. Proving consistency using these approaches typically involves giving a decision procedure for definitional equality, and equality for 2-dimensional type theory is not decidable, due to the equality reflection rules for identity types discussed in in Section 4. Given this, we also exploit equality reflection for 2 and $El(S)$, though these types could be treated in a more intensional way instead.

3.2 Canonicity

Our main new result in this paper is to check a kind of completeness, verifying that we have not omitted any necessary definitional equations for map, resp, etc. Specifically, we show that the equations of 2TT are sufficient to equate a closed term of type 2 to a value:

THEOREM 3.1: CANONICITY. *If* $\cdot \vdash M : 2$ *then either* $\cdot \vdash M \equiv \mathsf{true} : 2$ *or* $\cdot \vdash M \equiv \mathsf{false} : 2$.

This result fails for higher-dimensional type theory based on the univalence axiom [23], due to stuck applications of Id-elimination, as illustrated in the introduction—though it is currently conjectured that every closed term of boolean type may be *equivalent* to a value. In our setting, we prove the stronger result that canonicity holds for definitional equality.

The proof is organized as follows: First, we define the semantic domains into which we interpret, which consist of *syntactically presented groupoids and functors*. Using the usual terminology of logical relations, we say that an expression is "reducible" iff it is a member of these semantic domains. In a simple logical relations argument, open terms are reducible iff they take reducible arguments to reducible results. For 2-dimensional type theory, this generalizes to both (1) taking reducible terms to reducible terms and (2) taking reducible equivalences to reducible equivalences—i.e. functoriality. Analogous definitions are necessary for dependent types, dependently typed terms, and equivalences. Next, we show that the semantic domains are closed under the type formers, and prove the fundamental theorem, which says that all well-typed expressions are reducible. Finally, we obtain canonicity as a corollary.

3.2.1 Theorem Statement

Syntactically presented groupoids and functors

DEFINITION 3.2. *A groupoid G is presented by Γ (where Γ ctx) iff*

1. *The set of objects of G, written $Ob\ G$, is a subset of the equivalence classes of substitutions $\cdot \vdash \theta : \Gamma$ modulo definitional equality. Due to the preponderance of brackets in the syntax of 2TT, we write «θ» for the equivalence class of θ.*
2. *The set of morphisms «θ_1» $\longrightarrow_G$ «θ_2» is a subset of the equivalence classes of equivalences $\cdot \vdash \delta : \theta_1 \simeq_\Gamma \theta_2$ modulo definitional equality.*
3. *identity at «θ» is given by «refl_θ», composition of «δ_2» and «δ_1» is given by «$\delta_2 \circ \delta_1$», and the inverse of «δ» is given by «δ^{-1}».*

Note that we choose representatives of equivalence classes at various points in this definition; these uses are well-defined by the congruence laws for definitional equality. We write $\langle\Gamma\rangle$ to mean some groupoid that is presented by Γ, and use the analogous notation for the other syntactic presentations defined below.

Analogously, we define what it means for a groupoid to be presented by a closed type:

DEFINITION 3.3. *A groupoid G is presented by A (where $\cdot \vdash A$ type) iff*

1. *$Ob\ G$ is a subset of the equivalence classes of terms $\cdot \vdash M : A$ modulo definitional equality.*
2. *The set of morphisms «M_1» $\longrightarrow_G$ «M_2» is a subset of the equivalence classes of equivalences $\cdot \vdash \alpha : M_1 \simeq_\Gamma M_2$ modulo definitional equality.*
3. *identity at «M» is given by «refl_M», composition of «α_2» and «α_1» is given by «$\alpha_2 \circ \alpha_1$», and the inverse of «α» is given by «α^{-1}».*

A functor may be presented by a term with one free variable:

DEFINITION 3.4. *A functor $F : \langle A\rangle \longrightarrow \langle B\rangle$ is presented by M (where $x : A \vdash M : B$) iff*

1. *For all «N» $\in Ob\ \langle A\rangle$, F(«N») = «$M[N/x]$».*
2. *For all «α» : «N_1» $\longrightarrow_{\langle A\rangle}$ «N_2», F(«α») = «$M[\alpha/x]$».*

Let *GPD* be the category of groupoids and functors between them. A functor into *GPD* may be presented by a type:

DEFINITION 3.5. *A functor $F : \langle\Gamma\rangle \longrightarrow$ GPD is presented by A, where $\Gamma \vdash A$ type iff*

1. *For all «θ_1» $\in Ob\ \langle\Gamma\rangle$, F(«θ_1») is presented by $A[\theta_1]$.*
2. *For all «δ» : «θ_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ_2», F(«δ») is presented by $x : A[\theta_1].\mathsf{map}_A\ \delta\ x$.*

We write $\langle A\rangle$ for a functor presented by A.

Observe that all of these definitions respect definitional equality. For example, if G is presented by Γ, and $\Gamma \equiv \Gamma'$, then G is also presented by Γ'.

Reducible expressions Given these definitions, we can define the invariants about each syntactic category that are demanded by the proof:

DEFINITION 3.6. *Given groupoids $\langle\Gamma\rangle$ and $\langle\Delta\rangle$, define the set $RedSubst\ \langle\Gamma\rangle\ \langle\Delta\rangle$ (reducible substitutions) to be those substitutions $\Gamma \vdash \theta : \Delta$ such that*

1. *For all «θ_1» $\in Ob\ \langle\Gamma\rangle$, «$\theta[\theta_1]$» $\in Ob\ \langle\Delta\rangle$*
2. *For all «δ» : «θ_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ_2», «$\theta[\delta]$» : «$\theta[\theta_1]$» $\longrightarrow_{\langle\Delta\rangle}$ «$\theta[\theta_2]$»*

θ necessarily preserves identity and composition, by the corresponding rules of definitional equality. Thus, if $\theta \in RedSubst\langle\Gamma\rangle\langle\Delta\rangle$, then we can define a functor $\theta^* : \langle\Gamma\rangle \longrightarrow \langle\Delta\rangle$ using the given actions on objects and arrows.

DEFINITION 3.7. *Given $\langle\Gamma\rangle$ and $\langle\Delta\rangle$ and $\theta_1, \theta_2 \in RedSubst\langle\Gamma\rangle\langle\Delta\rangle$, define the set $RedEquiv^{\langle\Gamma\rangle}_{\langle\Delta\rangle}\ \theta_1\ \theta_2$ (reducible equivalences) to contain those $\Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2$ such that for all «δ'» : «θ'_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ'_2», «$\delta[\delta']$» : «$\theta_1[\theta'_1]$» $\longrightarrow_{\langle\Delta\rangle}$ «$\theta_2[\theta'_2]$»*

As a special case, this means that for any $\theta \in Ob\ \langle\Gamma\rangle$, «$\delta[\mathsf{refl}_\theta]$» : «$\theta_1[\theta]$» $\longrightarrow_{\langle\Delta\rangle}$ «$\theta_2[\theta]$». This family of morphisms is natural (a consequence of the interchange law rule of definitional equality). Consequently, if δ is reducible, then it determines a natural isomorphism $\delta^* : \langle\theta_1\rangle \simeq \langle\theta_2\rangle : \langle\Gamma\rangle \longrightarrow \langle\Delta\rangle$. We write $RedEquiv\ \theta_1\ \theta_2$ for $RedEquiv^{\langle\Gamma\rangle}_{\langle\Delta\rangle}\ \theta_1\ \theta_2$ when the contexts are unambiguous.

DEFINITION 3.8. *Given a groupoid $\langle\Gamma\rangle$ and a functor $\langle A\rangle : \langle\Gamma\rangle \longrightarrow$ GPD, define the set $RedTm^{\langle\Gamma\rangle}\ \langle A\rangle$ (reducible terms) to be those terms $\Gamma \vdash M : A$ such that*

1. *For all «θ_1» $\in Ob\ \langle\Gamma\rangle$, «$M[\theta_1]$» $\in Ob\ \langle A\rangle$(«θ_1»)*
2. *For all «δ» : «θ_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ_2»,*
   «$M[\delta]$» : «$\mathsf{map}_A\ \delta\ M[\theta_1]$» $\longrightarrow_{\langle A\rangle(«\theta_2»)}$ «$M[\theta_2]$»

We write $RedTm\ \langle A\rangle$ for $RedTm^{\langle\Gamma\rangle}\ \langle A\rangle$ when the context is unambiguous.

DEFINITION 3.9. *Given $M, N \in RedTm^{\langle\Gamma\rangle}\ \langle A\rangle$, define the set RedEquiv M N (reducible equivalences) to contain those $\Gamma \vdash \alpha : M \simeq_A N$ such that for all «δ» : «θ_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ_2», «$\alpha[\delta]$» : «$\mathsf{map}_A\ \delta\ M[\theta_1]$» $\longrightarrow_{\langle A\rangle(«\theta_2»)}$ «$N[\theta_2]$»*

Theorem Statement Using these definitions, we are in a position to state the fundamental theorem.

In the remainder of this section, we will define partial functions $[\![\Gamma]\!]$ and $[\![A]\!]$ such that:

THEOREM 3.10: FUNDAMENTAL THEOREM.

1. *If Γ ctx then $[\![\Gamma]\!]$ is a groupoid presented by Γ.*
2. *If $\Gamma \equiv \Gamma'$ then $[\![\Gamma]\!] = [\![\Gamma']\!]$.*
3. *If $\Gamma \vdash \theta : \Delta$ then $\theta \in RedSubst\ [\![\Gamma]\!]\ [\![\Delta]\!]$*
4. *If $\Gamma \vdash \delta : \theta_1 \simeq_\Delta \theta_2$ then $\delta \in RedEquiv^{[\![\Gamma]\!]}_{[\![\Delta]\!]}\ \theta_1\ \theta_2$.*
5. *If $\Gamma \vdash A$ type then $[\![A]\!]$ is a functor $[\![\Gamma]\!] \longrightarrow$ GPD presented by A.*
6. *If $\Gamma \vdash A \equiv A'$ type then $[\![A]\!] = [\![A']\!]$.*
7. *If $\Gamma \vdash M : A$ then $M \in RedTm^{[\![\Gamma]\!]}\ [\![A]\!]$*
8. *If $\Gamma \vdash \alpha : M \simeq_A N$ then $\alpha \in RedEquiv^{[\![\Gamma]\!]}_{[\![A]\!]}\ M\ N$*

The proof is by mutual induction on the given derivations.

As usual, the interpretation is defined compositionally, in terms of semantic counterparts of each type constructor:

$$\begin{array}{rcl}
[\![\cdot]\!] & = & \cdot \\
[\![\Gamma, x : A]\!] & = & \int_{[\![\Gamma]\!]} [\![A]\!] \\
[\![\Pi\, x{:}A.\, B]\!] & = & \mathbf{\Pi}_{[\![A]\!]}\ [\![B]\!] \\
[\![2]\!] & = & \mathbf{const}(\mathbf{2}) \\
[\![\mathsf{set}]\!] & = & \mathbf{const}(\mathbf{set}) \\
[\![El(M)]\!] & = & \mathbf{El}[M^*] \\
[\![A[\theta]]\!] & = & [\![A]\!][\theta^*]
\end{array}$$

These should be understood as partial functions, due to the typing constraints on the semantic type formers, as described below. The fundamental theorem states that they are defined on all well-typed expressions.

3.2.2 Definitions of Semantic Contexts and Types

First, we give the inductive steps of the interpretation of contexts and types, describing the semantic analogue of each context and type constructor. For each context, we define a groupoid, and for each type, we define a functor $\langle\Gamma\rangle \longrightarrow$ *GPD*. Moreover, we prove that these constructions are presented by the appropriate contexts and types. The type-directed definitional equalities for refl, $-^{-1}$, and $- \circ -$, as well as the equations defining map_A , are used in these verifications.

Contexts Corresponding to the empty context, let $\cdot$ be the groupoid with one object, «$\mathsf{id}_\cdot$», and one identity arrow, «$\mathsf{refl}_{\mathsf{id}_\cdot}$», and all compositions and inverses defined to be «$\mathsf{refl}_{\mathsf{id}_\cdot}$». Observe that this is presented by $\cdot$: the objects are classes of closed substitutions; the morphisms of closed equivalences; identity is «refl»; «δ»$^{-1}$ is «δ^{-1}» = «refl» by *2η*; and similarly for composition.

For context extension, given $\langle\Gamma\rangle$ and $\langle A\rangle : \langle\Gamma\rangle \longrightarrow$ *GPD*, the Grothendieck construction constructs a fibration from the total category $\int_{\langle\Gamma\rangle}\langle A\rangle$ to $\langle\Gamma\rangle$ given by a projection functor p. Concretely, we define $\int_{\langle\Gamma\rangle}\langle A\rangle$ so that it is presented by $\Gamma, x : A$:

1. An object «$(\theta, M/x)$» is the equivalence class of a pair, where «θ» $\in Ob\ \langle\Gamma\rangle$ and «M» $\in Ob\ \langle A\rangle$(«θ»).
2. A morphism «$(\delta, \alpha/x)$» : «$(\theta_1, M_1/x)$» $\longrightarrow_{\int_{\langle\Gamma\rangle}\langle A\rangle}$ «$(\theta_2, M_2/x)$» is the equivalence class of a pair, where «δ» : «θ_1» $\longrightarrow_{\langle\Gamma\rangle}$ «θ_2» and «α» : «$\mathsf{map}_A\ \delta\ M_1$» $\longrightarrow_{\langle A\rangle(«\theta_2»)}$ «M_2»
3. Identity is defined to be
   $$«\mathsf{refl}_{(\theta,M/x)}» : «(\theta, M/x)» \longrightarrow_{\int_{\langle\Gamma\rangle}\langle A\rangle} «(\theta, M/x)»$$
   To verify that this is a morphism, we must show that it is the equivalence class of a pair of morphisms, each in the appropriate category. By definitional equality, «$\mathsf{refl}_{(\theta,M/x)}$» = «$(\mathsf{refl}_\theta, \mathsf{refl}_M/x)$», so it remains to show that
   $$\begin{array}{l} «\mathsf{refl}_\theta» : «\theta» \longrightarrow_{\langle\Gamma\rangle} «\theta» \\ «\mathsf{refl}_M» : «\mathsf{map}_A\ \mathsf{refl}_\theta\ M» \longrightarrow_{\langle A\rangle(«\theta»)} «M». \end{array}$$
   For the first, because $\langle\Gamma\rangle$ is presented, «refl_θ» is a morphism in $\langle\Gamma\rangle$ (and is in fact the identity). For the second, «$\mathsf{map}_A\ \mathsf{refl}_\theta\ M$» = «$M$» by definitional equality, and because $\langle A\rangle$ is presented, $\langle A\rangle(\theta)$ is presented by $A[\theta]$, and therefore «refl_M» is a morphism (and moreover is the identity).
   The verification of the definition of symmetry by «δ»$^{-1}$ = «δ^{-1}» and composition by «δ_1» $\circ$ «δ_1» = «$\delta_2 \circ \delta_1$» is similar. The groupoid equations hold because they hold for definitional equality.

Observe that $\int_{\langle\Gamma\rangle}\langle A\rangle$ is presented by $\Gamma, x : A$. Because $\int_{\langle\Gamma\rangle}\langle A\rangle$ is only defined when the base is presented by a context Γ, and $\langle A\rangle : \langle\Gamma\rangle \longrightarrow$ *GPD*, the equation for $[\![\Gamma\ , x{:}A]\!]$ has tacit side conditions that $[\![\Gamma]\!]$ is presented by Γ, and that $[\![A]\!] : [\![\Gamma]\!] \longrightarrow$ *GPD*. In the proof of the fundamental theorem, we show that these are satisfied for well-formed syntax.

Types We define $[\![2]\!]$ to be the constant functor returning the groupoid $\mathbf{2}$, the discrete groupoid with two objects, «true» and «false», and only identity arrows «$\mathsf{refl}_{\mathsf{true}}$» and «$\mathsf{refl}_{\mathsf{false}}$». A priori, it is not necessarily the case that all terms of type 2 are objects of this groupoid—only the ones that are equal to true or false—but the fundamental theorem will show that, in fact, they all are. To see that this is presented by 2, we must show: (1) that for any θ, $\mathbf{2}$ is presented by $2[\theta]$. Because "presented by" respects definitional equality, and $2[\theta] \equiv 2$, it suffices to show that it is presented by 2, which it is—the conditions on inverses and composition follow from *uip*. (2) That for any δ, the identity functor (which is the action of a constant functor) is presented by $x.\mathsf{map}_2\ \delta\ x$. Again using respect for equality, by the definition of map for 2, it suffices to show that the identity functor is presented by $x.x$—which it is, using the β rules for $x[\theta]$ and $x[\delta]$.

Similarly, we define $[\![\mathsf{set}]\!]$ to be the constant functor delivering the groupoid $\mathbf{set}$ with one object, «bool», and two arrows, «$\mathsf{refl}_{\mathsf{bool}}$» and «$\mathsf{not}$», such that «$\mathsf{refl}_{\mathsf{bool}}$» is the identity and «$\mathsf{not}$»$^{-1}$ = «not» (the definitions of the remaining compositions are forced by the groupoid laws). The verification that it is presented by set is analogous to the above for 2.

For $\mathbf{El}$, first observe that because $\mathbf{const}(\mathbf{set})$ is a constant functor, an $M \in RedTm^{\langle\Gamma\rangle}\ \mathbf{const}(\mathbf{set})$ determines a functor $M^* : \langle\Gamma\rangle \longrightarrow \mathbf{set}$. Thus, it suffices to define a functor $\mathbf{El} : \mathbf{set} \longrightarrow$ *GPD* and interpret $El(M)$ by composition. For our simple universe, $\mathbf{El}$ is defined as follows: On objects, take $\mathbf{El}$(«bool») to be the discrete groupoid with two objects, «$\mathsf{in\,true}$» and «$\mathsf{in\,false}$». For a richer universe, with more type constructors, this definition would be extended to analyze the other possible codes for types. A functor between discrete groupoids is determined by a function between their sets of objects, so for morphisms, take $\mathbf{El}(«\mathsf{refl}_{\mathsf{bool}}»)$ to be the identity function, and $\mathbf{El}(\mathsf{not})$ to be the function that interchanges «in true» and «in false». This is well-defined because, by the semantic consistency argument, $\mathsf{true} \not\equiv \mathsf{false}$ and $\mathsf{refl}_{\mathsf{bool}} \not\equiv \mathsf{not}$, so we can map these equivalence classes to different results. It is simple to verify that this obeys the functor laws. It remains to check that $\mathbf{El}[«M»]$ is presented by $El(M)$. We have two obligations:

1. For all $«\theta» \in Ob\ \langle\Gamma\rangle$, $\mathbf{El}[«M»](\theta)$ is presented by $El(M)[\theta]$. Because M is reducible, $«M[\theta]»$ is an object of $\mathbf{set}$, and therefore equals «bool». This means that $M[\theta] \equiv \mathsf{bool}$, so in the syntax we can derive that $El(M)[\theta] \equiv El(\mathsf{bool})$ by pushing the substitution inside, and then using congruence and transitivity. The verification that $\mathbf{El}(«\mathsf{bool}»)$ is presented by $El(\mathsf{bool})$ is analogous to the above.
2. For all $«\delta» : «\theta_1» \longrightarrow «\theta_2»$, $\mathbf{El}[«M»](«\delta»)$ is presented by $x.\mathsf{map}_{El(M)}\ \delta\ x$. Due to discreteness/*uip*, the morphism part is trivial, so we show the objects, which amounts to: assuming $«N» \in Ob\ \mathbf{El}[«M»](\theta_1)$, we must show that $«\mathsf{map}_{El(M)}\ \delta\ N» \in Ob\ \mathbf{El}[«M»](\theta_2)$. As in the first part, we know that $\mathbf{El}[«M»](\theta_1)$ and $\mathbf{El}[«M»](\theta_2)$ are both $\mathbf{El}(«\mathsf{bool}»)$, and that $El(M)[\theta_1] \equiv El(M)[\theta_2] \equiv El(\mathsf{bool})$. Moreover, because M is reducible, $«M[\delta]»$ is a morphism in $\mathbf{set}$, which means that it is either $«\mathsf{refl}_{\mathsf{bool}}»$ or «not». In the first case, $(x.\mathsf{map}_{El(M)}\ \delta\ x) \equiv (x.\mathsf{map}_{El(M)}\ \mathsf{refl}\ x) \equiv x.x$ by *0-resp functoriality*. In the second, $(x.\mathsf{map}_{El(M)}\ \ /x)$ agrees with the semantic negation function on «in true» and «in false» by *0-resp* for $El(-)$.

To interpret $A[\theta]$, we overload $F[G]$ to mean functor composition. We verify that, when $\langle A\rangle : \langle\Delta\rangle \longrightarrow GPD$ and $\theta \in RedSubst\ \langle\Gamma\rangle\ \langle\Delta\rangle$, $\langle A\rangle[\theta^*]$ is presented by $A[\theta]$.

1. First, for an object $«\theta'» \in \langle\Gamma\rangle$, we must show that $\langle A\rangle(\theta^*(«\theta'»))$ is presented by $(A[\theta])[\theta']$. By respect for equality, it suffices to show presentation by $A[\theta[\theta']]$. By definition of θ^*, $\theta^*(«\theta'») = «\theta[\theta']» \in Ob\ \langle\Delta\rangle$. Thus, by definition of $\langle A\rangle$, $\langle A\rangle(«\theta[\theta']»)$ is presented by $A[(\theta[\theta'])]$, as we needed to show.
2. Second, for a morphism $«\delta» : «\theta_1» \longrightarrow_{\langle\Gamma\rangle} «\theta_2»$, we must show that $\langle A\rangle[\theta^*](«\delta»)$ is presented by $x.\mathsf{map}_{A[\theta]}\ \delta\ x$. This follows from the equation for map for composition, equating it to $x.\mathsf{map}_A\ (\mathsf{refl}_\theta[\delta])\ x$.

Finally, we come to $\mathbf{\Pi}_A\ B$. First we interpret closed Π-types: given a groupoid $\langle A\rangle$ and a functor $\langle B\rangle : \langle x : A\rangle \longrightarrow GPD$, we construct a groupoid $\pi_{\langle A\rangle}\ \langle B\rangle$, which is the dependent analogue of the functor category $\langle B\rangle^{\langle A\rangle}$:

1. An object is an equivalence class $«\lambda\, x.\, M»$, where $M \in RedTm\ \langle x : A\rangle\langle B\rangle$.
2. A morphism $«\lambda\, x.\, M» \longrightarrow_{(\pi_{\langle A\rangle}\ \langle B\rangle)} «\lambda\, x.\, N»$ is $«\lambda\, x.\, \alpha»$, where $\alpha \in RedEquiv\ M\ N$.
3. Identity at $«\lambda\, x.\, M»$ is $«\mathsf{refl}_{\lambda\, x.\, M}»$. To see that this is a morphism, we have to show that it is the equivalence class of a function, whose body is reducible. But $«\mathsf{refl}_{\lambda\, x.\, M}» = «\lambda\, x.\, \mathsf{refl}_M»$, so it suffices to show that $\mathsf{refl}_M \in RedEquiv\ M\ M$. By definition, this means that for any $\delta : \theta_1 \longrightarrow_{\langle x : A\rangle} \theta_2$, $«M[\delta]» : «\mathsf{map}_B\ \delta\ M[\theta_1]» \longrightarrow «M[\theta_2]»$. But this follows immediately from $M \in RedTm\ B$, which is necessary for $«\lambda\, x.\, M»$ to be an object. The definitions of inverse and composition are analogous; the groupoid equations hold because they hold for definitional equality.

Observe that $\pi_{\langle A\rangle}\ \langle B\rangle$ is presented by $\Pi\, x{:}A.\, B$.

Next, for a functor $\langle B\rangle : \langle\Gamma\ ,\ x{:}A\rangle \longrightarrow GPD$, its restriction to $«\theta» \in Ob\ \Gamma$, is the functor $\langle B\rangle|_{«\theta»} : \langle x : A(«\theta»)\rangle \longrightarrow GPD$ defined by

$$\begin{aligned}\langle B\rangle|_{«\theta»}«(M/x)» &= \langle B\rangle«(\theta, M/x)»\\ \langle B\rangle|_{«\theta»}«(\alpha/x)» &= \langle B\rangle«(\mathsf{refl}_\theta, \alpha/x)»\end{aligned}$$

Observe that $\langle B\rangle|_{«\theta»}$ is presented by $B[\theta, x/x]$. The restriction thus corresponds to holding the Γ part of the functor fixed at a particular object.

Now we can define the functor $\mathbf{\Pi}_{\langle A\rangle}\ \langle B\rangle : \langle\Gamma\rangle \longrightarrow GPD$, given $\langle A\rangle : \langle\Gamma\rangle \longrightarrow GPD$ and $\langle B\rangle : \langle\Gamma\ ,\ x{:}A\rangle \longrightarrow GPD$:

$$\begin{aligned}\mathbf{\Pi}_{\langle A\rangle}\ \langle B\rangle(«\theta») &= \pi_{\langle A\rangle(«\theta»)}\ \langle B\rangle|_{«\theta»}\\ \mathbf{\Pi}_{\langle A\rangle}\ \langle B\rangle(«\delta» : «\theta_1» \longrightarrow_{\langle\Gamma\rangle} «\theta_2») &= \langle x.\mathsf{map}_{\Pi\, x:A.\, B}\ \delta\ x\rangle\end{aligned}$$

First, observe that for each $«\theta»$, $\mathbf{\Pi}_A\ B(«\theta»)$ is presented by $\Pi\, x{:}A.\, B[\theta] \equiv \Pi\, x{:}A[\theta].\, B[\theta, x/x]$. Second, we must verify that $x.\mathsf{map}_{\Pi\, x:A.\, B}\ \delta\ x$ is reducible for any δ, so that it extends to a functor. Observe that the converse holds: If the action on morphisms of a functor $\langle C\rangle$ is presented by map_C , then for any δ, $x.\mathsf{map}_C\ \delta\ x$ is reducible. Thus, by the assumptions about $\langle A\rangle$ and $\langle B\rangle$, map_A and map_B are reducible for any δ. By the definition of map for Π, we have that

$$\mathsf{map}_{\Gamma.\Pi\, x:A.\, B}\ \delta\ f \equiv \lambda\, x.\ \mathsf{map}_{\Gamma, x:A.B}\ (\delta, \mathsf{refl})\ (f\ (\mathsf{map}_{\Gamma.A}\ \delta^{-1}\ x))$$

By respect for equality, it suffices to show that the RHS is reducible. This follows from the definition of π and the reducibility of map_A and map_B .

Using these definitions, it is simple to check the cases of the fundamental theorem for parts 1, 2, 5, and 6. For parts 1 and 5, in each case the inductive hypotheses satisfy the pre-conditions of the designated semantic construction, and we have already verified that these constructions define groupoids that are appropriately syntactically presented. For parts 2 and 6: the congruence rules hold because the interpretation is defined compositionally; the equivalence relation rules hold because equality in the meta-language is an equivalence relation. The only non-trivial equations between types are the rules that commute substitution with type formers (e.g. $\Pi\, x{:}A.\, B[\theta] \equiv \Pi\, x{:}A[\theta].\, B[\theta, x/x]$), which are simple to verify for the above definitions. Observe that the proof does not descend into equations between terms, because the only source of dependency is $El(M)$, where $[\![El(M)]\!] \equiv \mathbf{El}[M^*]$, and M^* automatically respects equality in M.

3.2.3 Reducibility of Substitutions/Terms and Equivalences

Finally, we must show that each θ, M, δ and α is reducible. In general: To show that an introduction forms is reducible, we will argue that $[\![-]\!]$ is defined to be "intro forms whose subterms are reducible", and that the subterms will be reducible by the inductive hypotheses. To show that an elimination form is reducible, we will argue that $[\![-]\!]$ tells you that it suffices to check the β-redices, that the result of reduction is reducible by induction, and that the elim. form is reducible because definitional equality contains reduction. To show that the judgemental operations are reducible, we will observe that syntactic presentation of $[\![-]\!]$ ensures that all closed instances of these operations exist, and that, using associativity laws, the definition of reducibility reduces to checking closed instances. We show some representative cases for δ, M, and α; the other cases, including those for θ, are available in the extended version [13].

Equivalences between Substitutions The case for refl follows from $\theta \in RedSubst\ [\![\Gamma]\!]\ [\![\Delta]\!]$ and *delegate*. The case for δ^{-1} reduces to symmetry of morphisms of $[\![\Gamma]\!]$ and $[\![\Delta]\!]$ using the equation $\delta^{-1}[\delta'] \equiv \delta[\delta'^{-1}]^{-1}$, which is derivable from interchange. The case for $\delta \circ \delta'$ is similar, using interchange to push the compositions inward. The case for $\theta[\delta]$ uses *1-resp assoc* to associate, while the case for $\delta[\delta']$ uses *2-resp assoc* to associate. The case for $(\delta, \alpha/x)$ uses the definition of *2-resp* for such pairs.

Terms Case for $\mathsf{map}_A\ \delta\ M$: On objects, we must show that «$(\mathsf{map}_A\ \delta\ M)[\theta]$» $\in$ $Ob\ [\![A[\theta_2]]\!](«\theta»)\ =\ Ob\ [\![A[\theta_2[\theta]]]\!]$. By *1-subst for* map, it suffices to show that «$\mathsf{map}_A\ \delta[\mathsf{refl}_\theta]\ M[\theta]$» is. This follows from the inductive hypotheses for M and δ, which show that their instances by θ are a morphism and object of the appropriate categories, and from syntactic presentation of $[\![A]\!]$, which shows that map_A is functorial, and therefore reducible. The argument for morphisms is similar, using *1-resp for* map.

Case for $\lambda\, x.\, M$: On objects, we must show that «$\lambda\, x.\, M[\theta]$» $\in$ $Ob\ [\![\Pi\, x{:}A.\, B]\!](«\theta»)\ =\ Ob\ \pi_{[\![A[\theta]]\!]}\ [\![B[\theta, x/x]]\!]$. By def. *subst*, this equals «$\lambda\, x.\, M[\theta, x/x]$». So it suffices to show that $x.M \in RedTm^{[\![x\,:\,A[\theta]]\!]}\ [\![B[\theta, x/x]]\!]$. The obligations for both objects and morphisms follow from the IH $M \in RedTm^{[\![\Gamma, x\,:\,A]\!]}\ [\![B]\!]$, holding the Γ part fixed at θ. The action on morphisms is similar, using «$(\lambda\, x.\, M)[\delta]$» = «$\lambda\, x.\, M[\delta, \mathsf{refl}_x/x]$».

Case for $M_1\ M_2$: On objects, we must show that «$M_1\, M_2[\theta]$» $=$ «$M_1[\theta]\ M_2[\theta]$» $\in$ $Ob\ [\![B[M_2/x]]\!](«\theta»)\ =\ Ob\ [\![B[\theta, M_2[\theta/x]]]\!]$. By the inductive hypothesis, we know that «$M_1[\theta]$» = «$\lambda\, x.\, M_1'$» where M_1' is reducible, and that «$M_2[\theta]$» is an appropriate object. Thus, «$M_1'[M_2[\theta]/x]$» = «$(\lambda\, x.\, M_1')\ M_2[\theta]$» = «$M_1[\theta]\ M_2[\theta]$» is an object of the result. The morphism part is analogous, because $M_1\ M_2[\delta] = M_1[\delta]\ M_2[\delta]$.

We omit the cases for $M[\theta]$, x, bool, true, false, if, in, and out.

Equivalences between Terms The cases for the judgemental framework operations (refl, α^{-1}, $\alpha \circ \alpha'$, $M[\delta]$, $\alpha[\delta]$) are analogous to those for substitutions above.

Case for $\lambda\, x.\alpha$: Given δ, «$(\lambda\, x.\, \alpha)[\delta]$» = «$\lambda\, x.\, \alpha[\delta, \mathsf{refl}_x/x]$», so it suffices to show that $\alpha[\delta, \mathsf{refl}_x/x]$ is reducible. Picking a substitution «β/x», the obligation is to show that «$\alpha[\delta, \mathsf{refl}_x/x][\beta/x]$» is a morphism. This holds by reassociating the substitution, and then using the IH that α is reducible.

Case for $\alpha\ \beta$: Pick «δ» : «θ_1» $\longrightarrow_{[\![\Gamma]\!]}$ «θ_2». To show:

$$«\alpha\ \beta[\delta]» : \mathsf{map}_{B[M_2/x]}\ \delta\ (M_1\ M_2)[\theta_1] \longrightarrow_{[\![B[N_1/x][\theta_2]]\!]} (N\ N_1)[\theta_2]$$

By equality, this is«$(\alpha[\delta])\ (\beta[\delta])$». The IH gives that α,β are reducible, so

$$\begin{aligned}\alpha[\delta] &: «\mathsf{map}_{\Pi\, x:A.\, B}\ \delta\ M[\theta_1]» \longrightarrow_{[\![\Pi\, x:A.\, B]\!](«\theta'»)} «N[\theta_2]»\\ \beta[\delta] &: «\mathsf{map}_A\ \delta\ M_1[\theta_1]» \longrightarrow_{[\![A]\!]} «N_1[\theta_2]»\end{aligned}$$

To complete the case, we (1) show that if «α'» is a morphism of $\pi_{\langle X\rangle}\ \langle Y\rangle$ and «β'» is a morphism of $\langle X\rangle$, then «$\alpha'\ \beta'$» is a morphism of $\langle Y\rangle$—the argument uses *2-β-expansion*, analogously to the case for application above. (2) deduce that

$$\begin{aligned}&(\mathsf{map}_{\Pi\, x:A.\, B}\ \delta\ (M[\theta_1]))\ (\mathsf{map}_A\ \delta\ (M_1[\theta_1]))\\ &\equiv \mathsf{map}_{B[M_2/x]}\ \delta\ (M_1\ M_2)[\theta_1]\end{aligned}$$

—the proof uses the definition of map for Π, functoriality of map, cancellation of inverses, and the *def.* map *for* $A[\theta]$. This shows that «$(\alpha[\delta])\ (\beta[\delta])$» is a morphism of the appropriate type.

We omit the cases for if and not.

3.2.4 Canonicity

Here, we check that Theorem 3.1 is a corollary of the fundamental theorem. Assume $\cdot\ \vdash M : 2$. Then $M \in RedTm^{[\![\cdot]\!]}\ [\![2]\!]$, so $M \in RedTm^{\cdot}\ \mathbf{const}(\mathbf{2})$. By definition, «id.» $\in Ob\ \cdot$, so «M[id.]» $\in$ $Ob\ (\mathbf{const}(\mathbf{2})(«\text{id.}»))$. But $M[\text{id.}] \equiv M$ and $\mathbf{const}(\mathbf{2})(«\text{id.}») =$ $\mathbf{2}$, so «M» $\in Ob\ \mathbf{2}$. By definition of $\mathbf{2}$, this means «M» = «true» or «M» = «false», which means that $M \equiv \mathsf{true}$ or $M \equiv \mathsf{false}$.

4. Identity Types

In this section, we show how to internalize the judgemental notion of equivalence as an identity type $\mathsf{Id}_A\ M\ N$, satisfying the usual rules. The rules for the identity type are presented in Figure 6. in, out, and the $\beta\eta$ rules for them state that the inhabitants of the identity type $\mathsf{Id}_A\ M\ N$ are exactly the equivalences $M \simeq_A N$. The *reflection* and *uip* rules express two-dimensionality: equivalences themselves are equivalent only if they are equal, and all equivalences between equivalences are the identity. These rules express *strict* 2-dimensionality, and are an inherent source of undecidability for a strictly 2-dimensional theory; Garner [9] discusses this point further. The *0-resp* rule expresses the action of the Hom-functor, given by pre- and post-composition. $\mathsf{out}\,M$ determines a 2-cell from a one-cell, and is the first mechanism in our calculus for introducing variable equivalences. Because *2-resp* is determined by the structure of the equivalence being substituted into, $(\mathsf{out}\,M)[\delta]$ is stuck until M is determined. Canonicity remains true in the presence of the identity type, interpreting $[\![\mathsf{Id}_A\ M\ N]\!]$ as the Hom-functor.

Surprisingly, we can derive the standard J elimination rule for the identity type. In traditional presentations of MLTT, map (which is usually called subst) does not entail J. In 2TT, it does, because of the following properties of the judgemental presentation of equivalence:

1. It axiomatizes the 2-cell structure of types using the operations refl, $-^{-1}$, $- \circ -$ $M[\delta]$, and map_C . Our derivation of J uses unit law equations for these operations:
$$\begin{aligned}&\mathsf{refl}^{-1} \equiv \mathsf{refl}\\ &M[\mathsf{refl}_\theta] \equiv \mathsf{refl}_{M[\theta]}\\ &\mathsf{refl} \circ \alpha \equiv \alpha\\ &\mathsf{map}_C\ \mathsf{refl}\ M \equiv M\end{aligned}$$
2. It interprets the identity type as the Hom-functor. Our derivation of J exploits the Hom-functor's action, which defines $\mathsf{map}_{\Gamma.\mathsf{Id}_A\ M\ N}$ in terms of pre- and post-composition.
3. It axiomatizes the 2-cell structure of Σ-types as context extension Γ , $x{:}A$. The derivation of J uses the pairing introduction rule for $\simeq_{\Gamma,x:A}$, and the definition of $\mathsf{refl}_{\theta,M/x}$ $\equiv$ $(\mathsf{refl}_\theta, \mathsf{refl}_M/x)$.

In traditional type theory, this 2-cell structure is instead derived from the identity type, which requires taking J as a primitive rule.

We state J in the Paulin-Mohring form [19]:

$$\frac{\begin{array}{c}\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash M : A\\ \Gamma, x : A, p : \mathsf{Id}_A\ M\ x \vdash C\ \mathsf{type}\\ \Gamma \vdash b : C[\mathsf{id}_\Gamma, M/x, \mathsf{refl}_M/p]\\ \Gamma \vdash N : A \quad \Gamma \vdash P : \mathsf{Id}_A\ M\ N\end{array}}{\Gamma \vdash \mathsf{J}_C(b, P) : C[\mathsf{id}_\Gamma, N/x, P/p]} \quad \mathsf{J}_C(b, \mathsf{in}\,\mathsf{refl}_M) \equiv b$$

and define it by

$$\mathsf{J}_C(b, \mathsf{in}\,\mathsf{refl}_M) := \mathsf{map}_{\Gamma, x:A, p:\mathsf{Id}_A\ M\ x.C}\ (\mathsf{refl}_{\mathsf{id}_\Gamma}, \mathsf{out}\,P/x, \mathsf{in}\,(\mathsf{refl}_P)/p)\ b$$

As Awodey has observed,[1] J can be defined in terms of map, given, for any A, M, N, P as in the premises of J, an equivalence

$$(M/x, \mathsf{refl}_M/p) \simeq_{(x\,:\,A, p\,:\,\mathsf{Id}_A\ M\ x)} (N/x, P/p)$$

Homotopically, this says that any path from M with a free endpoint, represented by the pair (N, P), is homotopic to the pair (M, refl) of M itself and the trivial path. Geometrically, (M, refl) can be expanded to (N, P) by dragging M along P.

Here, we observe that this equivalence is in fact provable, by $(\mathsf{out}\,P/x, \mathsf{in}\,(\mathsf{refl}_P)/p)$, using the rules described above. Because we work with total substitutions, we show that

$$\begin{aligned}&(\mathsf{refl}_{\mathsf{id}_\Gamma}, \mathsf{out}\,P/x, \mathsf{in}\,(\mathsf{refl}_P)/p) :\\ &(\mathsf{id}_\Gamma, M/x, \mathsf{refl}_M/p) \simeq_{(\Gamma, x\,:\,A, p\,:\,\mathsf{Id}_A\ M\ x)} (\mathsf{id}_\Gamma, N/x, P/p)\end{aligned}$$

so that map_C coerces the b to the appropriate type. Unpacking the pairing introduction rule for equivalence at Γ , $x{:}A$, our first goal

[1] Personal communication.

$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash \{M,N\} : A}{\Gamma \vdash \mathsf{Id}_A\ M\ N\ \mathsf{type}} \qquad \frac{\Gamma \vdash \alpha : M \simeq_A N}{\Gamma \vdash \mathsf{in}\,\alpha : \mathsf{Id}_A\ M\ N} \qquad \frac{\Gamma \vdash M : \mathsf{Id}_A\ M\ N}{\Gamma \vdash \mathsf{out}\,M : M \simeq_A N} \qquad (\mathsf{refl}_{\mathsf{in}\,\alpha} \text{ is canonical if } \alpha \text{ is}) \qquad \frac{\Gamma \vdash \alpha : P \simeq_{\mathsf{Id}_A\ M\ N} Q}{\Gamma \vdash P \equiv Q} \qquad \frac{\Gamma \vdash \alpha : P \simeq_{\mathsf{Id}_A\ M\ N} Q}{\Gamma \vdash \alpha \equiv \mathsf{refl}_P}$$

$$\begin{array}{rcll|rcll}
\mathsf{out}\,(\mathsf{in}\,\alpha) & \equiv & \alpha & \textit{1-}\beta & (\mathsf{in}\,\alpha)[\theta] & \equiv & \mathsf{in}\,(\alpha[\mathsf{refl}_\theta]) & \textit{1-subst} \\
\mathsf{in}\,(\mathsf{out}\,M) & \equiv & M & \textit{1-}\eta & (\mathsf{in}\,\alpha)[\delta] & \equiv & \mathsf{in}\,(\alpha[\delta]) & \textit{1-resp} \\
(\mathsf{Id}_A\ M\ N)[\theta] & \equiv & \mathsf{Id}_{A[\theta]}\ M[\theta]\ N[\theta] & \textit{0-subst} & \mathsf{refl} & & \text{is canonical} & \\
\mathsf{map}_{\Delta.\mathsf{Id}_A\ M\ N}\ \delta\ P & \equiv & & \textit{0-resp} & -^{-1}, - \circ - & & \text{trivial by extensionality} & \\
\multicolumn{4}{r|}{\mathsf{in}\,(N[\delta] \circ (\mathsf{resp}\ (x.\mathsf{map}_A\ \delta\ x)\ (\mathsf{out}\,P)) \circ M[\delta]^{-1})} & (\mathsf{out}\,M)[\delta] & & \text{stuck until } M \text{ reduces (neutral)} & \textit{2-resp}
\end{array}$$

Figure 6. Identity Types

is to show that $\mathsf{refl}_{\mathsf{id}_\Gamma} : \mathsf{id}_\Gamma \simeq_\Gamma \mathsf{id}_\Gamma$, which it clearly does. Next, the second component of the pair must have type $M \simeq_A N$ (the "adjustment" $\mathsf{map}_{\Gamma.A}\ \mathsf{refl}_{\mathsf{id}_\Gamma}\ M$ cancels by the unit law for map), which $\mathsf{out}\,P$ does. Our final goal is to prove

$$\mathsf{map}_{\Gamma,x:A.\mathsf{Id}_A\ M\ x}\ (\mathsf{refl}_{\mathsf{id}_\Gamma}, \mathsf{out}\,P/x)\ (\mathsf{in}\,\mathsf{refl}) \simeq P$$

In fact, the left-hand-side is definitionally equal to the right-hand, so refl_P gives the result. This follows from the fact that

$$\mathsf{map}_{\Gamma,x:A.\mathsf{Id}_A\ M\ x}\ (\mathsf{refl}_{\mathsf{id}_\Gamma}, \alpha/x)\ Q \equiv \mathsf{in}\,(\alpha \circ \mathsf{out}\,(Q))$$

I.e. map at this type is post-composition with α (the derivation is in the extended version [13]).

The computation rule holds because

$$\mathsf{J}_C(b, \mathsf{refl}) := \mathsf{map}_C\ (\mathsf{refl}_{\mathsf{id}_\Gamma}, \mathsf{out}\,(\mathsf{in}\,\mathsf{refl})/x, \mathsf{in}\,(\mathsf{refl}_P)/p)\ b$$

which, by *1-*β for Id and the definition of refl for $\Gamma\ ,\ x{:}A$, is $\mathsf{map}_C\ \mathsf{refl}\ b$, which is b by the unit law for map.

Conversely, the type-generic operations on equivalence that we have taken as primitive here (map, $M[\delta]$, refl, $-^{-1}\ - \circ -$, $\alpha[\delta]$) can be defined in traditional Martin-Löf type theory using the identity type $\mathsf{Id}_A\ M\ N$ in place of the equivalence judgement $M \simeq_A N$, as consequences of J. However, many of the equations on these operations hold only as higher-dimensional equivalences, rather than as definitional equalities, which breaks definitional canonicity. A resolution of the conjecture that univalent intensional type theory satisfies canonicity up to equivalence [23] is an important area of future work.

5. Related Work

Higher-dimensional type theory is based on a type-directed definition of equivalence, including extensional equality for functions and universes. The idea of defining *equality* in a type directed manner is central to Martin-Löf type theory Martin-Löf [17], especially as presented in NuPRL [5]. Relative to NuPRL, the main benefit of a 2-dimensional theory is that all types respect computationally-relevant notions of equivalence, such as isomorphism of sets; this generalizes NuPRL, where equality is computationally irrelevant.

A similar contrast applies to OTT [3]: In OTT, equality is a computationally irrelevant proposition, so all types are sets—OTT is 1-dimensional. In contrast, 2TT accounts for computationally relevant notions of equivalence—2-dimensional structure. For example, in OTT, coercing any term by any equality proof gives a result that is equal to the original term. This is not true in 2TT, because, for example, not exchanges true and false, which are not equal.

Hofmann and Streicher [12] give intensional type theory a semantics in groupoids. 2TT can be seen as an effort to read this semantics back into the syntax, enriching the type theory with a number of new equations, such as the computation rules for map and resp and the type-directed rules for equivalences. They use the groupoid interpretation to justify an axiomatic account of a universe of types modulo isomorphism, but this extension does not seem to enjoy canonicity.

Hofmann [10] justifies various extensional concepts, such as functional extensionality and quotient types, by a semantics constructed in intensional type theory; Altenkirch [2] adapts this construction to coexist better with other type-theoretic features, such as large eliminations. Canonicity is achieved by defining definitional equality as equality of denotations. Hofmann [10] does not take this approach for the groupoid model, because the groupoid model cannot be defined in intensional type theory without functional extensionality. An alternative to our current work would be to parallel this approach, and define a groupoid interpretation into *extensional* type theory, and thereby inherit equality from the meta-language. The benefit of the approach we take here is that it provides a more direct description of the equational theory, presenting it directly in terms of the source language.

Garner [9] studies a two-dimensional theory *ML*$_2$, and shows it sound and complete for a class of 2-categories. *ML*$_2$ introduced the two-dimensional identity types, with *reflection* and *uip* for identity types only, which we adopted in Section 4. It additionally included some new computation rules for Π-types, which in our notation would be written $\mathsf{refl}_{M\,:\,\Pi\,x:A.\,B} \equiv \lambda\,x.\,\mathsf{refl}_{M\ x}$ and $(\lambda\,x.\,\alpha : M \simeq_{\Pi\,x:A.\,B} N)\ \mathsf{refl}_{M_0} \equiv \alpha[M_0/x]$. However, it did not include, e.g., the type-directed rules for map and thus would not enjoy canonicity in the presence of something like univalence. In future work, we may consider identifying a class of two-categories that is sound and complete for 2TT.

Voevodsky's univalence axiom [23] equips intensional type theory with full homotopy equivalence, which includes isomorphism for sets, equivalence for categories, and so on. However, the axiomatic account does not satisfy canonicity for definitional equality. Voevodsky conjectures that every closed term of type nat is *equivalent* (using univalence) to one that does not use the univalence axiom (and therefore to a numeral, by canonicity for the base theory). However, this conjecture has not yet been proved, and our work suggests an approach to it, as we discuss below. Moreover, Voevodsky conjectures that this numeral can be computed, which is not something we have yet established here.

Our presentation here is based on our previous work on 2DTT [14], a 2-dimensional directed type theory, which generalizes equivalence to an asymmetric notion of transformation. Our presentation here avoids some of the complexities of 2DTT, like the need to account for variances of type constructors, but on the other hand shows that the presentational style can account for symmetry. The proof of canonicity and the derivation of identity types are novel to the present work; we conjecture that our proof of canonicity could be applied to 2DTT as well.

In concurrent work, de Queiroz and de Oliveira [6] have also developed a type theory with a judgemental notion of equivalence, which has explicit evidence with a groupoid structure, and which is internalized by the identity type. Their type theory is weaker (in the category-theoretic sense), in that every equation is part of equivalence—including $\beta\eta$-like rules, which we treat as equalities. However, there is no account of a higher-dimensional base type, nor a claim of canonicity. A number of equations on equivalences are oriented as rewrite rules, which are proved terminating and confluent, but it is unclear what equational theory this decides.

Interestingly, F_C [21], the calculus used as an intermediate language in the Haskell compiler GHC, has some similarities to 2TT. While there are no higher-dimensional types in F_C, type equalities are witnessed by explicit coercions, written in programs using a construct similar to map, and the class of coercions is closed under the 2-category operations considered here, such as refl, $- \circ -$, $-^{-1}$, and $-[-]$. Moreover, GHC includes a coercion simplification algorithm, whose purpose is to reduce the size of coercion terms, which exploits many of the equations on these 2-category operations. An interesting application of 2TT would be to analyze this coercion simplification algorithm by translating its reductions into 2TT equations, which might suggest some additional coercion reductions,

Other related work concerns categorical or homotopy-theoretic semantics of pure intensional type theory. On the homotopy-theoretic side, Awodey and Warren [4], Warren [24] show how to interpret intensional type theory into abstract homotopy theory (i.e. Quillen model categories). Lumsdaine [15] and van den Berg and Garner [22] show that the syntax of intensional type theory forms a weak ω-category, and Gambino and Garner [8] shows that identity types admit a weak factorization system.

6. Conclusion

We have presented a novel formulation of 2-dimensional type theory, 2TT, based on a judgemental account of higher-dimensional structure. This judgemental account reifies the groupoid structure of types and the functorial actions of families. 2TT enjoys a canonicity property for closed terms of observable type, which we have proved by a logical-relations style argument in which types are interpreted as groupoids rather than as equivalence relations. The identity type can be defined by internalizing this judgemental notion of equivalence, and the judgemental apparatus suffices to derive its standard elimination rule.

One direction for future work is to sharpen the canonicity theorem to state that $\cdot \vdash M : \mathsf{bool}$ evaluates to true or to false using a deterministic operational semantics. The result given here leaves open the possibility that the derivation of $M \equiv \mathsf{true}$ or $M \equiv \mathsf{false}$ proceeds by a non-operational rule, such as the various η principles, or an instance of equality reflection. The consistency of the equational theory suggests that such a maneuver cannot be essential, and hence only the β-like rules, including those for map and resp, are relevant to canonicity. A possible route to this result would be to define the groupoid interpretation in *extensional* type theory, and then prove a Plotkin-style computational adequacy theorem for this interpretation to obtain the sharper result.

Another direction for future work is to investigate canonicity for the many possible variations on 2TT: For example, we may extend 2TT with standard features in current proof assistants, such as inductive types. Another possible extension is an impredicative universe. A third possibility is to consider directed type theory [14], which has applications to generic programming with abstract syntax and directed homotopy theory. More ambitiously, we may consider an extension to a fully higher-dimensional univalent dependent type theory, for which canonicity remains an open problem. By analogy with our judgemental formulation, which reifies the groupoid structure of a type, we would present the syntax in such a way that it reifies the weak ω-groupoid structure of a type. However, an obstacle to this generalization is the complexity of the proposed definitions of “weak ω-groupoid” and their adaptation to the type-theoretic setting.

Acknowledgments We thank Thorsten Altenkirch, Jeremy Avigad, Steve Awodey, Caylee Hogg, Chris Kapulkin, Peter Lumsdaine, Gordon Plotkin, Mike Shulman, Kristina Sojakova, and the anonymous reviewers for helpful conversations about this work and feedback on this article.

References

[1] Homotopy type theory Web site. `www.homotopytypetheory.org`, 2011.

[2] T. Altenkirch. Extensional equality in intensional type theory. In *IEEE Symposium on Logic in Computer Science*, 1999.

[3] T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now! In *Programming Languages meets Program Verification Workshop*, 2007.

[4] S. Awodey and M. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 2009.

[5] 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, 1986.

[6] R. J. G. B. de Queiroz and A. G. de Oliveira. Propositional equality, identity types, and direct computational paths. *ArXiv e-prints*, July 2011.

[7] P. Dybjer and A. Filinski. Normalization and partial evaluation. In *Applied Semantics: International Summer School, APPSEM 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 137–192. Springer-Verlag, September 2000.

[8] N. Gambino and R. Garner. The identity type weak factorisation system. *Theoretical Computer Science*, 409(3):94–109, 2008.

[9] R. Garner. Two-dimensional models of type theory. *Mathematical. Structures in Computer Science*, 19(4):687–736, 2009.

[10] M. Hofmann. *Extensional Concepts in Intensional Type Theory.* PhD thesis, University of Edinburgh, 1995.

[11] M. Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

[12] M. Hofmann and T. Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory.* Oxford University Press, 1998.

[13] D. R. Licata and R. Hapner. Canonicity for 2-dimensional type theory (extended version). Technical Report CMU-CS-11-143, Carnegie Mellon University, 2011.

[14] D. R. Licata and R. Harper. 2-dimensional directed type theory. In *Mathematical Foundations of Programming Semantics (MFPS)*, 2011.

[15] P. L. Lumsdaine. Weak ω-categories from intensional type theory. In *International Conference on Typed Lambda Calculi and Applications*, 2009.

[16] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. Rose and J. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975.

[17] P. Martin-Löf. Constructive mathematics and computer programming. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 312(1522):501–518, 1984.

[18] B. Nordström, K. Peterson, and J. Smith. *Programming in Martin-Löf's Type Theory, an Introduction.* Clarendon Press, 1990.

[19] C. Paulin-Mohring. *Extraction de programmes dans le Calcul des Constructions.* PhD thesis, Université Paris 7, 1989.

[20] A. M. Pitts. Categorical logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000.

[21] M. Sulzmann, M. M. T. Chakravarty, S. P. Jones, and K. Donnell. System f with type equality coercions. In *ACM Workshop on Types in Language Design and Implementaion*, 2007. Appendix at `http://research.microsoft.com/ simonpj/papers/ext-f/`.

[22] B. van den Berg and R. Garner. Types are weak ω-groupoids. Available from `http://www.dpmms.cam.ac.uk/ rhgg2/Typesom/Typesom.html`, 2010.

[23] V. Voevodsky. Univalent foundations of mathematics. Invited talk at WoLLIC 2011 18th Workshop on Logic, Language, Information and Computation, 2011.

[24] M. A. Warren. *Homotopy theoretic aspects of constructive type theory.* PhD thesis, Carnegie Mellon University, 2008.