Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 276 (2011) 263–289

www.elsevier.com/locate/entcs

2-Dimensional Directed Type Theory

Daniel R. Licata and Robert Harper[1,2]

Carnegie Mellon University

Abstract

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.

Keywords: type theory, category theorey, dependent types, homotopy type theory

1 Introduction

In type theory, it is standard to define a type A 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

[1] 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.

[2] Email: {drl,rwh}@cs.cmu.edu

doi:10.1016/j.entcs.2011.09.026

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 $C[N/x]$:

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

The nature of this coercion is explained by the groupoid interpretation of type theory given by FIXME. 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 FIXME 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. [3] 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, FIXME generalizes the groupoid interpretation to a class of 2-categories where the 2-cells are invertible. FIXME and FIXME show that the syntax of intensional type theory forms a weak ω-category. On the homotopy-theoretic side, FIXME show how to interpret intensional type theory into abstract homotopy theory (i.e. Quillen model

[3] 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.

categories), and Voevodsky's equivalence axiom [36] 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 $\Psi : \mathsf{Ctx}$ is a representation of a context (e.g. a list of sorts). 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*. 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 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. 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. To summarize, the judgement forms of 2DTT are

- Contexts: $\Gamma\ \mathsf{ctx}$
- Substitutions: $\Gamma \vdash \theta : \Delta$ (where $\Gamma\ \mathsf{ctx}$ and $\Delta\ \mathsf{ctx}$)
- Transformations: $\Gamma \vdash \delta : \theta \Longrightarrow_\Delta \theta'$ (where $\Gamma\ \mathsf{ctx}$ and $\Delta\ \mathsf{ctx}$ and $\Gamma \vdash \theta, \theta' : \Delta$)
- Dependent Types: $\Gamma \vdash A\ \mathsf{type}$ (where $\Gamma\ \mathsf{ctx}$)
- Terms: $\Gamma \vdash M : A$ (where $\Gamma\ \mathsf{ctx}$ and $\Gamma \vdash A\ \mathsf{type}$)
- Term Transformations: $\Gamma \vdash \alpha : M \Longrightarrow_A M'$ (where $\Gamma\ \mathsf{ctx}$ and $\Gamma \vdash A\ \mathsf{type}$ and $\Gamma \vdash M, M' : A$)

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

Involution

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

$$\begin{array}{lll} (\Gamma^{\mathsf{op}})^{\mathsf{op}} & \equiv \Gamma & \textit{0-involution} \\ (\theta^{\mathsf{op}})^{\mathsf{op}} & \equiv \theta & \textit{1-involution} \\ (\delta^{\mathsf{op}})^{\mathsf{op}} & \equiv \delta & \textit{2-involution} \end{array}$$

Identity and composition for $\Gamma \vdash \theta : \Delta$

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

$$\begin{array}{lll} \theta_0[\theta[\theta']] & \equiv \theta_0[\theta][\theta'] & \textit{1-subst assoc/unit} \\ \theta_0[\mathsf{id}_{\Gamma}] & \equiv \theta_0 & \\ \mathsf{id}^{\Gamma}_{\Gamma}[\theta] & \equiv \theta & \end{array}$$

$$\begin{array}{lll} \theta[\delta[\delta']] & \equiv \theta[\delta][\delta'] & \textit{1-resp assoc} \\ \theta[\mathsf{refl}_{\theta'}] & \equiv \mathsf{refl}_{\theta[\theta']} & \textit{1-resp preserves refl.} \\ \theta[\theta'][\delta] & \equiv \theta[\theta'[\delta]] & \textit{1-resp for 1-subst} \end{array}$$

$$\begin{array}{lll} \mathsf{id}^{\mathsf{op}}_{\Gamma} & \equiv \mathsf{id}_{\Gamma^{\mathsf{op}}} & {}^{\mathsf{op}}\ \textit{interactions} \\ (\theta_1[\theta_2])^{\mathsf{op}} & \equiv \theta_1^{\mathsf{op}}[\theta_2^{\mathsf{op}}] & \\ (\theta[\delta])^{\mathsf{op}} & \equiv \theta^{\mathsf{op}}[\delta^{\mathsf{op}}] & \end{array}$$

Identity and Composition for $\Gamma \vdash \delta : \theta \Longrightarrow_{\Delta} \theta'$

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

$$\begin{array}{lll} (\delta_3 \circ \delta_2) \circ \delta_1 & \equiv \delta_3 \circ (\delta_2 \circ \delta_1) & \textit{trans assoc/unit} \\ (\delta \circ \mathsf{refl}) & \equiv \delta & \\ (\mathsf{refl} \circ \delta) & \equiv \delta & \end{array}$$

$$\begin{array}{lll} \delta_0[\delta[\delta']] & \equiv \delta_0[\delta][\delta'] & \textit{2-resp assoc/unit} \\ \delta_0[\mathsf{refl}_{\mathsf{id}}] & \equiv \delta_0 & \\ \mathsf{refl}_{\mathsf{id}^{\Gamma}_{\Gamma}}[\delta] & \equiv \delta & \end{array}$$

$$\begin{array}{lll} (\delta_1 \circ \delta_2)[\delta_3 \circ \delta_4] & \equiv \delta_1[\delta_3] \circ \delta_2[\delta_4] & \textit{interchange} \\ \mathsf{refl}_{\theta}[\delta] & \equiv \theta[\delta] & \textit{delegate} \end{array}$$

$$\begin{array}{lll} \mathsf{refl}^{\mathsf{op}}_{\theta} & \equiv \mathsf{refl}_{\theta^{\mathsf{op}}} & {}^{\mathsf{op}}\ \textit{interactions} \\ (\delta_1 \circ \delta_2)^{\mathsf{op}} & \equiv \delta_2^{\mathsf{op}} \circ \delta_1^{\mathsf{op}} & \\ (\delta_1[\delta_2])^{\mathsf{op}} & \equiv \delta_1^{\mathsf{op}}[\delta_2^{\mathsf{op}}] & \end{array}$$

Fig. 1. 2DTT: Identity, Composition, and Involution Principles (1)

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

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

$$\begin{aligned} A[\theta[\theta']] &\equiv A[\theta][\theta'] && \textit{0-subst assoc/unit}\\ A[\mathsf{id}_\Gamma] &\equiv A \end{aligned}$$

$$\begin{aligned} \mathsf{map}_{\Delta.C}\ \mathsf{refl}_\theta\ M &\equiv M && \textit{0-resp functoriality}\\ \mathsf{map}_{\Delta.C}\ (\delta_2 \circ \delta_1)\ M &\equiv \mathsf{map}_{\Delta.C}\ \delta_2\ (\mathsf{map}_{\Delta.C}\ \delta_1\ M) \end{aligned}$$

$$\begin{aligned} (\mathsf{map}_{\Delta.C}\ \delta\ M)[\theta_0] &\equiv \mathsf{map}_{\Delta.C}\ \delta[\mathsf{refl}_{\theta_0}]\ M[\theta_0] && \textit{1-subst for } \mathsf{map}\\ (\mathsf{map}_C\ (\delta : \theta_1 \Longrightarrow \theta_2)\ M)[\delta' : \theta'_1 \Longrightarrow \theta'_2] &\equiv \mathsf{resp}\ (x.\mathsf{map}_C\ (\delta[\mathsf{refl}_{\theta'_2}])\ x)\ (M[\delta']) && \textit{1-resp for } \mathsf{map}\\ \mathsf{map}_{\Delta.C[\theta:\Delta']}\ \delta\ M &\equiv \mathsf{map}_{\Delta'.C}\ \mathsf{refl}_\theta[\delta]\ M && \textit{def. } \mathsf{map} \textit{ for } A[\theta] \end{aligned}$$

Composition for $\Gamma \vdash M : A$

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

$$\begin{aligned} M[\theta[\theta']] &\equiv M[\theta][\theta'] && \textit{1-subst assoc/unit}\\ M[\mathsf{id}_\Gamma] &\equiv M \end{aligned}$$

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

Identity and Composition for $\Gamma \vdash \alpha : M \Longrightarrow_A N$

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

$$\begin{aligned} (\alpha_3 \circ \alpha_2) \circ \alpha_1 &\equiv \alpha_3 \circ (\alpha_2 \circ \alpha_1) && \textit{trans assoc/unit}\\ (\alpha \circ \mathsf{refl}) &\equiv \alpha\\ (\mathsf{refl} \circ \alpha) &\equiv \alpha \end{aligned}$$

$$\begin{aligned} \alpha[\delta[\delta']] &\equiv \alpha[\delta][\delta'] && \textit{2-resp assoc/unit}\\ \alpha[\mathsf{refl}_{\mathsf{id}}] &\equiv \alpha \end{aligned}$$

$$\begin{aligned} (\alpha_1 \circ \alpha_2)[\delta_3 \circ \delta_4] &\equiv \alpha_1[\delta_3] \circ \mathsf{resp}\ (x.\mathsf{map}\ \delta_3\ x)\ (\alpha_2[\delta_4]) && \textit{interchange}\\ \mathsf{refl}_M[\delta] &\equiv M[\delta] && \textit{delegate} \end{aligned}$$

Fig. 2. 2DTT: Identity, Composition, and Involution Principles (2)

All judgements respect equality:

$$\frac{\Gamma \equiv \Gamma' \quad \Delta \equiv \Delta' \quad \Gamma' \vdash \theta : \Delta'}{\Gamma \vdash \theta : \Delta} \qquad \frac{\Gamma \equiv \Gamma' \quad \Gamma' \vdash A\ \mathsf{type}}{\Gamma \vdash A\ \mathsf{type}} \qquad \frac{\Gamma \equiv \Gamma' \quad \Gamma \vdash A \equiv A'\ \mathsf{type} \quad \Gamma' \vdash M : A'}{\Gamma \vdash M : A}$$

$$\frac{\Gamma \equiv \Gamma' \quad \Delta \equiv \Delta' \quad \Gamma \vdash \theta_1 \equiv \theta_1' : A \quad \Gamma \vdash \theta_2 \equiv \theta_2' : A \quad \Gamma' \vdash \delta : \theta_1' \Longrightarrow_{\Delta'} \theta_2'}{\Gamma \vdash \delta : \theta_1 \Longrightarrow_{\Delta} \theta_2}$$

$$\frac{\Gamma \equiv \Gamma' \quad \Gamma \vdash A \equiv A'\ \mathsf{type} \quad \Gamma \vdash M \equiv M' : A \quad \Gamma \vdash N \equiv N' : A \quad \Gamma' \vdash \alpha : M' \Longrightarrow_{A'} N'}{\Gamma \vdash \alpha : M \Longrightarrow_{A} N}$$

Equality respects equality: Each equality judgement has an analogous respect-for-equality rule, which says that it respects equality in the context and classifier.

Congruence: Each equality judgement is a congruence, specified by reflexivity, symmetry, transitivity rules, and a compatibility rule for each term constructor.

Empty context:

$$\frac{}{\cdot\ \mathsf{ctx}} \qquad \frac{}{\Gamma \vdash \cdot : \cdot} \qquad \frac{}{\Gamma \vdash \cdot : \cdot \Longrightarrow_{\cdot} \cdot}$$

$$\begin{array}{lll}
\theta & \equiv_{\cdot} & \textit{1-}\eta \\
\delta & \equiv_{\cdot} & \textit{2-}\eta \\
.^{\mathsf{op}} & \equiv_{\cdot} & \textit{0,1,2-involution} \\
\mathsf{id}. & \equiv_{\cdot} & \textit{identity} \\
\cdot[\theta] & \equiv_{\cdot} & \textit{1-subst} \\
\cdot[\delta] & \equiv_{\cdot} & \textit{1-resp} \\
\mathsf{refl}. & \equiv_{\cdot} & \textit{reflexivity} \\
\cdot \circ \cdot & \equiv_{\cdot} & \textit{trans} \\
\cdot[\delta] & \equiv_{\cdot} & \textit{2-resp}
\end{array}$$

Fig. 3. 2DTT: General equality rules; Empty Context

2.1 Involution, Identity, and Composition Principles

In Figures 1 and 2, we present the generic involution, identity, and composition principles that define the basic structure of the theory.

The involution rules say that there is a dualizing operation $^{\mathsf{op}}$ on contexts, substitutions, and transformations. The equations say that this dualization operation is involutive. The rule for θ^{op} says that the opposite of a substitution proves the opposite of the contexts. 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, because $^{\mathsf{op}}$ is an involution, an "introduction" rule which concludes $\Gamma^{\mathsf{op}} \vdash \theta^{\mathsf{op}} : \Delta^{\mathsf{op}}$ from $\Gamma \vdash \theta : \Delta$ is derivable. The dual of a transformation not only dualizes the contexts and substitutions, but also reverses the direction of the transformation.

Covariant term variables:

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

$$\begin{array}{rcll}
\mathsf{id}_\Gamma^{\Gamma, x:A^+}[\theta, M^+/x] & \equiv & \theta & \textit{1-}\beta \\
x[\theta, M^+/x] & \equiv & M & \textit{1-}\beta \\
\theta : (\Gamma\ ,\ x{:}A^+) & \equiv & \mathsf{id}_\Gamma[\theta], x[\theta]^+/x & \textit{1-}\eta \\
\mathsf{id}_\Gamma^{\Gamma, x:A^+}[\delta, \alpha^+/x] & \equiv & \delta & \textit{2-}\beta \\
x[\delta, \alpha^+/x] & \equiv & \alpha & \textit{2-}\beta \\
\delta : \theta \Longrightarrow_{(\Gamma, x:A^+)} \theta' & \equiv & \mathsf{id}_\Gamma[\delta], x[\delta]^+/x & \textit{2-}\eta \\
 & & & \\
\mathsf{id}_{\Gamma, x:A^+} & \equiv & \mathsf{id}_\Gamma, x^+/x & \textit{1-id} \\
(\theta, M^+/x)[\theta_0] & \equiv & \theta[\theta_0], M[\theta_0]^+/x & \textit{1-subst} \\
(\theta, M^+/x)[\delta_0] & \equiv & \theta[\delta_0], M[\delta_0]^+/x & \textit{1-resp} \\
\mathsf{refl}_{\theta, M^+/x} & \equiv & \mathsf{refl}_\theta, \mathsf{refl}_{M^+}/x & \textit{refl} \\
(\delta_2, \alpha_2{}^+/x) \circ (\delta_1, \alpha_1{}^+/x) & \equiv & (\delta_2 \circ \delta_1), (\alpha_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ \delta_2\ x)\ \alpha_1)^+/x & \textit{trans} \\
(\delta, \alpha^+/x)[\delta_0] & \equiv & \delta[\delta_0], \alpha[\delta_0]^+/x & \textit{2-resp} \\
 & & & \\
(\Gamma\ ,\ x{:}A^+)^{\mathsf{op}} & \equiv & \Gamma^{\mathsf{op}}\ ,\ x{:}A^- & \textit{0-invol} \\
(\theta, M^+/x)^{\mathsf{op}} & \equiv & \theta^{\mathsf{op}}, M^-/x & \textit{1-invol} \\
(\delta, \alpha^+/x)^{\mathsf{op}} & \equiv & \delta^{\mathsf{op}}, \alpha^-/x & \textit{2-invol}
\end{array}$$

Contravariant term variables:

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

$$\begin{array}{rcll}
\mathsf{id}_\Gamma^{\Gamma, x:A^-}[\theta, M^-/x] & \equiv & \theta & \textit{1-}\beta \\
\theta : (\Gamma\ ,\ x{:}A^-) & \equiv & \mathsf{id}_\Gamma[\theta], x[\theta^{\mathsf{op}}]^-/x & \textit{1-}\eta \\
\mathsf{id}_\Gamma^{\Gamma, x:A^-}[\delta, \alpha^-/x] & \equiv & \delta & \textit{2-}\beta \\
\delta : \theta \Longrightarrow_{(\Gamma, x:A^-)} \theta' & \equiv & \mathsf{id}_\Gamma[\delta], x[\delta^{\mathsf{op}}]^-/x & \textit{2-}\eta \\
 & & & \\
\mathsf{id}_{\Gamma, x:A^-} & \equiv & \mathsf{id}_\Gamma, x^-/x & \textit{1-id} \\
(\theta, M^-/x)[\theta_0] & \equiv & \theta[\theta_0], M[\theta_0^{\mathsf{op}}]^-/x & \textit{1-subst} \\
(\theta, M^-/x)[\delta_0] & \equiv & \theta[\delta_0], M[\delta_0^{\mathsf{op}}]^-/x & \textit{1-resp} \\
\mathsf{refl}_{\theta, M^-/x} & \equiv & \mathsf{refl}_\theta, \mathsf{refl}_{M^-}/x & \textit{refl} \\
(\delta_2, \alpha_2{}^-/x) \circ (\delta_1, \alpha_1{}^-/x) & \equiv & (\delta_2 \circ \delta_1), (\alpha_2 \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta^{\mathsf{op}}.A}\ \delta_2^{\mathsf{op}}\ x)\ \alpha_1)^-/x & \textit{trans} \\
(\delta, \alpha^-/x)[\delta_0] & \equiv & \delta[\delta_0], \alpha[\delta_0^{\mathsf{op}}]^-/x & \textit{2-resp} \\
 & & & \\
(\Gamma\ ,\ x{:}A^-)^{\mathsf{op}} & \equiv & \Gamma^{\mathsf{op}}\ ,\ x{:}A^+ & \textit{0-invol} \\
(\theta, M^-/x)^{\mathsf{op}} & \equiv & \theta^{\mathsf{op}}, M^+/x & \textit{1-invol} \\
(\delta, \alpha^-/x)^{\mathsf{op}} & \equiv & \delta^{\mathsf{op}}, \alpha^+/x & \textit{2-invol}
\end{array}$$

Fig. 4. 2DTT: Co- and Contravariant Term Variables

Dependent functions:

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

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

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

Fig. 5. 2DTT: Dependent Function Types

Identity and Composition for Substitutions

The next three rules define identity and composition for substitutions. To make weakening admissible, id is really the composition of the identity substitution with *projections* that forget any number of variables. We write $\Gamma \supseteq \Delta$ to mean that Δ is obtained from Γ by dropping some number of variables:

$$\frac{}{\Gamma \supseteq \cdot}\ \textit{done} \qquad \frac{\Gamma \supseteq \Gamma'}{\Gamma\,,\, x{:}A^{\pm} \supseteq \Gamma'}\ \textit{skip} \qquad \frac{\Gamma \supseteq \Gamma'}{\Gamma\,,\, x{:}A^{\pm} \supseteq \Gamma'\,,\, x{:}A^{\pm}}\ \textit{keep}$$

We do not require a rule for $^{\mathsf{op}}$ because $^{\mathsf{op}}$ can always be expanded away using equalities. All judgements of the form $\Gamma \vdash J$ satisfy the following: If $\Gamma \vdash J$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash J$.

Composition of substitutions $\theta_2[\theta_2]$, which we refer to as 1-substitution, is standard in explicit substitution calculi. The additional composition operation, $\theta[\delta]$, forces substitutions to *respect transformation*: substitution instances by transformable substitutions are transformable. For this reason, we refer to it as *1-resp(ect)*. The first three equations say that 1-substitution is associative and unital. In the second equation, id_Γ can in fact be a weakening, in which case θ is tacitly weakened in the right-hand side. The third equation only makes sense when $\Gamma \vdash \mathsf{id} : \Gamma$, which we notate by $\mathsf{id}^\Gamma_\Gamma$. The next two rules say that 1-resp associates with 2-resp $(\delta[\delta'])$, which is the analogous operation for transformations (defined

Dependent pairs:

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

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

$$\frac{\Gamma \vdash \alpha : M \Longrightarrow_{\Sigma\, x:A.\, B} N}{\Gamma \vdash \mathsf{snd}\ \alpha : (\mathsf{map}^1_B\ (\mathsf{fst}\ \alpha)\ (\mathsf{snd}\ M)) \Longrightarrow_{B[\mathsf{fst}\ N/x]} \mathsf{snd}\ N}$$

$\mathsf{fst}\ (M, N)$	$\equiv M$	*1-β*
$\mathsf{snd}\ (M, N)$	$\equiv N$	*1-β*
$M : \Sigma\, x{:}A.\, B$	$\equiv (\mathsf{fst}\ M, \mathsf{snd}\ M)$	*1-η*
$\mathsf{fst}\ (\alpha_1, \alpha_2)$	$\equiv \alpha_1$	*2-β*
$\mathsf{snd}\ (\alpha_1, \alpha_2)$	$\equiv \alpha_2$	*2-β*
$\alpha : M \Longrightarrow_{\Sigma\, x:A.\, B} N$	$\equiv (\mathsf{fst}\ \alpha, \mathsf{snd}\ \alpha)$	*2-η*
$(\Sigma\, x{:}A.\, B)[\theta_0]$	$\equiv \Sigma\, x{:}A[\theta_0].\, B[\theta_0, x^{+}/x]$	*0-subst*
$\mathsf{map}_{\Delta.\Sigma\, x:A.\, B}\ \delta\ M$	$\equiv (\mathsf{map}_{\Delta.A}\ \delta\ (\mathsf{fst}\ M), \mathsf{map}_{\Delta, x:A^{+}.B}\ (\delta, \mathsf{refl}_{\mathsf{map}\ \delta\ (\mathsf{fst}\ M)})\ (\mathsf{snd}\ M))$	*0-resp*
$((M_1, M_2))[\theta_0]$	$\equiv (M_1[\theta_0], M_2[\theta_0])$	*1-subst*
$(\mathsf{fst}\ M)[\theta_0]$	$\equiv \mathsf{fst}\ (M[\theta_0])$	*1-subst*
$(\mathsf{snd}\ M)[\theta_0]$	$\equiv \mathsf{snd}\ (M[\theta_0])$	*1-subst*
$(M, N)[\delta]$	$\equiv (M[\delta], N[\delta])$	*1-resp*
$(\mathsf{fst}\ M)[\delta]$	$\equiv \mathsf{fst}\ M[\delta]$	*1-resp*
$(\mathsf{snd}\ M)[\delta]$	$\equiv \mathsf{snd}\ M[\delta]$	*1-resp*
refl_M	$\equiv (\mathsf{refl}_{\mathsf{fst}\ M}, \mathsf{refl}_{\mathsf{snd}\ M})$	*refl*
$(\alpha_2, \alpha_2') \circ (\alpha_1, \alpha_1')$	$\equiv (\alpha_2 \circ \alpha_1, \alpha_2' \circ \mathsf{resp}\ (x.\mathsf{map}_{\Delta.A}\ (\mathsf{refl}, \alpha_1)\ x)\ \alpha_1')$	*trans*
$(\alpha_1, \alpha_2)[\delta_0]$	$\equiv (\alpha_1[\delta_0], \alpha_2[\delta_0, \mathsf{refl}])$	*2-resp*
$(\mathsf{fst}\ \alpha)[\delta_0]$	$\equiv \mathsf{fst}\ \alpha[\delta_0]$	*2-resp*
$(\mathsf{snd}\ \alpha)[\delta_0]$	$\equiv \mathsf{snd}\ \alpha[\delta_0]$	*2-resp*

Fig. 6. 2DTT: Dependent Pairs

below), and preserves identities refl(defined below). The next three rules say that $^{\mathsf{op}}$ preserves identities (and projections), and distributes over compositions. As will often be the case, the second rule (for 1-cells) is necessary to type-check the third (for 2-cells).

Identity and Composition For Transformations

The next three rules define identity and composition for transformations. Transformations are always reflexive (refl) and transitive ($\delta_2 \circ \delta_1$). Additionally, transformations themselves respect transformation ($\delta[\delta_0]$), which we call 2-resp. The equations say that: Transitivity is associative and unital with reflexivity. 2-resp is also associative and unital. However, the unit of 2-resp is not an arbitrary refl_θ—which

Sets and elements:

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

$$\begin{array}{rcll}
\mathsf{map}_{El(S)}\ \delta\ M[\theta_1] & \equiv & M[\theta_2] & \textit{def. } El(-) \\
\alpha : S \Longrightarrow_{\mathsf{set}} S' & \equiv & x.\mathsf{map}_{a:Set.El(a)}\ (\cdot, \alpha)\ x & \textit{2-}\eta \\
\alpha : M \Longrightarrow_{El(S)} N & \equiv & \star & \textit{2-}\eta \\
 & & & \\
\mathsf{set}[\theta_0] & \equiv & \mathsf{set} & \textit{0-subst} \\
(El(S))[\theta_0] & \equiv & El(S[\theta_0]) & \\
\mathsf{map}_{\Delta.\mathsf{set}}\ \delta\ M & \equiv & M & \textit{0-resp} \\
\mathsf{map}_{\Delta.El(S)}\ \delta\ M & \equiv & N[\mathsf{id}, M^{+}/x] \ \text{ if } \ S[\delta] \equiv x.N & \\
\mathsf{refl}_{S\,:\,\mathsf{set}} & \equiv & x.x & \textit{refl} \\
\mathsf{refl}_{M\,:\,El(S)} & \equiv & \star & \textit{refl} \\
(x.M_1) \circ (x.M_2) & \equiv & x.M_2[M_1/x] & \textit{trans} \\
\star \circ \star & \equiv & \star & \textit{trans} \\
(x.M)[\delta_0] & \equiv & x.\mathsf{map}_{\Delta.S'}\ \delta_0\ M[\theta, x^{+}/x] \text{ if } \Delta\,,\, x{:}El(S)^{+} \vdash M : El(S') \text{ and } \delta_0 : \theta \Longrightarrow \theta' & \textit{2-resp} \\
\star_M[\delta_0] & \equiv & M[\delta_0] & \textit{2-resp}
\end{array}$$

Fig. 7. 2DTT: General Rules for Sets and Elements

would still require adapting a transformation $\delta : \theta \Longrightarrow \theta'$ to $\theta[\theta_0] \Longrightarrow \theta[\theta_0]$—but only $\Gamma \vdash \mathsf{refl}_{\mathsf{id}_\Gamma^\Gamma} : \mathsf{id}_\Gamma \Longrightarrow_\Gamma \mathsf{id}_\Gamma$ (by above, $\theta[\mathsf{id}_\Gamma]$ equals θ). As above, the second rule holds when id is in fact a projection, but the third requires that it really be the identity. The *interchange law* relates 2-resp and transitivity: transitivity followed by 2-resp is the same as 2-resp followed by transitivity. It has a variety of useful special cases:

$$\begin{array}{rcll}
\theta[\delta \circ \delta'] & \equiv & \theta[\delta] \circ \theta[\delta'] & \textit{1-resp preserves transitivities} \\
(\delta \circ \delta')[\mathsf{refl}_\theta] & \equiv & \delta[\mathsf{refl}_\theta] \circ \delta'[\mathsf{refl}_\theta] & \textit{2-resp preserves transitivities} \\
(\delta : \theta_1 \Longrightarrow \theta_2)[\delta' : \theta_1' \Longrightarrow \theta_2'] & \equiv & \mathsf{refl}_{\theta_2}[\delta'] \circ \delta[\mathsf{refl}_{\theta_1'}] & \textit{2-resp interchange 1} \\
(\delta : \theta_1 \Longrightarrow \theta_2)[\delta' : \theta_1' \Longrightarrow \theta_2'] & \equiv & \delta[\mathsf{refl}_{\theta_2'}] \circ \mathsf{refl}_{\theta_1}[\delta'] & \textit{2-resp interchange 2}
\end{array}$$

The first two equations say that resp preserves transitivities. The next two state that a 2-resp is equivalent to holding one part fixed while doing one transformation, then holding the other fixed while doing the other—in either order. Returning to the figure, the rule *delegate* delegates 2-resp at reflexivity to 1-resp. The final three rules say that $^{\mathsf{op}}$ preserves identities and compositions, reversing the order of composition in the case of transitivity.

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

$$\frac{\Gamma \vdash M : \Sigma\, x{:}El(S).\ El(S')}{\Gamma \vdash \mathsf{in}\, M : El(\Sigma\, x{:}S.\ S')} \qquad \frac{\Gamma \vdash M : El(\Sigma\, x{:}S.\ S')}{\Gamma \vdash \mathsf{out}\, M : \Sigma\, x{:}El(S).\ El(S')}$$

$$\frac{\Gamma \vdash M : \Pi\, x{:}El(S).\ El(S')}{\Gamma \vdash \mathsf{in}\, M : El(\Pi\, x{:}S.\ S')} \qquad \frac{\Gamma \vdash M : El(\Pi\, x{:}S.\ S')}{\Gamma \vdash \mathsf{out}\, M : \Pi\, x{:}El(S).\ El(S')}$$

$$\frac{}{\Gamma \vdash () : El(1)} \qquad \frac{}{\Gamma \vdash \mathsf{true} : El(2)} \qquad \frac{}{\Gamma \vdash \mathsf{false} : El(2)} \qquad \frac{\Gamma^{\pm} \vdash M : El(0)}{\Gamma \vdash \mathsf{abort}\, M : C} \qquad \frac{\Gamma^{\pm} \vdash M : El(0)}{\Gamma \vdash \mathsf{abort}\, M : M_1 \Longrightarrow_C M_2}$$

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

$$\frac{\Gamma \vdash \alpha : M \Longrightarrow_{El(S)} N}{\Gamma \vdash \mathsf{idi}\, \alpha : \mathsf{Id}_S\ M\ N} \qquad \frac{\Gamma \vdash P : \mathsf{Id}_{El(S)}\ M\ N}{\Gamma \vdash \mathsf{ide}\, P : M \Longrightarrow_{El(S)} N}$$

Rules for sets:

$$\begin{array}{lll}
\Pi\, x{:}S.\ S'[\theta] & \equiv \Pi\, x{:}S[\theta].\ S'[\theta, x^{-}/x] & \textit{1-subst}\\
\Sigma\, x{:}S.\ S'[\theta] & \equiv \Sigma\, x{:}S[\theta].\ S'[\theta, x^{+}/x] & \\
\{0,1,2\}[\theta] & \equiv \{0,1,2\} & \\
\mathsf{Id}_S\ M\ N[\theta] & \equiv \mathsf{Id}_{A[\theta]}\ M[\theta]\ N[\theta] & \\
\{0,1,2\}[\delta] & \equiv x.x & \textit{1-resp}\\
(\Pi\, x{:}S.\ S')[\delta : \theta \Longrightarrow \theta'] & \equiv x.\mathsf{in}\,(\mathsf{map}_{\Pi\, x:El(S).\ El(S')}\ \delta\ (\mathsf{out}\ M)) & \\
(\Sigma\, x{:}S.\ S')[\delta : \theta \Longrightarrow \theta'] & \equiv x.\mathsf{in}\,(\mathsf{map}_{\Sigma\, x:El(S).\ El(S')}\ \delta\ (\mathsf{out}\ M)) & \\
\mathsf{Id}_S\ M\ N[\delta : \theta \Longrightarrow \theta'] & \equiv x.\mathsf{idi}\ \star &
\end{array}$$

Fig. 8. 2DTT: Some Sets (1)

We do not define 2-subst, $\delta[\theta]$, directly, as this composition is definable as $\delta[\mathsf{refl}_\theta]$. Alternatively, we could take $\delta[\theta]$ as primitive and define $\delta[\delta']$ using the interchange law. However, it is in fact no harder to define $\delta[\delta']$, as the rules for the binary version also proceed compositionally in the term, just like ordinary substitution with a single θ would. This fact suggests that it may be possible to treat 2-resp as a meta-operation, which may be a helpful implementation technique.

Dependent Types

In Figure 2, we define identity and composition for dependent types, terms, and term transformations. A dependent type A can be pre-composed with a substitution, written $A[\theta]$; and has a functorial action $\mathsf{map}_{\Delta.A}\ \delta\ M$, which is the analogue of the subst elimination rule for propositional equality described above. map says that a

Rules for elements:

$$
\begin{array}{lll}
\mathsf{out}\,(\mathsf{in}\,M) & \equiv M & \beta\eta \\
\mathsf{in}\,(\mathsf{out}\,M) & \equiv M & \\
M : El(\mathsf{1}) & \equiv () & \\
M[(N : El(0))^{\pm}/x] & \equiv \mathsf{abort}\,N & \\
\mathsf{if}\,(\mathsf{true}, M_1, M_2) & \equiv M_1 & \\
\mathsf{if}\,(\mathsf{false}, M_1, M_2) & \equiv M_2 & \\
M[(N : El(2))^{\pm}/x] & \equiv \mathsf{if}\,(N, M[\mathsf{true}^{\pm}/x], M[\mathsf{false}^{\pm}/x]) & \\
\mathsf{ide}\,(\mathsf{idi}\,\alpha) & \equiv \alpha & \\
P : \mathsf{Id}_S\ M\ N & \equiv \mathsf{idi}\,(\mathsf{ide}\,P) & \\
 & & \\
(\mathsf{in}\,M)[\theta] & \equiv \mathsf{in}\,M[\theta] & \textit{1-subst} \\
(\mathsf{out}\,M)[\theta] & \equiv \mathsf{out}\,M[\theta] & \\
\{(), \mathsf{true}, \mathsf{false}\}[\theta] & \equiv \{(), \mathsf{true}, \mathsf{false}\} & \\
\mathsf{if}_{x\,:\,2^{\pm}.C}(M, M_1, M_2)[\theta^{\Gamma}_{\Delta}] & \equiv \mathsf{if}_{x\,:\,2^{\pm}.C[\theta, x^{\pm}/x]}(M[\theta], M_1[\theta], M_2[\theta]) & \\
(\mathsf{idi}\,\alpha)[\theta] & \equiv \mathsf{idi}\,\alpha[\mathsf{refl}_{\theta}] & \\
(\mathsf{ide}\,P)[\delta] & \equiv \star & \textit{2-resp} \\
 & & \\
(\mathsf{in}\,M)[\delta] & \equiv \star & \textit{1-resp} \\
(\mathsf{out}\,M)[\delta] & \equiv \star & \\
\{(), \mathsf{true}, \mathsf{false}\}[\delta] & \equiv \star & \\
\mathsf{idi}\,\alpha[\delta] & \equiv \mathsf{idi}\,\star & \\
\mathsf{if}\,(M, M_1, M_2)[\delta : \theta_1 \Longrightarrow \theta_2] & \equiv \mathsf{if}\,(M[\theta_2], M_1[\delta], M_2[\delta]) &
\end{array}
$$

Rules for transformation elims for positives:

$$
\begin{array}{rcll}
\mathsf{if}_{x.C}(\mathsf{true}, \alpha_1, \alpha_2) & \equiv & \alpha_1 & \beta \\
\mathsf{if}_{x.C}(\mathsf{false}, \alpha_1, \alpha_2) & \equiv & \alpha_2 & \\
\alpha[(\mathsf{refl}_{M\,:\,0})^{\pm}/x] & \equiv & \mathsf{abort}\,M & \eta \\
\alpha[(\mathsf{refl}_{M\,:\,2})^{\pm}/x] & \equiv & \mathsf{if}\,(M, \alpha[\mathsf{refl}_{\mathsf{true}}/x], \alpha[\mathsf{refl}_{\mathsf{false}}/x]) & \\
 & & & \\
(\mathsf{abort}\,M)[\delta : \theta_1 \Longrightarrow \theta_2] & \equiv & \mathsf{abort}\,M[\theta_2] & \textit{2-resp} \\
\mathsf{if}\,(M, \alpha_1, \alpha_2)[\delta : \theta_1 \Longrightarrow \theta_2] & \equiv & \mathsf{if}\,(M[\theta_2], \alpha_1[\delta], \alpha_2[\delta]) &
\end{array}
$$

Fig. 9. 2DTT: Some Sets (2)

transformation $\theta_1 \Longrightarrow \theta_2$ allows a term of type $A[\theta_1]$ to be coerced to a term of type $A[\theta_2]$. This says that *dependent types respect transformation*. We refer to these as *0-subst* and *0-resp*; there are no 0-subst/resp for contexts because contexts are not dependent.

The equations say: Substitution into types (0-subst) is associative with unit refl. map is functorial, preserving reflexivity and transitivity. The next two rules define 1-subst and 1-resp for map, which reassociate the 1-subst/1-resp with the 0-resp. The next rule defines map for a composition, again by reassociating.

Interactions between map and 2-resp are derivable from the above equations for transitivity, using the interchange law:

$$\mathsf{map}_C\ (\delta : \theta_1 \Longrightarrow \theta_2)[\delta' : \theta'_1 \Longrightarrow \theta'_2]\ M \equiv \mathsf{map}_C\ (\delta[\mathsf{refl}_{\theta'_2}])\ (\mathsf{map}_C\ \mathsf{refl}_{\theta_1}[\delta']\ M)$$

$$\mathsf{map}_C\ (\delta : \theta_1 \Longrightarrow \theta_2)[\delta' : \theta'_1 \Longrightarrow \theta'_2]\ M \equiv \mathsf{map}_C\ (\mathsf{refl}_{\theta_2}[\delta])\ (\mathsf{map}_C\ \delta[\mathsf{refl}_{\theta'_1}]\ M)$$

The interchange rule, along with the following, form the Godement calculus of functors and natural transformation composition [18]:

$$(\mathsf{refl}_{\theta[\theta']})[\delta] \equiv \mathsf{refl}_{\theta}[\mathsf{refl}_{\theta'}[\delta]]$$

This rule is derivable because $\mathsf{refl}_{\theta\circ\theta'} \equiv \mathsf{refl}_{\theta}[\mathsf{refl}'_{\theta}]$ by *1-resp-preserves-refl* and associativity.

There is also a right unit law for *resp* and $\mathsf{refl}_{\mathsf{id}}$:

$$\mathsf{refl}_{\theta}[\mathsf{refl}_{\mathsf{id}}] \equiv \mathsf{refl}_{\theta}$$

It is derivable using *1-resp preserves refl.* and unit of id.

Terms

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

An associativity rule for 1-subst followed by 1-resp in a term is derivable, using 1-resp-preserves-refl, congruence, and delegate and associativity for 2-resp (see below):

$$M[\theta][\delta] \equiv M[\mathsf{refl}_{\theta}[\delta]] \quad \textit{1-resp for } M[\theta]$$

Using interchange, we can derive that M preserves transitivities from the above interaction with 2-resp:

$$\mathsf{refl}_{M}[\delta \circ \delta'] \equiv \mathsf{refl}_{M}[\delta] \circ (\mathsf{resp}\ (x.\mathsf{map}\ \delta\ x)\ (\mathsf{refl}_{M}[\delta']))$$

Term Transformations

The rules for term transformations are entirely analogous to the rules for transformations, specifying reflexivity, transitivity, and 2-resp. The equations say that transitivity is associative and unital, that 2-resp is associative and unital, and that the order of trans and 2-resp can be interchanged. The interchange rule uses the derived form resp , which is explained below.

General equality rules

Figure 3 collects a variety of general equality rules: Each judgement, including equality, respects equality of its indices, and is a congruence.

2.2 Contexts

With the basic setup in hand, we are ready to define some concrete context formers. The general methodology for defining a context is to specify (1) A formation rule for Γ. (2) A substitution rule $\theta : \Gamma$, and a hypothesis rule for one of the other judgements

(e.g. the term rule for x for the context former $\Gamma\,,\,x{:}A^+$). These function as the introduction and elimination rules for the context, which are products of some sort, eliminated by first projections (which are implicit in id) and variables (representing projections). (3) A transformation rule for $\delta : \theta \Longrightarrow_\Gamma \theta'$ (4) Equations defining

1-$\beta\eta$ $\beta\eta$ for θ	*0-involution* Γ^{op}	*identity* id_Γ	*reflexivity* refl_θ
2-$\beta\eta$ $\beta\eta$ for δ	*1-involution* θ^{op}	*1-subst* $\theta[\theta']$	*transitivity* $\delta \circ \delta'$
	2-involution δ^{op}	*1-resp* $\theta[\delta']$	*2-resp* $\delta[\delta']$

In general, refl and $\delta \circ \delta'$ are defined in a type-directed manner, by giving one rule that covers arbitrary arguments. On the other hand, the subst/resp principles are defined in a syntax-directed manner, giving one rule for each syntactic construct.

In Figure 3 and Figure 4 we carry out this methodology for the basic contexts:

Empty context

The empty context has a trivial substitution into it, and a trivial transformation from this substitution to itself. Since the substitutions and transformations are singletons, the equations are all trivial.

Covariant context extensions

Next, we present the rules for covariant context extension: if A is a type well-formed in Γ, then Γ can be extended with a variable of type A. A covariant variable can be used as a term; the typing rule checks that the variable is in the context:

$$\frac{}{x : A^+ \in (\Gamma, A^+)} \qquad \frac{x : A^+ \in \Gamma}{x : A^+ \in (\Gamma\,,\,y{:}B^\pm)}$$

As with weakening, we do not include a rule for $^{\mathsf{op}}$, which can be expanded away. The substitution into an extended context $\Delta\,,\,x{:}A^+$ is a pair of a substitution θ into Δ and a term of type A, adjusted by θ (this is analogous to the usual introduction rule for a Σ-type). A transformation between such substitutions is a pair of transformations, one between the substitutions, and the other between the terms (adjusted by the first component). As these substitutions and transformations are pairs, the first set of rules gives the expected $\beta\eta$ rules, for the projections given by id and variables. The next rules define the identity, composition, and involution operations componentwise. The involution rules turn covariant context extension into contravariant context extension, which is defined below.

The familiar resp congruence rule derives respect for the last variable in the context:

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

by $\mathsf{resp}\ F\ \alpha = F[\mathsf{id}_{\mathsf{id}}, \alpha^+/x]$. This is well-typed because $\mathsf{map}\ \mathsf{id}$ cancels.

We will also make use of the corresponding rule for map, a derived form for transforming the last variable in the context:

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

This is defined by $\mathsf{map}^1_{x:A^+.B}\ \alpha\ M = \mathsf{map}_{\Gamma,x:A^+.B}\ \mathsf{id}, \alpha^+/x\ M$.

Contravariant context extensions

Next, we present the rules for contravariant context extension: if A is a type well-formed in Γ^{op}, then Γ can be extended with a variable of type A. For the most part, these are analogous to covariant context extension, except for the following: First, there is no rule for using a contravariant variable. This is because we reduce contravariant terms to covariant terms using $^{\mathsf{op}}$, and using the rules for $\Gamma, x{:}A^{-\mathsf{op}}$, a contravariant variable becomes a covariant variable. Second, the transformation rule reverses the order of M and N in the premise. Third, various $^{\mathsf{op}}$'s are inserted on substitutions and transformations to make the types work out.

A contravariant last-variable map rule is also definable:

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

by $\mathsf{map}^1_{x:A^-.B}\ \alpha\ M = \mathsf{map}_{\Gamma,x:A^-.B}\ (\mathsf{id}, \alpha^-/x)\ M$.

2.3 Types

With contexts in hand, we can move on to types and terms. In general, a type is specified by (1) A formation rule for A. (2) Introduction and elimination term rules, defining $M : A$. (3) Introduction and elimination transformation rules, defining $\alpha : M \Longrightarrow_A M'$. (4) Equations defining

1-$\beta\eta$ $\beta\eta$ for M	*0-substitution* $A[\theta]$	*1-substitution* $M[\theta]$	*reflexivity* refl_M
2-$\beta\eta$ $\beta\eta$ for α	*0-resp* $\mathsf{map}_A\ \delta\ M$	*1-resp* $M[\delta]$	*transitivity* $\alpha \circ \alpha'$
			2-resp $\alpha[\delta]$

Dependent functions

In Figure 5, we give the rules for dependent functions. The formation rule is standard, except that the domain is well-formed contravariantly in Γ, and thus assumed as a contravariant assumption. The intro and elim rules then insert the appropriate $^{\mathsf{op}}$'s. The transformation introduction rule says that a transformation at Π can be introduced by giving a family of transformations that work for each element—the extensionality rule. A transformation is eliminated by applying to transformable arguments. The symmetric variants of these transformation rules, and the equations for them described below, have been considered in prior categorically-motivated accounts of functionally extensional propositional equality [17].

The $\beta\eta$-rules are the expected rules for functions, both at the term and transformation levels. We write $M[N/x]$ to abbreviate $M[\mathsf{id}, N/x]$. Substitution into a Π-type proceeds compositionally, though we always $^{\mathsf{op}}$ the substitution in contravariant positions. $\mathsf{map}_{\Pi\, x:A.\, B}$ is given by pre- and post-composition. Note that the definition of transformation at $\Delta, x : A^-$ is just right so that δ, refl works as the post-composition. 1-subst and 1-resp are both defined compositionally, as is 2-resp. The rule for refl says that the identity at all elements is the identity. In the definition of transitivity, we again cheat by assuming the transformations are in introductory form, which makes sense because of η, but we could equivalently use the elimination rule instead.

Dependent pairs

The rules for Σ-types are mostly unsurprising, essentially a contextualized version of the rules for covariant context extension. The formation rule is analogous to Π, but the first component is well-formed covariantly, and the typing of the second component uses covariant context extension. The term rules are standard; the transformation rules say that a transformation between a pair is a pair of transformations. map is defined componentwise; in this case, the definition of transformation at $\Delta, x : A^+$ is just right so that δ, refl works as the second component. The $\beta\eta$-rules are standard, and the substitution rules are all defined compositionally. Identity and composition are defined componentwise.

Sets and elements

As our first example of a base type with non-trivial transformations, we consider a universe set that contains *discrete types*. That is, each term $S : \mathsf{set}$ will represent a type $El(S)$ whose elements have no non-identity transformations between them. However, set itself is *not* a discrete type: we take a transformation from S to S' to be a function from $El(S)$ to $El(S')$. Consequently, any type $s : \mathsf{set} \vdash C\ \mathsf{type}$ will admit a lifting of a function from $El(S)$ to $El(S')$ to a transformation from $C[S/s]$ to $C[S'/s]$. This is the common functor interface used in many programming languages. This universe of sets is *extensional*, in that transformation at sets satisfies equality reflection and definitional uniqueness of identity proofs—one can work with these sets as one would work in extensional type theory.

We populate the types $El(S)$ determined by a universe by giving inference rules for the members of these types. It is simpler to specify a set, than a type, because the transformations between elements are always only reflexivity. A set is specified by: (1) A formation rule for $S : \mathsf{set}$, with equations for (2) *1-substitution* $S[\theta]$ and *1-resp* $S[\delta]$. (3) Terms defining $M : El(S)$, as well as equations defining $\beta\eta$-rules, *1-substitution*, and *1-resp*. The equations for 1-resp for each S define the functorial action of the set former—which is used by map.

Sets and Elements. In Figure 7, we present the generic rules that apply to all sets: set is a type, as is $El(S)$ if S has type set. A transformation at set is a function from the elements of one to the elements of the other. The only transformation

between elements of sets is reflexivity, and such transformations are eliminated by equality reflection.

The first equation says that the 1-resp action of elements of sets is an equality, which is true because $El(S)$ is a discrete type: if $M : El(S)$ then M takes transformable arguments to *equal* results. The next two equations η-expand a transformation at set into a map, and a transformation at $El(S)$ into reflexivity. The rules for 0-subst are compositional.

map at set is a no-op, because set is a constant functor. map at $El(S)$ applies the function (open term) given by 1-resp of S, $S[\delta]$. We will give rules for each set-former defining $S[\delta]$. Reflexivity and transitivity are defined in the expected way for sets, and trivially for elements. The 2-resp rule for set says that a function $x.M$ respects a transformation δ by running M at the source and then applying δ. The 2-resp rule for $El(-)$ is analogous to *delegate*.

Basic Sets: Rules. In Figures 8 and 9, we present the rules for some basic sets: Π, Σ, $\mathsf{0}$ (the empty set), $\mathsf{1}$ (the unit set), $\mathsf{2}$ (booleans), and Id (identity between two elements of a set). In symmetric type theory, given Γ and $\Gamma \vdash A\ \mathsf{type}$ 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 $El(S)$ is discrete, and therefore trivially groupoidal. A more general alternative would be 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.

The rules for Π and Σ express that they are isomorphic to Π/Σ types of discrete elements. In fact, the domain of a Π need not be restricted to a set: $\Pi\, x{:}A.\, S$ is a set even if A is higher-dimensional. However, in our present theory types are both higher-dimensional than sets, and of higher *size* than sets: set itself is a type. So such a quantifier would not only be higher-dimensional, but impredicative. If additionally we had a universe typ of *small* types, then it would be appropriate to allow Π's to range over $A : \mathsf{typ}$.

$\mathsf{0}, \mathsf{1}, \mathsf{2}$ are introduced by the usual rules. The elimination rules for $\mathsf{0}$ and $\mathsf{2}$ have one subtlety: they allow elimination of both co- and contravariantly well-formed terms—we abbreviate a choice between Γ and Γ^{op} by $\Gamma^{\pm}$. The reason for this is that the natural deduction rules for positive types build in a cut, and the appropriate notion of cut includes both co- and contra-variant cut formulas (cf. the fact that substitutions allow for both co- and contravariant variables). These types can also be eliminated towards transformation judgements, so we add rules for eliminating $El(0)$ and $El(2)$ towards $M \Longrightarrow_A N$.

The rules for the identity type express an isomorphism with the corresponding

term transformations.

Basic Sets: Equalities. 1-subst into each set is as expected. 1-resp is the identity for constant sets, defined in terms of Π and Σ types for Π and Σ, and computationally trivial for identity (the verification that the right-hand-side is well-typed uses symmetry of equality). Next, in Figure 9, the $\beta\eta$ rules express the isomorphisms for Π and Σ and Id, and the usual equations for $\mathsf{0}$,$\mathsf{1}$,$\mathsf{2}$. 1-subst (and 2-resp for ide) are defined compositionally. The 1-resp rules are computationally trivial, but the fact that they are well-typed is interesting: e.g. the rule for in M requires showing that if two terms are transformable at $\Sigma\, x{:}El(S).\, El(S')$ then they are equal—which is true because the transformation provides equalities of each component. The 1-resp rule for if uses the transformation elimination form for booleans. The remaining rules state that the rules eliminating $0, 1, 2$ towards transformations satisfy similar $\beta\eta$ and *2-resp* laws.

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 FIXMEChapter 7]licata11thesis 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 $[\![A]\!] : [\![\Gamma]\!] \longrightarrow Cat$. 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 $\Gamma \longrightarrow Cat$ 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, $\int_\Gamma A$, with projection map $\mathsf{p} : \int_\Gamma A \longrightarrow \Gamma$. Thus, we can

define 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 FIXME, it is more convenient to use an equivalent explicit definition:

Definition 3.1 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 $\gamma \in Ob(\Gamma)$, $M_o(\gamma) \in Ob(A(\gamma))$
- 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 FIXME.

Theorem 3.3 *Soundness. There are total functions $[\![-]\!]$ that for each derivation $\mathcal{D} :: J$ 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. Γ^{op} is the action on objects; θ^{op} is the action on 1-cells; and δ^{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:

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

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 FIXME: 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 $M\ \mathsf{v} \Longrightarrow N\ \mathsf{v} : \Gamma.A^- \longrightarrow B$, where $\mathsf{v} : \mathsf{Tm}\ (\int_\Gamma A)\ (A[\mathsf{p}])$ is defined by second projection from $\int_\Gamma A$. The proof follows FIXME, 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 FIXME 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 $El(S)$ semantically by $\mathit{discrete} \circ S$. As usual, we overload notation and write $El(S)$ 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 $\mathsf{Tm}\ (\int_\Gamma (El(S)))\ (El(S'))$, 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 FIXME.

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 substitution 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 Ctx could 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; term-for-variable substitutions give substitution as well.

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:

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

As desired, this principle says that the renaming/substitution into the derivation proves the renaming/substitution 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 can obtain the structural properties for free even for a logic with such a rule, 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 set 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 FIXMEChapter 8]licata11thesis, 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 [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 [17,27,35,4,37,15,36] will be useful in generalizing 2DTT to additional models and higher-dimensions. An early connection between λ-calculus and 2-categories was made by FIXME, 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 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 [13,10,34]. 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 [30,31,32]; 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 equips 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 [33], 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 (FIXME 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.

Acknowledgement

We thank Steve Awodey, Peter Lumsdaine, Chris Kapulkin, Kristina Sojakova, and Thorsten Altenkirch for helpful conversations about this work.

References

[1] Abel, A., *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] Altenkirch, T. and P. Morris, *Indexed containers*, in: *IEEE Symposium on Logic in Computer Science* (2009), pp. 277–285.

[3] Altenkirch, T. and B. Reus, *Monadic presentations of lambda terms using generalized inductive types*, in: *CSL 1999: Computer Science Logic* (1999).

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

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

[6] Barthe, G., *Implicit coercions in type systems*, in: *International Workshop on Types for Proofs and Programs* (1996), pp. 1–15.

[7] Bellè, G., C. Jay and E. Moggi, *Functorial ML*, in: H. Kuchen and S. Doaitse Swierstra, editors, *Programming Languages: Implementations, Logics, and Programs*, Lecture Notes in Computer Science **1140**, Springer Berlin / Heidelberg, 1996 pp. 32–46.

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

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

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

[11] Coq Development Team, "The Coq Proof Assistant Reference Manual, version 8.2," INRIA (2009), available from http://coq.inria.fr/.

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

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

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

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

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

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

[18] Godement, R., "Théorie des faisceaux," Hermann, Paris, 1958.

[19] Hofmann, M., *Syntax and semantics of dependent types*, in: *Semantics and Logics of Computation* (1997), pp. 79–130.

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

[21] Hofmann, M. and T. Streicher, *The groupoid interpretation of type theory*, in: *Twenty-five years of constructive type theory* (1998).

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

[23] Laemmel, R. 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] Licata, D. R., "Dependently Typed Programming with Domain-Specific Logics," Ph.D. thesis, Carnegie Mellon University (2011), available from http://www.cs.cmu.edu/~drl/pubs/thesis/thesis.pdf.

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

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

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

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

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

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

[31] Pientka, B., *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*, 2008, pp. 371–382.

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

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

[34] Steffen, M., "Polarized Higher-Order Subtyping," Ph.D. thesis, Universitaet Erlangen-Nuernberg (1998).

[35] van den Berg, B. and R. Garner, *Types are weak ω-groupoids* (2010), available from http://www.dpmms.cam.ac.uk/~rhgg2/Typesom/Typesom.html.

[36] Voevodsky, V., *The equivalence axiom and univalent models of type theory* (2010), available from http://www.math.ias.edu/~vladimir/Site3/home_files/.

[37] Warren, M. A., "Homotopy theoretic aspects of constructive type theory," Ph.D. thesis, Carnegie Mellon University (2008).