2-Dimensional Directed Type Theory

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

Recent work on *higher-dimensional type theory* has explored connections between Martin-Löf type theory, higher-dimensional category theory, and homotopy theory. These connections suggest a generalization of dependent type theory to account for computationally relevant proofs of propositional equality—for example, taking $\mathsf{Id}_{Set}\ A\ B$ to be the isomorphisms between A and B. The crucial observation is that all of the familiar type and term constructors can be equipped with a functorial action that describes how they preserve such proofs. The key benefit of higher-dimensional type theory is that programmers and mathematicians may work up to isomorphism and higher equivalence, such as equivalence of categories.

In this paper, we consider a further generalization of higher-dimensional type theory, which associates each type with a *directed* notion of transformation between its elements. Directed type theory accounts for phenomena not expressible in symmetric higher-dimensional type theory, such as a universe set of sets and functions, and a type Ctx used in functorial abstract syntax. Our formulation requires two main ingredients: First, 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. Second, 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. We describe a *2-dimensional directed type theory*, or *2DTT*, which is validated by an interpretation into the strict 2-category *Cat* of categories, functors, and natural transformations. We also discuss applications of 2DTT for programming with abstract syntax, generalizing the functorial approach to syntax to the dependently typed and mixed-variance case.

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 $x{:}A \vdash C\,\mathsf{type}$ 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 $C[N/x]$:

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

*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.

Submitted to:
MFPS 2011

The nature of this coercion is explained by the groupoid interpretation of type theory given by Hofmann and Streicher [21]. 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 [21] 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 $\mathsf{El}(S_1) \to \mathsf{El}(S_2)$. 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, subst states that all type families respect isomorphism: for any $x : \mathsf{set} \vdash C : \mathsf{set}$, $A \cong B$ implies $C[A] \cong C[B]$. Computationally, the lifting of the isomorphism is given by the functorial action of the type family C.

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 [17] generalizes the groupoid interpretation to a class of 2-categories where the 2-cells are invertible. Lumsdaine [27] and van den Berg and Garner [36] 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 [37] 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 set whose elements are sets S and whose morphisms are functions $f : \mathsf{El}(S_1) \to \mathsf{El}(S_2)$, then any dependent type $x{:}\mathsf{set} \vdash C\,\mathsf{type}$ describes such a functor, and $\mathtt{subst}$ 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. In the functorial approach to syntax with binding [3, 20, 14], the syntactic expressions in context Ψ are represented by a family of types indexed by Ψ—e.g. a type $\mathsf{prop}(\Psi)$ classifying formulas in a first-order logic with free variables in Ψ—where $\Psi : \mathsf{lCtx}$ is a representation of a context (e.g. a list of sorts). Structural prop-

[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.

erties, 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*. 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, and therefore describing syntax functorially requires general, non-groupoidal, categories.

In this paper, we propose a new notion of *directed type theory*, which generalizes existing symmetric type theory by permitting an asymmetric notion of transformation between the elements of a type. This extends the connection between type theory, higher-dimensional category theory, and homotopy theory to the directed case. Our formulation requires two interesting technical ingredients: First, 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. Second, 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, account of transformations, and make explicit the action of transformations on families of types.

Here, we consider only the two-dimensional case of directed type theory, and define a type theory *2DTT* (Section 2). 2DTT admits a simple interpretation in the category *Cat* of categories, functors, and natural transformations (Section 3). 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. 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 are possible. Our main motivating application of 2DTT, which we sketch in Section 4, is extending functorial syntax [21, 14] to account for dependently typed and mixed variance syntax.

2 Syntax

In this section, we give a proof theory for 2DTT; we refer the interested reader to Licata [24, Chapter 7] for a more leisurely account of this material. 2DTT 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 asymmetric analogue of propositional "equality" proofs. As discussed above, the two main ingredients in 2DTT are these transformation judgements, and variance annotations on assumptions in the context.

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 substi-

$\boxed{\Gamma \,\mathsf{ctx}}$

$$\frac{\Gamma \,\mathsf{ctx}}{\Gamma^{\mathsf{op}} \,\mathsf{ctx}} \qquad \frac{}{\cdot \,\mathsf{ctx}} \qquad \frac{\Gamma \,\mathsf{ctx} \quad \Gamma \vdash A \,\mathsf{type}}{\Gamma, x{:}A^{+} \,\mathsf{ctx}} \qquad \frac{\Gamma \,\mathsf{ctx} \quad \Gamma^{\mathsf{op}} \vdash A \,\mathsf{type}}{\Gamma, x{:}A^{-} \,\mathsf{ctx}}$$

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

$$\frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash A \,\mathsf{type}}{\Gamma \vdash A[\theta] \,\mathsf{type}} \qquad \frac{\Gamma \vdash A \,\mathsf{type} \quad \Gamma, x{:}A^{+} \vdash B \,\mathsf{type}}{\Gamma \vdash \Sigma x{:}A.\, B \,\mathsf{type}} \qquad \frac{\Gamma^{\mathsf{op}} \vdash A \,\mathsf{type} \quad \Gamma, x{:}A^{-} \vdash B \,\mathsf{type}}{\Gamma \vdash \Pi x{:}A.\, B \,\mathsf{type}}$$

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

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

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

$$\frac{x{:}A^{+} \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash \theta : \Delta \quad \Delta \vdash M : A}{\Gamma \vdash M[\theta] : A[\theta]} \qquad \frac{\Delta \,\mathsf{ctx} \quad \Delta \vdash C \,\mathsf{type} \quad \Gamma \vdash \delta : \theta_1 \Longrightarrow_{\Delta} \theta_2 \quad \Gamma \vdash M : C[\theta_1]}{\Gamma \vdash \mathsf{map}_{\Delta.C}\ \delta\ M : C[\theta_2]}$$

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

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

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

$$\frac{\Gamma^{\mathsf{op}} \vdash \delta : \theta'^{\mathsf{op}} \Longrightarrow_{\Delta^{\mathsf{op}}} \theta^{\mathsf{op}}}{\Gamma \vdash \delta^{\mathsf{op}} : \theta \Longrightarrow_{\Delta} \theta'} \qquad \frac{}{\Gamma \vdash \cdot : \cdot \Longrightarrow_{\cdot} \cdot}$$

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

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

$$\frac{}{\Gamma \vdash \mathsf{id}^{A}_{M} : M \Longrightarrow_{A} M} \qquad \frac{\Gamma \vdash \alpha_1 : M_1 \Longrightarrow_{A} M_2 \quad \Gamma \vdash \alpha_2 : M_2 \Longrightarrow_{A} M_3}{\Gamma \vdash \alpha_2 \circ \alpha_1 : M_1 \Longrightarrow_{A} M_3}$$

$$\frac{\Gamma_0 \vdash \delta_0 : \theta_0 \Longrightarrow_{\Gamma} \theta_0' \quad \Gamma \vdash \alpha : M \Longrightarrow_{A} N}{\Gamma_0 \vdash \alpha[\delta_0] : (\mathsf{map}_{\Gamma.A}\ \delta_0\ (M[\theta_0])) \Longrightarrow_{A[\theta_0']} N[\theta_0']} \qquad \frac{\Delta \vdash M : A \quad \Gamma \vdash \delta : \theta_1 \Longrightarrow_{\Delta} \theta_2}{\Gamma \vdash M[\delta] : (\mathsf{map}_{\Delta.A}\ \delta\ (M[\theta_1])) \Longrightarrow_{A[\theta_2]} M[\theta_2]}$$

$$\frac{\Gamma, x{:}A^{-} \vdash \alpha : (M\, x) \Longrightarrow_{B} (N\, x)}{\Gamma \vdash \lambda\, x.\, \alpha : M \Longrightarrow_{\Pi x:A.\, B} N} \qquad \frac{\Gamma \vdash \alpha : M \Longrightarrow_{\Pi x:A.\, B} N \quad \Gamma^{\mathsf{op}} \vdash \beta : N_1 \Longrightarrow_{A} M_1}{\Gamma \vdash \alpha\, M_1\, N_1\, \beta : \mathsf{map}^{1}_{B}\ \beta\ (MM_1) \Longrightarrow_{B[N_1/x]} (NN_1)}$$

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

Figure 1: Formation Rules: Identity, Composition, Involution, Π, Σ

tutions 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. The treatment of dependent types in Pitts [32]'s survey article provides an introduction to this style of syntax, with an explicit substitution judgement and internalized composition principles.

2.1 Formation rules

Contexts and types. As discussed above, variables are annotated with a variance, $^+$ or $^-$, that determines in what positions they may be used. We also make use of 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}.

We require two auxiliary notions on contexts: (a) looking up a variable in a context, $x{:}A^\pm \in \Gamma$, and (b) one context being an extension of another, $\Gamma \supseteq \Delta$. These are defined by:

$$\frac{}{x{:}A^\pm \in \Gamma, x{:}A^\pm} \qquad \frac{x{:}A^\pm \in \Gamma}{x{:}A^\pm \in \Gamma, y{:}B^\pm} \qquad \frac{}{\Gamma \supseteq \cdot} \qquad \frac{\Gamma \supseteq \Gamma'}{\Gamma, x{:}A^\pm \supseteq \Gamma'} \qquad \frac{\Gamma \supseteq \Gamma'}{\Gamma, x{:}A^\pm \supseteq \Gamma', x{:}A^\pm}$$

Here we write $\Gamma, x{:}A^\pm$ for an assumption of either variance. We do not give rules for $^{\mathsf{op}}$ because, as discussed below, Γ^{op} can always be expanded away using definitional equality.

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). 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. 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 $\Gamma \vdash A\,\mathsf{type}$. All judgements of the form $\Gamma \vdash J$ satisfy the following: 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 we may also instantiate the rule to conclude $\Gamma^{\mathsf{op}} \vdash \theta^{\mathsf{op}} : \Delta^{\mathsf{op}}$ from $\Gamma \vdash \theta : \Delta$. 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:

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

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.

Transformations. Next, we consider transformations between substitutions θ and θ'. The judgement $\Gamma \vdash \delta : \theta_1 \Longrightarrow_\Delta \theta_2$ 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. A key property of 2DTT is that all constructions respect transformation. We already saw this with map, which says that dependent types respect transformation: given a type $\Gamma \vdash A\,\mathsf{type}$ and a transformation $\theta \Longrightarrow_\Delta \theta'$, map creates an open term $x{:}A[\theta]^+ \vdash -{:}A[\theta']$. The operations $\delta[\delta']$ and $\theta[\delta]$ similarly state that transformations and substitutions respect transformation—they permit one transformation to be substituted into another, or into a single substitution, yielding a transformation between the substitution instances. This operation corresponds to what is called horizontal composition in a 2-category, while $\delta_1 \circ \delta_2$ is called vertical composition. $\theta[\delta']$ is an instance of $\delta[\delta']$ where $\delta = \mathsf{id}_\theta$, but it is technically convenient to include both constructs in the language, as we discuss with the equations below. 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 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, which states that term transformations ($\alpha[\delta]$) and terms ($M[\delta]$) themselves respect transformation. 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 $\mathsf{resp}\ F\ \alpha = F[\mathsf{id}_{\mathsf{id}}, \alpha^+/x]$.

Next, we give introduction and elimination rules for transformations at the various connectives. 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.

2.2 Equality Rules

Next, we present the equality rules. Due to space constraints, we sketch the approach here, and refer the reader to Licata [24, Chapter 7] for details. For simplicity, we treat equality as in extensional type theory, giving equations for what is equal in the semantics, rather than restricting ourselves to a decidable fragment—we leave a more intensional presentation to future work. This allows us to focus on the computationally relevant intensional behavior of transformations.

There are several classes of equality rules: First, we have type-generic rules describing properties of $^{\mathsf{op}}$ and identity and composition. For example, $^{\mathsf{op}}$ is an involution, in that $(\Gamma^{\mathsf{op}})^{\mathsf{op}} \equiv \Gamma$ and $(\theta^{\mathsf{op}})^{\mathsf{op}} \equiv \theta$ and $(\delta^{\mathsf{op}})^{\mathsf{op}} \equiv \delta$. The identity and composition principles are associative and unital; e.g.

$$\theta_0[\theta[\theta']] \equiv \theta_0[\theta][\theta'] \qquad (\delta_3 \circ \delta_2) \circ \delta_1 \equiv \delta_3 \circ (\delta_2 \circ \delta_1) \qquad \delta_0[\delta[\delta']] \equiv \delta_0[\delta][\delta']$$

This rules are sound because our semantics below interprets the calculus into *Cat*, a strict 2-category, where associativity and unit laws for composition hold on the nose, not up to higher-dimensional morphisms. There are also equations stating how these operations interact. Examples include: First, the middle-four interchange law related horizontal and vertical composition: $(\delta_1 \circ \delta_2)[\delta_3 \circ \delta_4] \equiv \delta_1[\delta_3] \circ \delta_2[\delta_4]$. Second, $^{\mathsf{op}}$ preserves identities and compositions: $(\theta_1[\theta_2])^{\mathsf{op}} \equiv \theta_1^{\mathsf{op}}[\theta_2^{\mathsf{op}}]$ and $(\delta_1 \circ \delta_2)^{\mathsf{op}} \equiv \delta_2^{\mathsf{op}} \circ \delta_1^{\mathsf{op}}$. Third, for map and horizontal composition, there are rules expressing functoriality; e.g.:

$$\mathsf{map}_{\Delta.C}\ \mathsf{id}_\theta\ M \equiv M \qquad \mathsf{map}_{\Delta.C}\ (\delta_2 \circ \delta_1)\ M \equiv \mathsf{map}_{\Delta.C}\ \delta_2\ (\mathsf{map}_{\Delta.C}\ \delta_1\ M)$$

Next, for each context former, we give $\beta\eta$ rules expressing the product structure of substitutions and transformations, as well as rules defining the identity and composition and involution principles. As examples of the latter, we have:

$$\begin{array}{lll} (\Gamma, x{:}A^+)^{\mathsf{op}} \equiv \Gamma^{\mathsf{op}}, x{:}A^- & (\theta, M^+/x)^{\mathsf{op}} \equiv \theta^{\mathsf{op}}, M^-/x & (\delta, \alpha^+/x)^{\mathsf{op}} \equiv \delta^{\mathsf{op}}, \alpha^-/x \\ (\Gamma, x{:}A^-)^{\mathsf{op}} \equiv \Gamma^{\mathsf{op}}, x{:}A^+ & (\theta, M^-/x)^{\mathsf{op}} \equiv \theta^{\mathsf{op}}, M^+/x & (\delta, \alpha^-/x)^{\mathsf{op}} \equiv \delta^{\mathsf{op}}, \alpha^+/x \end{array}$$

Thus, $^{\mathsf{op}}$ interchanges the variance annotations on contexts, substitutions, and transformations.

For each type, we give $\beta\eta$-rules both for terms and transformations, and rules defining the various identity and composition principles. For example, the rules for Π-types are as follows:

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

The $\beta\eta$ rules for terms are standard; the $\beta\eta$ rules for transformations express the fact that a transformation at Π is isomorphic (in the meta-language) to a family of transformations. The `map` rules states that

map at Π is given by pre- and post-composition; the contravariance of A is essential to making this type check. `refl` and `trans` expand the definition of identity and composition for transformations at $\Pi x{:}A.\,B$, which can be expressed in terms of transformations at B. The remaining rules push the compositions inwards, exactly as in the standard definition of substitution. It is intriguing that 2-cell composition, $M[\delta]$ and $\alpha[\delta]$ can be defined in much the same manner as ordinary substitution. In ordinary symmetric type theory, these equations only hold up to propositional equality. While our current extensional presentation does not allow any claims about decidability, these equations suggest that it may be possible to build the computational behavior of the "respect for transformation/equality" operations into definitional equality, which would facilitate reasoning about these constructs.

2.3 Sets

Having presented the general framework, we can now consider a first example of a directed base type, a universe of sets and functions:

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

$$\mathsf{map}_{\Delta.\mathsf{set}}\ \delta\ M \equiv M$$

$$\mathsf{map}_{\Delta.\mathsf{el}\,S}\ \delta\ M \equiv N[\mathsf{id}, M^{+}/x] \ \text{ if } \ S[\delta] \equiv x.N$$

The terms of type set themselves represent types, but ones which are known to have trivial transformations: Each set S determines a type $\mathsf{el}\,S$, and the only transformation at $\mathsf{el}\,S$ is reflexivity, written $\star$. Here, we treat equality at $\mathsf{el}\,S$ as in extensional type theory, giving an equality reflection rule for $\alpha : M \Longrightarrow_{\mathsf{el}\,S} N$. On the other hand, a transformation between two sets $A \Longrightarrow_{\mathsf{set}} B$ is given by an open term $x.M$. For example, there is a transformation $0 \Longrightarrow_{\mathsf{set}} 1$ because there is a function from the empty type to the unit type, but not vice versa. Semantically, set is interpreted as the category of sets and functions. map at set itself is trivial, because set is constant, but map at $\mathsf{el}-$ uses horizontal composition $S[\delta]$ to compute a function (open term) that determines the transformation.

When we introduce a set, we must give rules for forming the set, and for forming its members, as in ordinary type theory, as well as a rule defining map. For example, sets are closed under many of the usual type constructors:

$$\frac{\Gamma^{\mathsf{op}} \vdash S : \mathsf{set} \quad \Gamma, x{:}\mathsf{el}\,S^{-} \vdash S' : \mathsf{set}}{\Gamma \vdash \Pi x{:}S.\,S' : \mathsf{set}} \qquad \frac{\Gamma \vdash S : \mathsf{set} \quad \Gamma, x{:}\mathsf{el}\,S^{+} \vdash S' : \mathsf{set}}{\Gamma \vdash \Sigma x{:}S.\,S' : \mathsf{set}} \qquad \frac{}{\Gamma \vdash 0, 1, 2 : \mathsf{set}} \qquad \frac{\Gamma \vdash S : \mathsf{set} \quad \Gamma \vdash M, N : \mathsf{el}\,A}{\Gamma \vdash \mathsf{Id}_S\,M\,N : \mathsf{set}}$$

For each set, we define its elements and define map by giving a rule for $S[\delta]$; e.g. for Π we have

$$\frac{\Gamma \vdash M : \Pi x{:}\mathsf{el}\,S.\,\mathsf{el}\,S'}{\Gamma \vdash \mathsf{in}\,M : \mathsf{el}\,(\Pi x{:}S.\,S')} \quad \frac{\Gamma \vdash M : \mathsf{el}\,(\Pi x{:}S.\,S')}{\Gamma \vdash \mathsf{out}\,M : \Pi x{:}\mathsf{el}\,S.\,\mathsf{el}\,S'} \quad (\Pi x{:}S.\,S')[\delta : \theta \Longrightarrow \theta'] \equiv x.\mathsf{in}\,(\mathsf{map}_{\Pi x:\mathsf{el}\,S.\,\mathsf{el}\,S'}\ \delta\ (\mathsf{out}\,M))$$

In addition, we must define $\beta\eta$ rules for the elements, as well as rules for the remaining substitution principles, $S[\theta]$, $M[\theta]$, and $M[\delta]$ [24].

Identity types and sets In symmetric type theory, given $\Gamma, \Gamma \vdash A\,\mathsf{type}$, and $\Gamma \vdash M, N : A$, one can define a type $\Gamma \vdash \mathsf{Id}_A\ M\ N\,\mathsf{type}$, whose functorial action on an equality $\delta : \mathsf{Id}_\Gamma\ \theta_1\ \theta_2$ is given by pre-composition with an equality determined from $M[\delta]$ and post-composition with an equality determined from $N[\delta]$.

However, this construction uses symmetry: the precomposition needs to be with with $(M[\delta])^{-1}$. In the directed setting, this is not necessarily possible, which motivates the judgemental approach to transformations that we take in this paper. However, when A is in fact groupoidal, one can internalize transformations as an identity type, whose functorial action is given as in symmetric type theory. The type $\mathsf{Id}_S\ M\ N$ defined here is a special case where $\mathsf{el}\,S$ is discrete, and therefore trivially groupoidal. A more general alternative is to consider a directed *Hom*-type, whose first argument is a contravariant position, which internalizes the notion of a dinatural transformation. However, the syntactic rules for such a type require further study.

3 Semantics

In this section, we sketch a semantics of 2DTT in *Cat*, the 2-category of categories, functors, and natural transformations, referring the reader to Licata [24, Chapter 7] for details.

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, the context, substitution, and transformation 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 $[\![\delta]\!] : [\![\theta_1]\!] \Longrightarrow [\![\theta_2]\!] : [\![\Gamma]\!] \longrightarrow [\![\Delta]\!]$.

The judgement $\Gamma \vdash A\ \mathsf{type}$ 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 $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathit{Cat}$. Here we take *Cat* to be the category of large categories, to interpret the type set.

As a notational convention, we will overload notation so that the semantics looks just like the syntax: First, we use the same letter for a piece of syntax and for the semantic concept it is interpreted as; e.g. we will write Γ for a category, θ for a functor, A for a functor into *Cat*, etc. Second, we use the same symbols as we use in the syntax for the 2-category structure on *Cat*: we write the identity functor as id, functor composition as $\theta[\theta']$, vertical composition of natural transformations as $\delta \circ \delta'$, horizontal composition as $\delta[\delta']$, and the identity natural transformation as id_θ, and the action of $^{\mathsf{op}}$ as Γ^{op}, etc. Additionally, we abbreviate $\Gamma \longrightarrow \mathit{Cat}$ by $\mathsf{Ty}\ \Gamma$.

The set of terms $\Gamma \vdash M : A$ is isomorphic to the one-element substitutions $\Gamma \vdash \mathsf{id}, M^+/x : \Gamma, x{:}A^+$, and $\Gamma, x{:}A^+$ is interpreted as the total category of the Grothendieck construction, $\int_\Gamma A$. Thus, we can define a the interpretation of a term $\Gamma \vdash M : A$ to be a functor $[\![M]\!] : \Gamma \longrightarrow \int_{[\![\Gamma]\!]} [\![A]\!]$ such that the Γ part of the functor is the identity—which we can formalize by saying that $[\![M]\!]$ is a section of p: $\mathsf{p} \circ [\![M]\!] = \mathsf{id}$. However, following Hofmann and Streicher [21], it is more convenient to use an equivalent explicit definition:

DEFINITION 3.1. For a category Γ and a functor $A : \Gamma \longrightarrow \mathit{Cat}$, 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 $\gamma \in \mathit{Ob}\Gamma$, $M_o(\gamma) \in \mathit{Ob}(A(\gamma))$
- For all $c : \gamma_1 \longrightarrow_\Gamma \gamma_2$, $M_a(c) : A(c)(M_o(\gamma_1)) \longrightarrow_{A(\gamma_2)} M_o(\gamma_2)$. Moreover, $M_a(\mathsf{id}) = \mathsf{id}$ and $M_a(c_2 \circ c_1) = M_a(c_2) \circ A(c)(M_a(c_1))$.

Similarly, we define the semantic counterpart of $\Gamma \vdash \alpha : M \Longrightarrow_A N$:

DEFINITION 3.2. 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_γ such that

- for $\gamma \in Ob\Gamma$, $\alpha_\gamma : M(\gamma) \longrightarrow_{A(\gamma)} N(\gamma)$
- for $c : \gamma_1 \longrightarrow_\Gamma \gamma_2$, $N(c) \circ A(c)(\alpha_{\gamma_1}) = \alpha_{\gamma_2} \circ M(c)$

The interesting part of the interpretation is showing that each inference rule is true: given the semantic domains corresponding to the premises, we can construct the semantic domain corresponding to the conclusion. Once we have defined the operations, we can validate each equation on the semantic counterparts of the terms in question. Taken together, these constructions and proofs represent the inductive steps of the interpretation. Then, we tie these pieces together with a soundness theorem, which is described in technical detail in Licata [24].

THEOREM 3.3. *Soundness. There are total functions $[\![-]\!]$ that for each derivation $\mathscr{D} :: J$ yield a semantic entity of type $[\![J]\!]$, validating the definitional equalities.*

We describe the inductive steps here:

Involution, Identity, and Composition The involutions are interpreted by the 2-functor $-^{\mathsf{op}} : Cat \longrightarrow Cat^{\mathsf{co}}$ which sends each category to its opposite category. $\Gamma^{\mathsf{op}}$is the action on objects; $\theta^{\mathsf{op}}$is the action on 1-cells; and $\delta^{\mathsf{op}}$is the action on 2-cells. The identity and composition principles for substitutions and transformations are interpreted as the identity, horizontal composition, and vertical composition operations of the 2-category *Cat*. The equations for them follow from the definition of a 2-category. A type is interpreted as a functor, and $A[\theta]$ as functor composition. map is an instance of whiskering a functor (into *Cat*) with a natural transformation, which can be thought of as the functorial action of the type on the transformation. It is simple to check that a term $\mathsf{Tm}\ \Delta\ A$ and a functor $\Gamma \longrightarrow \Delta$ can be composed as indicated by $M[\theta]$. Semantic identity and vertical and horizontal composition for term transformations are defined as follows:

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

In the final equation, we use N and A and θ as in the typing rule for the left-hand side.

Interpretation of Contexts The empty context is interpreted as the category 1, which has one object and its identity morphism. $\Gamma, x{:}A^+$ is interpreted by the Grothendieck construction $\int_\Gamma A$. $\Gamma, x{:}A^-$ is interpreted as $(\int_{\Gamma^{\mathsf{op}}} A)^{\mathsf{op}}$.

Interpretation of Types Π-types are defined as in Hofmann and Streicher [21]: 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 $\sigma \in Ob\Gamma$, we define $B_\sigma : \mathsf{Ty}\ A(\sigma)$ by

$$(B_\sigma)(\sigma') = B(\sigma, \sigma')$$
$$(B_\sigma)(c) = B(\mathsf{id}_\sigma, c)$$

For any Γ and A, the $\mathsf{Tm}\ \Gamma\ A$ are the objects of a category with morphisms given by term transformations α. This lets us define a Π type as follows:

$$(\Pi\ A\ B)_\sigma = \mathsf{Tm}\ A(\sigma)^{\mathsf{op}}\ B_\sigma$$

Functoriality is given by pre- and post-composition: the contravariance of A ensures that the pre-composition faces the right direction. λ and application and $\beta\eta$ rules are interpreted by giving a bijection between $\mathsf{Tm}\ \Gamma.A^{-}\ B$ and $\mathsf{Tm}\ \Gamma\ \Pi AB$. The transformation intro and elim and $\beta\eta$ 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 [21], Section 5.3, which observes that the groupoid interpretation justifies functional extensionality.

Because both subcomponents of $\Sigma x{:}A.\,B$ are covariant, the interpretation given in Hofmann and Streicher [21] adapts to our setting unchanged.

The type set is interpreted as the constant functor returning *Sets*, the category of sets and functions. Because the action on morphisms of a constant functor is the identity, $\mathsf{Tm}\ \Gamma\ \mathsf{set}$ is bijective with $\Gamma \longrightarrow \mathit{Sets}$. Thus, we can represent $\mathsf{el}\,S$ semantically by $\mathit{discrete} \circ S$. As usual, we overload notation and write $\mathsf{el}\,S$ for $\mathit{discrete} \circ S$. The transformation rule for set expresses (half of) an isomorphism between, on the one hand, natural transformations between two functors into *Sets*, and, on the other, terms $\mathsf{Tm}\ \int_{\Gamma} \mathsf{el}\,S\ \mathsf{el}\,S'$, which is given by currying.

More details on this soundness theorem, including the interpretation of particular sets and the outer induction that ties it all together, is described in Licata [24].

4 Applications and Extensions

4.1 Dependently Typed and Mixed Variance Syntax

First, we explain how 2DTT can be deployed to generalize the functorial approach to syntax [20, 14, 3] to dependently typed and mixed variance syntax. These examples are discussed in more detail in [24, Chapter 8].

Dependently Typed Syntax To illustrate the approach to representing dependently typed syntax, we represent a judgement $\Psi \vdash A$ as a type $\mathsf{nd}\ \Psi\ A$, where the proposition A can mention the variables in Ψ. Stating the structural properties for such judgements is tricky, because the substitution into the derivation must prove the substitution into the type: substitution maps derivations of $\Psi \vdash A$ to derivations of $\Psi' \vdash A[\theta]$, given a transformation θ from Ψ to Ψ'.

First, we define a type Ctx representing object-language contexts Ψ. For example, if the variables in Ψ are unsorted then the terms of type Ctxcould be natural numbers, with variables represented by inhabitants of $\mathsf{fin}(\Psi)$—numbers less than Ψ—i.e. we use dependent de Bruijn indices [9, 8]. Transformations at Ctx are taken to be substitutions $\Psi \vdash \Psi'$, which are chosen to give the desired structural properties. For example, representing substitutions by a function $\mathsf{fin}(\Psi') \to \mathsf{fin}(\Psi)$ gives weakening, exchange, and contraction but not substitution.

Next, we represent propositions by a set

$$\frac{\Gamma^{\mathsf{op}} \vdash \Psi : \mathsf{Ctx}}{\Gamma \vdash \mathsf{propo}\,\Psi : \mathsf{set}}$$

This typing says that propositions are contravariantly functorial in Ψ, meaning that

$$\frac{w : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi' \quad \phi : \mathsf{propo}\,\Psi'}{(\mathsf{map}_{\psi^{-}.\mathsf{propo}\,\psi}\ (w^{-}/\psi)\ e) : \mathsf{propo}\,\Psi}$$

Moreover, the functoriality equations for map stipulate that renaming by the identity is the identity, weakening by a composition is composition of the renamings, and so on. We will sometimes abbreviate $\mathsf{map}_{\psi^{-}.\mathsf{propo}\,\psi}\ w^{-}/\psi\ e$ by $\mathsf{map}\ w\ e$ when the meaning is clear from context.

Finally, we represent natural deduction derivations by a type

$$\frac{\Gamma^{\mathsf{op}} \vdash \Psi : \mathsf{Ctx} \quad \Gamma \vdash \phi : \mathsf{propo}\,\Psi}{\Gamma \vdash \mathsf{nd}\,\Psi\,\phi : \mathsf{set}}$$

The type-generic rule for map specializes to the appropriate renaming principle:

$$\frac{w : \Psi \Longrightarrow_{\mathsf{Ctx}} \Psi' \quad e : \mathsf{nd}\,\Psi'\,\phi}{(\mathsf{map}_{\psi^-,a^+.\mathsf{nd}\,\psi\,a}\ (w^-/\psi, \mathsf{id}^+/a)\ e) : \mathsf{nd}\,\Psi\,(\mathsf{map}\ w\ \phi)}$$

As desired, this principle says that the renaming of the derivation proves the renaming of the judgement. Thus, 2DTT's notion of transformation at a Σ-type naturally accounts for the structural properties of dependently typed syntax.

Mixed Variance 2DTT also accounts for mixed variance syntax, which mixes admissibility and derivability [26, 25]—such as a logic with the infinitary ω-rule for eliminating natural numbers. For example, in the rule

$$\frac{\Psi \vdash P(0) \quad \Psi \vdash P(1) \quad \dots}{\Psi \vdash \forall x.P(x)}$$

the infinitely many premises may be thought of as a function that yields $P(n)$ for each n, likely defined by induction. This will be encoded in 2DTT as a datatype constructor

$$\mathit{omega} : (\Pi n{:}\mathsf{nat}.\,\mathsf{nd}\,\Psi\,(\mathsf{subst}\,P\,n)) \to \mathsf{nd}\,\Psi\,(\mathsf{all}\,P)$$

where subst substitutes n for the last variable in P, and is defined using map. This datatype constructor can be used in existing dependent type theories. The advantage of 2DTT is that we will obtain the structural properties for free even for such a definition, because Π is equipped with a functorial action given by pre- and post-composition.

4.2 Extensions

Putting the above ideas into practice will require some interesting extensions of 2DTT: To define higher-dimensional types such as *Sets* and Ctx, we require an analogue of quotient types, where programmers specify a type by giving an *internal category*—a description of a category inside the theory. To define types such as propo and nd internally to the theory, we require an inductive datatype mechanism, adapting W-types [29] or indexed containers [2].

The connection between functorial syntax and higher-order abstract syntax is that in the category of presheaves, the exponential $\mathsf{exp}^{\mathsf{exp}} : \mathsf{Ctx} \to \mathit{Sets}$ is isomorphic to the type family $\mathsf{exp}(-+1)$ which adds an extra de Bruijn index [14, 20]. We can reproduce this result in 2DTT, but the exponential is *not* the contravariant Π we have considered so far, but a second, covariant, Π^{co}. A term $\psi : \mathsf{Ctx} \vdash M : \mathsf{exp}\,\psi \to^{\mathsf{co}} \mathsf{exp}\,\psi$, is explicitly parametrized over extensions in ψ (c.f. the Kripke interpretation of implication) and its functorial action is given by composing transformations—*not* by pre- and post-composition. This permits the argument position of a function to be treated as a covariant position, internalizing an assumption $x : A^+$.

Given these extensions, which are described in more detail in Licata [24, Chapter 8], 2DTT will afford an extremely general logical framework, in which programmers can specify logics using dependently typed and mixed-variance definitions, and automatically obtain implementations of the structural properties derived from the generic notion of functoriality built in to the calculus.

Another interesting avenue for future work is an A^{op} modality on types, given by the point-wise opposite of A, which may be useful for internalizing a directed Hom type, as discussed above. We also plan to consider generalizations to dimensions higher than 2, which will expose connections with weak ω-categories and directed homotopy theory, and to recover undirected type theory as a special case of directed type theory, by defining a universe of groupoids. On the semantic side, it will be interesting to consider semantics in 2-categories other than *Cat*.

5 Related Work

Of the many categorical accounts of Martin-Löf type theory Hofmann [19], our approach to the semantics of 2DTT most closely follows the groupoid interpretation [21]. Recent work connecting (symmetric) type theory with homotopy theory and higher-dimensional category theory [17, 27, 36, 4, 38, 15, 37] will be useful in generalizing 2DTT to additional models and higher-dimensions. An early connection between λ-calculus and 2-categories was made by Seely [34], who shows that simply-typed λ-calculus forms a (non-groupoidal) 2-category, with terms as 1-cells and reductions as 2-cells.

Functoriality of simple and polymorphic type constructors has been studied in previous work on generic traversals of data structures [23, 7] and compilation of subtyping [12]; our work generalizes this to the dependently typed case. In tactic-based proof assistants, it is possible to construct a library of tactics for showing that types and terms respect equivalence relations and order relations, such as Jackson's library for NuPRL [22] and setoid rewriting in Coq [11]. Our approach here is akin to building these tactics into the language, equipping *every* type and term with an action on transformations. This allows the computational content and equational behavior of these actions to be drawn out. Another application of functors in dependent type theory is indexed containers [2, 16], 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 are useful for specifying datatypes. Also, 2DTT is allows types indexed by an arbitrary category, but a container denotes a type indexed by a set.

Variance annotations on variables are common in simply-typed subtyping systems [13, 10, 35]. In the dependently typed case, variance annotations have been used to support termination-checking using sized types, as in MiniAgda [1].

Many systems support programming with dependently typed abstract syntax [30, 31, 33]; 2DTT will enable us to go beyond this previous work by generating the structural properties automatically for mixed-variance definitions.

6 Conclusion

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*. We have discussed an application to dependently typed and mixed variance syntax, and sketched some exciting avenues of future work.

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 [28]. 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 [34], 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 [18] 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.

Acknowledgments We thank Steve Awodey, Peter Lumsdaine, Chris Kapulkin, Kristina Sojakova, and Thorsten Altenkirch 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] G. Bellè, C. Jay, and E. Moggi. Functorial ML. In H. Kuchen and S. Doaitse Swierstra, editors, *Programming Languages: Implementations, Logics, and Programs*, volume 1140 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin / Heidelberg, 1996.

[8] F. Bellegarde and J. Hook. Substitution: A formal methods case study using monads and transformations. *Science of Computer Programming*, 23(2–3):287–311, 1994.

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

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

[11] Coq Development Team. *The Coq Proof Assistant Reference Manual, version 8.2*. INRIA, 2009. Available from `http://coq.inria.fr/`.

[12] K. Crary. Typed compilation of inclusive subtyping. In *ACM SIGPLAN International Conference on Functional Programming*, 2000.

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

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

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

[16] N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. In *Types for Proofs and Programs*, pages 210–225. Springer LNCS, 2004.

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

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

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

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

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

[22] P. Jackson. The nuprl proof development system (version 4.2) reference manual and user's guide. Available from `http://www.nuprl.org/documents/Jackson/Nuprl4.2Manual.html`, 1996.

[23] R. Laemmel and S. Peyton Jones. Scrap your boilerplate: a practical approach to generic programming. In *ACM SIGPLAN-SIGACT Symposium on Types in Language Design and Implementation*, 2003.

[24] D. R. Licata. *Dependently Typed Programming with Domain-Specific Logics*. PhD thesis, Carnegie Mellon University, 2011. Available from `http://www.cs.cmu.edu/~drl/pubs/thesis/thesis.pdf`.

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

[26] D. R. Licata, N. Zeilberger, and R. Harper. Focusing on binding and computation. In *IEEE Symposium on Logic in Computer Science*, 2008.

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

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

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

[30] 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.

[31] 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.

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

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

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

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

[36] 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.

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

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