Submitted to POPL 2011

2-Dimensional Directed Dependent Type Theory

Daniel R. Licata[*] Robert Harper [*]
Carnegie Mellon University
{drl,rwh}@cs.cmu.edu

Abstract

The groupoid interpretation of dependent type theory given by Hofmann and Streicher associates to each closed type a category whose objects represent the elements of that type and whose maps represent proofs of equality of elements. The categorial structure ensures that equality is reflexive (identity maps) and transitive (closure under composition); the groupoid structure, which demands that every map be invertible, ensures symmetry. Families of types are interpreted as functors; the action on maps (equality proofs) ensures that families respect equality of elements of the index type. The functorial action of a type family is computationally non-trivial in the case that the groupoid associated to the index type is non-trivial. For example, one may identity elements of a universe of sets up to isomorphism, in which case the action of a family of types indexed by sets must respect set isomorphism. The groupoid interpretation is 2-dimensional in that the coherence requirements on proofs of equality are required to hold "on the nose", rather than up to higher dimensional equivalences. Recent work by Awodey and Lumsdaine, Voevodsky, and others extends the groupoid interpretation to higher dimensions, exposing close correspondences between type theory, higher-dimensional category theory, and homotopy theory.

In this paper we consider another generalization of the groupoid interpretation that relaxes the symmetry requirement on proofs of "equivalence" to obtain a directed notion of *transformation* between elements of a type. Closed types may then be interpreted as categories, and families as functors that extend transformations on indices to transformations between families. Relaxing symmetry requires a reformulation of type theory to make the *variances* of type families explicit. The types themselves must be reinterpreted to take account of variance; for example, a Π type is contravariant in its domain, but covariant in its range. Whereas in symmetric type theory proofs of equivalence can be internalized using the Martin-Löf identity type, in directed type theory the two-dimensional structure must be made explicit at the judgemental level. The resulting *2-dimensional directed dependent type theory*, or *2DTT*, is validated by an interpretation into the strict 2-category Cat of categories, functors, and natural transformations, generalizing the groupoid interpretation. We conjecture that 2DTT can be given semantics in a broad class of 2-categories, and can be extended to make the higher dimensional structure explicit. We illustrate the use of 2DTT for writing dependently typed programs over representations of syntax and logical systems.

[*] This research was sponsored in part by the National Science Foundation under grant number CCF-0702381 and by the Pradeep Sindhu Computer Science Fellowship. 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.

1. Introduction

A type A is defined by introduction, elimination, and equality rules. The introduction and elimination rules describe how to construct and use terms M of type A, and the equality rules describe when two terms are equal. Intensional type theories distinguish two different notions of equality: a judgement of *definitional equality* ($M \equiv N : A$), containing the β- and perhaps some η-rules for the various types, and a type of *propositional equality* ($\mathsf{Id}_A\ M\ N$), which allows additional equalities that are justified by explicit proofs. The type theory ensures that all families of types $x{:}A \vdash C$ type respect equality, in the sense that equal terms M and N determine equal types $C[M/x]$ and $C[N/x]$. Definitionally equal terms give definitionally equal types, whereas propositionally equal terms induce a coercion between $C[M/x]$ and $C[N/x]$:

$$\frac{x{:}A \vdash C\ \mathsf{type} \quad P : \mathsf{Id}_A\ M\ N \quad Q : C[M/x]}{\mathsf{subst}_C\ P\ Q : C[N/x]}$$

The nature of this coercion is explained by the groupoid interpretation of type theory given by Hofmann and Streicher [17]. A closed type is interpreted as a groupoid (a category in which all morphisms are invertible), where the objects of the groupoid are the terms of the type, and the morphisms are proofs of propositional equality between terms. Open types and terms are interpreted as functors, whose object parts are (roughly) the usual types and terms of the set-theoretic semantics, and whose morphism parts show how those types and terms preserve propositional equality. An identity type $\mathsf{Id}_A\ M\ N$ is interpreted using the Hom set of A. Many types, such as natural numbers, are interpreted by discrete groupoids, where the only proofs of propositional equality are identities. Such types satisfy *uniqueness of identity proofs* (UIP) (see Hofmann and Streicher [17] for an introduction), which states that all terms of type $\mathsf{Id}_A\ M\ N$ are themselves equal. However, the groupoid interpretation also permits types of *higher dimension* that have a non-trivial notion of propositional equality.

One example of a higher-dimensional type is a universe, set, a type whose elements are themselves classifiers: associated to each element S of set, there is a type $\mathsf{El}(S)$ classifying the elements of S. The groupoid interpretation permits sets to be considered modulo isomorphism, by taking the propositional equalities between S_1 and S_2 to be invertible functions $\mathsf{El}(S_1) \rightarrow \mathsf{El}(S_2)$. Semantically, set may be interpreted as the category of sets and isomorphisms.[1] This interpretation of set does not satisfy UIP, as there can be many different isomorphisms between two sets. Given this definition of propositional equality, the rule subst states that all type families respect isomorphism: for any $C : \mathsf{set} \rightarrow \mathsf{set}$, $A \cong B$ implies $C[A] \cong C[B]$. Computationally, the lifting of the isomorphism is given by the functorial action of the type family, C.

[1] This works if the sets S themselves are discrete; otherwise, set can be interpreted as the groupoid of small groupoids, which permits non-trivial maps between elements of sets.

The groupoid interpretation accounts for types of dimension 2, but not higher. For example, while the groupoid interpretation permits a universe of sets modulo isomorphism, it does not provide the appropriate notion of equality for a universe containing a universe, where equality should be categorical equivalence, which may be described as "isomorphism-up-to-isomorphism". Recent work has generalized this interpretation to higher dimensions, exploiting connections between type theory and homotopy theory or higher-dimensional category theory (which, under the homotopy hypothesis [5] are two sides of the same coin). On the categorical side, Garner [12] generalizes the groupoid interpretation to a class of 2-categories where the 2-cells are invertible. Lumsdaine [19] and van den Berg and Garner [26] show that the syntax of intensional type theory forms a weak ω-category. On the homotopy-theoretic side, Awodey and Warren [4] show how to interpret intensional type theory into abstract homotopy theory (i.e. Quillen model categories), and Voevodsky's equivalence axiom [27] equips a type theory with a notion of homotopy equivalence, which provides the appropriate notion of equality for types of any dimension.

However, the groupoid interpretation, and all of these generalizations of it, make essential use of the fact that proofs of equivalence are symmetric, interpreting types as groupoids or homotopy spaces. For some applications, it would be useful to consider types with an asymmetric notion of transformation between elements. For example, functors have proved useful for generic programming, because every functor provides a way for a programmer to apply a transformation to the components of a data structure. If we consider a universe setwhose elements are sets S and whose morphisms are functions $f : \mathsf{El}(S_1) \rightarrow \mathsf{El}(S_2)$, then any dependent type $x{:}\mathsf{set} \vdash C\ \mathsf{type}$ describes such a functor, and `subst` can be used to apply a function f to the components of the data structure described by C.

Another application concerns programming with abstract syntax and logical derivations. Abstract syntax can be represented in a dependently typed programming language using *well-scoped de Bruijn indices* [3, 7]. For example, formulas in a first-order logic are represented by a type $\mathsf{prop}(\Psi : \mathsf{ICtx})$, where ICtx is a type representing the context of individual variables (e.g. a list of sorts), and $\mathsf{prop}(\Psi)$ is a type representing the formulas with free variables in Ψ. Structural properties, such as weakening, exchange, contraction, and substitution can be cast as showing that $\mathsf{prop}(-)$ is the object part of a functor from a *context category* [10, 16]. The context category has contexts Ψ as objects, while the choice of morphisms determines which structural properties are provided: variable-for-variable substitutions give weakening, exchange, and contraction; term-for-variable substitutions additionally give substitution. However, these context morphisms are not in general invertible.

In this paper, we propose a new notion of *directed dependent type theory*, which permits an asymmetric notion of transformation between the elements of a type. The contributions of the paper may be summarized as follows:

- Directed type theory differs from conventional type theory in that it must account for *variances* of families of types. In conventional symmetric type theory there is no need to account for variance, because the proofs of equivalence of two indices are invertible. To relax this restriction requires that the syntax distinguish between co- and contra-variant dependencies. This has implications for the type structure as well, so that, for example, dependent function types are contravariant in the domain and covariant in their range.
- Directed type theory exposes higher-dimensional structure at the judgemental, rather than the propositional level. In particular the Martin-Löf identity type is no longer available, because the usual elimination rule implies symmetry, which we explicitly wish to relax. Moreover, in the absence of invertibility, the identity type cannot be formed as a type. We must instead give a judgemental, rather than propositional, account of transformations, and make explicit the action of transformations on families of types.
- Analogously to the groupoid interpretation of type theory given by Hofmann and Streicher [17], we consider only the two-dimensional case of directed dependent type theory, which we call *2DTT*. 2DTT admits a simple interpretation in the category Cat of categories, functors, and natural transformations. Specifically, a closed type denotes a category, whose objects are the elements of the type, and whose maps are (not necessarily invertible) transformations between them. The groupoid interpretations of the dependent type constructors, such as Π and Σ, can be adapted to the asymmetric case by taking careful account of variances. The syntax of 2DTT reflects the fact that Cat is a strict 2-category, in that various associativity, unit, and functoriality laws hold definitionally, rather than propositionally. (A propositional account would require a generalization to higher dimensions, which we do not consider here.)
- A motivating application of 2DTT is programming with dependently typed abstract syntax. The structural properties of variables (weakening, exchange, contraction, substitution) arise as transformations between binding contexts. Structurality of representations of logics, such as a type $\mathsf{nd}\ \Psi\ A$ of derivations of $\Psi \vdash A$ in natural deduction, is represented in 2DTT by functoriality of nd in its context argument Ψ. The categorical semantics of 2DTT provides the usual substitution map from derivations of $\Psi \vdash A$ to derivations of $\Psi' \vdash A[\theta]$, given a transformation θ from Ψ to Ψ'.

Although it is not necessary for the applications we consider here, it seems likely that 2DTT could be extended to higher dimensions, and that more general interpretations of both the two- and higher-dimensional cases are possible. We leave these as interesting directions for future work.

2. Syntax

In this section, we give a proof theory for 2DTT. This proof theory has three main judgements, defining contexts Γ, substitutions θ, and transformations δ. In the semantics given below, these are interpreted as categories, functors, and natural transformations, respectively. Using the terminology of 2-categories, we will refer to a context Γ as a "0-cell", a substitution as a "1-cell", and a transformation as a "2-cell". Each of these three levels has a corresponding contextualized version, which is judged well-formed relative to a context Γ. Contextualized contexts and substitutions are dependent types A and terms M, while contextualized transformations are an asymmetric variant of propositional equality proofs.

Because 2-cell structure is not commonly described type-theoretically, we have chosen to make many rules derivable, rather than admissible, so that the typing rules give a complete account of the theory. For example, we make use of explicit substitutions, which internalize the composition principles of a 2-category, rather than treating substitution as a meta-level operation. The defining equations of substitutions are included as definitional equality rules. However, we leave weakening admissible, as the de Bruijn form that results from explicit weakening is difficult to read.

In this section, we present a base theory with just enough types to illustrate the main ideas; we consider extensions below.

2.1 Formation rules

Contexts and types. As discussed above, variables are annotated with a variance, $^+$ or $^-$, that determines in what positions they may

$$\boxed{\Gamma\ \mathsf{ctx}} \qquad\qquad \boxed{\text{All judgements respect equality}}$$

$$\frac{}{\cdot\ \mathsf{ctx}} \quad \frac{\Gamma\ \mathsf{ctx} \quad \Gamma \vdash A\ \mathsf{type}}{\Gamma\,,\, x{:}A^{+}\ \mathsf{ctx}} \quad \frac{\Gamma\ \mathsf{ctx} \quad \Gamma^{\mathsf{op}} \vdash A\ \mathsf{type}}{\Gamma\,,\, x{:}A^{-}\ \mathsf{ctx}} \quad \frac{\Gamma\ \mathsf{ctx}}{\Gamma^{\mathsf{op}}\ \mathsf{ctx}} \qquad\qquad \frac{\Gamma \equiv \Gamma' \quad \Gamma \vdash J \equiv J' \quad \Gamma' \vdash t : J'}{\Gamma \vdash t : J}$$

$$\boxed{\Gamma \vdash A\ \mathsf{type}}$$

$$\frac{\begin{array}{l}\Gamma \vdash A\ \mathsf{type}\\ \Gamma\,,\, x{:}A^{+} \vdash B\ \mathsf{type}\end{array}}{\Gamma \vdash \Sigma\, x{:}A.\, B\ \mathsf{type}} \quad \frac{\begin{array}{l}\Gamma^{\mathsf{op}} \vdash A\ \mathsf{type}\\ \Gamma\,,\, x{:}A^{-} \vdash B\ \mathsf{type}\end{array}}{\Gamma \vdash \Pi\, x{:}A.\, B\ \mathsf{type}} \quad \frac{}{\Gamma \vdash \mathsf{set}\ \mathsf{type}} \quad \frac{\Gamma \vdash S : \mathsf{set}}{\Gamma \vdash \mathsf{el}\ S\ \mathsf{type}} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash A\ \mathsf{type}}{\Gamma \vdash A[\theta]\ \mathsf{type}}$$

$$\boxed{\Gamma \vdash \theta : \Delta}$$

$$\frac{\Gamma \supseteq \Delta}{\Gamma \vdash \mathsf{id}_{\Delta} : \Delta} \quad \frac{\Gamma_2 \vdash \theta_2 : \Gamma_3 \quad \Gamma_1 \vdash \theta_1 : \Gamma_2}{\Gamma_1 \vdash \theta_2[\theta_1] : \Gamma_3} \quad \frac{\Gamma^{\mathsf{op}} \vdash \theta : \Delta^{\mathsf{op}}}{\Gamma \vdash \theta^{\mathsf{op}} : \Delta} \quad \frac{}{\Gamma \vdash \cdot : \cdot} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma \vdash M : A[\theta]}{\Gamma \vdash \theta, M^{+}/x : \Delta, x : A^{+}} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Gamma^{\mathsf{op}} \vdash M : A[\theta^{\mathsf{op}}]}{\Gamma \vdash \theta, M^{-}/x : \Delta, x : A^{-}}$$

$$\boxed{\Gamma \vdash M : A}$$

$$\frac{x{:}A^{+} \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash M : A}{\Gamma \vdash M[\theta] : A[\theta]} \quad \frac{\Delta\ \mathsf{ctx} \quad \Delta \vdash C\ \mathsf{type} \quad \Gamma \vdash \delta : \theta_1 \Longrightarrow_{\Delta} \theta_2 \quad \Gamma \vdash M : C[\theta_1]}{\Gamma \vdash \mathsf{map}_{\Delta.C}\ \delta\ M : C[\theta_2]} \quad \text{(rules for sets and elements of sets)}$$

$$\frac{\Gamma \vdash M_1 : A \quad \Gamma \vdash M_2 : B[M_1/x]}{\Gamma \vdash (M_1, M_2) : \Sigma\, x{:}A.\, B} \quad \frac{\Gamma \vdash M : \Sigma\, x{:}A.\, B}{\Gamma \vdash \mathsf{fst}\ M : A} \quad \frac{\Gamma \vdash M : \Sigma\, x{:}A.\, B}{\Gamma \vdash \mathsf{snd}\ M : B[\mathsf{fst}\ M/x]} \quad \frac{\Gamma\,,\, x{:}A^{-} \vdash M : B}{\Gamma \vdash \lambda\, x.\, M : \Pi\, x{:}A.\, B} \quad \frac{\Gamma \vdash M_1 : \Pi\, x{:}A.\, B \quad \Gamma^{\mathsf{op}} \vdash M_2 : A}{\Gamma \vdash M_1\ M_2 : B[M_2/x]}$$

$$\boxed{\Gamma \vdash \delta : \theta \Longrightarrow_{\Delta} \theta'}$$

$$\frac{}{\Gamma \vdash \mathsf{id}^{\Delta}_{\theta} : \theta \Longrightarrow_{\Delta} \theta} \quad \frac{\Gamma \vdash \delta_1 : \theta_1 \Longrightarrow_{\Delta} \theta_2 \quad \Gamma \vdash \delta_2 : \theta_2 \Longrightarrow_{\Delta} \theta_3}{\Gamma \vdash \delta_2 \circ \delta_1 : \theta_1 \Longrightarrow_{\Delta} \theta_3} \quad \frac{\Gamma \vdash \delta : \theta \Longrightarrow_{\Delta} \theta' \quad \Gamma_0 \vdash \delta_0 : \theta_0 \Longrightarrow_{\Gamma} \theta'_0}{\Gamma_0 \vdash \delta[\delta_0] : \theta[\theta_0] \Longrightarrow_{\Delta} \theta'[\theta'_0]} \quad \frac{\Gamma^{\mathsf{op}} \vdash \delta : \theta'^{\mathsf{op}} \Longrightarrow_{\Delta^{\mathsf{op}}} \theta^{\mathsf{op}}}{\Gamma \vdash \delta^{\mathsf{op}} : \theta \Longrightarrow_{\Delta} \theta'}$$

$$\frac{}{\Gamma \vdash \cdot : \cdot \Longrightarrow_{\cdot} \cdot} \quad \frac{\Gamma \vdash \delta : \theta \Longrightarrow_{\Delta} \theta' \quad \Gamma \vdash \alpha : (\mathsf{map}_{\Delta.A}\ \delta\ M) \Longrightarrow_{A[\theta']} N}{\Gamma \vdash (\delta, \alpha^{+}/x) : (\theta, M^{+}/x) \Longrightarrow_{\Delta, x:A^{+}} (\theta', N^{+}/x)} \quad \frac{\Gamma \vdash \delta : \theta \Longrightarrow_{\Delta} \theta' \quad \Gamma^{\mathsf{op}} \vdash \alpha : (\mathsf{map}_{\Delta^{\mathsf{op}}.A}\ (\delta^{\mathsf{op}})\ N) \Longrightarrow_{A[\theta]} M}{\Gamma \vdash (\delta, \alpha^{-}/x) : (\theta, M^{-}/x) \Longrightarrow_{\Delta, x:A^{-}} (\theta', N^{-}/x)}$$

$$\boxed{\Gamma \vdash \alpha : M \Longrightarrow_{A} N}$$

$$\frac{}{\Gamma \vdash \mathsf{id}^{A}_{M} : M \Longrightarrow_{A} M} \quad \frac{\Gamma \vdash \alpha_1 : M_1 \Longrightarrow_{\Delta} M_2 \quad \Gamma \vdash \alpha_2 : M_2 \Longrightarrow_{\Delta} M_3}{\Gamma \vdash \alpha_2 \circ \alpha_1 : M_1 \Longrightarrow_{A} M_3} \quad \frac{\Gamma_0 \vdash \delta_0 : \theta_0 \Longrightarrow_{\Gamma} \theta'_0 \quad \Gamma \vdash \alpha : M \Longrightarrow_{A} N}{\Gamma_0 \vdash \alpha[\delta_0] : (\mathsf{map}_{\Gamma.A}\ \delta_0\ (M[\theta_0])) \Longrightarrow_{A[\theta'_0]} N[\theta'_0]}$$

$$\frac{\Gamma\,,\, x{:}(\mathsf{el}\ S)^{+} \vdash M : \mathsf{el}\ S'}{\Gamma \vdash x.M : S \Longrightarrow_{\mathsf{set}} S'} \quad \frac{}{\Gamma \vdash \star : M \Longrightarrow_{\mathsf{el}\ S} M} \quad \frac{\Gamma \vdash \alpha : M \Longrightarrow_{\mathsf{el}\ S} N}{\Gamma \vdash M \equiv N : \mathsf{el}\ S}$$

$$\frac{\Gamma\,,\, x{:}A^{-} \vdash \alpha : (M\ x) \Longrightarrow_{B} (N\ x)}{\Gamma \vdash \lambda\, x.\, \alpha : M \Longrightarrow_{\Pi\, x:A.\, B} N} \quad \frac{\Gamma \vdash \alpha : M \Longrightarrow_{\Pi\, x:A.\, B} N \quad \Gamma \vdash \beta : M_1 \Longrightarrow_{A} N_1}{\Gamma \vdash \alpha\ M_1\ N_1\ \beta : \mathsf{map}^{1}_{B}\ \beta\ (M M_1) \Longrightarrow_{B[N_1/x]} (N N_1)}$$

$$\frac{\begin{array}{l}\Gamma \vdash \alpha_1 : \mathsf{fst}\ M \Longrightarrow_{A} \mathsf{fst}\ N\\ \Gamma \vdash \alpha_2 : (\mathsf{map}^{1}_{B}\ \alpha_1\ (\mathsf{snd}\ M)) \Longrightarrow_{B[\mathsf{fst}\ N/x]} \mathsf{snd}\ N\end{array}}{\Gamma \vdash (\alpha_1, \alpha_2) : M \Longrightarrow_{\Sigma\, x:A.\, B} N} \quad \frac{\Gamma \vdash \alpha : M \Longrightarrow_{\Sigma\, x:A.\, B} N}{\Gamma \vdash \mathsf{fst}\ \alpha : \mathsf{fst}\ M \Longrightarrow_{A} \mathsf{fst}\ N} \quad \frac{\Gamma \vdash \alpha : M \Longrightarrow_{\Sigma\, x:A.\, B} N}{\Gamma \vdash \mathsf{snd}\ \alpha : (\mathsf{map}^{1}_{B}\ (\mathsf{fst}\ \alpha)\ (\mathsf{snd}\ M)) \Longrightarrow_{B[\mathsf{fst}\ N/x]} \mathsf{snd}\ N}$$

Figure 1. Formation Rules

Congruence

All judgements have reflexivity, symmetry, transitivity, and a compatibility rule for each term constructor.

β-η for terms and transformations

$$\begin{aligned}
(\lambda\, x.\, M)\ N &\equiv M[N^{-}/x]\\
M : \Pi\, x{:}A.\, B &\equiv \lambda\, x.\, M\ x\\
\mathsf{fst}\ (M, N) &\equiv M\\
\mathsf{snd}\ (M, N) &\equiv N\\
M : \Sigma\, x{:}A.\, B &\equiv (\mathsf{fst}\ M, \mathsf{snd}\ M)
\end{aligned}$$

(additional rules for elements of sets)

$$\begin{aligned}
(\lambda\, x.\, \alpha_1)\ \alpha_2 &\equiv \alpha_1[\mathsf{id}, {\alpha_2}^{-}/x]\\
\alpha : M \Longrightarrow_{\Pi\, x:A.\, B} N &\equiv \lambda\, x.\, \alpha\ (\mathsf{id}_x)\\
\mathsf{fst}\ (\alpha_1, \alpha_2) &\equiv \alpha_1\\
\mathsf{snd}\ (\alpha_1, \alpha_2) &\equiv \alpha_2\\
\alpha : M \Longrightarrow_{\Sigma\, x:A.\, B} N &\equiv (\mathsf{fst}\ \alpha, \mathsf{snd}\ \alpha)\\
\alpha : S \Longrightarrow_{\mathsf{set}} S' &\equiv x.\mathsf{map}_{a:Set.\mathsf{el}\ a}\ (\cdot, \alpha)\ x\\
\alpha : M \Longrightarrow_{\mathsf{el}\ S} N &\equiv \star
\end{aligned}$$

$\{\Gamma, \theta, \delta\}^{\mathsf{op}}$

$$\begin{aligned}
.^{\mathsf{op}} &\equiv .\\
(\Gamma\ , x{:}A^{\pm})^{\mathsf{op}} &\equiv \Gamma^{\mathsf{op}}\ , x{:}A^{\mp}\\
(\Gamma^{\mathsf{op}})^{\mathsf{op}} &\equiv \Gamma
\end{aligned}$$

$$\begin{aligned}
\mathsf{id}^{\mathsf{op}}_{\Gamma'} &\equiv \mathsf{id}_{\Gamma'^{\mathsf{op}}}\\
(\theta_1[\theta_2])^{\mathsf{op}} &\equiv \theta^{\mathsf{op}}_1[\theta^{\mathsf{op}}_2]\\
(\theta^{\mathsf{op}})^{\mathsf{op}} &\equiv \theta\\
.^{\mathsf{op}} &\equiv .\\
(\theta, M^{\pm}/x)^{\mathsf{op}} &\equiv \theta^{\mathsf{op}}, M^{\mp}/x
\end{aligned}$$

$$\begin{aligned}
\mathsf{id}^{\mathsf{op}}_{\theta} &\equiv \mathsf{id}_{\theta^{\mathsf{op}}}\\
(\delta_1 \circ \delta_2)^{\mathsf{op}} &\equiv \delta^{\mathsf{op}}_2 \circ \delta^{\mathsf{op}}_1\\
(\delta_1[\delta_2])^{\mathsf{op}} &\equiv \delta^{\mathsf{op}}_1[\delta^{\mathsf{op}}_2]\\
(\delta^{\mathsf{op}})^{\mathsf{op}} &\equiv \delta\\
.^{\mathsf{op}} &\equiv .\\
(\delta, \alpha^{\pm}/x)^{\mathsf{op}} &\equiv \delta^{\mathsf{op}}, \alpha^{\mp}/x
\end{aligned}$$

$\mathsf{id}_{\{\Gamma,\theta\}}$

$$\begin{aligned}
\mathsf{id}_{.} &\equiv \cdot\\
\mathsf{id}_{\Gamma, x:A^{\pm}} &\equiv \mathsf{id}_{\Gamma}, x^{\pm}/x
\end{aligned}$$

$$\begin{aligned}
\mathsf{id}_{.} &\equiv \cdot\\
\mathsf{id}_{\theta, M^{\pm}/x} &\equiv \mathsf{id}_{\theta}, \mathsf{id}_{M^{\pm}}/x
\end{aligned}$$

$t[\theta]$ (horizontal composition of objects)

$$\begin{aligned}
t[\theta[\theta']] &\equiv t[\theta][\theta']\\
t[\mathsf{id}_{\Gamma}] &\equiv t\\
\mathsf{id}_{\Gamma}[\theta] &\equiv \theta
\end{aligned}$$

$$\begin{aligned}
\cdot[\theta_0] &\equiv \cdot\\
(\theta, M^{+}/x)[\theta_0] &\equiv \theta[\theta_0], M[\theta_0]^{+}/x\\
(\theta, M^{-}/x)[\theta_0] &\equiv \theta[\theta_0], M[\theta^{\mathsf{op}}_0]^{-}/x\\
(\Sigma\, x{:}A.\, B)[\theta_0] &\equiv \Sigma\, x{:}A[\theta_0].\, B[\theta_0, x^{+}/x]\\
(\Pi\, x{:}A.\, B)[\theta_0] &\equiv \Pi\, x{:}A[\theta^{\mathsf{op}}_0].\, B[\theta_0, x^{-}/x]\\
\mathsf{set}[\theta_0] &\equiv \mathsf{set}\\
(\mathsf{el}\ S)[\theta_0] &\equiv \mathsf{el}\ (S[\theta_0])\\
x[\theta_0] &\equiv \theta_0(x)\\
(\mathsf{map}_{\Delta.C}\ \delta\ M)[\theta_0] &\equiv \mathsf{map}_{\Delta.C}\ \delta[\mathsf{id}_{\theta_0}]\ M[\theta_0]
\end{aligned}$$

(similarly for remaining terms M)

map

$$\begin{aligned}
\mathsf{map}_{\Delta.C}\ \mathsf{id}_{\theta}\ M &\equiv M\\
\mathsf{map}_{\Delta.C}\ (\delta_2 \circ \delta_1)\ M &\equiv \mathsf{map}_{\Delta.C}\ \delta_2\ (\mathsf{map}_{\Delta.C}\ \delta_1\ M)
\end{aligned}$$

$$\begin{aligned}
\mathsf{map}_{\Delta.C[\theta\,:\,\Delta']}\ \delta\ M &\equiv \mathsf{map}_{\Delta'.C}\ \mathsf{id}_{\theta}[\delta]\ M\\
\mathsf{map}_{\Delta.C}\ \delta\ M &\equiv M\ \text{ if }\ \Delta\#C\ \text{(includes } \mathsf{set}\text{)}\\
\mathsf{map}_{\Delta.\Pi\, x:A.\, B}\ \delta\ M &\equiv \lambda\, x.\, \mathsf{map}_{\Delta, x:A^{-}}\ (\delta, \mathsf{id})\ (M\ (\mathsf{map}_{\Delta^{\mathsf{op}}.A}\ \delta^{\mathsf{op}}\ x))\\
\mathsf{map}_{\Delta.\Sigma\, x:A.\, B}\ \delta\ M &\equiv (\mathsf{map}_{\Delta.A}\ \delta\ \mathsf{fst}\ M, \mathsf{map}_{\Delta, x:A^{+}.B}\ (\delta, \mathsf{id}_{\mathsf{map}\ \delta\ (\mathsf{fst}\ M)})\ \mathsf{snd}\ M)\\
\mathsf{map}_{\Delta.\mathsf{el}\ (S)}\ \delta\ M &\equiv N[M^{+}/x]\ \text{ if }\ \mathsf{id}_S[\delta] \equiv x.N : S[\theta] \Longrightarrow_{\mathsf{set}} S'[\theta]
\end{aligned}$$

$h[\delta]$ (horizontal composition of morphisms)

For $h \in \{\delta, \alpha\}$:

$$\begin{aligned}
h[\delta[\delta']] &\equiv h[\delta][\delta']\\
h[\mathsf{id}_{\mathsf{id}}] &\equiv t\\
\mathsf{id}_{\mathsf{id}}[\delta] &\equiv \delta
\end{aligned}$$

$$\begin{aligned}
\delta_1 \circ \delta_2[\delta_3 \circ \delta_4] &\equiv \delta_1[\delta_3] \circ \delta_2[\delta_4]\\
(\mathsf{id}_{o[\theta]})[\delta] &\equiv \mathsf{id}_o[\mathsf{id}_{\theta}[\delta]]\ \text{for}\ o \in \{\theta, M\}
\end{aligned}$$

$$\begin{aligned}
\cdot[\delta_0] &\equiv \cdot\\
(\delta, \alpha^{+}/x)[\delta_0] &\equiv \delta[\delta_0], \alpha[\delta_0]^{+}/x\\
(\delta, \alpha^{-}/x)[\delta_0] &\equiv \delta[\delta_0], \alpha[\delta^{\mathsf{op}}_0]^{-}/x
\end{aligned}$$

$$\begin{aligned}
(x.M)[\delta_0] &\equiv x.\mathsf{map}_{\Delta.S'}\ \delta_0\ M[\theta, x^{+}/x]\ \text{if}\ \Gamma\ , x{:}\mathsf{el}\ S^{+} \vdash M : \mathsf{el}\ S'\ \text{and}\ \delta_0 : \theta \Longrightarrow \theta'\\
\star[\delta_0] &\equiv \star\\
(\lambda\, x{:}A.\, \alpha)[\delta_0] &\equiv \lambda\, x{:}A[\theta'].\, \alpha[\delta_0, \mathsf{id}/a]\\
\alpha_1\ \alpha_2[\delta_0] &\equiv (\alpha_1[\delta_0])\ (\alpha_2[\delta_0])\\
(\alpha_1, \alpha_2)[\delta_0] &\equiv (\alpha_1[\delta_0], \alpha_2[\delta_0, \mathsf{id}])\\
(\mathsf{fst}\ \alpha)[\delta_0] &\equiv \mathsf{fst}\ \alpha[\delta_0]\\
(\mathsf{snd}\ \alpha)[\delta_0] &\equiv \mathsf{snd}\ \alpha[\delta_0]
\end{aligned}$$

Writing $M[\delta]$for $\mathsf{id}_M[\delta]$:

$$\begin{aligned}
x[\delta] &\equiv \delta(x)\\
M[\theta][\delta] &\equiv M[\mathsf{id}_{\theta}[\delta]]\\
(\mathsf{map}_A\ \delta\ M)[\delta_0] &\equiv \text{???}\\
(\lambda\, x.\, M)[\delta] &\equiv \lambda\, x.\, M[\delta, \mathsf{id}]\\
(M\ N)[\delta] &\equiv M[\delta]\ N[\delta]\\
(M, N)[\delta] &\equiv (M[\delta], N[\delta])\\
(\mathsf{fst}\ M)[\delta] &\equiv \mathsf{fst}\ M[\delta]\\
(\mathsf{snd}\ M)[\delta] &\equiv \mathsf{snd}\ M[\delta]
\end{aligned}$$

(additional rules for $S[\delta]$ for $S : \mathsf{set}$)

$h \circ h'$ (vertical composition)

$$\begin{aligned}
(h_3 \circ h_2) \circ h_1 &\equiv h_3 \circ (h_2 \circ h_1)\\
(h \circ \mathsf{id}) &\equiv h\\
(\mathsf{id} \circ h) &\equiv h
\end{aligned}$$

$$\begin{aligned}
\cdot \circ \cdot &\equiv \cdot\\
(\delta_2, {\alpha_2}^{+}/x) \circ (\delta_1, {\alpha_1}^{+}/x) &\equiv (\delta_2 \circ \delta_1), (\alpha_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta_1\ x)\ \alpha_1)^{+}/x\\
(\delta_2, {\alpha_2}^{-}/x) \circ (\delta_1, {\alpha_1}^{-}/x) &\equiv (\delta_2 \circ \delta_1), (\alpha_1 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta^{\mathsf{op}}.A}\ \delta^{\mathsf{op}}_2\ x)\ \alpha_2)^{-}/x
\end{aligned}$$

$$\begin{aligned}
(x.M_1) \circ (x.M_2) &\equiv x.M_2[M_1/x]\\
\star \circ \star &\equiv \star\\
(\lambda\, x.\, \alpha_2) \circ (\lambda\, x.\, \alpha_1) &\equiv \lambda\, x.\, \alpha_2 \circ \alpha_1\\
(\alpha_2, \alpha'_2)[(\alpha_1, \alpha'_1)] &\equiv (\alpha_2 \circ \alpha_1, \alpha'_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ (\mathsf{id}, \alpha_1)\ x)\ \alpha'_1)
\end{aligned}$$

Weakening

$t[\theta, M^{\pm}/x] \equiv t[\theta]$ if $\Delta \vdash t : J$ and $\Gamma \vdash \theta : \Delta$
$t[\delta, \alpha^{\pm}/x] \equiv t[\delta]$ similarly
$\mathsf{map}_{\Delta, x\,:\,A^{\pm}.C}\ (\delta, \alpha^{\pm}/x)\ M \equiv \mathsf{map}_{\Delta.C}\ \delta\ M$ if $\Delta \vdash C$ type

Figure 2. Definitional Equality ($^{\mathsf{op}}$, identity, and composition)

be used. We also require an operation Γ^{op}, which flips the variance of the variables in Γ. Contexts then consist of the empty context $\cdot$, as well as context extension with a variable of either variance. A covariant extension $\Gamma\,,\, x{:}A^{+}$ is well-formed if A is well-formed in Γ, while a contravariant extension $\Gamma\,,\, x{:}A^{-}$ is well-formed if A is well-formed in Γ^{op}.

The judgement $\Gamma \vdash A\ \mathsf{type}$ means that A is a type using the variables in Γ with their designated variances. The types consist of dependent functions (Π) and pairs (Σ), as well as a universe of sets (set) and elements of them ($\mathsf{el}\, S$), and explicit substitution $A[\theta]$. Functions are contravariant in their domain type, so in $\Pi\, x{:}A.\, B$, A is type-checked contravariantly, and x is assumed contravariantly in B. On the other hand, in $\Sigma\, x{:}A.\, B$, x is assumed covariantly in B. In general, a universe contains terms representing classifiers, with the type $\mathsf{el}\, S$ representing their elements. Here, set contains *discrete types*: types whose elements have no non-identity transformations between them (it is also possible to define other forms of universes, as we discuss in 6). However, set itself is *not* a discrete type; it is an example of a base type with a non-trivial notion of transformation. In particular, we take a transformation from S to S' to be a function from $\mathsf{el}\, S$ to $\mathsf{el}\, S'$. Consequently, any type $s : \mathsf{set} \vdash C\ \mathsf{type}$ will admit a lifting of a function from $\mathsf{el}\, S$ to $\mathsf{el}\, S'$ to a transformation from $C[S/s]$ to $C[S'/s]$; thus, we recover the common functor interface used in many programming languages as a special case. The remaining type former, $A[\theta]$, represents an explicit substitution into a type. We use the abbreviation $t[N^{\pm}/x]$ for $t[\mathsf{id}, N^{\pm}/x]$, for substituting for a single variable in any expression t of the calculus.

Weakening is admissible for judgements such as $\Gamma \vdash A\ \mathsf{type}$. To state weakening, we require a notion of looking up a variable in a context:

$$\frac{}{x{:}A^{\pm} \in \Gamma, x{:}A^{\pm}} \qquad \frac{x{:}A^{\pm} \in \Gamma}{x{:}A^{\pm} \in \Gamma, y{:}B^{\pm}} \qquad \frac{x{:}A^{\mp} \in \Gamma}{x{:}A^{\pm} \in \Gamma^{\mathsf{op}}}$$

Next, we write $\Gamma \supseteq \Gamma'$ iff for all $x{:}A^{\pm} \in \Gamma', x{:}A^{\pm} \in \Gamma$. All judgements of the form $\Gamma \vdash J$ satisfy the following:

LEMMA 2.1. *Weakening. If* $\Gamma \vdash J$ *and* $\Gamma' \supseteq \Gamma$ *then* $\Gamma' \vdash J$.

Substitutions and terms. The judgement $\Gamma \vdash \theta : \Delta$ means that θ is a substitution from Γ to Δ. This judgement is well-formed when $\Gamma\ \mathsf{ctx}$ and $\Delta\ \mathsf{ctx}$. Below, we will see that contexts and substitutions between them form a category, so we have the identity substitution id and composition of substitutions $\theta[\theta']$. The next rule states that $^{\mathsf{op}}$ has an action on substitutions. To avoid specializing the context in the conclusion of the rules, we phrase this as an "elimination" rule, removing $^{\mathsf{op}}$ from the two premise contexts. However, $^{\mathsf{op}}$ is an involution: we have a definitional equality between $(\Gamma^{\mathsf{op}})^{\mathsf{op}}$ and Γ, so the following "introduction" rule is derivable:

$$\frac{\Gamma \vdash \delta : \Delta}{\Gamma^{\mathsf{op}} \vdash \delta^{\mathsf{op}} : \Delta^{\mathsf{op}}}$$

The remaining rules substitute for the empty context, and for co- and contravariant context extension. A substitution for $\Delta, x : A^{+}$ is a dependent pair of a substitution θ and a term of type $A[\theta]$. On the other hand, the term substituted for a contravariant assumption must be well-formed contravariantly, so we require a substitution from Γ^{op} to Δ^{op}, which is θ^{op}.

The judgement $\Gamma \vdash M : A$ means that M is a term of type A in Γ. The judgement is well-formed when $\Gamma\ \mathsf{ctx}$ and $\Gamma \vdash A\ \mathsf{type}$. The formation rule for variables x says that a covariant variable may be used; contravariant variables are are turned into covariant variables by Γ^{op}, which allows them to be used in contravariant positions (see the definitional equality rules below). The explicit substitution rule is analogous to that for types.

The map rule allows a transformation to be applied to a term; its type is analogous to the standard subst rule for identity types. Given a type C in context Δ, if M has type $C[\theta_1]$, and there is a transformation from θ_1 to θ_2, then the transformation applied to M has type $C[\theta_2]$. The computational action of map is dependent on the functorial action of each type C. We can define derived forms for replacing the last variable in the context:

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

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

are defined by $\mathsf{map}^{1}_{x:A^{\pm}.B}\ \alpha\ M = \mathsf{map}_{\Gamma,x:A^{\pm}.B}\ \mathsf{id}, \alpha^{\pm}/x\ M$. Note that we use the corresponding abbreviation $t[N^{\pm}/x]$ for $t[\mathsf{id}, N^{\pm}/x]$ for substituting for a single variable.

The remaining rules are variants of the usual rules for Σ and Π. In $\lambda\, x.\, M$, x is hypothesized contravariantly, and in an application the argument is a contravariant position. We give rules for particular sets and elements below.

Transformations. Transformations between substitutions θ and θ' are a variant of propositional equality proofs in intensional type theory. The judgement $\Gamma \vdash \alpha : \theta_1 \Longrightarrow_{\Delta} \theta_2$ is well-formed when $\Gamma\ \mathsf{ctx}$ and $\Delta\ \mathsf{ctx}$ and $\Gamma \vdash \theta_i : \Delta$. Substitutions are reflexive (id) and transitive ($\delta_2 \circ \delta_1$) but not necessarily symmetric. For any given Γ and Γ', the substitutions and transformations between them form a category (the hom-category between θ and θ'), with these as identity and composition. The operation $\delta[\delta']$ says that transformation respects transformation—it allows one formation to be substituted into another, yielding a transformation between the substitution instances. This operation corresponds to what is called horizontal composition of maps in a 2-category, while $\delta_1 \circ \delta_2$ is called vertical composition.

Next, $^{\mathsf{op}}$ has an action on transformations, but it interchanges θ and θ'—it is a contravariant functor on the hom-category. Finally, we give rules for relating the empty substitution to itself, and context extensions if their components are related. The auxiliary judgement $\Gamma \vdash \alpha : M \Longrightarrow_A N$ represents a transformations from the term M to the term N, both of which have type A. For covariant context extension, M and N have different types—$M : A[\theta]$ and $N : A[\theta']$ by the rules for substitution formation—so we cannot compare them directly. The correct rule is to transform the application of δ to M into N. The rule for contravariant extension is similar: we apply δ to N and transform the result into M.

The judgement $\Gamma \vdash \alpha : M_1 \Longrightarrow_A M_2$ represents transformations between terms. It is well-formed when $\Gamma\ \mathsf{ctx}$ and $\Gamma \vdash A\ \mathsf{type}$ and $\Gamma \vdash M_i : A$. Term transformations also have identity, vertical composition, and horizontal composition. Horizontal composition can be used to define the usual resp congruence rule

$$\frac{\Gamma \vdash \alpha : M \Longrightarrow_A N \quad \Gamma \vdash B\ \mathsf{type} \quad \Gamma\,,\, x{:}A^{+} \vdash F : B}{\Gamma \vdash \mathsf{resp}\ F\ \alpha : F[M/x] \Longrightarrow_B F[N/x]}$$

by $\mathsf{resp}\ F\ \alpha = \mathsf{id}_F[\mathsf{id}_{\mathsf{id}}, \alpha^{+}/x]$.

Next, we give introduction and elimination rules for transformations at the various connectives. Transformation from a set S to a set S' is given by a function from $\mathsf{el}\, S$ to $\mathsf{el}\, S'$, represented as an open term. The corresponding elimination rule should allow this function to be applied; $\mathsf{map}_{x:\mathsf{set}.\mathsf{el}\ x}$ does exactly this. Transformation of elements of a set is just equality, expressing that sets are discrete. As an elimination, we give an *equality reflection* rule, which states that a transformation at $\mathsf{el}\, S$ induces a definitional equality. Transformation at Π types is given by extensionality, and eliminated by application. Transformation at Σ is given by pairing, and eliminated by projection. In the type of α_2, the second component must be transformed along α_1 to make the types line up, as in the rules for context extension above.

One issue in syntactic presentations of dependent type theory is whether the rules presuppose or guarantee that their subjects are well-formed—e.g. in $\Gamma \vdash M : A$, is it assumed that $\Gamma \vdash A\,\mathsf{type}$, or does $\Gamma \vdash M : A$ imply $\Gamma \vdash A\,\mathsf{type}$? Each of our rules presupposes that the context Γ is well-formed—that $\Gamma\,\mathsf{ctx}$. Additionally, we employ a convention that ensures that the subjects of each judgement are well-formed: the rules have implicit premises stating that each metavariable occuring free in the rule schema is well-formed, such that under these additional assumptions the subjects of each judgement are be well-formed. For example, the above $\mathsf{snd}\ M$ rule abbreviates the following:

$$\frac{\Gamma \vdash A\,\mathsf{type} \quad \Gamma, x{:}A^{+} \vdash B\,\mathsf{type} \quad \Gamma \vdash M : \Sigma\, x{:}A.\, B}{\Gamma \vdash \mathsf{snd}\ M : B[\mathsf{fst}\ M/x]}$$

Under these assumptions, $\Sigma\, x{:}A.\, B$ and $B[\mathsf{fst}\ M/x]$ are well-formed, as required.

2.2 Equality Rules

Next, we present the equality rules. In addition to the familiar $\beta\eta$-rules for terms, we give an equational behavior to identity, composition, and $^{\mathsf{op}}$, as well as $\beta\eta$-rules for transformations. Our semantics below interprets the calculus into Cat, a strict 2-category, which means that the associativity and unit laws for composition hold up to equality, not up to higher-dimensional morphisms. Correspondingly, we give definitional equality rules for associativity and unit of composition, functoriality of map , and so on.

We define one definitional equality judgement corresponding to each of the six judgements above. To improve readability, we elide the formation premises which ensure that both subjects of the equation are well-formed. We also write $A^{\pm}$ for either A^{+} or A^{-}, with $A^{\mp}$ standing for the opposite.

$\beta\eta$ The $\beta\eta$-rules for terms are standard. The rules for transformations for Π and Σ are analogous to the rules for terms. For set, we give an η-rule that expands any transformation into a map . The only elimination form for transformations at $\mathsf{set}$is map , so the β-like rule will be discussed with it below. For $\mathsf{el}\ S$, we give an η-rule that expands any transformation into $\star$; this is well-typed because of the equality-reflection elimination rule for $\mathsf{el}\ S$.

Op (*$^{\mathsf{op}}$*) Next, we give rules that define $^{\mathsf{op}}$, by giving its action on each Γ, θ, and δ. These rules state that: (1) $^{\mathsf{op}}$ flips the variance bits on context extensions. (2) $^{\mathsf{op}}$ is an involution. (3) $^{\mathsf{op}}$ is an functor which is covariant on 1-cells and contravariant on 2-cells: it preserves identities and compositions, flipping $\delta \circ \delta'$ (the technical term for this is that $^{\mathsf{op}}$ is a 2-functor from the 2-category $Cat$to its two-cell dual Cat^{co}).

Identity (*id*) The rules for identity expand substitutions and transformations at the empty context and context extensions. The following rules for expanding identities at terms are derivable, using the rules below for horizontal composition with the identity:

$$\begin{array}{lcl}
\mathsf{id}^{\mathsf{set}}_{S} & \equiv & x.x \\
\mathsf{id}^{\mathsf{el}\ S}_{\star} & \equiv & \star \\
\mathsf{id}_{\lambda\, x.\, M} & \equiv & \lambda\, x.\, \mathsf{id}_M \\
\mathsf{id}_{M\ N} & \equiv & \mathsf{id}_M\ \mathsf{id}_N \\
\mathsf{id}_{(M,N)} & \equiv & (\mathsf{id}_M, \mathsf{id}_N) \\
\mathsf{id}_{\mathsf{fst}\ M} & \equiv & \mathsf{fst}\ \mathsf{id}_M \\
\mathsf{id}_{\mathsf{snd}\ M} & \equiv & \mathsf{snd}\ \mathsf{id}_M
\end{array}$$

Substitution $t[\theta]$***, i.e. horizontal composition of objects.*** The rules for horizontal composition $t[\delta]$, for t in $\{M, A, \theta\}$ define substitution in a fairly standard manner. The first three rules state that substitution is associative and unital. The remaining rules push the substitution into the remaining terms (the case of $\theta^{\mathsf{op}}$is covered by the functoriality rule for $^{\mathsf{op}}$ above). There are two subtleties: first, in contravariant positions, we substitute by θ^{op}. Second, in the rule for map , we need to recursively substitute into δ. However, pushing a substitution into a transformation with $\delta[\theta]$ can be defined to mean pushing the identity transformation on θ, id_θ, into δ, so we may use the judgement $\delta[\delta_0]$, defined below.

Map (*map *) The first two rules for map express that it preserves identities and vertical compositions. The remaining rules implement the functorial action of each type constructor. The first rule says that the map of a type defined by composition $C[\theta]$is given by reassociating, mapping C over the composition of θ and δ. The next rules says that the action of a constant type is the identity. The rules for Σ η-expands the term and push the transformation inside. When transforming at B, we must extend the transformation, but because we have assumed $x : A^{+}$ covariantly, the required transformation is the identity at $\mathsf{map}\ \delta\ (\mathsf{fst}\ M)$. The rule for Π is similar: for $\delta : \theta \Longrightarrow_{\Delta} \theta'$, it precomposes the given function with the transformation applied contravariantly to $x : A[\theta']$, and then postcomposes with the transformation at B. For the latter, we assume $x : A^{-}$ contravariantly, so that we may extend the transformation by the identity at $\mathsf{map}\ \delta^{\mathsf{op}}\ x$'. Finally, the rule for $\mathsf{el}\ S$ uses the composition $\mathsf{id}_S[\delta]$ to compute an open term, and plugs M into the result.

Substitution of transformations into transformations (*$h[\delta]$*), i.e. horizontal composition of morphisms The first three rules state that horizontal composition is associative and unital, with $\Gamma \vdash \mathsf{id}_{\mathsf{id}} : \mathsf{id} \Longrightarrow_{\Gamma} \mathsf{id}$ as the unit; we do not give a rule for substituting into $\mathsf{id}_\theta$for other θ, as such transformations can be expanded using the identity rules. The next rule states the *middle-four interchange law* for interchanging horizontal and vertical composition. The next rule pushes a horizontal composition into the identity on a horizontal composition. Together with the interchange law, this rule forms the Godement calculus of natural transformations [14]. The next two rules state that substitution for canonical transformations proceeds compositionally.

Next, we give rules for substituting into a transformation between terms α. For substituting $\delta_0 : \theta \Longrightarrow \theta'$ into a transformation $x.M$, we apply M at θ, x^{+}/x, and then use map to transform by δ_0. For the remaining canonical transformations, substitution proceeds compositionally.

The interesting case is $\Gamma \vdash \mathsf{id}^{A}_{M}[\delta] : M[\theta] \Longrightarrow_{\mathsf{map}_{\delta.A}\ \delta\ M} M[\theta']$, which we will abbreviate as $M[\delta]$. In the semantics, this is interpreted using the action on morphisms of a term M. For a variable x, we look the appropriate transformation up in δ. For composition, we can reassociate. For the remaining terms, substitution proceeds compositionally. Below, we give rules for each particular set S, which implement the functorial action of each type constructor. The case of $M : \mathsf{el}\ S$ is covered by the η-rule for transformations of this type.

Vertical composition (*$h_1 \circ h_2$*) The first three rules state that $\delta \circ \delta'$ and $\alpha \circ \alpha'$ are associative and unital. Unlike horizontal composition, the unit of $\delta \circ \delta'$ is $\mathsf{id}_\theta$for an arbitrary θ (not just $\mathsf{id}_{\mathsf{id}}$). On transformations δ, vertical composition proceeds componentwise, though it is necessary to "adjust" the second components of transformations between contexts and Σ-types by the first components—in the semantics, this is explained by the definition of composition for the total category of the Grothendieck construction. The remaining rules define composition for canonical transformations between terms, stating in each case that composition commutes with the transformation constructor.

2.3 Sets

In Figure 3, we give typing and equality rules for Π, Σ, and empty ($\mathsf{0}$), unit ($\mathsf{1}$), and boolean ($\mathsf{2}$) sets. Modulo contravariance, the typing and $\beta\eta$-rules are standard. Contravariance shows up in the formation rules for Π and λ, as above, and additionally in the

$$\frac{\begin{array}{l}\Gamma^{\mathsf{op}} \vdash S_1 : \mathsf{set}\\ \Gamma\,,\, x{:}\mathsf{el}\, S_1{}^{-} \vdash S_2 : \mathsf{set}\end{array}}{\Gamma \vdash \Pi\, x{:}S_1 .\, S_2 : \mathsf{set}} \qquad \frac{\begin{array}{l}\Gamma \vdash S_1 : \mathsf{set}\\ \Gamma\,,\, x{:}\mathsf{el}\, S_1{}^{+} \vdash S_2 : \mathsf{set}\end{array}}{\Gamma \vdash \Sigma\, x{:}S_1 .\, S_2 : \mathsf{set}} \qquad \frac{}{\Gamma \vdash 0, 1, 2 : \mathsf{set}}$$

$$\frac{\begin{array}{l}M_1 : \mathsf{el}\, S\\ M_2 : \mathsf{el}\, S'[M_1/x]\end{array}}{(M_1 , M_2) : \mathsf{el}\, \Sigma\, x{:}S.\, S'} \qquad \frac{M : \mathsf{el}\, \Sigma\, x{:}S.\, S'}{\mathsf{fst}\ M : \mathsf{el}\, S} \qquad \frac{M : \mathsf{el}\, \Sigma\, x{:}S.\, S'}{\mathsf{snd}\ M : \mathsf{el}\, S'[\mathsf{fst}\ M/x]}$$

$$\frac{x{:}\mathsf{el}\, S^{-} \vdash M : \mathsf{el}\, S'}{\lambda\, x.\, M : \mathsf{el}\, \Pi\, x{:}S.\, S'} \qquad \frac{\Gamma \vdash M_1 : \mathsf{el}\, \Pi\, x{:}S.\, S' \quad \Gamma^{\mathsf{op}} \vdash M_2 : \mathsf{el}\, S}{\Gamma \vdash M_1\ M_2 : \mathsf{el}\, S'[M_2/x]}$$

$$\frac{}{() : 1} \qquad \frac{}{\mathsf{true}, \mathsf{false} : 2} \qquad \frac{\Gamma^{\pm} \vdash M : 0}{\Gamma \vdash \mathsf{abort}\, M : C}$$

$$\frac{\begin{array}{ll}\Gamma^{\pm} \vdash M : 2 & \Gamma \vdash M_1 : C[\mathsf{true}/x]\\ \Gamma\,,\, x{:}2^{\pm} \vdash C\ \mathsf{type} & \Gamma \vdash M_2 : C[\mathsf{false}/x]\end{array}}{\Gamma \vdash \mathsf{if}_C(M, M_1 , M_2) : C[M/x]}$$

$$\begin{array}{lcl}
(\lambda\, x.\, M)\ N & \equiv & M[N^{-}/x]\\
M : \mathsf{el}\, \Pi\, x{:}S.\, S' & \equiv & \lambda\, x.\, M\ x\\
\mathsf{fst}\ (M, N) & \equiv & M\\
\mathsf{snd}\ (M, N) & \equiv & N\\
M : \mathsf{el}\, \Sigma\, x{:}S.\, S' & \equiv & (\mathsf{fst}\ M, \mathsf{snd}\ M)\\
M : \mathsf{el}\, 1 & \equiv & ()\\
\mathsf{if}\,(\mathsf{true}, M_1 , M_2) & \equiv & M_1\\
\mathsf{if}\,(\mathsf{false}, M_1 , M_2) & \equiv & M_2
\end{array}$$

$$\begin{array}{lcl}
0, 1, 2[\delta] & \equiv & x.x
\end{array}$$

$$\begin{array}{l}
(\Pi\, x{:}S.\, S')[\delta : \theta \Longrightarrow \theta'] \equiv\\
\ f : \mathsf{el}\, (\Pi\, x{:}S[\theta].\, S'[\theta, x^{-}/x]).\\
\qquad \lambda\, x{:}\mathsf{el}\, S[\theta'].\, \mathsf{map}_{\mathsf{el}\, S'}\ (\delta, \mathsf{id})\ (f\ (\mathsf{map}_{\mathsf{el}\, S}\ \delta^{\mathsf{op}}\ x))\\
(\Sigma\, x{:}S.\, S')[\delta : \theta \Longrightarrow \theta'] \equiv\\
\ p : \mathsf{el}\, (\Sigma\, x{:}S[\theta].\, S'[\theta, x^{+}/x]).(\mathsf{map}_{\mathsf{el}\, S}\ \delta\ \mathsf{fst}\ p, \mathsf{map}_{\mathsf{el}\, S'}\ (\delta, \mathsf{id})\ \mathsf{snd}\ p)
\end{array}$$

Figure 3. Sets

elimination rules for 0 and 2. For example, the rule for if allows the principal premise to be well-formed either in Γ or in Γ^{op} (we abbreviate this choice by $\Gamma^{\pm}$), with the motive of the eliminator, C, well-formed under the appropriate variance extension. This is definable because 2 is discrete. The rule for abort is similar.

When we introduce a set, we must give $\beta\eta$-rules for its elements, and horizontal composition rules that define its functorial action. In the figure, we give $\beta\eta$-rules for $\Pi, \Sigma, 1$ and β-rules for $0, 2$. We also define the functorial action of each type constructor in the expected manner.

3. Semantics

In this section, we give a semantics in Cat, the 2-category of categories, functors, and natural transformations. For notational convenience, we write $[\![\Gamma]\!]$, $[\![\theta]\!]$, etc., but formally the semantics is defined on typing derivations, not raw terms.

The intuition for this interpretation is that a context, or a closed type, is interpreted as a category, whose objects are the members of the type, and whose morphisms are the transformations between members. Thus, a substitution (an "open object") is interpreted as a functor—a family of objects that preserves transformations. A transformation (an "open morphism") is interpreted as a natural transformation—a family of morphisms that respects substitution. More formally, judgements are interpreted as follows:

- $[\![\Gamma]\!]$ is a category
- $[\![\Gamma \vdash \theta : \Delta]\!]$ is a functor $[\![\theta]\!] : [\![\Gamma]\!] \longrightarrow [\![\Delta]\!]$
- $[\![\Gamma \vdash \delta : \theta_1 \Longrightarrow_{\Delta} \theta_2]\!]$ is a natural transformation $[\![\delta]\!] : [\![\theta_1]\!] \Longrightarrow [\![\theta_2]\!] : [\![\Gamma]\!] \longrightarrow [\![\Delta]\!]$

We summarize the interpretation with the following theorem:

THEOREM 3.1. *Soundness*

- *There are total functions $[\![-]\!]$ that for each derivation $\mathcal{D} :: J$ yield a semantic entity of type $[\![J]\!]$.*
- *If $\mathcal{D}_i :: \Gamma \vdash t_i : J$ and $\Gamma \vdash t_1 \equiv t_2 : J$ then $[\![\mathcal{D}_1]\!] = [\![\mathcal{D}_2]\!]$.*

3.1 Interpretation of Formation Judgements

In this section, we describe the interpretation of contexts, substitutions, and transformations. This requires specifying the interpretations of types, terms, and term transformations. We also give the semantics of the judgemental rules for types, terms, and transformations (composition, identity, map). We discuss discuss the semantics of particular types below.

3.1.1 Contexts and types

Semantic Types Before giving the interpretation of contexts, we must state the specification for the interpretation of types. The judgement $\Gamma \vdash A$ type represents an open type. Correspondingly, it should be interpreted as a functor that assigns a closed type to each object of Γ, preserving transformations. Since closed types are represented by categories, this is modeled by a functor into Cat:

$$[\![\Gamma \vdash A\ \mathsf{type}]\!] \text{ is a functor } [\![A]\!] : [\![\Gamma]\!] \longrightarrow Cat$$

In general, we will use the same letter for a piece of syntax and for the semantic concept it is interpreted as; e.g. we will sometimes write Γ for a category, θ for a functor, A for a functor into Cat, etc. Further, we use the same notation for $^{\mathsf{op}}$ and composition in the semantics, writing e.g. $\theta[\theta']$ for composition of functors, $\delta[\delta']$ for horizontal composition of natural transformations, and $\delta \circ \delta'$ for vertical composition. Additionally, we abbreviate $\Gamma \longrightarrow Cat$ by $\mathsf{Ty}\,\Gamma$.

Interpretation of Contexts Contexts are interpreted as follows:

$$\begin{array}{l}
[\![\cdot]\!] = 1\\
[\![\Gamma^{\mathsf{op}}]\!] = [\![\Gamma]\!]^{\mathsf{op}}\\
[\![\Gamma\,,\, x{:}A^{+}]\!] = \int_{[\![\Gamma]\!]} [\![A]\!]\\
[\![\Gamma\,,\, x{:}A^{-}]\!] = (\int_{[\![\Gamma]\!]^{\mathsf{op}}} [\![A]\!])^{\mathsf{op}}
\end{array}$$

The empty context is interpreted as the one-object category 1. Opposite contexts are interpreted as the opposite category.

To interpret context extension, we use the total category given by the Grothendieck construction (see Hofmann and Streicher [17] for an introduction), a categorical analogue of Σ-types. Given a category Γ and a functor $A : \Gamma \longrightarrow Cat$, the Grothendieck construction constructs a *fibration*, i.e. a *total category* $\int_\Gamma A$ along with a functor p from the total category to the indexing category Γ ($\mathsf{p} : \int_\Gamma A \longrightarrow \Gamma$):

- an object of $\int_\Gamma A$ is a pair (σ, m) where $\sigma \in \mathsf{Ob}\ \Gamma$, and $m \in \mathsf{Ob}\ A(\sigma)$.
- a morphism from (σ, m) to (σ', m') is a pair (c, a) where $c : \sigma \longrightarrow_\Gamma \sigma'$ and $a : A(c)m \longrightarrow_{A(\sigma')} m'$.

Identity and composition are defined component-wise, with $\mathsf{id}_{(\sigma,m)} = (\mathsf{id}_\sigma, \mathsf{id}_m)$ (note that this is well-typed because $A(\mathsf{id}) = \mathsf{id}$), and $(c, a) \circ (c', a') = (c \circ c', a \circ A(c)a')$ (which is well-typed because A preserves compositions). We will refer to the fibration $\mathsf{p} : \int_\Gamma A \longrightarrow \Gamma$as a *weakening map*; it is defined by first project on objects and morphisms.

For a covariant context extension $\Gamma\,,\, x{:}A^{+}$, we have that $[\![\Gamma]\!]$ is a category and $[\![A]\!] : [\![\Gamma]\!] \longrightarrow Cat$. Thus, we can interpret the extended context as the total category construction of $[\![A]\!]$. The weakening map p interprets a substitution given by weakening (which is tacit in the syntax) $\Gamma\,,\, x{:}A^{+} \vdash \mathsf{id} : \Gamma$.

For a contravariant context extension $\Gamma\ ,\ x{:}A^{-}$, we have that $[\![\Gamma]\!]$ is a category and $[\![A]\!] : [\![\Gamma]\!]^{\mathsf{op}} \longrightarrow Cat$. Thus, we can form $\int_{[\![\Gamma]\!]^{\mathsf{op}}} [\![A]\!]$. In the syntax, we have weakening for a contravariant context extension as well, so we require a functor $[\![\Gamma\ ,\ x{:}A^{-}]\!] \longrightarrow [\![\Gamma]\!]$. However, $\mathsf{p} : \int_{[\![\Gamma]\!]^{\mathsf{op}}} [\![A]\!] \longrightarrow [\![\Gamma]\!]^{\mathsf{op}}$ faces in the wrong direction. Thus, we interpret $\Gamma\ ,\ x{:}A^{-}$ as $(\int_{[\![\Gamma]\!]^{\mathsf{op}}} [\![A]\!])^{\mathsf{op}}$. For any functor $F : C \longrightarrow D$, there is a functor $F^{\mathsf{op}} : C^{\mathsf{op}} \longrightarrow D^{\mathsf{op}}$ given by the same data, so $\mathsf{p}^{\mathsf{op}} : (\int_{[\![\Gamma]\!]^{\mathsf{op}}} [\![A]\!])^{\mathsf{op}} \longrightarrow [\![\Gamma]\!]$ provides the required weakening map.

Judgemental rules for types The only judgemental rule for types is substitution, which is represented by functor composition: $[\![A[\theta]]\!] = [\![A]\!][[\![\theta]\!]]$.

3.1.2 Substitutions

Semantic Terms One possibility is to identity a term $\Gamma \vdash M : A$ with a one-element substitution $\Gamma \vdash \mathsf{id}, M^{+}/x : \Gamma\ ,\ x{:}A^{+}$. Thus, we would interpret $\Gamma \vdash M : A$ as a functor $\overline{M} : \Gamma \longrightarrow \int_{[\![\Gamma]\!]} [\![A]\!]$ such that the Γ part of the functor is the identity—which we can formalize by saying that $\overline{M}$ is a section of p: $\mathsf{p} \circ \overline{M} = \mathsf{id}$. However, following Hofmann and Streicher [17], it is more convenient to give the following equivalent definition:

DEFINITION 3.2. For a category Γ and a functor $A : \Gamma \longrightarrow Cat$, the set of terms over Γ of type A, written $\mathsf{Tm}\ \Gamma\ A$, consists of pairs (M_o, M_a) such that

- For all $\sigma \in \mathsf{Ob}\ \Gamma$, $M_o(\sigma) \in \mathsf{Ob}\ (A(o))$
- For all $c : \sigma_1 \longrightarrow_{\Gamma} \sigma_2$, $M_a(c) : A(c)(M_o(\sigma_1)) \longrightarrow_{A(\sigma_2)} M_o(\sigma_2)$. Moreover, $M_a(\mathsf{id}) = \mathsf{id}$ and $M_a(c_2 \circ c_1) = M_a(c_2) \circ A(c)(M_a(c_1))$.

A $\mathsf{Tm}\ \Gamma\ A$ is a "dependently typed functor." When A is a constant functor, the definition reduces to that of a functor $M : \Gamma \longrightarrow A$. However, in general, the type of the object that M_o returns depends on its argument, and the action on morphisms relates $M_o(\sigma_2)$ to $(M_o(\sigma_1))$ "adjusted" by applying the functor $A(c)$. As with functors, we elide the projections from M, writing $M(\sigma)$ and $M(c)$.

As Hofmann and Streicher [17] discusses, there is a bijection between functors $\theta : \Gamma \longrightarrow \int_{\Delta} A$ and pairs (θ_1, M) where $\theta_1 : \Gamma \longrightarrow \Delta$ and $M : \mathsf{Tm}\ \Gamma\ A[\theta]$, given by pairing and projection at the meta-level: the above functor p projects the first component, $\mathsf{v} : \mathsf{Tm}\ (\int_{\Gamma} A)\ (A[\mathsf{p}])$ projects the second, and we write (θ_1, M) for the reverse direction.

Now, the rules for substitutions are interpreted as follows:

$$\begin{aligned}
&[\![\mathsf{id}]\!] = \mathsf{p}^{*}\\
&[\![\theta_1[\theta_2]]\!] = [\![\theta_1]\!][[\![\theta_2]\!]]\\
&[\![\theta^{\mathsf{op}}]\!] = [\![\theta]\!]^{\mathsf{op}}\\
&[\![\cdot]\!] = !\\
&[\![\theta, M^{+}/x]\!] = ([\![\theta]\!], [\![M]\!])\\
&[\![\theta, M^{-}/x]\!] = ([\![\theta]\!]^{\mathsf{op}}, [\![M]\!])^{\mathsf{op}}
\end{aligned}$$

The syntactic identity function builds in weakening, so it is interpreted as an appropriate composition of weakening maps p. Substitution is interpreted as composition of functors. $\theta^{\mathsf{op}}$is interpreted as the action of o^{op}n the semantics of θ, which simply "retypes" a functor $C \longrightarrow D$ to a functor $C^{\mathsf{op}} \longrightarrow D^{\mathsf{op}}$. Covariant substitution extension is interpreted using the pairing map discussed above for functors into $\int_{[\![\Gamma]\!]} [\![A]\!]$. Contravariant extension uses pairing as well, inserting $^{\mathsf{op}}$'s to make the types work out.

Judgemental rules for terms To interpret explicit substitution $M[\theta]$, observe that a term $\mathsf{Tm}\ \Delta\ A$ can be composed with a functor $\theta : \Gamma \longrightarrow \Delta$ to give a $\mathsf{Tm}\ \Gamma\ A[\theta]$:

$$\begin{aligned}
(M[\theta])(\sigma) &= M(\theta(\sigma))\\
(M[\theta])(c) &= M(\theta(c))
\end{aligned}$$

Variables x are interpreted using the v and p projections from the total category: $[\![x]\!]$ is v composed with enough projections p to weaken past the remainder of the context.

To interpret map, given a type $C : \mathsf{Ty}\ \Gamma$, a natural transformation $\delta : \theta \Longrightarrow \theta' : \Gamma \longrightarrow \Delta$, and a term $M : \mathsf{Tm}\ \Gamma\ C[\theta]$, we define a term $\mathsf{Tm}\ \Gamma\ C[\theta']$ as follows:

$$\begin{aligned}
(\mathsf{map}_C\ \delta\ M)(\sigma) &= C(\delta_\sigma)(M(\sigma))\\
(\mathsf{map}_C\ \delta\ M)(c : \sigma_1 \longrightarrow_{\Gamma} \sigma_2) &= C(\delta_{\sigma_2})(M(c))
\end{aligned}$$

The action on morphisms has the appropriate type because of the naturality square for $\delta(c)$: the domain of $C(\delta_{\sigma_2})(M(c))$ is $C(\theta'(c))(C(\delta_{\sigma_1})(e(\sigma_1)))$, which equals $C(\theta'(\delta_{\sigma_2}))(C(\theta(c))(e(\sigma_1)))$ by naturality of δ and functoriality of C.

3.1.3 Transformations

Semantic Term Transformations We now define the semantic counterpart of $\Gamma \vdash \alpha : M \Longrightarrow_A N$. We characterized terms M as one-element substitutions, or sections of weakening. Similarly, we can define a transformation between M and N to be a transformation $(\mathsf{id}, M) \Longrightarrow (\mathsf{id}, N) : \Gamma \longrightarrow \int_{\Gamma} A$ that projects to the identity transformation on id. However, it is more convenient to work with the following equivalent definition:

DEFINITION 3.3. Given a category Γ, $A : \mathsf{Ty}\ \Gamma$, and $M, N : \mathsf{Tm}\ \Gamma\ A$, a *dependent natural transformation* $\alpha : M \Longrightarrow N$ consists of a family of maps α_σ such that

- for $\sigma \in \mathsf{Ob}\ \Gamma$, $\alpha_\sigma : M(\sigma) \longrightarrow_{A(\sigma)} N(\sigma)$
- for $c : \sigma_1 \longrightarrow_{\Gamma} \sigma_2$, $N(c) \circ A(c)(\alpha_{\sigma_1}) = \alpha_{\sigma_2} \circ M(c)$

Using this definition, we interpret transformations as follows:

$$\begin{aligned}
&[\![\mathsf{id}]\!] = \mathsf{id}\\
&[\![\delta_1 \circ \delta_2]\!] = [\![\delta_1]\!] \circ [\![\delta_2]\!]\\
&[\![\delta_1[\delta_2]]\!] = [\![\delta_1]\!][[\![\delta_2]\!]]\\
&[\![\delta^{\mathsf{op}}]\!] = [\![\delta]\!]^{\mathsf{op}}\\
&[\![\cdot]\!] = \mathsf{id}\\
&[\![\delta, \alpha^{+}/x]\!] = [\![\delta]\!], [\![\alpha]\!]\\
&[\![\delta, \alpha^{-}/x]\!] = [\![\delta]\!], [\![\alpha]\!]
\end{aligned}$$

Identity, vertical, and horizontal composition are interpreted as their categorical analogues, which we write with the corresponding notation. The family of maps δ_σ comprising a natural formation from F to G also defines a transformation from F^{op} to G^{op}, and we write δ^{op} for this "retyping" in the interpretation of $^{\mathsf{op}}$. To interpret covariant transformation extension, we observe that, because of the interpretation of map, and the interpretation of δ and α, we can define a natural transformation of the appropriate type by $(\delta, \alpha)_\sigma = (\delta_\sigma, \alpha_\sigma)$, and that the commutativity condition follows from commutativity for δ and α. The natural transformation for contravariant context extension is similarly defined by $(\delta, \alpha)_\sigma = (\delta_\sigma, \alpha_\sigma)$.

Judgemental rules for term transformations We define semantic identity, vertical, and horizontal composition as follows:

$$\begin{aligned}
(\mathsf{id}_M)_\sigma &= \mathsf{id}_{M(\sigma)}\\
(\alpha_2 \circ \alpha_1)_\sigma &= \alpha_{2\sigma} \circ \alpha_{1\sigma}\\
(\alpha[\delta])_\sigma &= N(\delta_\sigma) \circ A(c)(\alpha(\theta(\sigma)))
\end{aligned}$$

Identity $\mathsf{id} : M \Longrightarrow M$ is the pointwise identity; naturality holds by the unit laws. Vertical composition of $\alpha_2 : M_2 \Longrightarrow M_3$ and $\alpha_1 : M_1 \Longrightarrow M_2$ is given pointwise as well; naturality holds using naturality for the components and functoriality. For horizontal composition of $\alpha : M \Longrightarrow N$ and $\delta : \theta \Longrightarrow \theta'$, naturality again holds by naturality of the components and functoriality.

Then we interpret $[\![\mathsf{id}_M]\!] = \mathsf{id}_{[\![M]\!]}$, $[\![\beta \circ \alpha]\!] = [\![\beta]\!] \circ [\![\alpha]\!]$, and $[\![\alpha[\delta]]\!] = [\![\alpha]\!][[\![\delta]\!]]$.

3.1.4 Types, Terms, and Transformations

Next, we discuss the interpretation of each type constructor, its terms, and its transformations.

Π Π-types are defined as in Hofmann and Streicher [17]: we follow their construction, checking that everywhere they depend on symmetry of equality, we have inserted the appropriate $^{\mathsf{op}}$'s. For a category Γ and a $A : \mathsf{Ty}\ \Gamma^{\mathsf{op}}$, we abbreviate semantic contravariant context extension $(\int_{\Gamma^{\mathsf{op}}} A)^{\mathsf{op}}$ by $\Gamma.A^{-}$. Given a $B : \mathsf{Ty}\ \Gamma.A^{-}$ and an object $\sigma \in \mathsf{Ob}\ \Gamma$, we define $B_\sigma : \mathsf{Ty}\ A(\sigma)$ by

$$\begin{array}{l}(B_\sigma)(\sigma') = M(\sigma, \sigma') \\ (B_\sigma)(c) = M(\mathsf{id}_\sigma, c)\end{array}$$

For any Γ and A, the $\mathsf{Tm}\ \Gamma\ A$ are the objects of a category with morphisms given by term transformations α. Thus, we define $(\Pi\ A\ B)_\sigma$ to be the category $\mathsf{Tm}\ A(\sigma)\ B_\sigma$. Functoriality is given as in [17], where the contravariance of A replaces the use of inverses. λ and application are interpreted by giving a bijection between $\mathsf{Tm}\ \Gamma.A^{-}\ B$ and $\mathsf{Tm}\ \Gamma\ \Pi AB$. The transformation rules express a bijection between $M \Longrightarrow N : \Gamma \longrightarrow \Pi AB$ and $M\ \mathsf{v} \Longrightarrow N\ \mathsf{v} : \Gamma.A^{-} \longrightarrow B$. The proof follows Hofmann and Streicher [17], Section 5.3, which observes that the groupoid interpretation justifies functional extensionality. The equality rules, validated below, show that this construction gives a dependent product.

Σ Because both subcomponents of $\Sigma\, x{:}A.\, B$ are covariant, the interpretation given in Hofmann and Streicher [17] adapts to our setting unchanged. We define $([\![\Sigma\, x{:}A.\, B]\!])_\sigma$ to be $\int_{A(\sigma)} B_\sigma$, with functoriality defined componentwise. Pairing and projection, and pairing and projection for transformations, following from the definition of the Grothendieck construction.

set $[\![\mathsf{set}]\!]$ is the constant functor returning $Sets$, the category of sets and functions. Because the action on morphisms of the constant functor is the identity, $\mathsf{Tm}\ \Gamma\ \mathsf{set}$is bijective with $\Gamma \longrightarrow Sets$. The particular sets defined above are interpreted as the corresponding constructions in sets, equipped with their usual functorial action:

$$\begin{array}{lcl}[\![k = 0,1,2]\!] & = & const(k) \\ [\![\Pi\, x{:}S.\, S']\!] & = & \Pi[\![S]\!][\![S']\!] \\ [\![\Sigma\, x{:}S.\, S']\!] & = & \Sigma[\![S]\!][\![S']\!]\end{array}$$

where semantic Π and Σ are defined as follows:

$$\begin{array}{lcl}(\Sigma\ S\ S')(\sigma) & = & \Sigma_{m \in S(\sigma)} S'(\sigma, m) \\ (\Sigma\ S\ S')(c) & = & (m, m') \mapsto (S(c)(m), S(c, \mathsf{id})(m')) \\ \\ (\Pi\ S\ S')(\sigma) & = & \Pi_{m \in S(\sigma)} S'(\sigma, m)\} \\ (\Pi\ S\ S')(c) & = & f \mapsto (x \mapsto S(c, \mathsf{id})(f(S(c)(x))))\end{array}$$

To interpret the transformation rule, we must define a transformation $\widetilde{M}$: $S \Longrightarrow S' : \Gamma \longrightarrow Sets$ given a $M : \mathsf{Tm}\ (\int_\Gamma (\mathsf{el}\ S))\ (\mathsf{el}\ S')$:

$$(\widetilde{M})(\sigma) = x \mapsto M(\sigma, x)$$

For a map c, naturality is given by $x \mapsto M(c, \mathsf{id}_x)$.

It is also possible to define a universe types of non-discrete types, interpreted as a category of small categories. However, in this case, a $M : \mathsf{Tm}\ (\int_\Gamma A)\ B$ does not induce a transformation between A and B. The reason is that the action on morphisms of M gives a natural transformation between the two sides of the required naturality square, whereas the naturality condition requires equality of these compositions. We conjecture that a higher-dimensional universe can be handled better by a higher-dimensional generalization of our theory, where these naturality conditions would be up to higher-dimensional structure.

$\mathsf{el}\ S$ $[\![\Gamma\ \vdash\ S : \mathsf{set}]\!]$ amounts to a functor $\Gamma \longrightarrow Sets$, and there is a functor $discrete : Sets \longrightarrow Cat$ that takes each set to the discrete category on that set, and a morphism f between sets to the functor between discrete categories whose action on objects is given by f (the action on morphisms is a just the fact that f takes equals to equals). Thus we define:

$$[\![\mathsf{el}\ S]\!] = discrete \circ [\![S]\!]$$

As usual, we overload notation and write $\mathsf{el}\ S$ for $discrete \circ S$.

On objects, terms $\lambda\, x.\, M$, $M_1\ M_2$, (M, N), $\mathsf{fst}\ M$, $\mathsf{snd}\ M$, $()$, true, and false are interpreted as the corresponding set-theoretic concepts. In each case, the action on maps is just a proof that action on objects respects equality. Because case and abort eliminate towards an arbitrary type, we check them in more detail. For the covariant if $(M : \mathsf{Tm}\ \Gamma\ 2)$, we proceed as follows:

$$\begin{array}{lcl}(\mathsf{if}(M, M_1, M_2))(\sigma) & = & M_1(\sigma) \text{ if } M(\sigma) \\ (\mathsf{if}(M, M_1, M_2))(\sigma) & = & M_2(\sigma) \text{ otherwise} \\ \\ (\mathsf{if}(M, M_1, M_2))(c : \sigma_1 \longrightarrow \sigma_2) & = & M_1(c) \text{ if } M(\sigma_1) \\ (\mathsf{if}(M, M_1, M_2))(c : \sigma_1 \longrightarrow \sigma_2) & = & M_2(c) \text{ otherwise}\end{array}$$

The action on morphisms is well-typed because $M(c)$ shows that $M(\sigma_1) = M(\sigma_2)$. The contravariant if is defined similarly, as 2 is discrete and therefore symmetric. The semantic version of abort is defined by elimination on the empty set.

The transformation rules are justified as follows: $(\star)$ is just the identity transformation; in this case, each component is just reflexivity of equality. To interpret equality reflection, observe that (1) by proof-irrelevance, two $\mathsf{Tm}\ \Gamma\ \mathsf{el}\ S$ are determined entirely by their action on objects, as the action on morphisms produces a proof of equality and (2) the interpretation of the premise says that $[\![M]\!](\sigma) = [\![N]\!](\sigma)$ for all σ.

3.2 Interpretation of Equality Rules

We validate the equality rules as follows: The congruence rules hold because the interpretation is defined compositionally, and meta-level equality is a congruence. The $\beta\eta$-rules for terms are validated as in Hofmann and Streicher [17]. The $\beta\eta$-rules for transformations at Π and Σ hold because the hom-sets for these connectives are meta-level functions and pairs. The rule for $\mathsf{set}$is an η-like rule, and follows from η for functions and pairs at the meta-level. The rule for $\mathsf{el}\ S$ follows from uniqueness of equality proofs at the meta-level. It is simple to verify that $^{\mathsf{op}}$ is an involution, and that it interacts with $\int$ as the rules describe. The identity and composition rules for $^{\mathsf{op}}$ are validated by the fact that it is a 2-functor from the 2-category $Cat$to its two-cell dual Cat^{co}, so it preserves identities and compositions. The identity expansion rules are the definition of identity for 1 and $\int$. For horizontal composition, the first three rules express associativity and unit laws for functor composition. The rules commuting substitution with each term constructor also follow from associativity of functor composition, as the type-theoretic term formers build in a composition. Functoriality of map holds because map is interpreted using the action on morphisms of the functor C. The remaining rules follow from the action on morphisms of the interpretation of each type constructor. The first three rules for horizontal composition of morphisms are validated by associativity and unit of this operation. The next two are the rules of the Godement calculus [14]. The remaining compatibility rules are validated by simple calculations. The rules for $M[\delta]$ follow from the action on morphisms of $[\![M]\!]$. The vertical composition rules follow from associativity and unit, and the meaning of composition for the semantics of the types. The rules for weakening follow from the definition of p, which is inserted in the interpretation of terms that are constant with respect to a variable (the interpretation of these rules requires an inductive argument, as weakening is inserted only at the leaves of the typing derivation).

3.3 Identity types

In the symmetric case, given Γ, $A : \mathsf{Ty}\ \Gamma$, and $M, N : \mathsf{Tm}\ \Gamma\ A$, one can define a $\Gamma \vdash \mathsf{Id}_A\ M\ N$ type by $(\mathsf{Id}_A\ M\ N)(o) = discrete(Hom_{A(o)}(M(o), N(o)))$. However, this family of categories is not functorial in Γ in the asymmetric setting. Consider $\mathsf{Id}_{Sets}\ S\ R$, and note that $Hom_{Sets}(A, B)$ is just $A \to B$. The functoriality condition requires us to produce a a function $S(r) \to R(r)$ given $f : S(s) \to R(s)$, $M(c) : S(s) \to S(r)$, and $N(c) : R(s) \to R(r)$. In the groupoid model, we can take the required map to be $N_c \circ f \circ M_c^{-1}$, but in the absence of symmetry it is not clear how to proceed. This motivates the judgemental approach to transformations that we take in this paper. It may be possible to consider a directed Hom-type, whose first argument is a contravariant position, though the syntactic rules for such a type require further study.

4. Application: Abstract Syntax

In this section, we illustrate the use of directed type theory for programming with abstract syntax and logical derivations. First, we illustrate the main ideas with a standard example of natural deduction derivations. Below, we sketch an application of 2DTT to mixed-variance syntax [18].

Representing syntax To illustrate the main ideas, we represent a very small first-order logic, defined at the top of Figure 4. Contexts Ψ represent the free individual variables of the logic. Contexts are natural numbers, written with zero as ϵ and successor as , i—the approach readily generalizes to sorted syntax. The type of variables, represented as well-scoped de Bruijn indices [3, 7], is indexed by a context Ψ, and represents a pointer into Ψ. To illustrate the binding structure of the language, we require one proposition that binds a variable ($\forall$) and one that allows a variable to be used (an atomic proposition b). For concision, we inline the scoping rules into the grammar, annotating each meta-variable with the context in which it occurs. A *renaming* $r : \Psi \to \Psi'$ is a function that maps variables n^{Ψ} to variables $n^{\Psi'}$. This notion of map gives the structural properties of weakening, exchange, and contraction (mapping variables in Ψ instead to terms in Ψ' would give substitution as well). The identity function is a renaming $\mathsf{id} : \Psi \to \Psi$, and we can pair a renaming $\Psi \to \Psi'$ with a variable $n^{\Psi'}$ to make a renaming $(\Psi, \mathsf{i}) \to \Psi'$, which on z gives n and on $\mathsf{s}(k)$ gives $r(k)$ (i.e. we treat the contexts as coproducts [10]). A renaming induces a meta-operation on propositions: given ϕ^{Ψ} and a renaming $r : \Psi \to \Psi'$, we construct $(\phi\{r\})^{\Psi'}$ by traversing the syntax and applying r to each variable. We give inference rules for $\forall$ introduction and elimination; the latter uses renaming. There is a renaming operation on derivations as well: if $\mathcal{D} :: \Psi \vdash \phi$ and $r : \Psi \to \Psi'$ then $\mathcal{D}\{r\} :: \Psi' \vdash \phi\{r\}$—the renaming of the derivation proves the renaming of the proposition.

To represent this syntax in directed type theory, we first define a type Ctx, whose terms are contexts Ψ and whose transformations are renamings r. After set, this is our second example of a non-discrete base-type. Second, we define a set $\mathsf{var}(\Psi)$ representing variables. Variables are functorial in Ψ, with the action on morphisms given by looking the variable up in the renaming. Third, we define a set $\mathsf{prop}(\Psi)$ representing propositions. It too is functorial in Ψ, with the action on morphisms given by the renaming operation on propositions. Finally, we define a set $\mathsf{nd}\ \Psi\ \phi$. The action on morphisms of nd gives the renaming principle described above. We elide the el around sets in the rules in Figure 4.

The types and terms are as discussed above. We represent renamings with three constructors ϵ (the map out of the empty context), pairing (the map out of a coproduct), and an explicit shift $\uparrow$.

Contexts	Ψ	$::=$	$\epsilon \mid \Psi, \mathsf{i}$
Variables	n^{Ψ}	$::=$	$\mathsf{z}^{\Psi,\mathsf{i}} \mid (\mathsf{s}(n^{\Psi}))^{\Psi,\mathsf{i}}$
Propositions	ϕ^{Ψ}	$::=$	$\forall(\phi^{\Psi,\mathsf{i}}) \mid \mathsf{b}(n^{\Psi}) \mid \ldots$

$$\frac{\Psi, \mathsf{i} \vdash \phi}{\Psi \vdash \forall\phi}\ \forall I \qquad \frac{\Psi \vdash \forall\phi}{\Psi \vdash \phi\{\mathsf{id}, n^{\Psi}\}}\ \forall E \qquad \ldots$$

$\boxed{\text{Contexts}}$

$$\frac{}{\Gamma \vdash \mathsf{Ctx\ type}} \qquad \frac{}{\Gamma \vdash \epsilon : \mathsf{Ctx}} \qquad \frac{\Gamma \vdash \Psi : \mathsf{Ctx}}{\Gamma \vdash \Psi, \mathsf{i} : \mathsf{Ctx}}$$

$$\frac{}{\Gamma \vdash \epsilon : \epsilon \Longrightarrow_{\mathsf{Ctx}} \Psi'} \qquad \frac{\Gamma \vdash \alpha : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi' \quad \Gamma \vdash n : \mathsf{var}(\Psi')}{\Gamma \vdash (\alpha, n) : \Psi, \mathsf{i} \Longrightarrow_{\mathsf{Ctx}} \Psi'}$$

$$\frac{\Gamma \vdash \alpha : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi'}{\Gamma \vdash \uparrow \alpha : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi', \mathsf{i}}$$

$\boxed{\text{Variables}}$

$$\frac{\Gamma \vdash \Psi : \mathsf{Ctx}}{\Gamma \vdash \mathsf{var}(\Psi) : \mathsf{set}} \qquad \frac{}{\Gamma \vdash \mathsf{z} : \mathsf{var}(\Psi, \mathsf{i})} \qquad \frac{\Gamma \vdash n : \mathsf{var}(\Psi)}{\Gamma \vdash \mathsf{s}(n) : \mathsf{var}(\Psi, \mathsf{i})}$$

$\boxed{\text{Propositions and Derivations}}$

$$\frac{\Gamma \vdash \Psi : \mathsf{Ctx}}{\Gamma \vdash \mathsf{prop}(\Psi) : \mathsf{set}} \qquad \frac{\Gamma \vdash n : \mathsf{var}(\Psi)}{\Gamma \vdash \mathsf{b}(n) : \mathsf{prop}(\Psi)} \qquad \frac{\Gamma \vdash \phi : \mathsf{prop}(\Psi, \mathsf{i})}{\Gamma \vdash \forall\phi : \mathsf{prop}(\Psi)}$$

$$\frac{\Gamma \vdash \Psi : \mathsf{Ctx} \quad \Gamma \vdash \phi : \mathsf{prop}(\Psi)}{\Gamma \vdash \mathsf{nd}\ \Psi\ \phi : \mathsf{set}} \qquad \frac{\Gamma \vdash D : \mathsf{nd}\ (\Psi, \mathsf{i})\ \phi}{\Gamma \vdash \forall_I(D) : \mathsf{nd}\ \Psi\ \forall\phi}$$

$$\frac{\Gamma \vdash D : \mathsf{nd}\ \Psi\ (\forall\phi) \quad \Gamma \vdash n : \mathsf{var}(\Psi)}{\Gamma \vdash \forall_E(D, n) : \mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ (\mathsf{id}, n)\ \phi}$$

$\boxed{\text{Equality rules}}$

$$\begin{array}{l}
\epsilon[\delta] \equiv \epsilon \\
(\Psi, \mathsf{i})[\delta] \equiv (\uparrow \Psi[\delta], \mathsf{z}) \\
\epsilon \circ \alpha = \epsilon \\
(\alpha, n) \circ \alpha' = (\alpha \circ \alpha', \mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ \alpha'\ n) \\
\uparrow \epsilon \equiv \epsilon \\
\uparrow (\alpha, n) \equiv (\uparrow \alpha, \mathsf{s}(n)) \\
\mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ (\alpha, n)\ \mathsf{z} \equiv n \\
\mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ (\alpha, n)\ \mathsf{s}(k) \equiv \mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ \alpha\ k \\
\mathsf{map}_{\Delta.\mathsf{prop}(\Psi)}\ \delta\ (\mathsf{b}(n)) \equiv \mathsf{b}(\mathsf{map}_{\Delta.\mathsf{var}(\Psi)}\ \delta\ n) \\
\mathsf{map}_{\Delta.\mathsf{prop}(\Psi)}\ \delta\ (\forall\phi) \equiv \forall(\mathsf{map}_{\Delta.\mathsf{prop}(\Psi,\mathsf{i})}\ \delta\ \phi) \\
\mathsf{map}_{\Delta.\mathsf{nd}\ \Psi\ (\forall\phi)}\ \delta\ (\forall_I D) \equiv \forall_I(\mathsf{map}_{\Delta.\mathsf{nd}\ (\Psi,\mathsf{i})\ \phi}\ \delta\ D) \\
\mathsf{map}_{\Delta.\mathsf{nd}\ \Psi\ \mathsf{map}^{1}_{\Psi.\mathsf{var}(\Psi)}\ (\mathsf{id},n)\ \phi}\ \delta\ (\forall_E(D, n)) \equiv \\
\qquad \forall_E(\mathsf{map}_{\Delta.\mathsf{nd}\ \Psi\ \forall\phi}\ \delta\ D, \mathsf{map}_{\Delta.\mathsf{var}(\Psi)}\ \delta\ n)
\end{array}$$

Figure 4. Representation of the logic

When we introduce a new non-discrete type, we must give $\beta\eta$ rules for its types and terms, and define horizontal and vertical composition for its canonical elements. Horizontal composition into the empty context still relates the empty context to itself. Horizontal composition into Ψ, i gives the familiar parallel extension of a renaming $r : \Psi[\theta] \to \Psi[\theta']$ to a renaming $(\Psi, \mathsf{i})[\theta] \to (\Psi, \mathsf{i})[\theta']$ that leaves the last variable unchanged. This is defined using the shift $\uparrow$. The next two rules define vertical composition, and the following rules define the action of shift on canonical transformations.

When we introduce a new set, we must say how map acts on it. Variables are transformed by looking them up in the renaming (by the rule for $\mathsf{map}_{\Delta.A[\theta]}$, it is enough to consider the case where the type is $\cdot, \Psi{:}\mathsf{Ctx}^{+}.\mathsf{var}(\Psi)$). Propositions are transformed recursively. In the $\forall$ case, the definition of $(\Psi, \mathsf{i})[\delta]$ accomplishes the parallel extension of the renaming that is necessary when descending under a binder. Derivations are transformed recursively as well. In the $\forall_E$ case, associativity of composition is used to show that the right-hand side has the required type. The definition of transformation for prop and nd could alternatively be given using the elimination rules described below, rather than by giving a clause for each constructor.

Now, we observe that map gives the appropriate renaming principle for the indexed type nd:

$$\frac{\Gamma \vdash \alpha : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi' \quad \Gamma \vdash D : \mathsf{nd}\ \Psi\ \phi}{\Gamma \vdash (\mathsf{map}^{1}_{\psi, a.\mathsf{nd}\ \psi\ a}\ (\alpha, \mathsf{id})\ D) : \mathsf{nd}\ \Psi'(\mathsf{map}^{1}_{\psi.\mathsf{var}(\psi)}\ \alpha\ \phi)}$$

Thus, 2DTT provides the appropriate vocabulary to describe structural properties for dependently typed abstract syntax.

Extending the language with types prop, nd, etc. amounts to implementing renaming for these types explicitly: the clauses defining them are added to the equality rules. However, these clauses are determined entirely by the types of the constructors of prop and nd, so we conjecture that it would be possible to generate them automatically from a schema for these inductive types.

Elimination rules Directed type theory provides some insight into what elimination rules we can have for these types while respecting the structural properties. For example, we cannot add the usual recursor for Ctx, as this would permit functions that do not respect renaming, such as:

$$\begin{array}{l}\Psi : \mathsf{Ctx}^{+} \vdash \mathsf{tail}(\Psi) : \mathsf{Ctx} \\ \mathsf{tail}(\epsilon) = \epsilon \\ \mathsf{tail}(\Psi, \mathsf{i}) = \Psi\end{array}$$

tail term would not respect transformation, in that if $\Psi \Longrightarrow \Psi'$ there is not necessarily a transformation $\mathsf{tail}(\Psi) \Longrightarrow \mathsf{tail}(\Psi')$. To see this, take $\Psi' = (\epsilon, \mathsf{i})$ and $\Psi = (\epsilon, \mathsf{i}, \mathsf{i})$, and let the original renaming map both variables in Ψ to the one variable in Ψ'. There is no renaming $\mathsf{i} \Longrightarrow \epsilon$, so this renaming cannot be preserved by tail. Because the functoriality component of a terms requires it to respect transformation, tail cannot have this type.

However, certain functions on contexts do respect transformation. For example, we can define context concatenation Ψ_1, Ψ_2 that interacts with ϵ and $, \mathsf{i}$ in the expected ways. This respects transformation because every variable in Ψ_1, Ψ_2 is in Ψ'_1, Ψ'_2 if every variable in Ψ_i is in Ψ'_i.

What are we to make of this situation? At a first cut, having a restricted set of elimination rules for contexts may be sufficient for programming with variable binding—for example, the Beluga system [22] provides only context nil, cons, and append, all of which preserve transformations. More generally, we may consider allowing the programmer to define any function that respects renaming: by analogy with quotient types, one should be able to define a function on a higher-dimensional type by defining a function on the raw terms and proving separately that it respects transformation.

For similar reasons, we cannot give the standard case-analysis principle for variables $\mathsf{var}(\Psi)$, giving a case for z and a case for s. The reason is that functoriality imposes a coherence constraint between the transformation of the result of the zero case and the result of the successor case. However, we can show that it is possible to define an equality test for variables:

$$x : \mathsf{var}(\Psi)^{+}, y : \mathsf{var}(\Psi)^{+} \vdash \mathsf{eq}\ x\ y : 2$$

(and more informatively-typed variants), which is all one typically uses in recurring over abstract syntax. The type var is an example of a general class of inductive families indexed non-uniformly by higher-dimensional types, which require further study.

On the other hand, prop and nd *can* be given their usual elimination principles, as their higher-dimensional index Ψ is uniform (always fully general in the conclusion of a constructor).

Mixed variance syntax Languages such as Twelf, Delphin, and Beluga [21, 22, 23] distinguish a logical framework (LF) from a meta-language in which LF terms are treated as inductively defined data. Moreover, they do not allow meta-language types, such as functions, to be used in abstract syntax or derivations (though they do allow framework functions, which are used to represent binding). This stratified approach corresponds to allowing indexed inductive definitions that are constructed recursively from Σ-types, de Bruijn indices (corresponding to LF variables), and recursive instances in extended Ψ's (corresponding to LF functions). In previous work [18], we explored extending such schemas to additionally allow meta-level functions ($\rightarrow$-types) in a simply-typed setting.

Because of contravariance, not all such definitions are functorial—i.e. the structural properties do not always hold. However, we did analyze certain circumstances under which functoriality holds. In light of the present work, we can recast this analysis as investigating functors from different context categories to $Sets$. For example, all schemas generate types that admit exchange: all schemas generate types that are functorial in the category $\mathsf{Ctx}^{\cong}$ of contexts and bijections between them. Second, all schemas generating a type A admit weakening and strengthening of variables whose types are *insubordinate* to A, where subordination tracks when one type can appear in the values of another type: all A are functorial in $\mathsf{Ctx}^{\geq A}$, the category of contexts that differ only by types that are insubordinate to A. Third, variables are functorial in renamings, and terms are additionally functorial in renamings and substitutions if each of their constructors are. These constraints can be composed. For example, a schema with positive and negative occurrences of the context (see the arith example [18]) is functorial in renamings, which additionally only weaken or contract variables in subordinate to its contravariant types. This can be represented as a category of contexts that has projections to both Ctx and $\mathsf{Ctx}^{\geq A}$. 2DTT is a natural setting to explore this hierarchy of contexts, and to generalize our previous work to the dependently typed case. We anticipate giving schemas for indexed inductive definitions such as nd, where the functoriality of the types generated by these schemas automatically equips the type with the desired structural properties.

5. Related Work

Of the many categorical accounts of Martin-Löf type theory Hofmann [15], our approach to the semantics of 2DTT most closely follows the groupoid interpretation [17]. Recent work connecting (symmetric) type theory with homotopy theory and higher-dimensional category theory [4, 11, 12, 19, 26, 27, 28] will be useful in generalizing 2DTT to additional models and higher-dimensions.

Another application of functors in dependent type theory is indexed containers [2**?**], a mechanism for specifying inductive families. Whereas we associate a functorial action with every type constructor, containers are deliberately restricted to strictly positive functors, which is necessary to use them to specify datatypes. Also, 2DTT is compatible with types indexed by an arbitrary category, but a container denotes a type indexed by a set.

Many systems support programming with dependently typed abstract syntax [3, 21, 22, 23]; 2DTT will enable us to go beyond this previous work by generating the structural properties automatically for mixed-variance definitions.

The following additional sources have informed our work: An early connection between λ-calculus and 2-categories was made by

Seely [24], who shows that simply-typed λ-calculus forms a (non-groupoidal) 2-category, with terms as 1-cells and reductions as 2-cells. Variance annotations on variables are common in simply-typed subtyping systems [8, 9, 25]. In the dependently typed case, variance annotations have been used to support termination-checking using sized types, as in MiniAgda [1].

6. Conclusion and Future Work

We introduce directed type theory, which gives an account of types with an asymmetric notion of transformation between their elements. Examples include a universe of sets with functions between them, and a type of variable contexts with renamings or substitutions between them. We show that the groupoid interpretation of type theory generalizes to the directed case, giving our language a semantics in Cat.

Numerous generalizations and applications of directed dependent type theory are possible. On the semantic side, we may consider semantics in 2-categories other than Cat. On the syntactic side, we may explore additional type constructors: Defining type families such as nd internally to the theory requires schemas for indexed inductive definitions. Defining 2-dimensional types such as Ctx requires an analogue of quotient types, as discussed above. A directed hom-type would provide a generic first-class notion of transformation. We may also recover undirected type theory as a special case of directed type theory, so that current dependently typed programming practices can be imported. Semantically, every groupoid is a category, so it is possible to isolate a universe of symmetric types, but we have not yet explored how to make this universe available syntactically. Of course, the generalization to dimensions beyond 2 is another important direction for future work, as it will expose connections with weak ω-categories and directed homotopy theory. Higher dimensions may provide a guide in obtaining a theory with a decidable notion of definitional equality, as many of the equations that are definitional in 2DTT will hold only up to higher structure.

Finally, we speculate on some additional applications of our theory. First, we may be able to recover existing examples of directed phenomena in dependent type systems, such variance annotations for sized types [1], implicit coercions [6], and coercive subtyping [20]. For example, we may consider a translation of coercive subtyping into our system, using functoriality to model the lifting of a coercion by the subtyping rules. Because uses of map are explicit, our approach additionally supports non-coherent systems of coercions, and it will be interesting to explore applications of this generality; but the coherent case may provide a guide as to when instances of map can be inferred. Second, directed type theory may be useful as a meta-language for formalizing directed concepts, such as reduction [24], or category theory itself. Third, directed type theory may be useful for reasoning about effectful programs or interactive systems, which evolve in a directed manner (Gaucher [13] connects homotopy theory and concurrency). For example, we could define a type of interactive processes with transformations given by their operational semantics, or a type of processes with the transformations given by simulation.

Acknowledgements We thank Steve Awodey and Peter Lumsdaine for helpful conversations about this work.

References

[1] A. Abel. Miniagda: Integrating sized and dependent types. In A. Bove, E. Komendantskaya, and M. Niqui, editors, *Workshop on Partiality And Recursion in Interative Theorem Provers*, 2010.

[2] T. Altenkirch and P. Morris. Indexed containers. In *IEEE Symposium on Logic in Computer Science*, pages 277–285, Washington, DC, USA, 2009. IEEE Computer Society.

[3] T. Altenkirch and B. Reus. Monadic presentations of lambda terms using generalized inductive types. In *CSL 1999: Computer Science Logic*. LNCS, Springer-Verlag, 1999.

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

[5] J. C. Baez and M. Shulman. Lectures on n-categories and cohomology. Available from `http://arxiv.org/abs/math/0608420v2`, 2007.

[6] G. Barthe. Implicit coercions in type systems. In *International Workshop on Types for Proofs and Programs*, pages 1–15, London, UK, 1996. Springer-Verlag.

[7] R. S. Bird and R. Paterson. De Bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999.

[8] L. Cardelli. Notes about $F^{\omega}_{<:}$. Unpublished., 1990.

[9] D. Duggan and A. Compagnoni. Subtyping for object type constructors. In *Workshop On Foundations Of Object-Oriented Languages*, 1999.

[10] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *IEEE Symposium on Logic in Computer Science*, 1999.

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

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

[13] P. Gaucher. A model category for the homotopy theory of concurrency. *Homology, Homotopy, and Applications*, 5(1):549–599, 2003.

[14] R. Godement. *Théorie des faisceaux*. Hermann, Paris, 1958.

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

[16] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *IEEE Symposium on Logic in Computer Science*, 1999.

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

[18] D. R. Licata and R. Harper. A universe of binding and computation. In *ACM SIGPLAN International Conference on Functional Programming*, 2009.

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

[20] Z. Luo. Coercive subtyping. *Journal of Logic and Computatio*, 9(1), 1999.

[21] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. In H. Ganzinger, editor, *International Conference on Automated Deduction*, pages 202–206, 1999.

[22] B. Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 371–382, 2008.

[23] A. Poswolsky and C. Schürmann. Practical programming with higher-order encodings and dependent types. In *European Symposium on Programming*, 2008.

[24] R. Seely. Modeling computations: a 2-categorical framework. In *IEEE Symposium on Logic in Computer Science*, pages 65–71, 1987.

[25] M. Steffen. *Polarized Higher-Order Subtyping*. PhD thesis, Universitaet Erlangen-Nuernberg, 1998.

[26] 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.

[27] V. Voevodsky. The equivalence axiom and univalent models of type theory. Available from `http://www.math.ias.edu/~vladimir/Site3/home_files/CMU_talk.pdf`, 2010.

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