Contributions to Multimode and Presheaf Type Theory - Motivation, Overview and Discussion

Andreas Nuyts

May 9, 2022

Abstract

This note describes and discusses my PhD dissertation 'Contributions to Multimode and Presheaf Type Theory'. The PhD project was motivated by the quest for a higher-dimensional directed type theory with interoperating functoriality-for-free and naturality-for-free (NatDTT). We discuss the various contributions – to wit: a study of the internalization of (natural transformations between / adjunctions of) morphisms of CwFs, parametric quantifiers (ParamDTT), degrees of relatedness (RelDTT), multimode type theory (MTT), the transpension type, the robustness criterion for (contextual) fibrancy and a study of internal fibrancy – as well as how they fit together and serve the higher purpose of NatDTT.

This text reuses parts from [Nuy20a, ch. 1 and 10]. That chapter 1 in turn reuses parts of a non-public grant renewal application at the Research Foundation - Flanders (FWO), authored by myself in collaboration with Dominique Devriese.

1 Motivation: Higher-dimensional Directed Type Theory

The goal we originally set out for this PhD project [Nuy20a, henceforth referred to as 'PhD'] was to establish higher-dimensional directed dependent type theories by formulating them, implementing them, proving their consistency and demonstrating their use. By a directed dependent type theory, we mean a dependent type theory which has not only a built-in notion of equality, but also of transformation. Such theories would provide a powerful framework for computer-assisted reasoning about asymmetric phenomena such as subtyping, syntactic substitution and various kinds of non-invertible transformations. In particular, we expect to obtain functoriality-for-free, relieving programmers from the burden of explicitly implementing the 'map' operation of functors and proving that it respects identity and composition. Moreover, we aim to generalize the notion of parametricity in programming languages to one of naturality, interacting smoothly with the aforementioned functoriality. This would first of all extend the concept of naturality beyond natural transformations but also relieve mathematicians from the burden to prove that their operations are natural by giving them a method that asserts naturality by construction. On top of that, the intention was for naturality theorems to be provable *within* dependent type theory.

Unsurprisingly for those who tried, this turned out to be a project for more than a single PhD. None of the original goals has been fully achieved, but I believe that with this dissertation and its associated papers and technical reports, my co-authors and I make several important contributions in mapping out the road and freeing it from some important obstructions, and each of these contributions has collateral benefits.

2 Context

Static type systems provide a way of guaranteeing safety and termination of computer programs, by preventing at compile-time that variables are assigned inappropriate values. They are also interesting from a logical perspective, as the Curry-Howard correspondence associates type operators to logical operators, so that propositions may be translated into types and vice versa [How80]. Proving a proposition then corresponds to constructing an element of the associated type.

Dependent type theories such as Martin-Löf type theory (MLTT) [ML82, ML98] include type theoretic counterparts for universal (∀) and existential (∃) quantification in logic. As such, all possible propositions can be translated into dependent types, and they can be proven formally by constructing a program of the corresponding type and having it type-checked by a computer. This crucial observation allows the use of dependently typed functional programming languages such as Coq [Coq14], Agda [Nor09] and Idris [Bra13], as proof assistants for proving either mathematical theorems or program correctness.

An important ingredient of logical reasoning is the concept of **equality**. This, too, has a type theoretic counterpart in MLTT: given values $a, b : A$, we have an 'identity type' or $a \equiv_A b$ of proofs that a and b are equal. The precise definition of the identity type is subtle, but we have three operators at hand to ensure that equality is an equivalence relation on the elements of A: a reflexivity constructor ensuring that everything is equal to itself, a proof-composition operator $(x \equiv_A y) \to (y \equiv_A z) \to (x \equiv_A z)$ ensuring transitivity, and an inversion operator ensuring symmetry. As such, (classical) MLTT equips every type with a built-in equivalence relation called '(propositional) equality'.

Homotopy Type Theory (HoTT [Uni13]) starts from the observation that two objects can be identified in multiple ways. For example, a boolean is essentially the same thing as a bit: it takes one out of two values. But if I want to encode a boolean as a bit, do I encode true as 1 or as 0? I have to pick one of two ways in which a boolean is the same as a bit. While MLTT is usually equipped with a 'uniqueness of identity proofs' axiom (UIP) [Uni13, §7.2], which states that all elements (proofs) of $a \equiv_A b$ are equal and hence that the only information encoded in an equality proof, is its existence; HoTT abandons this principle and interprets the type $a \equiv_A b$ as the type of all isomorphisms between a and b. This is formalized by Voevodsky's **univalence axiom** [KLV12, Uni13], which essentially states that equality of types simply means isomorphism. The reflexivity, transitivity and symmetry operations now serve as the identity, composition and inversion operations of a (higher) *groupoid*: a category in which all morphisms are invertible.

If MLTT is a type theory with good support for notions of equality and HoTT is one with good support for notions of isomorphism, then **directed type theories** [LH11, Nuy15, RS17, Nor19, WL20] are aimed at supporting various asymmetric phenomena, such as a subtyping relation on types [Abe08], syntactic substitutions in programming language formalization [LH11], transformations between mathematical structures such as vector spaces or monads, functorial behaviour etc.

The key idea is to abandon the symmetry of the equality relation, leading to a type $a \leq_A b$ of inequality proofs. We can do this in the spirit of classical MLTT, with a 'uniqueness of inequality proofs' axiom, and obtain a theory that automatically equips any type with a kind of order relation. Or we can do this in the spirit of HoTT, interpreting $a \leq_A b$ as the type of transformations from a to b. In that case, we are equipping types with a much richer structure of transformations; what is mathematically called a (higher) category. An important aspect to consider in directed type theories is variance: dependencies can be increasing/covariant, decreasing/contravariant, or can disrespect inequality altogether. Modalities are annotations on function types and can be used to keep track of the variance of functions.

3 Higher Directed Type Theory in Practice

Although no type theory for higher[1] categories exists yet, we will demonstrate with an example what we hope to achieve with such a system.

Example Problem In purely functional programming languages like Haskell, functions behave (almost) the way they do in mathematics: calling the same function with the same inputs multiple times, will yield the same output, and no side effects occur in the process.

If we do want to allow a function to cause side-effects, then we can give it a monadic return type [Mog89]. For example, if we want a function with output type A to be able to log messages of type W, then we give it output type Writer $W\,A := W \times A$. The type W should have the structure of a monoid, with an associative binary operation $* : W \times W \to W$ for concatenating messages, and a unit element $e : W$ that serves as the empty message. Then the functor Writer W is a monad, whose unit (a.k.a. return) function

[1]In the sense of (n, r)-categories. Type systems for $(\infty, 1)$-categories do exist [RS17, WL20].

$\eta : A \to \mathsf{Writer}\, W\, A : a \mapsto (e, a)$ creates programs making no use of the logging functionality in the sense that they return the empty message, and whose bind operation concatenates programs' messages.

If we want to *add* logging functionality to an existing monad M, we use the monad transformer $\mathsf{WriterT}\, W$, where $\mathsf{WriterT}\, W\, M\, A := M(W \times A)$. For example M could be the Maybe monad, where elements of $\mathsf{Maybe}\, A$ are either $\mathsf{nothing}$ or $\mathsf{just}\, a$ where $a : A$ (i.e. $\mathsf{Maybe}\, A \cong A \uplus \mathsf{Unit}$). A function of output type $\mathsf{Maybe}\, A$ is conceptually a function of output type A that has the option to fail. Then a function of output type $\mathsf{WriterT}\, W\, \mathsf{Maybe}\, A := \mathsf{Maybe}(W \times A)$ is a function that has the option to fail and, if it doesn't, will yield an output of type A and may log messages of type W.

A monad morphism is nothing but a natural transformation that respects the unit and bind operations. In Moggi's framework of monadic side effects, a monad morphism $m : M_0 \to M_1$ can be thought of as a compiler that compiles the effectful operations available in M_0 to effects in M_1.

Recall that the free monoid over S is given by $(\mathsf{List}\, S, [\,], {+}{+})$. Thus, an arbitrary function $s : S_0 \to S_1$ gives rise to a monoid morphism $\mathsf{List}\, s : (\mathsf{List}\, S_0, [\,], {+}{+}) \to (\mathsf{List}\, S_1, [\,], {+}{+})$, which in turn gives rise to a monad morphism $\mathsf{WriterT}\, (\mathsf{List}\, s)\, M : \mathsf{WriterT}\, (\mathsf{List}\, S_0)\, M \to \mathsf{WriterT}\, (\mathsf{List}\, S_1)\, M$ for any monad M. On the other hand, a monad morphism $m : M_0 \to M_1$ should give rise to a monad morphism $\mathsf{WriterT}\, W\, m : \mathsf{WriterT}\, W\, M_0 \to \mathsf{WriterT}\, W\, M_1$ for any monoid W.

The challenge is now to construct the aforementioned monad morphisms and to prove commutativity of the following diagram:

$$\begin{tikzcd}
\mathsf{WriterT}\,(\mathsf{List}\, S_0)\, M_0 \arrow[r, "\mathsf{WriterT}\,(\mathsf{List}\, S_0)\, m"] \arrow[d, "\mathsf{WriterT}\,(\mathsf{List}\, s)\, M_0"'] & \mathsf{WriterT}\,(\mathsf{List}\, S_0)\, M_1 \arrow[d, "\mathsf{WriterT}\,(\mathsf{List}\, s)\, M_1"] \\
\mathsf{WriterT}\,(\mathsf{List}\, S_1)\, M_0 \arrow[r, "\mathsf{WriterT}\,(\mathsf{List}\, S_1)\, m"'] & \mathsf{WriterT}\,(\mathsf{List}\, S_1)\, M_1
\end{tikzcd}$$

If $s = \mathsf{capitalize} : \mathsf{String} \to \mathsf{String}$ and $m = \mathsf{just} : \mathsf{Id} \to \mathsf{Maybe}$, then what this says is that it doesn't matter whether we first capitalize all logged messages and then decide to allow but not use the option to fail, or the other way around.

Existence of the functions underlying the monad morphisms may be needed already when defining a program. The fact that the functions are monad morphisms and that the diagram commutes, may be necessary to prove correctness of a program. However, since all of these results are completely obvious and dull from a categorical viewpoint, we want to spend minimal time on implementing and proving them, and in particular we prefer not to bother with list induction. This is why we want native support for category theory in our programming language.

In Plain Dependent Type Theory If we want to face the challenge in plain dependent type theory in a somewhat principled manner, we would do the following:

- Show that List is a functor from types to monoids:
  - For every $f : A \to B$, show that there is a monoid morphism $\mathsf{List}\, f : \mathsf{List}\, A \to \mathsf{List}\, B$:
    - Define $\mathsf{List}\, f$, by list recursion,
    - Show that it respects the empty list (trivial) and list concatenation (by list induction).
  - Show that this operation respects identity and composition,[2] by list induction.
- Show that $\mathsf{WriterT}$ is a functor from monoids to covariant monad transformers:
  - Show that $\mathsf{WriterT}\, W$ is a covariant monad transformer for every monoid W:
    - Show that $\mathsf{WriterT}\, W\, M$ is a monad for every monad M.
    - Show that, for any monad morphism $m : M_0 \to M_1$, we get a monad morphism $\mathsf{WriterT}\, W\, m : \mathsf{WriterT}\, W\, M_0 \to \mathsf{WriterT}\, W\, M_1$:
      - Define $\mathsf{WriterT}\, W\, m\, A : \mathsf{WriterT}\, W\, M_0\, A \to \mathsf{WriterT}\, W\, M_1\, A$ for any type A,

[2] This is not actually needed for the challenge at hand, but is a matter of not doing half work.

            · Show that it is natural in A,
            · Show that it respects the monad operations.
        * Show that this operation respects identity and composition.
    - For any monoid morphism $w : W_0 \to W_1$, show that there is a morphism of covariant monad transformers $\mathsf{WriterT}\, w : \mathsf{WriterT}\, W_0 \to \mathsf{WriterT}\, W_1$:
        * Define $\mathsf{WriterT}\, w\, M\, A : \mathsf{WriterT}\, W_0\, M\, A \to \mathsf{WriterT}\, W_1\, M\, A$ for any monad M and type A,
        * Show that it is natural in M,
        * Show that it is natural in A,
        * Show that it respects the monad transformer operation lift.[3]
    - Show that this operation respects identity and composition.

In Homotopy Type Theory (HoTT) Let us see how this simplifies in homotopy type theory (HoTT) [Uni13]. Of course, HoTT only has native support for isomorphisms, so we will assume that s and m are isomorphisms. We will also assume that Haskell types are sets (in the HoTT sense) and thus that kinds are 1-groupoids. We then have to do the following:

- Show that List is a groupoid functor from types to monoids.
    - For every $f : A \cong B$, the univalence axiom provides an equality proof (a.k.a. path) $\mathsf{ua}\, f : A \equiv B$. The function $\lambda X.(\mathsf{List}\, X, [], ++, \_)$ sending the type X to the free monoid[4] over X respects equality, so we get a proof of $(\mathsf{List}\, A, [], ++, \_) \equiv_{\mathsf{Monoid}} (\mathsf{List}\, B, [], ++, \_)$, which by the structure identity principle (SIP) [Uni13, §9.8] is the same as a monoid isomorphism.
    - This operation automatically respects identity and composition, because all HoTT functions respect identity and composition of paths.

  So it turns out that we can prove this without knowing the implementation of List and with little knowledge of the definition of a monoid (we merely need to know that it is a 'standard notion of structure'). We get this result essentially for free.

- Show that $\mathsf{WriterT}$ is a groupoid functor from monoids to groupoid-functorial monad transformers:
    - Show that $\mathsf{WriterT}\, W$ is a groupoid-functorial monad transformer for every monoid W:
        * Show that $\mathsf{WriterT}\, W\, M$ is a monad for every monad M.
        * Again, we get groupoid functoriality for free. Indeed, given a monad isomorphism $m : M_0 \cong M_1$, we get $M_0 \equiv_{\mathsf{Monad}} M_1$ by the SIP, whence a proof of $\mathsf{WriterT}\, W\, M_0 \equiv_{\mathsf{Monad}} \mathsf{WriterT}\, W\, M_1$, which by the SIP is the same as an isomorphism between the writer monads.
    - By similar reasoning, groupoid functoriality of $\mathsf{WriterT}$ is also for free.

In Higher Directed Type Theory with Naturality (NatDTT) In the previous subsection, we saw that we could greatly shorten our todo list by moving to HoTT, and moreover that we are rid of all list inductions. The price we paid and which we seek to unpay by moving to higher DDTT, is that we had to assume that m and s are isomorphisms. In higher DDTT, we expect that our todo list will look like this:

- Let the type-checker check that List is covariant (i.e. can be annotated with the covariance modality). In fact, let it check that the function $\lambda X.(\mathsf{List}\, X, [], ++, \_)$ sending the type X to the free monoid over X, is covariant. This requires that monoids depend on their structure by a special modality: a directed analogue of the structural modality which we introduced in Degrees of Relatedness [ND18a][PhD, ch. 9].

[3]https://hackage.haskell.org/package/transformers-0.5.6.2/docs/Control-Monad-Trans-Class.html
[4]The underscore stands for the proofs of the monad laws.

- Show that List is a functor from types to monoids.
  - For every $f : A \to B$, the directed univalence axiom [WL20] provides an inequality proof (a.k.a. morphism or directed path) $\mathsf{dua}\, f : A \leq B$. By covariance, we get a proof of $(\mathsf{List}\, A, [], {+\!+}, \_) \leq_{\mathsf{Monoid}} (\mathsf{List}\, B, [], {+\!+}, \_)$, which by an expected directed SIP is the same as a monoid morphism.
  - This operation automatically respects identity and composition, because all covariant functions respect identity and composition of morphisms.

  Again, we get this result essentially for free.

- Show that $\mathsf{WriterT}$ is a functor from monoids to covariant monad transformers:
  - Show that $\mathsf{WriterT}\, W$ is a covariant monad transformer for every monoid W:
    * Show that $\mathsf{WriterT}\, W\, M$ is a monad for every monad M.
    * Let the type-checker check that $\mathsf{WriterT}\, W\, M$ satisfies the covariance modality w.r.t. M.
    * Again, we get the covariant action and laws for free. Indeed, given a monad morphism $m : M_0 \to M_1$, we get $M_0 \leq_{\mathsf{Monad}} M_1$ by the directed SIP, whence a proof of $\mathsf{WriterT}\, W\, M_0 \leq_{\mathsf{Monad}} \mathsf{WriterT}\, W\, M_1$, which by the directed SIP is the same as a morphism between the writer monads.
  - By similar reasoning, functoriality of $\mathsf{WriterT}$ is also for free.

Thus, we expect that higher DDTT can drastically simplify proofs of boring properties where HoTT can already do so for isomorphisms. This generalization is necessary because most transformations are not invertible. It is also complex, because while in HoTT all functions respect equality/isomorphism, in higher DDTT it is not reasonable to expect that all functions respect inequality/morphisms. Therefore, we need to keep track of the behaviour of functions in order to assert covariance, contravariance, naturality etc. of functions *by construction*, in a way that can (hopefully) be verified by a type-checker. This thesis is not concerned with the variance *checking* aspect, but with paving the road towards the design of a sound system in the first place.

4 Contributions

Note: a diagram of the contributions in my dissertation, together with some important prior work and some intended future work is given on page 10.

4.1 ParamDTT: Parametric Quantifiers

In my master thesis [Nuy15], I studied higher DDTT from a purely type theoretic point of view, trying to set up a system of typing rules that appeal to category-theoretic intuition and do not obviously introduce contradictions. At the start of my PhD, I wanted to underpin the work of my master thesis with a denotational semantics. The most natural setting to formulate these semantics seemed to be (higher) category theory, but attempts to model the style of directed type theory from my master thesis in this setting kept failing.[5] For the category theorist, the problem can be succinctly described by saying that the functor category functor $\mathsf{Cat}^{\mathsf{op}} \times \mathsf{Cat} \to \mathsf{Cat} : (\mathcal{C}, \mathcal{D}) \mapsto \mathcal{D}^{\mathcal{C}}$ does not preserve composition of profunctors[6], failing the interpretation of the function type. For the type theorist, this problem is the reason that Reynolds' relationally parametric interpretation of System F [Rey83] features an identity extension lemma but no composition extension lemma as it would be violated by the function type, and that later models of parametricity [AM13, AGJ14] are formulated in reflexive graphs, which are exactly categories *without composition.*

[5]Licata and Harper [LH11] do provide a model in category theory, but this work has a coupling of variance of types and terms (covariant terms live in covariant types) that we seek to relax.

[6]It does laxly, but this can be broken by exponentiating again [PhD, ex. 8.1.27].

The second most natural setting to work in, are **presheaf categories** – such as the category of reflexive graphs mentioned above – which are automatically models of dependent type theory [Hof97] but are also used as models of higher category theory (e.g. via the notion of quasi-categories [nLa20b]) and homotopy type theory [CCHM17, Hub16, BCH14, KLV12].

The idea arose in discussion with Andrea Vezzosi and Andreas Abel to model the undirected part of the intended NatDTT, i.e. a type system with a naturality modality, but no modalities for functoriality. Naturality is then more typically called parametricity. This simplification of the ideas in my master thesis led to a dependently typed system ParamDTT in which function types could be annotated as parametric or non-parametric, i.e. a system featuring a parametric $\forall$ alongside the non-parametric Π [NVD17][PhD, §9.2]. The idea to use a modality to keep track of parametricity, turned out to be an answer to an open question in the literature. Indeed, parametricity results about simpler type systems such as System F (the polymorphic λ-calculus) and System Fω had not been properly carried over to dependent type theory where large types are involved.

The original paper on ParamDTT [NVD17] is not subsumed in this thesis, but in [PhD, §9.2] we give a high-level discussion of the system and its model, and explain how the system can be constructed more cleanly and efficiently with the tools that are available today.

The parametricity modality is modelled as a CwF morphism [Dyb96], which prompted a study of the **internalization of CwF morphisms**, as well as natural transformations between and adjunctions of CwF morphisms, into type theory [Nuy17, ch. 2][PhD, §5.1]. The results on adjunctions overlap with independent work by Birkedal et al. on dependent right adjoints [BCM+20], as discussed in [PhD, §5.2].

Following Bernardy, Coquand and Moulin [BCM15, Mou16] (henceforth: BCM), ParamDTT was modelled in cubical sets, whose edges however we annotated as expressing either equality (paths) or relatedness (bridges) [Nuy17]. Following Atkey, Ghani and Johann [AGJ14], Reynolds' identity extension lemma [Rey83] was modelled by restricting to *discrete* types.

We also wanted free parametricity theorems to be provable internally in ParamDTT. We could not rely on the **internal parametricity operators** by BCM [BCM15, Mou16], because these require an *affine* cubical model whereas discreteness of bridge and path types requires a *cartesian* cubical model. Instead, we used **the** Glue **type** from cubical HoTT [CCHM17] (stripped of its Kan fibrancy requirements) and introduced **a dual type** Weld. We showed that these types can be modelled in arbitrary presheaf models [Nuy17] and discuss them in [PhD, ch. 6] on presheaf type theory. We refer to the paper [NVD17] for examples on how to apply these operators.

4.2 RelDTT: Degrees of Relatedness

In follow-up work [ND18a, Nuy18a], we abandon the idea that types and kinds should be the same thing, inspired via directed type theory by the fact that in category theory, the collection of n-categories is of course an n-category but, much more interestingly, is an $(n+1)$-category. This solves technical inconveniences in ParamDTT, such as the fact that small types contain unnecessary relational structure, whereas universes seemed to lack structure.

Following [LS16], we move from a modal type theory where function types are annotated by a modality (e.g. parametric or not) to a multimode type theory. In a multimode type theory, every judgement is annotated by a mode, which is of course just a syntactic feature but conceptually tells you in what category the judgement should be interpreted. Modalities then have a domain and a codomain and are modelled by CwF morphisms between the corresponding categories.

We exhibit parametricity as just one out of many interesting and less interesting modalities, including ad hoc polymorphism, irrelevance (at type-checking type) [Pfe01, MS08, BB08, AS12], shape-irrelevance [AVW17], as well as a novel *structural* modality which explains how algebras (living in a kind) depend on their structure (a lower-level object living in a type).

The original paper on this type system RelDTT [ND18a] is not subsumed in this thesis, but again a high-level discussion that also relates it to today's state of the art is given in a dedicated chapter [PhD, ch. 9]. Other aspects of the system are handled the same way as for ParamDTT, see the previous section.

Theoretical importance My research indicates that, if we care for dependently typed parametricity with identity extension even for large types, then we should be looking towards modalities or at least a stratification of types based on their relational complexity (which may or may not be decoupled from the level). I would argue that RelDTT is so general that such modal or stratified systems should be almost always explicable as a subsystem of RelDTT, e.g. the following theorem is proven by translation to RelDTT [PhD, thm. 9.5.1]:

Theorem 4.1. There exists a non-trivial model of DTT with Agda-style cumulativity, in which any function $f : \mathsf{U}_\ell \to A$ where $A : \mathsf{U}_\ell$, is constant.

Practical importance A type system like RelDTT [ND18a], with infinitely many modes and modalities, is at first sight likely unappetizing to the practical programmer, even when they are familiar with DTT. However, a few modes RelDTT already appear in a language like Haskell, with programs living at mode 0, types at mode 1 and kinds at mode 2. Adding a mode −1 for proofs does not seem outrageous. As shown in the original paper [ND18a, fig. 2], *all* modalities up to mode 1 can be expressed in terms of modalities that were known prior to RelDTT and/or structurality, the novel modality by which algebras depend on their structure.

Haskell unfortunately has completely different languages at mode 0 and 1. A general theory of relational modalities may advise more consistent language development in the future. And while full RelDTT provides ω modes, most non-logicians will only need the lowest few which are in fact relatively familiar, and need not be hampered by the existence of others.

An understandable concern is that programming becomes 'very complicated' if we have to constantly think about which modality we should annotate our functions with. I suggest to look at this from a different angle. *Supposing* one needs to rely on a 'free' parametricity theorem, would programmers rather go back to all code they have been relying on and recursively prove that it satisfies its parametricity predicate, or would they rather put a few modalities here and there to point out to the type-checker that, by non-violation, their program is parametric?

4.3 Transpension: The Right Adjoint to the Π-type

The observation by Dominique Devriese that it seemed impossible to prove parametricity of System F in ParamDTT, sparked an investigation of the comparative expressivity of internal parametricity operators [ND18b]. The crux, it turned out, is that the operators by BCM [BCM15, Mou16] do something that Glue and Weld do not: promote cells of a cubical set to a higher dimension. Of course abstraction over a dimension does the opposite, e.g. a square in $\Pi(i : \mathbb{I}).A$ where i ranges over the interval (i.e. a line), is a cube in A. In [PhD, ch. 7][ND20] and its associated technical report [Nuy20b], we introduce the *transpension type former* $\between i.A$ which is right adjoint to the function type. We explain how this operation, together with the strictness axiom [OP18] and a pushout type former [PhD, ch. 6] can be used to reconstruct all existing internal presheaf operators that we are aware of.

The semantics of the transpension type and its associated operators are parametrized by an almost arbitrary functor which we call a multiplier and which interprets context extension with a *shape variable* $u : \mathbb{U}$. We introduce a series of criteria (including 'affine' and 'cartesian') for classifying multipliers and deduce internal properties depending on those criteria.

4.4 Robust Notions of Fibrancy

Many type systems modelled in presheaf categories interpret the type judgement not in the standard way, but have to restrict to a subset of all presheaf types. As mentioned, in order to validate Reynolds' identity extension lemma, in models of parametricity we need to restrict to *discrete* types [AGJ14, NVD17, ND18a, CH19]. In models of HoTT [KLV12, CCHM17], one restricts to *Kan fibrant* types, which are types equipped with appropriate composition operations for (higher) paths. In presheaf models of directed type theory [RS17, WL20], one is interested in *Segal fibrant* types (with composition operations for (higher) morphisms), covariant types (essentially Haskell's functors), and other notions.[7] In models of guarded type theory [BM18], one restricts to *clock-irrelevant* types. All of these conditions are notions of *fibrancy*, which means that they arise from a *factorization system* on the presheaf CwF.

[7]Restricting to those types is in general not feasible, as they are not closed under important type formers.

Because the most obvious interpretation for the parametric quantifier ∃ in ParamDTT does not automatically preserve discreteness, we instead have to use its 'discrete replacement' to actively force it to be discrete. Hence, we want this discrete replacement operation to be stable under substitution. The technical report on ParamDTT [Nuy17] contains an unwieldy ad hoc proof that this is the case. The prospect of developing RelDTT and NatDTT asked for a more principled approach, so I developed the *robustness* criterion [Nuy18b][PhD, §8.4]. Robust notions of fibrancy, such as discreteness [PhD, ex. 8.4.7], automatically come with a fibrant replacement monad which is stable under substitution and can be axiomatized internally. Fibrant types can then be defined internally as algebras of this monad.

Moreover, robust fibrancy has the property that the function type is fibrant as soon as its codomain is. Although Segal fibrancy [PhD, ex. 8.1.8] is not robust, we can follow Boulier and Tabareau's approach for Kan fibrancy [BT17] in moving to contextual fibrancy, in which case we *can* satisfy the robustness criterion and thus model the directed function type under restricted circumstances [PhD, prop. 8.6.2] for *some* notion of Segal fibrancy.

4.5 MTT: Well-Behaved Multimode Type Theory

Following Pfenning [Pfe01] and Abel [Abe06, Abe08], we had formulated ParamDTT and RelDTT with a left division operation on contexts: whenever the type-checker moves into a modal subterm, its left Galois connection (left adjoint) is applied to all modality annotations in the context.

While trying to implement a proof-assistant for these type systems [Nuy19, ND19], I noticed that computation of the left division could be postponed until usage of a variable subject to the division, and even then could be brought to the right again, so that the division in fact *never* needs to be computed. As such, I was able to turn the division from a context operation into a context constructor, inadvertently creating a hybrid with the Fitch-style approach of modal type theory [BGM17, BCM$^+$20, GSB19].

This hybrid multimode type system subsequently underwent the scrutiny of my co-authors Daniel Gratzer, G. A. Kavvos and Lars Birkedal – who praised it for having a cleaner substitution calculus than other modal type systems, be they Fitch-style or based on left division – which resulted in a paper [GKNB20b][PhD, §5.3], a journal paper [GKNB21] and an extensive technical report [GKNB20a].

Applications MTT is parametrized by a mode theory, specifying the available modes and modalities and their laws, which can be instantiated at will. Applications of modal type theory in general include: modal logic (eponymously) [PD01], variance of functors [Abe06, Abe08, LH11], intensionality vs. extensionality [Pfe01], irrelevance [Pfe01, Miq01, BB08, MS08, Ree03, AS12, AVW17, ND18a], shape-irrelevance [AVW17, ND18a], parametricity [NVD17], axiomatic cohesion [LS16], globality [LOPS18], guarded type theory [Nak00, CBGB16, BGC$^+$16, VvdW19, Gua18] and the metatheory of programming languages [Ste22, BKS21].

Sikkel and general modular presheaf type theory Different extensions of type theory have been developed for different purposes. This is not a good situation from a programmer's perspective: when faced with a certain problem that is solved by a language extension, the entire program needs to be moved to a different flavour of type theory. When faced with a second such problem, the corresponding extension may not be (known to be) compatible with the previous one.

We would instead prefer a situation where a language extension can be used only in the program module that requires it. Sikkel [CND22] is a library for Agda, based on (currently simply-typed) MTT, that achieves exactly that. Users can write MTT programs in a deeply embedded implementation of MTT, which are interpreted in a presheaf model built in Agda. If the final result lives at the trivial mode, interpreted in a set model, then we can extract an ordinary Agda program.

5 Towards Natural Dependent Type Theory (NatDTT)

A higher directed type system providing functoriality and naturality for free as sketched in sections 1 and 3 has several requirements, which are also found in the diagram on page 10:

Variance Some functions will be increasing/covariant, some decreasing/contravariant, some neither. In order to keep track of this, we need a modal type system such as MTT.

Non-self-classification In RelDTT [ND18a], we have abandoned the idea that types and their kinds should be the same thing, and instead used a multimode system [LS16] to embrace the diversity. This still applies – and perhaps in a more familiar way – to a directed system. There, we may still be concerned with types that are essentially sets, such as Bool or $\mathbb{N}$, and it will be pointless to consider the variance of functions to or from such types. On the other hand, a universe of such sets is of course a category, or indeed a pro-arrow equipped category [nLa20a]. A universe of (pro-arrow equipped) categories, is then a 2-category, or indeed a 2-dimensional generalization of a pro-arrow equipped category. On a 2-category C, we can reverse 1-arrows (yielding C^{op}) or 2-arrows (yielding C^{co}), so that we clearly need more modalities for variance of functors between 2-categories than between 1-categories. A multimode type system instantiating MTT is an excellent answer to this phenomenon.

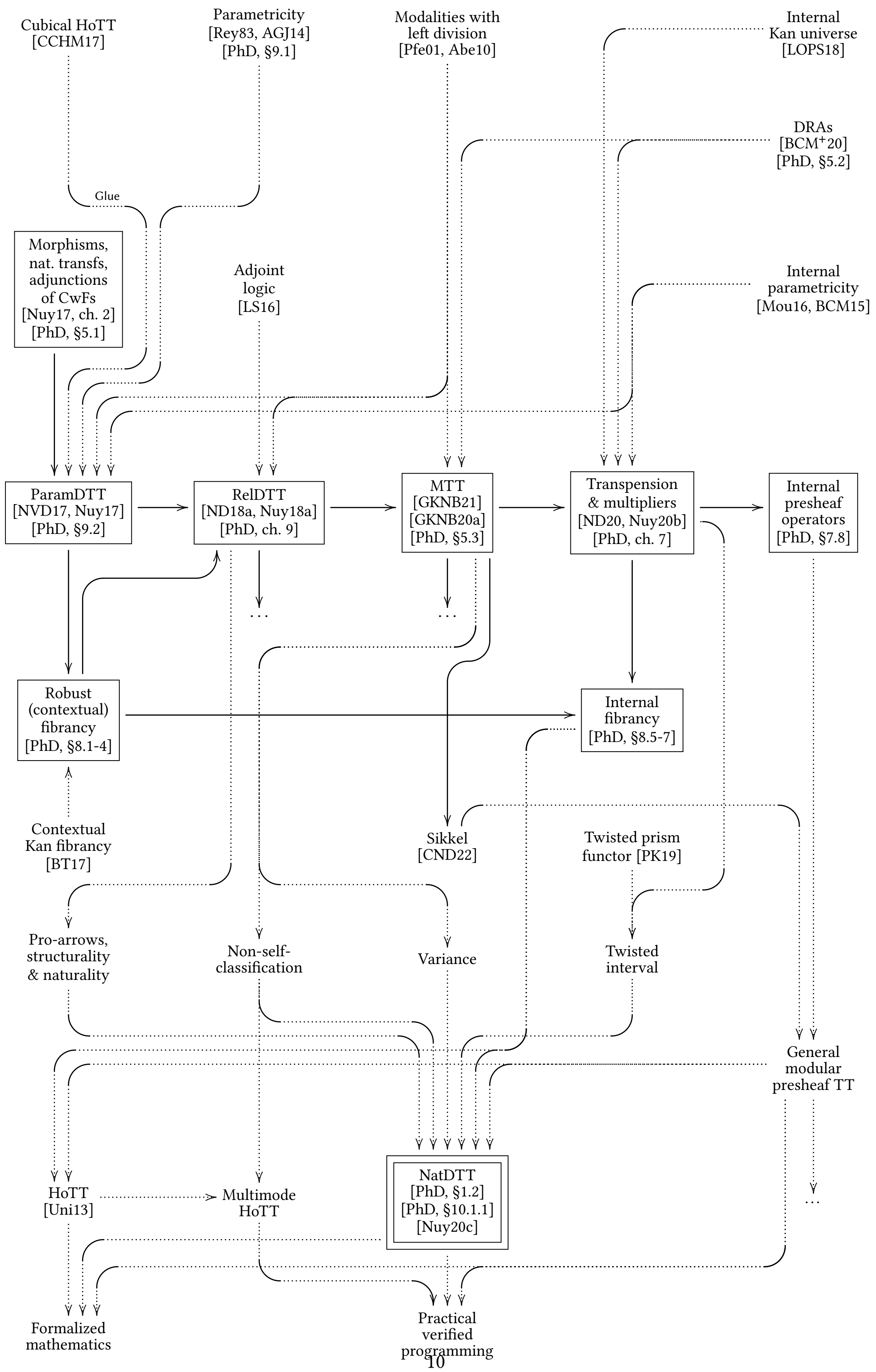
Multimode HoTT This idea already pays off in undirected HoTT. Types in HoTT are usually viewed as ∞-groupoids, which can be thought of as topological spaces. This is great for topologists seeking to formalize their results in a proof assistant, but of little use for practical programmers, who are typically only faced with data types (sets or 0-groupoids), kinds (1-groupoids), propositions (-1-groupoids) and very occasionally a universe of kinds (2-groupoid). These people will find little joy in having to *prove* the groupoid dimension (h-level) of their types. A multimode system with a mode for every h-level would be more suited when applying HoTT to practical programming.

Pro-arrows, structurality & naturality One of our main desiderata was to be able to reason about naturality, for which we would like to have a modality. ParamDTT [NVD17] features bridges (relations) between types, across which we can consider heterogeneous paths (proofs of relatedness). Parametric dependencies were identified to be those that promote bridges to paths. In RelDTT, we additionally discovered the novel *structural* modality by which algebras depend on their structure and which is right adjoint to parametricity. Directifying all of this, bridges (relations) should turn into pro-arrows (profunctors), across which we can consider heterogeneous morphisms. Natural dependencies will be those that promote pro-arrows to morphisms and the structural modality will be paramount to obtaining a sensible directed structure identity principle (SIP) [Uni13, §9.8].

Twisted interval In cubical homotopy type theory [CCHM17, for example], propositional equality $a \equiv_A b$ is proven by giving a function $f : \mathbb{I} \to A$ from the *interval* $\mathbb{I}$, a special type which has essentially two elements 0 and 1 that are considered equal, such that the 'endpoints' $f\,0$ and $f\,1$ are *definitionally* (computably) equal to a and b respectively. In directed type theory, this is difficult, because the Hom-type is contravariant in the source and covariant in the target, so that application to 0 of functions over the interval would have to have different variance than application to 1. Strikingly, Pinyo and Kraus's twisted prism functor [PK19][PhD, ex. 7.4.11] gives us exactly that: it comes with natural transformations $(\mathrm{id}_W, 0) : W^{\mathrm{op}} \to W \ltimes \mathbb{I}$ and $(\mathrm{id}_W, 1) : W \to W \ltimes \mathbb{I}$. The techniques related to the transpension type [ND20][PhD, ch. 7] will allow us to some extent to treat the twisted prism functor internally as though it were a (special) type. Weinberger [Wei22, §7.2.3] makes a similar observation.

Fibrancy See section 4.4.

Internal presheaf operators Generalizing internal parametricity operators, we would like internal operators that allow us to inhabit naturality squares simply from the knowledge that a function type-checks as natural. From [ND20][PhD, ch. 7], we know that the transpension type and the strictness axiom together give us all the wealth of currently known presheaf operators, which is encouraging.



Cubical HoTT [CCHM17]
Parametricity [Rey83, AGJ14] [PhD, §9.1]
Modalities with left division [Pfe01, Abe10]
Internal Kan universe [LOPS18]
DRAs [BCM$^+$20] [PhD, §5.2]
Glue
Morphisms, nat. transfs, adjunctions of CwFs [Nuy17, ch. 2] [PhD, §5.1]
Adjoint logic [LS16]
Internal parametricity [Mou16, BCM15]
ParamDTT [NVD17, Nuy17] [PhD, §9.2]
RelDTT [ND18a, Nuy18a] [PhD, ch. 9]
MTT [GKNB21] [GKNB20a] [PhD, §5.3]
Transpension & multipliers [ND20, Nuy20b] [PhD, ch. 7]
Internal presheaf operators [PhD, §7.8]
...
...
Robust (contextual) fibrancy [PhD, §8.1-4]
Internal fibrancy [PhD, §8.5-7]
Contextual Kan fibrancy [BT17]
Sikkel [CND22]
Twisted prism functor [PK19]
Pro-arrows, structurality & naturality
Non-self-classification
Variance
Twisted interval
General modular presheaf TT
HoTT [Uni13]
Multimode HoTT
NatDTT [PhD, §1.2] [PhD, §10.1.1] [Nuy20c]
...
Formalized mathematics
Practical verified programming

References

[Abe06] Andreas Abel. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006. URL: `http://www.cse.chalmers.se/abela/diss.pdf`.

[Abe08] Andreas Abel. Polarised subtyping for sized types. *Mathematical Structures in Computer Science*, 18(5):797–822, 2008. `doi:10.1017/S0960129508006853`.

[Abe10] Andreas Abel. Miniagda: Integrating sized and dependent types. In *PAR 2010*, 2010.

[AGJ14] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *Principles of Programming Languages*, 2014. `doi:10.1145/2535838.2535852`.

[AM13] Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 197–208. ACM, 2013. `doi:10.1145/2500365.2500597`.

[AS12] Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1):1–36, 2012. TYPES'10 special issue. `doi:http://dx.doi.org/10.2168/LMCS-8(1:29)2012`.

[AVW17] Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. Normalization by evaluation for sized dependent types. *Proc. ACM Program. Lang.*, 1(ICFP):33:1–33:30, August 2017. URL: `http://doi.acm.org/10.1145/3110277`, `doi:10.1145/3110277`.

[BB08] Bruno Barras and Bruno Bernardo. *The Implicit Calculus of Constructions as a Programming Language with Dependent Types*, pages 365–379. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. `doi:10.1007/978-3-540-78499-926`.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, Dagstuhl, Germany, 2014. URL: `http://drops.dagstuhl.de/opus/volltexte/2014/4628`, `doi:10.4230/LIPIcs.TYPES.2013.107`.

[BCM15] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. *Electron. Notes in Theor. Comput. Sci.*, 319:67 – 82, 2015. `doi:http://dx.doi.org/10.1016/j.entcs.2015.12.006`.

[BCM$^+$20] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. `doi:10.1017/S0960129519000197`.

[BGC$^+$16] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal. Guarded dependent type theory with coinductive types. In *FOSSACS '16*, 2016. `doi:10.1007/978-3-662-49630-5“2`.

[BGM17] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The clocks are ticking: No more delays! In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. `doi:10.1109/LICS.2017.8005097`.

[BJP12] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012. URL: `https://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=135303`, `doi:10.1017/S0956796812000056`.

[BKS21] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. Induction principles for type theories, internally to presheaf categories. *CoRR*, abs/2102.11649, 2021. URL: https://arxiv.org/abs/2102.11649, arXiv:2102.11649.

[BM18] Aleš Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. *CoRR*, abs/1802.03744, 2018. URL: http://arxiv.org/abs/1802.03744, arXiv:1802.03744.

[Bra13] Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23:552–593, 2013.

[BT17] Simon Boulier and Nicolas Tabareau. Model structure on the universe in a two level type theory. Working paper or preprint, 2017. URL: https://hal.archives-ouvertes.fr/hal-01579822.

[CBGB16] Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. The guarded lambda-calculus: Programming and reasoning with guarded recursion for coinductive types. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:7)2016.

[CCHM17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. *FLAP*, 4(10):3127–3170, 2017. URL: http://www.cse.chalmers.se/simonhu/papers/cubicaltt.pdf.

[CH19] Evan Cavallo and Robert Harper. Parametric cubical type theory. *CoRR*, abs/1901.00489, 2019. arXiv:1901.00489.

[CND22] Joris Ceulemans, Andreas Nuyts, and Dominique Devriese. Sikkel: Multimode simple type theory as an agda library. In *MSFP*, 2022. URL: http://eptcs.web.cse.unsw.edu.au/paper.cgi?MSFP2022.5.pdf.

[Coq14] The Coq Development Team. *The Coq proof assistant: reference manual*, 2014. v8.4, https://coq.inria.fr/refman/.

[Dyb96] Peter Dybjer. *Internal type theory*, pages 120–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61780-966.

[GKNB20a] Daniel Gratzer, Alex Kavvos, Andreas Nuyts, and Lars Birkedal. Type theory à la mode. Preprint, 2020. URL: https://anuyts.github.io/files/mtt-techreport.pdf.

[GKNB20b] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. doi:10.1145/3373718.3394736.

[GKNB21] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. URL: https://lmcs.episciences.org/7713, doi:10.46298/lmcs-17(3:11)2021.

[GSB19] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. Implementing a modal dependent type theory. *Proc. ACM Program. Lang.*, pages 107:1–107:29, 2019. doi:10.1145/3341711.

[Gua18] Adrien Guatto. A generalized modality for recursion. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18. ACM, 2018. doi:10.1145/3209108.3209148.

[Hof97] Martin Hofmann. *Syntax and Semantics of Dependent Types*, chapter 4, pages 79–130. Cambridge University Press, 1997.

[How80] William A. Howard. *The formulae-as-types notion of construction*, pages 479–490. Academic press, 1980. Original manuscript from 1969.

[Hub16] Simon Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, Sweden, 2016. URL: `http://www.cse.chalmers.se/simonhu/misc/thesis.pdf`.

[KL12] Chantal Keller and Marc Lasson. Parametricity in an impredicative sort. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 381–395, 2012. `doi:10.4230/LIPIcs.CSL.2012.381`.

[KLA$^+$] Chantal Keller, Marc Lasson, Abhishek Anand, Pierre Roux, Emilio Jesús Gallego Arias, Cyril Cohen, and Matthieu Sozeau. Paramcoq. 2012-2018. URL: `https://github.com/coq-community/paramcoq`.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. 2012. Preprint, `http://arxiv.org/abs/1211.2851`.

[LH11] Daniel R. Licata and Robert Harper. 2-dimensional directed type theory. *Electr. Notes Theor. Comput. Sci.*, 276:263–289, 2011. `doi:10.1016/j.entcs.2011.09.026`.

[LOPS18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, pages 22:1–22:17, 2018. `doi:10.4230/LIPIcs.FSCD.2018.22`.

[LS16] Daniel R. Licata and Michael Shulman. *Adjoint Logic with a 2-Category of Modes*, pages 219–235. Springer International Publishing, 2016. `doi:10.1007/978-3-319-27683-016`.

[Miq01] Alexandre Miquel. The implicit calculus of constructions. In *TLCA*, pages 344–359, 2001. `doi:10.1007/3-540-45413-627`.

[ML82] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175, 1982.

[ML98] Per Martin-Löf. An intuitionistic theory of types. In *Twenty-five years of constructive type theory*, pages 127–172. Oxford University Press, 1998.

[Mog89] Eugenio Moggi. Computational lambda-calculus and monads. In *4th annual symposium on logic in computer science*, pages 14–23. IEEE Press, 1989.

[Mou16] Guilhem Moulin. *Internalizing Parametricity*. PhD thesis, Chalmers University of Technology, Sweden, 2016. URL: `publications.lib.chalmers.se/records/fulltext/235758/235758.pdf`.

[MS08] Nathan Mishra-Linger and Tim Sheard. *Erasure and Polymorphism in Pure Type Systems*, pages 350–364. 2008. `doi:10.1007/978-3-540-78499-925`.

[Nak00] Hiroshi Nakano. A modality for recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 255–266. IEEE Computer Society, 2000. `doi:10.1109/LICS.2000.855774`.

[ND18a] Andreas Nuyts and Dominique Devriese. Degrees of relatedness: A unified framework for parametricity, irrelevance, ad hoc polymorphism, intersections, unions and algebra in dependent type theory. In *Logic in Computer Science (LICS) 2018, Oxford, UK, July 09-12, 2018*, pages 779–788, 2018. doi:10.1145/3209108.3209119.

[ND18b] Andreas Nuyts and Dominique Devriese. Internalizing Presheaf Semantics: Charting the Design Space. In *Workshop on Homotopy Type Theory / Univalent Foundations*, 2018. URL: https://hott-uf.github.io/2018/abstracts/HoTTUF18paper1.pdf.

[ND19] Andreas Nuyts and Dominique Devriese. Menkar: Towards a multimode presheaf proof assistant. In *TYPES*, 2019.

[ND20] Andreas Nuyts and Dominique Devriese. Transpension: The right adjoint to the pi-type, 2020. arXiv:2008.08533.

[nLa20a] nLab authors. 2-category equipped with proarrows, April 2020. Revision 32. URL: http://ncatlab.org/nlab/show/2-category%20equipped%20with%20proarrows.

[nLa20b] nLab authors. quasi-category, April 2020. Revision 69. URL: http://ncatlab.org/nlab/show/quasi-category.

[Nor09] Ulf Norell. Dependently typed programming in Agda. In *Advanced Functional Programming*, pages 230–266. Springer, 2009.

[Nor19] Paige Randall North. Towards a directed homotopy type theory. *Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2019, London, UK, June 4-7, 2019*, pages 223–239, 2019. doi:10.1016/j.entcs.2019.09.012.

[Nuy15] Andreas Nuyts. Towards a directed homotopy type theory based on 4 kinds of variance. Master's thesis, KU Leuven, Belgium, 2015. URL: https://anuyts.github.io/files/mathesis.pdf.

[Nuy17] Andreas Nuyts. A model of parametric dependent type theory in bridge/path cubical sets. Technical report, KU Leuven, Belgium, 2017. Subsumed in [Nuy18a]. URL: https://arxiv.org/abs/1706.04383.

[Nuy18a] Andreas Nuyts. Presheaf models of relational modalities in dependent type theory. *CoRR*, abs/1805.08684, 2018. arXiv:1805.08684.

[Nuy18b] Andreas Nuyts. Robust notions of contextual fibrancy. In *Workshop on Homotopy Type Theory / Univalent Foundations*, 2018. URL: https://hott-uf.github.io/2018/abstracts/HoTTUF18paper2.pdf.

[Nuy19] Andreas Nuyts. Menkar. https://github.com/anuyts/menkar, 2019.

[Nuy20a] Andreas Nuyts. *Contributions to Multimode and Presheaf Type Theory*. PhD thesis, KU Leuven, Belgium, 8 2020. URL: https://anuyts.github.io/files/phd.pdf.

[Nuy20b] Andreas Nuyts. The transpension type: Technical report, 2020. arXiv:2008.08530.

[Nuy20c] Andreas Nuyts. A vision for natural type theory. Unpublished note, 2020. URL: https://anuyts.github.io/files/nattt-vision.pdf.

[NVD17] Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. Parametric quantifiers for dependent type theory. *PACMPL*, 1(ICFP):32:1–32:29, 2017. URL: http://doi.acm.org/10.1145/3110276, doi:10.1145/3110276.

[OP18] Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. *Logical Methods in Computer Science*, 14(4), 2018. doi:10.23638/LMCS-14(4:23)2018.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001. doi:10.1017/S0960129501003322.

[Pfe01] Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS '01*, pages 221–230, 2001. doi:10.1109/LICS.2001.932499.

[PK19] Gun Pinyo and Nicolai Kraus. From cubes to twisted cubes via graph morphisms in type theory. *CoRR*, abs/1902.10820, 2019. URL: http://arxiv.org/abs/1902.10820, arXiv:1902.10820.

[PMD15] Andrew M. Pitts, Justus Matthiesen, and Jasper Derikx. A dependent type theory with abstractable names. *Electronic Notes in Theoretical Computer Science*, 312:19 – 50, 2015. Ninth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2014). URL: http://www.sciencedirect.com/science/article/pii/S1571066115000079, doi:https://doi.org/10.1016/j.entcs.2015.04.003.

[Ree03] Jason Reed. Extending higher-order unification to support proof irrelevance. In *TPHOLs 2003*, pages 238–252. 2003. doi:10.1007/1093075516.

[Rey83] John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983.

[RS17] E. Riehl and M. Shulman. A type theory for synthetic ∞-categories. *ArXiv e-prints*, May 2017. arXiv:1705.07442.

[Ste22] Jonathan Sterling. First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory. 4 2022. URL: https://kilthub.cmu.edu/articles/thesis/FirstStepsinSyntheticTaitComputability TheObjectiveMetatheoryofCubicalTypeTheory/19632681, doi:10.1184/R1/19632681.v1.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, IAS, 2013.

[VvdW19] Niccolò Veltri and Niels van der Weide. Guarded recursion in agda via sized types. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 32:1–32:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.32.

[Wad89] Philip Wadler. Theorems for free! In *FPCA '89*, pages 347–359, New York, NY, USA, 1989. ACM. doi:10.1145/99370.99404.

[Wei22] Jonathan Weinberger. *A Synthetic Perspective on (∞, 1)-Category Theory: Fibrational and Semantic Aspects*. PhD thesis, TU Darmstadt, Germany, 2022. URL: https://arxiv.org/abs/2202.13132, arXiv:2202.13132.

[WL20] Matthew Z. Weaver and Daniel R. Licata. A constructive model of directed univalence in bicubical sets. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 915–928. ACM, 2020. doi:10.1145/3373718.3394794.