Parametric Quantifiers for Dependent Type Theory

ANDREAS NUYTS, KU Leuven, Belgium
ANDREA VEZZOSI, Chalmers University of Technology, Sweden
DOMINIQUE DEVRIESE, KU Leuven, Belgium

Polymorphic type systems such as System F enjoy the parametricity property: polymorphic functions cannot inspect their type argument and will therefore apply the same algorithm to any type they are instantiated on. This idea is formalized mathematically in Reynolds's theory of relational parametricity, which allows the metatheoretical derivation of parametricity theorems about all values of a given type. Although predicative System F embeds into dependent type systems such as Martin-Löf Type Theory (MLTT), parametricity does not carry over as easily. The identity extension lemma, which is crucial if we want to prove theorems involving equality, has only been shown to hold for small types, excluding the universe.

We attribute this to the fact that MLTT uses a single type former Π to generalize both the parametric quantifier $\forall$ and the type former $\to$ which is non-parametric in the sense that its elements may use their argument as a value. We equip MLTT with parametric quantifiers $\forall$ and $\exists$ alongside the existing Π and Σ, and provide relation type formers for proving parametricity theorems internally. We show internally the existence of initial algebras and final co-algebras of indexed functors both by Church encoding and, for a large class of functors, by using sized types.

We prove soundness of our type system by enhancing existing iterated reflexive graph (cubical set) models of dependently typed parametricity by distinguishing between edges that express relatedness of objects (bridges) and edges that express equality (paths). The parametric functions are those that map bridges to paths.

We implement an extension to the Agda proof assistant that type-checks proofs in our type system.

CCS Concepts: • **Software and its engineering** → **Polymorphism**; *Formal methods*; *Functional languages*; *Syntax*; *Semantics*; • **Theory of computation** → *Proof theory*;

Additional Key Words and Phrases: Parametricity, cubical type theory, presheaf semantics, sized types, Agda

ACM Reference Format:
Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. 2017. Parametric Quantifiers for Dependent Type Theory. *Proc. ACM Program. Lang.* 1, ICFP, Article 32 (September 2017), 29 pages.
https://doi.org/10.1145/3110276

1 INTRODUCTION

Many type systems and functional programming languages support functions that are parametrized by a type. For example, we may create a tree flattening function $\mathsf{flatten}\,\alpha : \mathsf{Tree}\,\alpha \to \mathsf{List}\,\alpha$ that works for any type α. If the implementation of a parametrized function does not inspect the particular type α that it is operating on, possibly because the type system prohibits this, then the function is said to be *parametric*: it applies the same algorithm to all types. From this knowledge, we obtain various useful 'free theorems' about the function [Reynolds 1983; Wadler 1989]. For example, if we have a function $f : A \to B$, then we know that $\mathsf{listmap}\,f \circ \mathsf{flatten}\,A = \mathsf{flatten}\,B \circ \mathsf{treemap}\,f$. If parametricity is enforced by the type system, as is the case in System F but also in a programming

2475-1421/2017/9-ART32
https://doi.org/10.1145/3110276

language like Haskell, then we can deduce such free theorems purely from a function's type signature, without knowledge of its implementation. This allows parts of a function's contract to be enforced by the type-checker; a powerful feature.

Existing work on parametricity in dependent type systems such as Martin-Löf Type Theory (MLTT) has been able to show that the expected parametricity results hold for functions that produce values of a small type [Atkey et al. 2014; Krishnaswami and Dreyer 2013; Takeuti 2001]. Below, we show with a simple example that in existing dependent type systems, parametricity theorems can break down where large types are involved. The central aim of this paper is to resolve this issue by equipping dependent type theory with additional parametric quantifiers.

Representation independence in System F. In order to expose the problem that occurs in dependent type theory, we will elaborate an example that shows the power of parametricity in System F, but which breaks down in dependent type theory. Assume that A is a type that is essentially an interface, listing the operations that its elements provide. Then typically, we will not directly construct values of type A; rather, we will construct representations of them in some type C, from which we can extract the operations using a function $r : C \to A$. A function f of type $\mathsf{RI}\ A\ B :\equiv \forall\gamma.(\gamma \to A) \to (\gamma \to B)$ for some fixed type B, is then a function that, for any type γ that implements the interface A as witnessed by $r : \gamma \to A$, produces a map $\gamma \to B$. Parametricity now asserts that f is representation independent: it can only use the argument $c : \gamma$ through its operations $r\ c : A$ and is thus oblivious to the particular implementation. Hence, elements of type $\mathsf{RI}\ A\ B$ are in one-to-one correspondence with functions $A \to B$.

Representation polymorphism in dependent type theory. Dependent type theory departs from System F in that it erases the strict dichotomy between types and values. The result is a system in which types can depend on values, and can appear as values themselves, possibly as computational content of other values (e.g. we can consider lists of types).

The function type former $\to$ from System F, is replaced with the type former Π (called the product type, dependent function type or simply Π-type) in dependent type theory. If S is a type and T is a type depending on a variable $x : S$, then the type $\Pi(x : S).T$ contains functions f that map any value $s : S$ to a value $f\,s : T[s/x]$. When T does not depend on x, we simply write $S \to T$ and have recovered the ordinary function type from System F.

If we disregard parametricity, we may also use Π to recover the $\forall$ type former from System F. If the domain S is some type of types $\mathcal{U}$, also called a universe, then the function type $\Pi(\alpha : \mathcal{U}).T$ corresponds to the polymorphic type $\forall\alpha.T$ from System F. So we can translate RI to dependent types as $\mathsf{RI}\ A\ B :\equiv \Pi(C : \mathcal{U}).(C \to A) \to (C \to B)$. However, representation independence is not enforced for this type, and an easy counterexample can be constructed if we let B be the universe $\mathcal{U}$ itself. Then we can break representation independence by directly leaking the implementation type C to the end user:

$$\mathsf{leak} = \lambda C.\lambda r.\lambda c.C : \Pi(C : \mathcal{U}).(C \to A) \to (C \to \mathcal{U}) \tag{1}$$

Wrapping up. We claim that while dependent type theory clearly takes a step forward from System F in that it allows any kind of dependencies, it takes a step back by unifying $\forall$ and $\to$ in a single type constructor. The problem is that functions $f : \forall\alpha.T$ and $g : P \to Q$ differ not only in that f is dependent and takes a type as argument whereas g is non-dependent and takes a value as argument; they also differ in that f is parametric and uses its argument solely for type-checking purposes, whereas g is non-parametric and is allowed to use its argument as a value. It is the second property of $\forall$ that produces the free theorems we want.

In order to restore parametricity for large types in dependent type theory, we reinstate the parametric quantifier $\forall$ from System F alongside the non-parametric quantifier Π (also $\to$) in dependent type theory. The type formation rules for both quantifiers have the exact same premises.

This means that we can quantify parametrically or not over either type-like or value-like arguments, making the distinction between parametricity and non-parametricity orthogonal to the distinction between type-level and value-level arguments, which we seek to erase in dependent type theory. This leads to four situations of which only two existed (at the value level) in System F.

(1) As in System F, we can quantify parametrically over a type argument. An example is the (proper) Church encoding of lists: $\mathsf{ChList}\, B = \forall(X : \mathcal{U}).¶X \to ¶(B \to X \to X) \to X$ (the ¶ modality is explained later).
(2) Unlike in System F, we can quantify non-parametrically over a type argument. A (non-dependent) example is the cons constructor of $\mathsf{List}\,\mathcal{U}$, which has the type $\mathsf{cons} : \mathcal{U} \to \mathsf{List}\,\mathcal{U} \to \mathsf{List}\,\mathcal{U}$. Clearly, this function uses its first argument not just for type checking purposes. Rather, the *value* X can be retrieved from the list $\mathsf{cons}\, X\, Xs$ using the list recursor. Another example is the function type former $\lambda X.\lambda Y.(X \to Y) : \mathcal{U} \to \mathcal{U} \to \mathcal{U}$. The arguments X and Y are not used just for type-checking (in fact they do not even occur in the type $\mathcal{U} \to \mathcal{U} \to \mathcal{U}$); rather, they determine the output *value* $X \to Y$. Non-parametric *dependent* quantification over a type variable is also common. For example, algebraic structures can be represented as dependent tuples and these must be non-parametric in their underlying type, lest we identify structures as soon as there is a homomorphism between them.
(3) Unlike in System F, we can quantify parametrically over a value argument. We have a type Size that is similar to the natural numbers but enforces a form of well-behavedness for functions that have Size as their domain. Let $\mathsf{Vec}_i\, A$ be the type of vectors of length i : Size over A, where vectors of different lengths are considered equal if the truncation of the longer one is equal to the shorter one. Now consider the type $\forall(i : \mathsf{Size}).\mathsf{Vec}_i\, A$. Parametricity ensures that a function of this type will produce equal (i.e. compatible) vectors for all sizes. In other words, this is the type of infinite streams of elements of A.
(4) Like in System F, we can quantify non-parametrically over a value argument. Any ordinary term-level function from System F is an example of this. A more intriguing example is the function $\lambda i.\mathsf{Vec}_i\, A : \mathsf{Size} \to \mathcal{U}$ which maps sizes to types, but also provides a notion of heterogeneous (cross-type) equality between elements of $\mathsf{Vec}_i\, A$ and $\mathsf{Vec}_j\, A$.

Although up until this point, we appealed to the intuition of parametric functions as 'not inspecting an argument', this intuition diverges from the relational formulation when we consider parametric quantification over a data type. Relational parametricity asserts that related inputs will lead to related outputs. The **identity extension lemma** (IEL) moreover implies that relatedness in a closed type, means equality. No assertions are made, however, about unrelated inputs. If we have a type Nat in which (unlike in Size) different natural numbers are considered unrelated, then we can allow to pattern match on them. However, a function $T : \mathsf{Nat} \to \mathcal{U}$ may then not provide a notion of equality between $T\, m$ and $T\, n$ and a function of type $\forall(n : \mathsf{Nat}).T\, n$ need not produce equal output for different numbers (as we even lack the notion of equality). This is a situation that does not apply in basic System F (or Fω), where any two elements of a given kind, are related.

Contributions.

(1) We present a dependent type system **ParamDTT** in which dependencies can be either *parametric* (♯) or *continuous* (**id**). Correspondingly, we obtain (predicative) relationally parametric quantifiers $\forall$ and $\exists$ alongside the usual continuous quantifiers Π and Σ.
(2) We make parametricity theorems provable internally using a type former called Glue (first used by Cohen et al. [2016] in order to achieve computational univalence), and its (novel) dual which we call Weld. These are an alternative for the operators by Bernardy et al. [2015]. Both Glue and Weld have some dependencies that are not continuous and that we cannot prove further parametricity theorems about. This is represented by a third *pointwise* modality

(¶). As such, these type formers cannot be self-applied and iterated parametricity is not fully available internally. We reframe and internalize IEL in the form of an axiom called the *path degeneracy axiom*, enabling us to prove parametricity theorems involving equality.

(3) We construct Church initial algebras and final co-algebras of indexed functors, showing that indexed (co)-recursive types can be built up from simpler components. We prove their universal properties (up to universe level issues) internally, which to our knowledge has not been done before in any type system. These internal proofs have some pointwise dependencies, indicating that internal parametricity does not apply again to those dependencies.

(4) Annotating (co)-recursive types with a size bound on their elements is a modular way to enforce termination and productivity of programs. This is a use case for parametric quantification over values, as we do not want to view an object's size bound as computational content of the object. We construct initial algebras and final co-algebras of a large class of indexed functors using induction on, and parametric quantification over size bounds. We again prove their universal properties internally.

(5) We implement an extension to the dependently typed language Agda, which type-checks ParamDTT and thus shows that its computational behaviour is sufficiently well-behaved to allow for automated type-checking. [1] We expect that ParamDTT minus its equality axioms, which block computation of the J-rule, satisfies all desired computational properties.

(6) We prove the soundness of the type system by constructing a presheaf model in terms of iterated reflexive graphs (more commonly called cubical sets), based on the reflexive graph model by Atkey et al. [2014] and enhancements by Bernardy et al. [2015]. An important innovation in our model is that our iterated graphs have two flavours of edges: bridges express relatedness, and paths express heterogeneous equality of objects living in related types. If we were to model parametricity of System F in the same model, we would use bridges to represent relatedness of types, and paths to represent relatedness of terms. Correspondingly, continuous functions are those that respect edge flavours, whereas parametric functions are those that strengthen bridges to paths.

Overview. In Section 2, we give an informal overview of ParamDTT and its features. In Section 3, we present the formal typing rules and relate the system to MLTT and predicative System Fω. In Section 4, we treat Church encoding and sized types. In Section 5, we give an overview of the presheaf model that proves soundness of ParamDTT. A more complete treatment of the model is found in [Nuyts 2017]. We conclude in Section 6 with a discussion of related work and future research directions.

2 A PROGRAMMER'S PERSPECTIVE

Before we show the formal rules of our type system, we present the system from a programmer's perspective. We consider the typical but simple example type of polymorphic identity functions: $\forall(X : \mathcal{U}).X \to X$.

Modalities. The $\forall$ quantifier is syntactic sugar for $\Pi^\sharp(X : \mathcal{U}).X \to X$: a Π-type annotated with the parametric modality $\sharp$. We can construct a value of this type by annotating a lambda with the same modality: $\lambda(X^\sharp : \mathcal{U}).\sqcup$. In the body, the variable X is available in the context, annotated as $X^\sharp : \mathcal{U}$ to remind us that it should only be used in parametric positions (which we color magenta as a guide to the reader). A variable that is in the context parametrically ($X^\sharp : \mathcal{U}$) does not type-check as a value of type $\mathcal{U}$. Luckily, when we next use a normal lambda $\lambda(x : X).\sqcup$ (i.e. annotated with the continuous modality **id**) to construct a value of type $X \to X$, the type annotation X for x is a parametric position. As such, it is not type-checked in the current context ($X^\sharp : \mathcal{U}$), but in

[1] Available in the artifact, and on Github as the 'parametric' branch of Agda: https://github.com/agda/agda/tree/parametric

the context $(X : \mathcal{U})$, where all parametric variables have been rendered continuous. This context modification is a common theme in modal typing rules: ♯-modal subterms like our X are checked in modified contexts, defined by formally *left-dividing* the current context $(X^\sharp : \mathcal{U})$ by the ♯-modality: $(\sharp \setminus (X^\sharp : \mathcal{U})) = (X : \mathcal{U})$. In the body of the second lambda, we simply return variable x of type X to finish our example.

Another parametric position is the argument that we pass to a parametric function. For example, with $f : \forall(X : \mathcal{U}).X \to X$ in the context, we can construct another value of the same type by composing f with itself: $\lambda(X^\sharp : \mathcal{U}).\lambda(x : X).f\, X^\sharp\, (f\, X^\sharp\, x)$. The variable X is used (twice) as the argument to a parametric function, and as such, it is not type-checked in context $\Gamma = (X^\sharp : \mathcal{U}, x : X)$ (where it would not be accepted), but in $(\sharp \setminus \Gamma) = (X : \mathcal{U}, x : X)$.

As a guide to the reader, we color subterms and variable bindings according to their modality: magenta for parametricity (♯), black for continuity (**id**), blue for the pointwise modality (denoted ¶, see further below), and orange for any unknown modality (some typing rules work for an arbitrary modality μ). Because we want our language to be unambiguous even without color, we will sometimes additionally insert a modality symbol to disambiguate. Continuity (**id**) is considered the default and will be omitted.

Internal parametricity: paths and bridges. A compelling feature of our type system is that we have internal parametricity: free theorems about parametric functions can be derived internally. Imagine that we have a function $f : \forall(X : \mathcal{U}).X \to X$, a type $X^\sharp : \mathcal{U}$ and a value $x : X$ and we want to use parametricity of f to prove that $f\, X^\sharp\, x$ is equal to x. We can do this inside the language, using essentially the same approach that one would take when using binary relational parametricity of System F. There, for every $x : X$, we would construct a relation R_x between the unit type $\top$ and X such that $R_x(u, y)$ iff $x = y$. Since f, being parametric in its first argument, maps R_x-associated second arguments to R_x-associated output, we have that $R_x(f\, \top^\sharp\, \mathrm{tt}, f\, X^\sharp\, x)$, i.e. $f\, X^\sharp\, x = x$.

To construct the relation R_x inside the language, we will construct a *bridge* from $\top$ to X. Such a bridge can best be thought of as a line connecting two values, possibly living in different types. The precise meaning of a bridge depends on the types concerned; a bridge from $B_0 : \mathcal{U}$ to $B_1 : \mathcal{U}$ gives meaning to statements of the form '$b_0 : B_0$ and $b_1 : B_1$ are related' (expressed by a heterogeneous (cross-type) bridge from b_0 to b_1) or '$p_0 : B_0$ and $p_1 : B_1$ are equal' (expressed by a *path* from p_0 to p_1). Note that, unlike in existing accounts of relational parametricity, we distinguish between relatedness and (possibly heterogeneous) equality.

A proof of $R_x(u, y)$ will be represented internally as a *path* from $u : \top$ to $y : X$. A path between values p_0 and p_1 can also be thought of as a line connecting these values. Just like bridges, paths may be heterogeneous, but when they are homogeneous (i.e. when they stay within a single type), they are necessarily constant, implying equality of their endpoints. Moreover, paths are respected by all functions. These properties make the path relation a good notion of heterogeneous equality: a congruence that reduces to equality whenever equality is meaningful. So in order to prove that $f\, X^\sharp\, x = x$, it will be sufficient to construct a path from $f\, X^\sharp\, x$ to x.

We internalize both bridges and paths using a special pseudo-type $\mathbb{I}$ called the interval, which consists of two elements 0 and 1 connected by a bridge. Since continuous functions, by definition in the model, respect bridges, a bridge from B_0 to B_1 can be represented as a function $B : \mathbb{I} \to \mathcal{U}$ such that $B\, 0 \equiv B_0$ and $B\, 1 \equiv B_1$ definitionally. Since parametric functions, by definition in the model, strengthen bridges to paths, a path from $p_0 : B_0$ to $p_1 : B_1$ can be represented as a function $p : \forall(i : \mathbb{I}).B\, i$ such that $p\, 0^\sharp \equiv p_0$ and $p\, 1^\sharp \equiv p_1$. Meanwhile, heterogeneous bridges take the form $b : \Pi(i : \mathbb{I}).B\, i$. The typing rules make sure that the types $\forall(x : A).T\, x$ and $\Pi(x : A).T\, x$ can be formed precisely when we have a continuous function $T : A \to \mathcal{U}$; when $\mathbb{I}$ is the domain, this says that there needs to be a bridge between the types before we can consider bridges or paths

between their values. An internal *path degeneracy axiom* degax finally asserts that non-dependent (homogeneous) paths are in fact constant.

Turning a function into a bridge. Note that the relation R_x we used in the System F proof, is in fact the graph of the function $\lambda u.x : \top \to X$. As mentioned above, we intend to internalize R_x as a bridge from $\top$ to X, i.e. a function $/\lambda u.x\backslash : \mathbb{I} \to \mathcal{U}$ such that $/\lambda u.x\backslash 0$ reduces to $\top$ and $/\lambda u.x\backslash 1$ reduces to X. In fact, an operator $/\_\backslash$ that turns a function into the bridge representing its graph relation, can be implemented using either the primitive Glue type former, or its dual called Weld, which we introduce in Section 3.2. For now, we just assume that we have a bridge $/\lambda u.x\backslash$ and that it comes with a function push : $\forall(i : \mathbb{I}).\top \to /\lambda u.x\backslash i$ from the domain, such that push $0^\sharp \equiv \mathrm{id}_\top : \top \to \top$ and push $1^\sharp \equiv \lambda u.x : \top \to X$; and a function pull : $\forall(i : \mathbb{I})./\lambda u.x\backslash i \to X$ to the codomain, such that pull $0^\sharp \equiv \lambda u.x : \top \to X$ and pull $1^\sharp \equiv \mathrm{id}_X : X \to X$. Now consider the following composite:

$$\top \xrightarrow{\text{push } i^\sharp} /\lambda u.x\backslash i \xrightarrow{f\,(/\lambda u.x\backslash i)^\sharp} /\lambda u.x\backslash i \xrightarrow{\text{pull } i^\sharp} X.$$

For $i \equiv 0$, it reduces to the constant function $\lambda u.x$, while for $i \equiv 1$, it reduces to $\lambda u.f\ X^\sharp\ x$. Thus, applying this to $\mathrm{tt} : \top$, we obtain a homogeneous (non-dependent) path

$$p :\equiv \lambda i^\sharp.\ (\text{pull } i^\sharp \circ f\ (/\lambda u.x\backslash i)^\sharp \circ \text{push } i^\sharp)\ \mathrm{tt} : \sharp\mathbb{I} \to X, \qquad p\,0^\sharp \equiv x, \qquad p\,1^\sharp \equiv f\,X^\sharp\,x.$$

Finally, since this is a non-dependent (homogeneous) path, the aforementioned path degeneracy axiom asserts that it is constant, implying that $x =_X f\,X^\sharp\,x$.

Before we proceed: the pointwise modality. One important aspect of our system has been tucked under the carpet in the above example: the Glue and Weld type formers, as well as the graph type former $/\_\backslash$ implemented in terms of either of them, break the relational structure. This is reflected syntactically by a third *pointwise* modality (¶), which annotates dependencies that have no action on bridges. So to be precise, the above example does not show that any function of type $\forall(X : \mathcal{U}).X \to X$ is the identity; rather, every such function can be weakened to the type $\forall(X : \mathcal{U}).¶X \to X$ (forgetting its action on bridges in the second argument) and we have proven that all functions of *that* type are the identity. In practice, this means the proof is perfectly usable and valid, but we cannot apply another parametricity argument to the proof term. This restriction is a limitation of our current model, but an acceptable one, as we will argue in what follows.

3 THE TYPE SYSTEM, FORMALLY

With the general ideas of ParamDTT established, this section presents the system formally. The first part treats the core type system, which is just MLTT with modality annotations. The second part explains the machinery we use for internal parametricity. The third part adds two types Nat and Size of natural numbers, and we conclude in the fourth part by embedding MLTT and predicative System Fω in our system in two ways.

3.1 Core Typing Rules: Annotating Martin-Löf Type Theory

Modalities. The judgements and typing rules of our system are similar to MLTT, except that, as discussed, every dependency is equipped with a modality: either pointwise (¶), continuous (**id**) or parametric (♯). We follow the general approach developed by Pfenning [2001] and Abel [2006, 2008]. A dependency's modality expresses how it acts on bridges and paths. Functions of all three modalities respect paths. Continuous functions moreover respect bridges, while parametric functions strengthen them to paths, and pointwise functions do not act on bridges whatsoever. Every dependency can be viewed as pointwise by ignoring its action on bridges. Because equal values are also related, our model allows to weaken paths to bridges; this weakening allows to view parametric dependencies as continuous. We express these findings as an order relation on modalities (Fig. 1).

If a term t_1 depends μ_1-modally on a variable x and t_2 depends μ_2-modally on y, then the term $t_2[t_1/y]$ depends on x under a composed modality $\mu_2 \circ \mu_1$. This composition of modalities is defined by the first table in Fig. 1, and follows immediately from the action on bridges and paths. E.g. $\mu \circ \P = \P$, because if the inner function forgets bridges, then the outer one cannot retrieve them, and $\mu \circ \sharp = \sharp$ because if the inner function strengthens bridges to paths, then the outer function has to respect those paths. Note that composition preserves order in both operands.

$\P \le \mathbf{id} \le \sharp$

$\downarrow \circ \to$	$\P$	$\mathbf{id}$	$\sharp$
$\P$	$\P$	$\P$	$\sharp$
$\mathbf{id}$	$\P$	$\mathbf{id}$	$\sharp$
$\sharp$	$\P$	$\sharp$	$\sharp$

$\downarrow \setminus \to$	$\P$	$\mathbf{id}$	$\sharp$
$\P$	$\P$	$\sharp$	$\sharp$
$\mathbf{id}$	$\P$	$\mathbf{id}$	$\sharp$
$\sharp$	$\P$	$\mathbf{id}$	$\mathbf{id}$

Fig. 1. Composition, left division and order of modalities.

Formally, modalities come into play wherever dependencies appear in the type system. First of all, the term on the right of a judgement depends on the context variables, so we annotate those with modalities: we write $x : A$ for continuous, $x^\sharp : A$ for parametric and $x^\P : A$ for pointwise variables. Secondly, a term in the conclusion of an inference rule, also depends on the terms $t : T$ in the premises. As explained before, such terms may be in μ-modal position, in which case the premise will not be $\Gamma \vdash t : T$ but rather $\mu \setminus \Gamma \vdash t : T$, which can be read as $\Gamma \vdash t^\mu : T$ (a non-existent judgement form). This left-division of context Γ by modality μ replaces every dependency $x^\nu : A$ by $x^{\mu\setminus\nu} : A$, defined by the second table in Fig. 1. The left division selects the least modality $\mu \setminus \nu$ such that $\nu \le \mu \circ (\mu \setminus \nu)$. In other words, $\mu \setminus \nu \le \rho \Leftrightarrow \nu \le \mu \circ \rho$ for all ρ, i.e. left division by μ is left adjoint to postcomposition with μ. In general, we take care to maintain admissibility of the following structural rules:

$$\frac{\Gamma, x^\mu : T, \Delta \vdash J \qquad \mu \setminus \Gamma \vdash t : T}{\Gamma, \Delta[t/x] \vdash J[t/x]}\text{subst} \qquad \frac{\Gamma, x^\mu : T, \Delta \vdash J \qquad \nu \le \mu}{\Gamma, x^\nu : T, \Delta \vdash J}\text{mod-wkn}$$

With this modality machinery in place, we can now discuss our typing rules.

Contexts. Contexts are formed by starting from the empty context (c-em) and adding variables in modalities of your choice (c-ext). Context variables are valid terms if their modality is continuous or less (t-var).

$$\frac{}{\vdash \mathrm{Ctx}}\text{c-em} \qquad \frac{\Gamma \vdash T\ \mathrm{type}}{\Gamma, x^\mu : T \vdash \mathrm{Ctx}}\text{c-ext} \qquad \frac{\Gamma \vdash \mathrm{Ctx} \quad (x^\mu : T) \in \Gamma \quad \mu \le \mathbf{id}}{\Gamma \vdash x : T}\text{t-var}$$

Universes. There is a countable hierarchy of universes, each one living in the next (t-Uni). Elements of a universe can be coerced to higher universes (t-lift). Elements of the universe can be turned into types (ty), and this operation is parametric because it shifts T to the type level, preventing any further use as a value.

$$\frac{\Gamma \vdash \mathrm{Ctx} \qquad \ell \in \mathbb{N}}{\Gamma \vdash \mathcal{U}_\ell : \mathcal{U}_{\ell+1}}\text{t-Uni,} \qquad \frac{\Gamma \vdash T : \mathcal{U}_k \qquad k \le \ell \in \mathbb{N}}{\Gamma \vdash T : \mathcal{U}_\ell}\text{t-lift,} \qquad \frac{\sharp \setminus \Gamma \vdash T : \mathcal{U}_\ell}{\Gamma \vdash T\ \mathrm{type}}\text{ty,}$$

Definitional equality. There are equality judgements $\Gamma \vdash s \equiv t : T$ and $\Gamma \vdash S \equiv T\ \mathrm{type}$. We omit the rules that make definitional equality a congruence and an equivalence relation. The conversion rule (t-conv) allows to convert terms between equal types.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B\ \mathrm{type}}{\Gamma \vdash a : B}\text{t-conv,}$$

Quantification. We have universal and existential quantifiers for every modality (t-Π, t-Σ). We denote them as Π^μ and Σ^μ for general modalities, but we abbreviate the continuous ones as Π and Σ and the parametric ones as $\forall$ and $\exists$. Moreover, in the non-dependent case we will write $\mu A \to B$ and $\mu A \times B$. Quantified types are continuous in both domain A and codomain B. Indeed, A and B are not just provided for type-checking but they *determine* the output value $\mathrm{Q}(x : A).B$. The variance of B's dependency on x is worth a mention: we want $b : B$ to be meaningful if an element $a^\mu : A$ is given μ-modally. Since in the claim $b : B$, the type B is in a parametric position, it needs to be

defined for $a^{\sharp\backslash\mu} : A$. For $\mu = \mathbf{id}$ or $\mu = \sharp$, we have that $\sharp \setminus \mu$ is $\mathbf{id}$, so B depends continuously on x. However, if $\mu = ¶$, then the codomain may be pointwise in x, since $\sharp \setminus ¶ = ¶$.

$$\frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma, x^{\sharp\backslash\mu} : A \vdash B : \mathcal{U}_\ell}{\Gamma \vdash \Pi^\mu(x : A).B : \mathcal{U}_\ell}\text{t-}\Pi \qquad \frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma, x^{\sharp\backslash\mu} : A \vdash B : \mathcal{U}_\ell}{\Gamma \vdash \Sigma^\mu(x : A).B : \mathcal{U}_\ell}\text{t-}\Sigma$$

Functions. μ-Modal functions are created using λ-abstraction over μ-modal values (t-λ). μ-Modal function applications depend μ-modally on the argument provided (t-ap). We omit β- and η-rules for functions.

$$\frac{\Gamma, x^\mu : A \vdash b : B}{\Gamma \vdash \lambda(x^\mu : A).b : \Pi^\mu(x : A).B}\text{t-}\lambda \qquad \frac{\Gamma \vdash f : \Pi^\mu(x : A).B \qquad \mu \setminus \Gamma \vdash a : A}{\Gamma \vdash f\ a^\mu : B[a/x]}\text{t-ap}$$

Pairs. The μ-modal existential type $\Sigma^\mu(x : A).B$ contains pairs (a^μ, b) where $a^\mu : A$ and $b : B[a/x]$ (t-pair). We have a dependent eliminator for ν-modal use of pairs (t-indpair), for which we omit the β-rule. Note that the non-dependent specialization of $\mathsf{ind}^{\mathbf{id}}_\exists$ corresponds to the 'unpack' eliminator for existentials in System F [see e.g. Pierce 2002, ch. 24].

$$\frac{\begin{array}{l}\Gamma \vdash \Sigma^\mu(x : A).B\ \mathsf{type}\\ \mu \setminus \Gamma \vdash a : A\\ \Gamma \vdash b : B[a/x]\end{array}}{\Gamma \vdash (a^\mu, b) : \Sigma^\mu(x : A).B}\text{t-pair} \qquad \frac{\begin{array}{l}\Gamma, z^\nu : \Sigma^\mu(x : A).B \vdash C\ \mathsf{type}\\ \Gamma, x^{\nu\circ\mu} : A, y^\nu : B \vdash c : C[(x^\mu, y)/z]\\ \nu \setminus \Gamma \vdash p : \Sigma^\mu(x : A).B\end{array}}{\Gamma \vdash \mathsf{ind}^\nu_{\Sigma^\mu}(z.C, x.y.c, p) : C[p/z]}\text{t-indpair}$$

Using $\mathsf{ind}^{\mathbf{id}}_\Sigma$, we can implement continuous projections for $\Sigma(x : A).B$ as

$$\mathsf{fst} : \Sigma(x : A).B \to A \qquad \mathsf{snd} : \Pi(p : \Sigma(x : A).B).B[\mathsf{fst}\,p/x]$$
$$\mathsf{fst}\,p = \mathsf{ind}^{\mathbf{id}}_\Sigma(z.A, x.y.x, p), \qquad \mathsf{snd}\,p = \mathsf{ind}^{\mathbf{id}}_\Sigma(z.B[\mathsf{fst}\,z/x], x.y.y, p).$$

The model supports the η-rule for $\mathsf{ind}^\nu_{\Sigma^\mu}$, allowing us to assume the η-rule $p \equiv (\mathsf{fst}\,p, \mathsf{snd}\,p)$. For $\Sigma^¶(x : A).B$, we can similarly implement a parametric first and a continuous second projection:

$$\mathsf{fst}^¶ : ♯(\Sigma^¶(x : A).B) \to A \qquad \mathsf{snd}^¶ : \Pi\Big(p : \Sigma^¶(x : A).B\Big).B[\mathsf{fst}\,p/x]$$
$$\mathsf{fst}^¶\,p = \mathsf{ind}^♯_{\Sigma^¶}(z.A, x.y.x, p), \qquad \mathsf{snd}^¶\,p = \mathsf{ind}^{\mathbf{id}}_{\Sigma^¶}(z.A, x.y.y, p).$$

Since p depends on its first component pointwise and $♯ \circ ¶ = ¶$, the term $\mathsf{fst}^¶\,p$ gets pointwise access, hence continuous access, to the first component of p. Again, we assume the η-rule for these projections.

Example 3.1. Consider the Church encoding of streams: $\mathsf{ChStr}\,A = \exists(X : \mathcal{U}_\ell).((X \to A \times X) \times X)$. The above rules allow us to build head and tail functions for this type, left as an exercise to the reader.

Identity types. We have an identity type $a =_A b$, continuous in A, a and b (t-Id).[2] The reflexivity constructor is parametric (t-refl). We can use an equality proof e with modality ν using the dependent eliminator J^ν.[3] We omit the β-rule. Our model supports the reflection rule (t=-rflct), which we will often not want to include as it breaks decidability of type-checking. Instead, we can take some of its consequences as axioms, such as function extensionality. A program that applies the J-rule to an instance of this axiom, will block. The model also supports definitional uniqueness of identity

[2] It may be surprising that the identity type $a =_A b$ is not parametric in the type A, as one might think it is only there for type-checking. However, if this were the case, then by parametricity, the type former $\sqcup =_\sqcup \sqcup$, of type $\forall(X : \mathcal{U}_\ell).X \to X \to \mathcal{U}_\ell$ would have to be constant (this claim holds pointwise in a and b). As such, the type $a =_A b$ would have to remain unchanged if we were to replace the type A, for example, with the related type $\top$ and $a, b : A$ with the heterogeneously equal values $\mathsf{tt}, \mathsf{tt} : \top$.

[3] A J-eliminator for every modality is actually a bit overkill, since the J-rule for a lesser modality follows from the J-rule for a greater one.

proofs (t=-UIP); which could be added to the type system either as-is, or in the form of special-case propositional or definitional rules (e.g. proofs of $a =_A a$ reduce to $\mathsf{refl}\, a$).

$$\frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma \vdash a, b : A}{\Gamma \vdash a =_A b : \mathcal{U}_\ell}\text{t-Id} \qquad \frac{\sharp \setminus \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}\, a : a =_A a}\text{t-refl}$$

$$\frac{\begin{array}{ll}\sharp \setminus \Gamma \vdash a, b : A & \Gamma, y^\sharp : A, w^\nu : a =_A y \vdash C\ \mathsf{type} \\ \nu \setminus \Gamma \vdash e : a =_A b & \Gamma \vdash c : C[a/y, \mathsf{refl}\, a/w]\end{array}}{\Gamma \vdash \mathsf{J}^\nu(a, b, y.w.C, e, c) : C[b/y, e/w]}\text{t-J} \qquad \frac{\begin{array}{l}\sharp \setminus \Gamma \vdash f, g : \Pi^\mu(x : A).B \\ \Gamma \vdash p : \Pi^\mu(x : A).f\, x^\mu =_B g\, x^\mu\end{array}}{\Gamma \vdash \mathsf{funext}^\mu\, p : f =_{\Pi^\mu(x:A).B} g}$$

$$\left(\frac{\Gamma \vdash a, b : A \qquad \Gamma \vdash e : a =_A b}{\Gamma \vdash a \equiv b : A}\text{t=-rflct}\right) \qquad \left(\frac{\Gamma \vdash e, e' : a =_A b}{\Gamma \vdash e \equiv e' : a =_A b}\text{t=-UIP}\right)$$

Example 3.2. The continuous J-rule allows us to prove transport: $\forall(X, Y : \mathcal{U}_\ell).(X =_{\mathcal{U}_\ell} Y) \to X \to Y$. Plugging in reflexivity, we get a term of type $\forall(X : \mathcal{U}_\ell).X \to X$. Parametricity will allow us to conclude, solely from the type, that this term is (pointwise) the identity function.

3.2 Internal Parametricity: Glueing and Welding

With the core typing rules established, let us now turn to the operators that will allow us to prove parametricity theorems internally. In Section 2, we used the type former $/\sqcup\setminus$ for turning a function $f^\P : C \to D$ into a bridge $/f\setminus : \mathbb{I} \to \mathcal{U}$ from C to D. As mentioned, this type former is not a primitive; rather, it can be implemented in terms of either of the primitive type formers Glue and Weld, which we introduce in this section. Because they require a bit of machinery, we start the section with an introductory example in which we implement $/\sqcup\setminus$ in terms of both Glue and Weld. The Glue type former was originally introduced with one additional prerequisite by Cohen et al. [2016] in a type system without modalities.

3.2.1 Introduction: Turning a Bridge into a Function. Let us take an $f^\P : C \to D$ and reiterate from Section 2 the properties that we need the type $/f\setminus : \mathbb{I} \to \mathcal{U}$ to satisfy. The type is a bridge between C and D, so we want to have that $/f\setminus 0 \equiv C$ and $/f\setminus 1 \equiv D$. Additionally, we want there to be a function $\mathsf{push} : \forall(i : \mathbb{I}).C \to /f\setminus i$ such that $\mathsf{push}\, 0^\sharp \equiv \mathsf{id}_C : C \to C$ and $\mathsf{push}\, 1^\sharp \equiv f : C \to D$. Finally, we also need $\mathsf{pull} : \forall(i : \mathbb{I})./f\setminus i \to D$ to the codomain, such that $\mathsf{pull}\, 0^\sharp \equiv f : C \to D$ and $\mathsf{pull}\, 1^\sharp \equiv \mathsf{id}_D : D \to D$. The property $\mathsf{pull}\, i^\sharp \circ \mathsf{push}\, i^\sharp \equiv f$ holds if i equals 0 or 1 and we want it to hold on the entire interval.

A construct called *systems* allows us to construct partially defined terms. For example, we can define $T :\equiv (i \doteq 0\,?\,C \mid i \doteq 1\,?\,D)$, a type that reduces to C if $i \doteq 0$ and to D if $i \doteq 1$. For general i, it is not defined; it only makes sense when the predicate $P :\equiv (i \doteq 0 \vee i \doteq 1)$ holds. Clearly, $/f\setminus i$ should extend T, in the sense that both types should be equal whenever T is defined. A naive solution would be to add a default clause, i.e. something like:

$$/f\setminus i :\equiv (i \doteq 0\,?\,C \mid i \doteq 1\,?\,D \mid \mathsf{else}\ E) \qquad \text{(not ParamDTT syntax).}$$

This approach is not quite right: it would allow us to relate any two types C and D by adding an arbitrary third type as a default clause and it is all but clear what that would mean. But let's consider it anyway. What would be the meaning of a path $p : \forall(i : \mathbb{I})./f\setminus i$ with this definition? The endpoints of the path would live in C and D, while the rest of the path lives in E. So we need some condition to decide whether elements $c : C$ and $d : D$ qualify as endpoints of a path in E (which normally has endpoints also in E). This condition could arise from a relation between C and $E[0/i]$ and one between D and $E[1/i]$, or in short a relation between T and E defined only when P holds. This is essentially how Glue and Weld work; however, they do not allow any kind of relation. The Weld type former takes a partially defined function $g^\P : E \to T$ whereas Glue takes $h^\P : T \to E$. The resulting types are denoted

$$\mathsf{Weld}\{E \to (P\,?\,T, g)\}, \qquad \mathsf{Glue}\{E \leftarrow (P\,?\,T, h)\},$$

and reduce to T when P is true. When P is false, they will be isomorphic to E.[4]

Using the Weld *operation.* In order to form $/f\setminus i$ using Weld, we need a diagram of the form $C \leftarrow E \to D$ that somehow encodes the graph of the function f.[5] In general, a relation can be represented by such a diagram if we let E be the type of related pairs, which for functions is isomorphic to the domain. So we set $E :\equiv C$ (Fig. 2, left column) and $g :\equiv (i \doteq 0\,?\,\mathsf{id}_C \mid i \doteq 1\,?\,f) : C \to T$ (Fig. 2, full arrows from left to middle column). With some syntactic sugar to avoid repeating the same predicate, we can then write (Fig. 2, middle column)

$$/f\setminus i :\equiv \mathsf{Weld}\{C \to (i \doteq 0\,?\,C, \mathsf{id}_C \mid i \doteq 1\,?\,D, f)\}, \qquad /f\setminus 0 \equiv C, \qquad /f\setminus 1 \equiv D.$$

Thinking of $/f\setminus i$ as a system with a default clause, it is clear how we can obtain elements of it: if P holds, we can take elements of T. If it does not hold, then we are (intuitively) in the default case and we can use a constructor-like function $\mathsf{push}\,i^\sharp :\equiv \mathsf{weld}\,(P\,?\,g) \equiv \mathsf{weld}\,(i \doteq 0\,?\,\mathsf{id}_C \mid i \doteq 1\,?\,f) : C \to /f\setminus i$ (Fig. 2, dashed arrow). Moreover, this function extends to the case where P *does* hold, and then it specializes to g, i.e. $\mathsf{push}\,0^\sharp \equiv \mathsf{id}_C$ and $\mathsf{push}\,1^\sharp \equiv f$. This internalizes the idea that the paths in the welded type come from the default case, while their endpoints are associated to them by the function g. Since $\mathsf{weld}\,(P\,?\,g)$ is meaningful regardless of whether P is true or false, it is a total function extending g. To summarize: given a partial type T, a total type E and a partially defined $g : E \to T$, welding extends T to a total type and g to a total function which takes the role of a constructor.

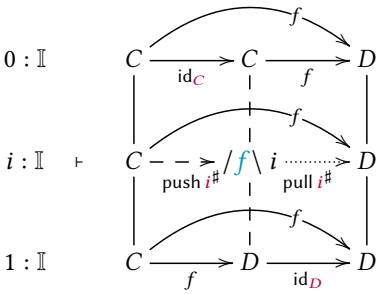

Fig. 2. Formation and elimination of $/f\setminus$ using Weld. The middle row depends on i and reduces to the top/bottom row when i equals 0/1. Welding produces $/f\setminus i$ and the dashed lines.

Finally, in order to construct $\mathsf{pull} : \forall(i : \mathbb{I}).\,/f\setminus i \to D$, we use the eliminator $\mathsf{ind}_{\mathsf{Weld}}$. It allows us to eliminate a value $w : /f\setminus i$ into a goal type D (which could in general depend on w) by inspecting how w was obtained: we need to handle elements created using $\mathsf{weld}\,(P\,?\,g)$ (Fig. 2, curved arrows) and, in the event that P holds, elements living in T (Fig. 2, full arrows from middle to right column). These cases are not disjoint: if P is true, then $\mathsf{weld}\,(P\,?\,g)$ creates elements of type T, so we need to handle them compatibly (Fig. 2, commutation of diagrams in top and bottom row). Thus, we can define $\mathsf{pull}\,i^\sharp$ (with some predicates-related syntactic sugar) as

$$\mathsf{pull}\,i^\sharp :\equiv \mathsf{ind}_{\mathsf{Weld}}(w.D, (i \doteq 0\,?\,c.(f\ c) \mid i \doteq 1\,?\,d.d), c.(f\ c), \sqcup) : /f\setminus i \to D.$$

Note that modalities play an inconspicuous but utterly important role here. The predicate former $\doteq$ is continuous in its endpoints and the Weld type former is continuous in the predicate P, so that $/f\setminus$ is a bridge relating types and not a path equating them. On the other hand, weld and $\mathsf{ind}_{\mathsf{Weld}}$ are parametric in the predicate P, so that push and pull *are* heterogeneous paths, allowing us to prove parametricity theorems involving equality.

Using the Glue *operation.* Glueing is the dual operation to welding. In order to form $/f\setminus i$ using Glue, we need a diagram of the form $C \to E \leftarrow D$ that encodes the graph of the function f. A well-behaved relation can be represented by letting E be the disjoint union of C and D,

[4]The particular predicate $i \doteq 0 \vee i \doteq 1$ will never reduce to $\bot$ as there are no mid-interval constants, so we will never really end up in the default case. A predicate that can become false, is $i \doteq 0$.
[5]In fact, E may depend on i so we can be more general: we need functions $E[0/i] \to C$ and $E[1/i] \to D$. However, glueing and welding over constant types will be powerful enough for all applications in this paper.

quotiented by the relation. For a function, this is isomorphic to the codomain. So we set $E :\equiv D$ and $h :\equiv (i \doteq 0\,?\,f \mid i \doteq 1\,?\,\mathrm{id}_D) : T \to D$. Then we get

$$/f \setminus i :\equiv \mathsf{Glue}\{D \leftarrow (i \doteq 0\,?\,C, f \mid i \doteq 1\,?\,D, \mathrm{id}_D)\}, \qquad /f \setminus 0 \equiv C, \qquad /f \setminus 1 \equiv D.$$

Whereas the Weld type – like an inductive type – lets us eliminate elements by inspecting their construction, the Glue type – like a record type – lets us construct elements by saying how they eliminate. Thinking of $/f \setminus i$ as a system with a default clause, it is clear how we can eliminate an element b: if P holds, then b becomes an element of T; otherwise, we are in the default case and we can use a projection-like function $\mathsf{pull}\ i^\sharp :\equiv \mathsf{unglue}\,(P\,?\,h) \equiv \mathsf{unglue}\,(i \doteq 0\,?\,f \mid i \doteq 1\,?\,\mathrm{id}_D) : /f \setminus i \to D$. Again, this function extends to the case where P does hold, and then it specializes to h. Thus, in order to construct an element $b : /f \setminus i$, we need to say what element t of T it extends, and what element $a : A$ it projects to, and moreover we need that $g\ t \equiv a$. This is then assembled as $b :\equiv \mathsf{glue}\{a \leftarrow (P\,?\,t)\}$. In particular, we can define $\mathsf{push}\ i^\sharp :\equiv \lambda c.\mathsf{glue}\{f\ c \leftarrow (i \doteq 0\,?\,c \mid i \doteq 1\,?\,f\ c)\}$.

The rest of this section is dedicated to making formal every concept encountered in the above examples, to wit: the interval $\mathbb{I}$, a calculus of *face predicates* such as $i \doteq 0$, *systems* for case distinction over face predicates, and the Glue and Weld type formers. We conclude by postulating the path degeneracy axiom.

3.2.2 The Interval. The interval $\mathbb{I}$ is what we use to reason internally about bridges and paths. Although the model treats it as a type like any other, containing just two points 0 and 1 connected by a bridge, we choose to give it an exceptional syntactic treatment with the purpose of preserving the following syntactic property:

Lemma 3.3. *If* $\Gamma \vdash t : \mathbb{I}$, *then* t *is either* 0, 1 *or a variable from* Γ *with modality* $\mu \leq \mathbf{id}$. □

In other words, we want to avoid neutral interval terms. Then we should not have functions with codomain $\mathbb{I}$, and so $\mathbb{I}$ cannot be part of a universe. Instead, we consider $\Gamma \vdash i : \mathbb{I}$ a new class of judgements. So we postulate two constants $0, 1 : \mathbb{I}$ and furthermore allow: context extension with interval variables $i^\mu : \mathbb{I}$ (c-ext), use of interval variables (t-var), the construction of Π^μ- and Σ^μ-types with $\mathbb{I}$ as their domain (t-Π, t-Σ), λ-abstraction over $i^\mu : \mathbb{I}$ (t-λ) and application to interval terms (t-ap), formation of interval pairs (i^μ, y) (t-pair) and pattern matching over such pairs (t-indpair) but *not* first projections for such pairs, as they would pollute the interval term judgement.

3.2.3 Face Predicates and Face Unifiers. If we have m continuous and n parametric interval variables in the context, we can think of the term on the right as ranging over an $(m+n)$-dimensional cube that has m bridge-dimensions and n path-dimensions. A calculus of face predicates allows us to make assumptions about where we are on that cube, and to use those assumptions *definitionally*. Face predicates can be thought of as propositions, i.e. types that have at most one element. For that reason, and because we make sure face predicates are decidable, we will never explicitly write their proofs or hypotheses. Because propositions have no relational structure, we can ignore modalities for their elements. Again, while the model allows us to treat the universe of face predicates $\mathbb{F}$ as an ordinary type, we will be more restrictive in order to preserve decidability of definitional equality.

$$\frac{\Gamma \vdash i, j : \mathbb{I}}{\Gamma \vdash i \doteq j : \mathbb{F}}\text{f-eq}$$

$$\frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash \top : \mathbb{F}}\text{f-tt} \qquad \frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash \bot : \mathbb{F}}\text{f-ff}$$

$$\frac{\Gamma \vdash P, Q : \mathbb{F}}{\Gamma \vdash P \wedge Q : \mathbb{F}}\text{f-}\wedge \qquad \frac{\Gamma \vdash P, Q : \mathbb{F}}{\Gamma \vdash P \vee Q : \mathbb{F}}\text{f-}\vee$$

$$\frac{\Gamma \vdash \mathsf{Ctx} \qquad \sharp \setminus \Gamma \vdash P : \mathbb{F}}{\Gamma, P \vdash \mathsf{Ctx}}\text{c-f}$$

Fig. 3. Formation rules for face predicates.

Figure 3 lists the rules for generating predicates: we can equate interval terms (f-eq), we have true and false predicates (f-tt, f-ff) and conjunction and disjunction (f-$\wedge$, f-$\vee$). We can also extend contexts with a face predicate assumption (c-f). As usual, we omit congruence rules for definitional

equality. Before we can define equality for face predicates, we need two definitions. First, we say that a face predicate *holds* when its translation to a metatheoretical predicate holds, i.e. $i \doteq j$ holds if and only if $i \equiv j$, $\top$ holds, $\bot$ does not hold, $P \wedge Q$ holds when both P and Q hold, and $P \vee Q$ holds when either P or Q holds. Secondly, a *face unifier* $\sigma : \Delta \to \Gamma$ for a context Γ is a substitution of interval variables from a face-predicate-free context. To be precise, they are generated as follows:

$$\frac{}{() : () \to ()}\text{u-em} \qquad \frac{\sigma : \Delta \to \Gamma \qquad \Gamma \vdash T\,\text{type}}{\sigma : (\Delta, x^\mu : T[\sigma]) \to (\Gamma, x^\mu : T)}\text{u-ext} \qquad \frac{\sigma : \Delta \to \Gamma}{\sigma : (\Delta, i^\mu : \mathbb{I}) \to \Gamma}\text{u-wkn}$$

$$\frac{\sigma : \Delta \to \Gamma \qquad \mu \setminus \Delta \vdash j : \mathbb{I}}{(\sigma, j/i) : \Delta \to (\Gamma, i^\mu : \mathbb{I})}\text{u-sub} \qquad \frac{\sigma : \Delta \to \Gamma \qquad \sharp \setminus \Gamma \vdash P : \mathbb{F} \qquad P[\sigma]\,\text{holds}}{\sigma : \Delta \to (\Gamma, P)}\text{u-ext-f}$$

Given $\Gamma \vdash P, Q : \mathbb{F}$, we write $P \Rightarrow Q$ if, for every face unifier σ for Γ such that $P[\sigma]$ holds, $Q[\sigma]$ also holds. We equate predicates that are in this sense equivalent (f=). If an equality predicate is satisfied by any face unifier of the context, we can use it definitionally (i=-f).

$$\frac{\Gamma \vdash P, Q : \mathbb{F} \qquad P \Leftrightarrow Q}{\Gamma \vdash P \equiv Q : \mathbb{F}}\text{f=} \qquad \frac{\Gamma \vdash i, j : \mathbb{I} \qquad \top \Rightarrow (i \doteq j)}{\Gamma \vdash i \equiv j : \mathbb{I}}\text{i=-f}$$

The joint effect of these rules is that, as soon as there are face predicates in the context, type checking no longer happens in the current context, but only after face unification. Even though a given context has infinitely many face unifiers (by weakening), it is always sufficient to check finitely many, and further optimizations are possible. Indeed, if the face predicates in the context do not use disjunctions, then one unifier is sufficient.

Example 3.4. Terms in the context $\Gamma, i : \mathbb{I}, j : \mathbb{I}$ can be thought of as living in context Γ and varying over a two-dimensional square. If we extend the context further with the assumption $P :\equiv i \doteq 0 \vee i \doteq 1 \vee j \doteq 0 \vee j \doteq 1$, then we are restricting ourselves to the sides of that square. Terms in the extended context will be type-checked 4 times, under each of the face unifiers $(0/i), (1/i) : (\Gamma, j : \mathbb{I}) \to (\Gamma, i : \mathbb{I}, j : \mathbb{I}, P)$ and $(0/j), (1/j) : (\Gamma, i : \mathbb{I}) \to (\Gamma, i : \mathbb{I}, j : \mathbb{I}, P)$.

3.2.4 Systems. Systems are the eliminator for proofs of disjunctions ($\vee$) and contradictions ($\bot$). Assuming that $P \vee Q$ is true, we can define a term by giving its value when either P or Q is true, such that the given values match when both are true (t-sys2). Assuming a contradiction, we can spawn terms at will using the empty system (t-sys0). We also give the β-rules for systems and the η-rule for $\lightning$, which states that $\lightning$ is equal to anything.

$$\frac{\begin{array}{l}\Gamma \vdash A\,\text{type} \qquad \Gamma, P \vdash a : A \qquad \Gamma, Q \vdash b : A \\ \Gamma, P \wedge Q \vdash a \equiv b : A \qquad \sharp \setminus \Gamma \vdash P \vee Q \equiv \top : \mathbb{F}\end{array}}{\Gamma \vdash (P\,?\,a \mid Q\,?\,b) : A}\text{t-sys2} \qquad \frac{\begin{array}{l}\Gamma \vdash A\,\text{type} \\ \sharp \setminus \Gamma \vdash \bot \equiv \top : \mathbb{F}\end{array}}{\Gamma \vdash \lightning : A}\text{t-sys0}$$

$$(\top\,?\,a \mid Q\,?\,b) \equiv a, \qquad (P\,?\,a \mid \top\,?\,b) \equiv b, \qquad \lightning \equiv a.$$

In order to avoid repeating predicates, we will denote $(P\,?\,a \mid Q \vee R\,?\,(Q\,?\,b \mid R\,?\,c))$ shorter as $(P\,?\,a \mid Q\,?\,b \mid R\,?\,c)$.

3.2.5 Welding. As clarified in Section 3.2.1, the Weld type (t-Weld) comes with a constructor (t-weld) and a variance-polymorphic eliminator (t-indweld). We supplement these rules with three equations that express that Weld and weld extend the partial objects that they should extend, as well as two β-rules for $\mathsf{ind}_{\mathsf{Weld}}$. The β-rules are compatible due to the equality required by $\mathsf{ind}_{\mathsf{Weld}}$.

$$\frac{\begin{array}{l}\Gamma \vdash P : \mathbb{F} \qquad \Gamma, P \vdash T : \mathcal{U}_\ell \\ \Gamma \vdash A : \mathcal{U}_\ell \qquad \P \setminus \Gamma, P \vdash f : A \to T\end{array}}{\Gamma \vdash \mathsf{Weld}\{A \to (P\,?\,T, f)\} : \mathcal{U}_\ell}\text{t-Weld} \qquad \frac{\begin{array}{l}\Gamma \vdash \mathsf{Weld}\{A \to (P\,?\,T, f)\}\,\text{type} \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{weld}\,(P\,?\,f)\,a : \mathsf{Weld}\{A \to (P\,?\,T, f)\}}\text{t-weld}$$

$$\frac{\begin{array}{ll}\Gamma, y^\nu : \mathrm{Weld}\{A \to (P\,?\,T, f)\} \vdash C\,\mathrm{type} & \Gamma, P, y^\nu : T \vdash d : C \\ \Gamma, x^\nu : A \vdash c : C[\mathrm{weld}\,(P\,?\,f)\,x/y] & \Gamma, P, x^\nu : A \vdash c \equiv d[f\,x/y] : C[f\,x/y] \\ \nu \setminus \Gamma \vdash b : \mathrm{Weld}\{A \to (P\,?\,T, f)\} & \end{array}}{\Gamma \vdash \mathrm{ind}^\nu_{\mathrm{Weld}}(y.C, (P\,?\,y.d), x.c, b) : C[b/y]}\text{t-indweld}$$

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{Weld}\{A \to (\top\,?\,T, f)\} & \equiv & T, & \mathrm{ind}^\nu_{\mathrm{Weld}}(y.C, (\top\,?\,y.d), x.c, b) & \equiv & d[b/y], \\ \mathrm{weld}\,(\top\,?\,f)\,a & \equiv & f\,a, & \mathrm{ind}^\nu_{\mathrm{Weld}}(y.C, (P\,?\,y.d), x.c, \mathrm{weld}\,(P\,?\,f)\,a) & \equiv & c[a/x]. \end{array}$$

3.2.6 Glueing. Dually, the Glue type (t-Glue) comes with a projection (t-unglue) and a constructor (t-glue). We supplement these with three equations stating what happens when $P \equiv \top$, a β-rule and an η-rule.

$$\frac{\begin{array}{ll}\Gamma \vdash P : \mathbb{F} & \Gamma, P \vdash T : \mathcal{U}_\ell \\ \Gamma \vdash A : \mathcal{U}_\ell & \P \setminus \Gamma, P \vdash f : T \to A \end{array}}{\Gamma \vdash \mathrm{Glue}\{A \leftarrow (P\,?\,T, f)\} : \mathcal{U}_\ell}\text{t-Glue} \qquad \frac{\Gamma \vdash b : \mathrm{Glue}\{A \leftarrow (P\,?\,T, f)\}}{\Gamma \vdash \mathrm{unglue}\,(P\,?\,f)\,b : A}\text{t-unglue}$$

$$\frac{\Gamma \vdash \mathrm{Glue}\{A \leftarrow (P\,?\,T, f)\}\,\mathrm{type} \qquad \Gamma, P \vdash t : T \qquad \Gamma \vdash a : A \qquad \Gamma, P \vdash f\,t \equiv a : A}{\Gamma \vdash \mathrm{glue}\{a \leftarrowtail (P\,?\,t)\} : \mathrm{Glue}\{A \leftarrow (P\,?\,T, f)\}}\text{t-glue}$$

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{Glue}\{A \leftarrow (\top\,?\,T, f)\} & \equiv & T, & \mathrm{unglue}\,(P\,?\,f)\,(\mathrm{glue}\{a \leftarrowtail (P\,?\,t)\}) & \equiv & a, \\ \mathrm{glue}\{a \leftarrowtail (\top\,?\,t)\} & \equiv & t, & \mathrm{glue}\{(\mathrm{unglue}\,(P\,?\,f)\,b) \leftarrowtail (P\,?\,b)\} & \equiv & b. \\ \mathrm{unglue}\,(\top\,?\,f)\,b & \equiv & f\,b, & & & \end{array}$$

3.2.7 From Identity Extension to the Path Degeneracy Axiom. Relational parametricity in System F [Reynolds 1983] asserts that, at both the type and the value level, related inputs will lead to related outputs; thus, 'relatedness' is a congruence. In this formulation, to say that types A_1 and A_2 are related, is to give a relation $[A] : \mathrm{Rel}(A_1, A_2)$. Relatedness of values $a_1 : A_1$ and $a_2 : A_2$ is then defined by that same relation $[A]$. The identity extension lemma (IEL) asserts that if, at the type level, all inputs are of the form $\mathrm{Eq}_A : \mathrm{Rel}(A, A)$, then the output will also be of that form. In particular, relatedness in a closed type (with no type-level input) means equality. Then 'relatedness' at the value level can be thought of as heterogeneous equality: it is a congruence that boils down to equality when it becomes homogeneous. However, 'relatedness' at the type level does not mean equality, as $\mathrm{Rel}(A, B)$ can be inhabited for any two types A and B. This explains the difficulty in extending IEL to large types in dependent type theory.

Our approach in the transition to dependent types, is to mix value and type levels, while maintaining two separate relations: the bridge relation is like type-level relatedness in System F, whereas the path relation is like value-level relatedness in System F and expresses heterogeneous equality. Parametric functions in System F map types to values; hence our parametric functions map bridges to paths. We already know that all functions preserve paths (since $\mu \circ \sharp = \sharp$), so we only need to add that path-connectedness in closed types means equality. We assert this by postulating that all non-dependent paths are constant, implying that their endpoints are equal:

$$\frac{\Gamma \vdash A\,\mathrm{type} \qquad \sharp \setminus \Gamma \vdash p : \forall(i : \mathbb{I}).A}{\Gamma \vdash \mathrm{degax}\,p : p =_{\forall(i:\mathbb{I}).A} (\lambda(i^\sharp : \mathbb{I}).p\,0^\sharp)}\text{t-degax}$$

3.3 Related and Unrelated Naturals

We include two types Nat and Size which both represent the natural numbers, but with different relational structure. In Nat, numbers are only related to themselves, i.e. every bridge $\mathbb{I} \to \mathrm{Nat}$ is constant. In Size, any two sizes are related, i.e. the bridge relation is codiscrete. As such, it is easier to create functions of domain Nat, but these functions come with fewer type-guaranteed properties. The type of naturals Nat is highly similar to that of MLTT:

$$\frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash \mathsf{Nat} : \mathcal{U}_0}\text{t-Nat} \qquad \frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash 0 : \mathsf{Nat}}\text{t-0} \qquad \frac{\Gamma \vdash n : \mathsf{Nat}}{\Gamma \vdash \mathsf{s}\, n : \mathsf{Nat}}\text{t-s}$$

$$\frac{\begin{array}{l}\Gamma, m^\nu : \mathsf{Nat} \vdash C\,\mathsf{type}\\ \Gamma \vdash c_0 : C[0/m]\\ \Gamma, m^\nu : \mathsf{Nat}, c : C \vdash c_\mathsf{s} : C[\mathsf{s}\, m/m]\\ \nu \setminus \Gamma \vdash n : \mathsf{Nat}\end{array}}{\Gamma \vdash \mathsf{ind}^\nu_\mathsf{Nat}(m.C, c_0, m.c.c_\mathsf{s}, n) : C[n/m]}\text{t-indnat}$$

We omit the β-rules. Perhaps surprisingly, the eliminator allows us to create parametric functions from the naturals by pattern matching. For example, we can have a parametric identity function:

$$\lambda n^\sharp.\mathsf{ind}^\sharp_\mathsf{Nat}(m.\mathsf{Nat}, 0, m.c.(\mathsf{s}\, c), n) : \sharp\mathsf{Nat} \to \mathsf{Nat}. \tag{2}$$

This makes sense, because Nat has no interesting bridges, so every function trivially maps bridges to paths, as required by the parametric modality $\sharp$. This is in line with the known theorem [Atkey et al. 2014; Takeuti 2001] that any function in MLTT with small codomain, is parametric, even though such functions may pattern match on natural numbers. The reason for the surprise may lie in the fact that parametricity over a discrete domain typically does not arise in e.g. System Fω, where elements of the most common kinds are always related. In Section 6, we contrast parametricity with irrelevance.

The type Size, in contrast, has the following constructors:

$$\frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash \mathsf{Size} : \mathcal{U}_0}\text{t-Size} \qquad \frac{\Gamma \vdash \mathsf{Ctx}}{\Gamma \vdash 0_\mathsf{S} : \mathsf{Size}}\text{t-Size-0} \qquad \frac{\Gamma \vdash n : \mathsf{Size}}{\Gamma \vdash \uparrow n : \mathsf{Size}}\text{t-Size-s} \qquad \frac{\begin{array}{l}\Gamma \vdash P : \mathbb{F}\\ \Gamma, P \vdash n : \mathsf{Size}\end{array}}{\Gamma \vdash \mathsf{fill}(P\,?\,n) : \mathsf{Size}}\text{t-Size-fill}$$

where $\mathsf{fill}(P\,?\,n)$ extends n, i.e. $\mathsf{fill}(\top\,?\,n) \equiv n : \mathsf{Size}$. Moreover, we have $\mathsf{fill}(\bot\,?\,n) \equiv 0_\mathsf{S}$. So the fill operation completes a partial size by adding 0_S in missing vertices, and filling up the relational parts using the fact that any two sizes are related. However, we want more: if the bridge relation is to be codiscrete, then the relational action of any continuous function to Size should be void of information. This means not only that any two sizes are related, as is asserted by $\lambda i.\mathsf{fill}(i \doteq 0\,?\,m \mid i \doteq 1\,?\,n)$, but also that they are related in a unique way. To that end, we add the following, somewhat unconventional typing rule:

$$\frac{(i^\mu : \mathbb{I}) \in \Gamma \qquad \mu \leq \mathbf{id} \qquad \Gamma \vdash m, n : \mathsf{Size} \qquad \Gamma, (i \doteq 0 \vee i \doteq 1) \vdash m \equiv n : \mathsf{Size}}{\Gamma \vdash m \equiv n : \mathsf{Size}}\text{t=-Size-codisc}$$

Repeated application shows that two sizes are equal as soon as they are equal on all bridge-vertices of Γ, i.e. they become equal after substituting every continuous and pointwise interval variable with a constant.

An eliminator that simply takes images for each of these constructors, satisfying the necessary equations, would be very complicated to use. Instead, we choose to provide the strong principle of induction (t-fix) which uses an inequality type (t-$\leq$). Since instances of the fix^ν combinator expand non-terminatingly, we include its β-rule as a (non-computational) equality axiom (t-fix-eq), although the model supports the definitional equality.

$$\frac{\Gamma, n^\nu : \mathsf{Size} \vdash A\,\mathsf{type} \qquad \Gamma \vdash f : \Pi^\nu(n : \mathsf{Size}).(\Pi^\nu(m : \mathsf{Size}).(\uparrow m \leq n) \to A[m/n]) \to A}{\Gamma \vdash \mathsf{fix}^\nu\, f : \Pi^\nu(n : \mathsf{Size}).A}\text{t-fix}$$

$$\frac{\sharp \setminus \Gamma \vdash \mathsf{fix}^\nu\, f : \Pi^\nu(n : \mathsf{Size}).A \qquad (\sharp \circ \nu) \setminus \Gamma \vdash m : \mathsf{Size}}{\Gamma \vdash \mathsf{fix}^\nu_=\, f\, m : \mathsf{fix}^\nu\, f\, m^\nu =_A f\, m^\nu\, (\lambda n^\nu.\lambda e.\mathsf{fix}^\nu\, f\, n^\nu)}\text{t-fix-eq} \qquad \frac{\Gamma \vdash m, n : \mathsf{Size}}{\Gamma \vdash m \leq n : \mathcal{U}_0}\text{t-}\leq$$

$$\frac{\sharp \setminus \Gamma \vdash n : \mathsf{Size}}{\Gamma \vdash \mathsf{refl}_\leq\, n : n \leq n}\text{t-}\leq\text{-refl} \qquad \frac{\Gamma \vdash e_1 : n_0 \leq n_1 \qquad \Gamma \vdash e_2 : n_1 \leq n_2}{\Gamma \vdash \mathsf{trans}_\leq\, e_1\, e_2 : n_0 \leq n_2}\text{t-}\leq\text{-trans}$$

$$\frac{\sharp \setminus \Gamma \vdash n : \mathsf{Size}}{\Gamma \vdash \mathsf{zero}_\leq\, n : 0 \leq n}\text{t-}\leq\text{-zero} \qquad \frac{\Gamma \vdash e : m \leq n}{\Gamma \vdash \mathsf{step}_\leq\, e : \uparrow m \leq \uparrow n}\text{t-}\leq\text{-step}$$

$$\frac{\Gamma, P \vdash e : m \leq n}{\Gamma \vdash \mathsf{fill}_{\leq}(P\,?\,e) : \mathsf{fill}(P\,?\,m) \leq \mathsf{fill}(P\,?\,n)}\text{t-}\leq\text{-fill} \qquad \begin{array}{lcl}\mathsf{fill}_{\leq}(\top\,?\,e) & \equiv & e \\ \mathsf{fill}_{\leq}(\bot\,?\,e) & \equiv & \mathsf{zero}_{\leq}\,0_{\mathsf{S}}\end{array}$$

$$\frac{(i^{\mu} : \mathbb{I}) \in \Gamma \qquad \Gamma \vdash e, e' : m \leq n \qquad \Gamma, (i \doteq 0 \vee i \doteq 1) \vdash e \equiv e' : m \leq n}{\Gamma \vdash e \equiv e' : m \leq n}\text{t=-}\leq\text{-codisc}$$

The combinator fix^{ν} allows us to define $g\,n^{\nu}$ in terms of the restriction of g to $\mathsf{Size}_{<n}$:

$$g|_{<n} :\equiv \lambda(m^{\nu} : \mathsf{Size}).\lambda(e : \uparrow m \leq n).g\,m^{\nu}. \tag{3}$$

An intuitive argument why fix^{ν} can have the modalities it does, is the following: since we have $\mathsf{fix}^{\nu}\,f\,n^{\nu} = f\,n^{\nu}\,(\lambda m^{\nu}.\lambda e.\mathsf{fix}^{\nu}\,f\,m^{\nu})$ and the right hand side is ν-modal in n, so is the left hand side. This reasoning shows that, when we pass a bridge in the argument n, the action of ν on bridges will be respected during expansion of fix^{ν}. Note that although the 0_{S} and $\uparrow$ constructors allow us to create a function $\mathsf{Nat} \to \mathsf{Size}$, we cannot construct an inverse. Indeed, fix^{ν} does not allow arbitrary distinctions between 0_{S} and $\uparrow n$.

We could add many more Size-related primitives, e.g. for deriving contradictions from assumptions like $e : \uparrow n \leq n$ or to prove that all inequality proofs are definitionally equal. However, the current set of rules, combined with the following maximum operator (t-$\sqcup$) is more than sufficient for the practical applications in Section 4.2.

$$\frac{\Gamma \vdash m, n : \mathsf{Size}}{\Gamma \vdash m \sqcup n : \mathsf{Size}}\text{t-}\sqcup \qquad \frac{\sharp \leq \Gamma \vdash m, n : \mathsf{Size}}{\Gamma \vdash \mathsf{lmax}_{\leq}\,m\,n : m \leq m \sqcup n}\text{t-}\leq\text{-}\sqcup\text{-l} \qquad \frac{\sharp \leq \Gamma \vdash m, n : \mathsf{Size}}{\Gamma \vdash \mathsf{rmax}_{\leq}\,m\,n : n \leq m \sqcup n}\text{t-}\leq\text{-}\sqcup\text{-r}$$

3.4 Embedding Other Systems

To understand precisely how powerful our type system is, it is useful to compare it to others. In this section, we show that MLTT and a predicative variant of System Fω can be embedded into our system using either the continuous or the pointwise modality. The latter shows that the pointwise modality of Glue and Weld's dependency on the diagram arrows will only interfere with special features of ParamDTT and not with features already present in MLTT or System Fω. Indeed, everything that can be done in MLTT, can be done pointwise in ParamDTT. Each of these results can be proven by induction on the derivation tree of the translated judgement.

Lemma 3.5 (Embedding of MLTT). *Let μ be either* **id** *or* ¶. *Then every derivable judgement of MLTT can be translated to a derivable judgement of ParamDTT by inserting μ wherever a modality is required.*[6] *Thus, we have extended MLTT.*

We can make System Fω [see e.g. Pierce 2002, ch. 30] predicative by annotating the kind $*$ with a level $\ell \in \mathbb{N}$ and assigning levels to types in the style of MLTT, e.g. if $\varphi : *_j \to *_k \vdash A : *_\ell$, then $\forall(\varphi : *_j \to *_k).A : *_{\max\{j+1,k+1,\ell\}}$. This extends the predicative variant of System F by Leivant [1991].

Lemma 3.6 (Embedding of predicative System Fω). *Let μ be either* **id** *or* ¶. *Then every derivable judgement of predicative System Fω can be translated to a derivable judgement of ParamDTT by the tables in Fig. 4 (where we omit the translation of terms and equality judgements).*[7]

Kinds	$\to$	Types
$[\![*_\ell]\!]$	=	$\mathcal{U}_\ell$
$[\![\kappa \to \kappa']\!]$	=	$[\![\kappa]\!] \to [\![\kappa']\!]$

Kinding contexts	$\to$	Contexts
$[\![()]\!]$	=	$()$
$[\![\Delta, \alpha : \kappa]\!]$	=	$[\![\Delta]\!], \alpha^{\sharp} : [\![\kappa]\!]$

Contexts	$\to$	Contexts
$[\![\Delta \mid ()]\!]$	=	$[\![\Delta]\!]$
$[\![\Delta \mid \Gamma, x : A]\!]$	=	$[\![\Delta \mid \Gamma]\!], x^{\mu} : A$

Types	$\to$	Types
$[\![\forall(\alpha : \kappa).A]\!]$	=	$\forall(\alpha : [\![\kappa]\!]).[\![A]\!]$
$[\![\exists(\alpha : \kappa).A]\!]$	=	$\exists(\alpha : [\![\kappa]\!]).[\![A]\!]$
$[\![A \to B]\!]$	=	$\mu[\![A]\!] \to [\![B]\!]$
$[\![A \times B]\!]$	=	$\mu[\![A]\!] \times [\![B]\!]$

Judgements	$\to$	Judgements
$[\![\Delta \vdash \mathsf{KindingCtx}]\!]$	=	$[\![\Delta]\!] \vdash \mathsf{Ctx}$
$[\![\Delta \mid \Gamma \vdash \mathsf{Ctx}]\!]$	=	$[\![\Delta \mid \Gamma]\!] \vdash \mathsf{Ctx}$
$[\![\Delta \vdash T : \kappa]\!]$	=	$\sharp \setminus [\![\Delta]\!] \vdash [\![T]\!] : [\![\kappa]\!]$
$[\![\Delta \mid \Gamma \vdash t : T]\!]$	=	$[\![\Delta \mid \Gamma]\!] \vdash [\![t]\!] : [\![T]\!]$

Fig. 4. Embedding predicative System Fω into ParamDTT.

4 APPLICATIONS

Mechanized Agda proofs for the results of this section, are available in the artifact or online[8].

[6] Alternatively, all Σ-types can be annotated with **id** instead of μ.
[7] Alternatively, all product types can be annotated with **id** instead of μ.
[8] https://github.com/Saizan/parametric-demo

4.1 Church Encoded (Co)-Recursive Types

In System F, we can use Church encoding to represent data types. For example, for a fixed type B, we can encode the type of lists over B as $\mathsf{ChList}\ B :\equiv \forall\alpha.\alpha \to (B \to \alpha \to \alpha) \to \alpha$. The list $[b_1, b_2]$ is then encoded as $\Lambda\alpha.\lambda\mathsf{nil}'.\lambda\mathsf{cons}'.\mathsf{cons}'\ b_1\ (\mathsf{cons}'\ b_2\ \mathsf{nil}')$. With dependent types, we would define $\mathsf{ChList}_\ell\ B :\equiv \Pi(X : \mathcal{U}_\ell).X \to (B \to X \to X) \to X$. Now if B happens to be some universe $\mathcal{U}_j$ with $j \geq \ell$, then the following term should arouse suspicion:

$$\mathsf{exoticList} = \lambda X.\lambda\mathsf{nil}'.\lambda\mathsf{cons}'.\mathsf{cons}'\ X\ \mathsf{nil}' : \mathsf{ChList}_\ell\ \mathcal{U}_j$$

If this is a list of types, then it is one whose contents depend on the type X which we are eliminating to. This may not be detrimental – for example an analysis based on universe levels may reveal that we cannot extract the contents of such a list if $j \geq \ell$. However, using $\forall$ instead of Π, we can instead forbid exoticList altogether.

In this section, we use parametric Church encoding to construct indexed (co)-recursive data types. More formally put, we construct, up to universe level issues, initial algebras and final co-algebras for indexed functors, and prove their universal properties internally. We assume throughout this section that we have a context Γ and an index type $\sharp \setminus \Gamma \vdash Z : \mathcal{U}_0$ (which we take, for simplicity, in $\mathcal{U}_0$ although our formal proof in Agda is more general). We will write $X \Rightarrow Y$ for $\forall(z : Z).X\ z \to Y\ z$ and $\circledcirc$ for the corresponding composition operator.[9] We moreover assume a functor F which is a scheme consisting of

$$\sharp \setminus \Gamma \vdash F_\ell : (Z \to \mathcal{U}_\ell) \to (Z \to \mathcal{U}_\ell)$$
$$¶ \setminus \Gamma \vdash \vec{F}^\ell_k : \forall(X : Z \to \mathcal{U}_k).\forall(Y : Z \to \mathcal{U}_\ell).(X \Rightarrow Y) \to (F\,X \Rightarrow F\,Y)$$

for all metatheoretic $k, \ell \in \mathbb{N}$, such that identity and composition are preserved definitionally, even across universes. We will omit level annotations on F and $\vec{F}$, as well as the first two arguments of $\vec{F}$. Throughout the section, we will write $t^\mu : T$ for $\mu \setminus \Gamma \vdash t : T$. Note that theorems are strongest when their assumptions are parametric and their conclusions are pointwise.

4.1.1 *Initial Algebras.* [10] Define

$$\sharp \setminus \Gamma \vdash \mathsf{Mu}_\ell :\equiv \lambda z.\forall(X : Z \to \mathcal{U}_\ell).¶(F\,X \Rightarrow X) \to X\ z : Z \to \mathcal{U}_{\ell+1},$$
$$\Gamma \vdash \mathsf{fold} :\equiv \lambda X^\sharp.\lambda\mathsf{mk}X^¶.\lambda z^\sharp.\lambda m.m\ X^\sharp\ \mathsf{mk}X^¶ : \forall(X : Z \to \mathcal{U}_\ell).¶(F\,X \Rightarrow X) \to (\mathsf{Mu}_\ell \Rightarrow X),$$
$$\Gamma \vdash \mathsf{mkMu}_\ell :\equiv \lambda z^\sharp.\lambda\widehat{m}.\lambda X^\sharp.\lambda\mathsf{mk}X^¶.(\mathsf{mk}X \circledcirc \vec{F}\,(\mathsf{fold}\,X^\sharp\ \mathsf{mk}X^¶))\ z^\sharp\ \widehat{m} : F\,\mathsf{Mu}_\ell \Rightarrow \mathsf{Mu}_\ell,$$
$$\Gamma \vdash \downarrow :\equiv \lambda z^\sharp.\lambda m.\lambda X^\sharp.\lambda\mathsf{mk}X^¶.m\ X^\sharp\ \mathsf{mk}X^¶ : \mathsf{Mu}_{\ell+1} \Rightarrow \mathsf{Mu}_\ell.$$

Lemma 4.1 (Naturality of fold). *Assume $A^\sharp, B^\sharp : Z \to \mathcal{U}_\ell$ with algebra structures $\mathsf{mk}A^¶ : F\,A \Rightarrow A$ and $\mathsf{mk}B^¶ : F\,B \Rightarrow B$. Assume a morphism of algebras $f^¶ : A \Rightarrow B$ (i.e. $f \circledcirc \mathsf{mk}A \equiv \mathsf{mk}B \circledcirc \vec{F}\,f$). Then there exists a term $\mathsf{natlemma}^¶ : f \circledcirc \mathsf{fold}\,A^\sharp\ \mathsf{mk}A^¶ =_{\mathsf{Mu}_\ell \Rightarrow B} \mathsf{fold}\,B^\sharp\ \mathsf{mk}B^¶$.*

Sketch of proof. The proof boils down to the construction of the diagram of algebra morphisms in Fig. 5, where $/f\setminus$ differs from the type defined in Section 3.2.1 using Glue in that it is now a Z-indexed type formed from a Z-indexed function, i.e. we currently write $/f\setminus\ i\ z$ instead of $/f\ z^\sharp\setminus\ i$. The path degeneracy axiom then asserts that the top and bottom rows compose to equal functions.

A major hurdle is that we need a proof $(\mathsf{mk}/f\setminus i^\sharp)^¶ : F(/f\setminus\ i) \Rightarrow /f\setminus\ i$ that $/f\setminus\ i$ is an algebra, so that we can apply fold. Here, the push and pull functions are insufficient, and it becomes important that we use the Glue-implementation of $/f\setminus$. Indeed, the weld constructor only takes input of type

[9] One can argue that it is more general to consider continuously indexed functions: $\Pi(z : Z).X\ z \to Y\ z$. However, those interfere with the ¶ modality. In Section 4.2, we do use continuously indexed functions.

[10] For initial algebras, we have been able to refine the proofs so as to obtain the same results, but with continuous instead of pointwise dependencies. There are some complications in dualizing the technique we used for final co-algebras.

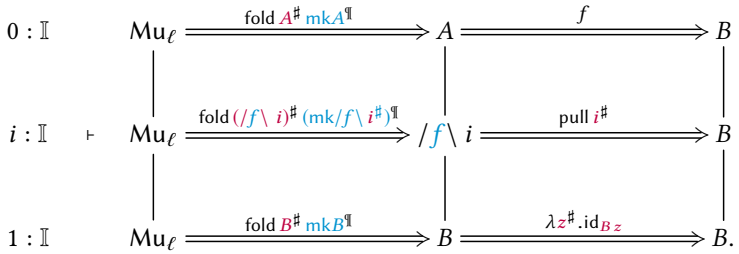

Fig. 5. Proving naturality of fold. The middle row depends on i and reduces to the top/bottom row for $i \doteq 0$ and $i \doteq 1$ respectively.

A, and applying $\mathsf{ind}_{\mathsf{Weld}}$ under the functor F would not allow us to escape F. However, we do have $\mathsf{mk}B \circledcirc \vec{F}\ \mathsf{pull}\ i^\sharp{}^\P : F(/f\setminus i) \Rightarrow B$, which is enough for glue. See the Agda proofs for details. □

We can prove, up to universe level issues, that folding to Mu is the identity:

Lemma 4.2. *We have* $\mathsf{loweringlemma}^\P : \downarrow =_{\mathsf{Mu}_{\ell+1} \Rightarrow \mathsf{Mu}_\ell} \mathsf{fold}_{\ell+1}\ \mathsf{Mu}_\ell{}^\sharp\ \mathsf{mkMu}_\ell{}^\P$.

Proof. By function extensionality, it is sufficient to prove the equation only when postcomposed with a general $\mathsf{fold}_\ell\ X^\sharp\ \mathsf{mk}X^\P : \mathsf{Mu}_\ell \Rightarrow X$. It is clear that $\mathsf{fold}_\ell\ X^\sharp\ \mathsf{mk}X^\P \circledcirc \downarrow$ is equal to $\mathsf{fold}_{\ell+1}\ X^\sharp\ \mathsf{mk}X^\P$, so we can apply the previous theorem to $\mathsf{fold}_\ell\ X^\sharp\ \mathsf{mk}X^\P$. □

Combining these lemmas, we can conclude up to universe level issues that $\mathsf{fold}\ B^\sharp\ \mathsf{mk}B^\P$ is the only algebra morphism $\mathsf{Mu} \Rightarrow B$:

Theorem 4.3. *Assume we have* $B^\sharp : Z \to \mathcal{U}_{\ell+1}$, *an algebra structure* $\mathsf{mk}B^\P : F\,B \Rightarrow B$ *and an algebra morphism* $f^\P : \mathsf{Mu}_\ell \Rightarrow B$ *(i.e.* $f \circledcirc \mathsf{mkMu}_\ell \equiv \mathsf{mk}B \circledcirc \vec{F} f$*). Then* $\mathsf{initiality}^\P : f \circledcirc \downarrow =_{\mathsf{Mu}_{\ell+1} \Rightarrow B} \mathsf{fold}_{\ell+1}\ B^\sharp\ \mathsf{mk}B^\P$. □

This theorem is perhaps less interesting than the proof technique. For example, the type of Church lists $\forall(X : \mathcal{U}_\ell).\P X \to \P(B \to X \to X) \to X$ and the Church-encoded unit type $\forall(X : \mathcal{U}_\ell).\P X \to X$ mentioned in Section 2 do not precisely fit the constraints of the theorem. However, they do up to isomorphism if there is a unit type and a coproduct type former, and the proof technique even applies if those types do not exist. What we have demonstrated is in fact a more general thesis, namely that Church encodings of indexed recursive types can be shown internally to work.

4.1.2 Final Co-algebras. Define, using some syntactic sugar for (elimination of) triples:

$\sharp \setminus \Gamma \vdash \mathsf{Nu}_\ell :\equiv \lambda(z : Z).\exists(X : Z \to \mathcal{U}_\ell).\P(X \Rightarrow F\,X) \times X\ z : Z \to \mathcal{U}_{\ell+1}$,

$\Gamma \vdash \mathsf{unfold} :\equiv \lambda X^\sharp.\lambda \mathsf{match}X^\P.\lambda z^\sharp.\lambda x.(X^\sharp, \mathsf{match}X^\P, x) : \forall(X : \mathcal{U}_\ell).\P(X \Rightarrow F\,X) \to (X \Rightarrow \mathsf{Nu}_\ell)$,

$\Gamma \vdash \mathsf{matchNu}_\ell :\equiv \lambda z^\sharp.\lambda(X^\sharp, \mathsf{match}X^\P, x).(\vec{F}\,(\mathsf{unfold}\ X^\sharp\ \mathsf{match}X^\P) \circledcirc \mathsf{match}X)\ z^\sharp\ x : \mathsf{Nu}_\ell \Rightarrow F\,\mathsf{Nu}_\ell$,

$\Gamma \vdash \uparrow :\equiv \lambda z^\sharp.\lambda(X^\sharp, \mathsf{match}X^\P, x).(X^\sharp, \mathsf{match}X^\P, x) : \mathsf{Nu}_\ell \Rightarrow \mathsf{Nu}_{\ell+1}$.

Then similar reasoning as before, but with $/\sqcup\setminus$ implemented using Weld instead of Glue, shows:

Theorem 4.4. *Assume we have* $B^\sharp : Z \to \mathcal{U}_{\ell+1}$, *a co-algebra structure* $\mathsf{match}B^\P : B \Rightarrow F\,B$ *and a co-algebra morphism* $f^\P : B \Rightarrow \mathsf{Nu}_\ell$ *(i.e.* $\mathsf{matchNu}_\ell \circledcirc f \equiv \vec{F} f \circledcirc \mathsf{match}B$*). Then* $\mathsf{finality}^\P : \uparrow \circledcirc f =_{B \Rightarrow \mathsf{Nu}_{\ell+1}} \mathsf{unfold}\ B^\sharp\ \mathsf{match}B^\P$.

4.2 Sized Types

As an example of the use of parametric quantification over values, in this section we show how it can express irrelevance properties of definitions that use sized types. By indexing data-types with a bound to the height of their elements, sized types reduce both terminating recursion and productive

	universal			existential		
a	$\forall\_.T$	$\cong$	T	$\exists\_.T$	$\cong$	T
b	$\forall n.A\,n$	$\cong$	$\forall n.\forall m < n.A\,m$	$\exists n.A\,n$	$\cong$	$\exists n.\exists m < n.A\,m$
c	$\Pi(x:T).\forall n.B\,x\,n$	$\cong$	$\forall n.\Pi(x:T).B\,x\,n$	$\Sigma(x:T).\exists n.B\,x\,n$	$\cong$	$\exists n.\Sigma(x:T).B\,x\,n$
d	$(\forall n.A\,n)\times(\forall n.B\,n)$	$\cong$	$\forall n.(A\,n\times B\,n)$	$(\exists n.A\,n)+(\exists n.B\,n)$	$\cong$	$\exists n.(A\,n+B\,n)$
e	$\forall n.\Sigma(x:T).B\,x\,n$	$\cong$	$\Sigma(x:T).\forall n.B\,x\,n$			
f	$\forall n.(A\,n+B\,n)$	$\cong$	$(\forall n.A\,n)+(\forall n.B\,n)$			
g				$(\exists n.A\,n)\times(\exists n.B\,n)$	$\cong$	$\exists n.(A^<\,n\times B^<\,n)$
				where $C^<\,n$	$:\equiv$	$\exists m < n.C\,m$

Fig. 6. Isomorphisms for working with Size quantifiers. Isomorphisms between dually quantified types are placed side by side.

co-recursion to well-founded induction on sizes [Abel and Pientka 2013]. This allows to enforce the totality of programs through typing and allows more recursion patterns to be recognized as well-founded when compared to more syntactic checkers.

However, while these sizes are just natural (or in some applications ordinal) numbers, they require a different treatment than other natural numbers, as a bound on the length of a list should not be considered computational content of that list. Consider the following sized list type and its constructors:

$$\begin{array}{lcl} \mathsf{SList}\,A & : & \mathsf{Size} \to \mathcal{U} \\ \mathsf{nil} & : & \Pi(n:\mathsf{Size}).\Pi(m:\mathsf{Size}).(m<n) \to \mathsf{SList}\,A\,n \\ \mathsf{cons} & : & \Pi(n:\mathsf{Size}).\Pi(m:\mathsf{Size}).(m<n) \to A \to \mathsf{SList}\,A\,m \to \mathsf{SList}\,A\,n \end{array}$$

If we treat sizes as computational content, then we obtain two different empty lists of size 2:

$$\mathsf{n}_0 :\equiv \mathsf{nil}\,(\uparrow\uparrow 0_\mathsf{S})\,0_\mathsf{S}\,(\ldots), \qquad \mathsf{n}_1 :\equiv \mathsf{nil}\,(\uparrow\uparrow 0_\mathsf{S})\,(\uparrow 0_\mathsf{S})\,(\ldots).$$

In ParamDTT we can fix this problem by making the constructors parametric in their size arguments. Then we can use the codiscrete bridge structure of Size, to build a path $\lambda i^\sharp.\mathsf{nil}\,(\uparrow\uparrow 0_\mathsf{S})^\sharp\,(\mathsf{fill}(i \doteq 0\,?\,0_\mathsf{S} \mid i \doteq 1\,?\,\uparrow 0_\mathsf{S}))^\sharp\,(\ldots)$ from n_0 to n_1. To show that this approach to sized types is valid we will build initial algebras and final co-algebras, and their sized version, for sufficiently well-behaved functors. Our approach is immature for practical use as it relies on many propositional equalities, some of which are non-computing axioms. Throughout the section, we will omit the domain of quantifiers over Size, and we will moreover write $\exists m < n.T$ for $\exists m.(m<n)\times T\,m$ and $\forall m < n.T$ for $\forall m.(m<n) \to T$.

4.2.1 Isomorphisms for Size Quantifiers. To do so we will make use of the isomorphisms in Fig. 6 which describe how parametric quantification over Size commutes with other connectives. We highlight the central ideas here, see the full Agda proofs for details. We get to remove quantifiers over constant types (a), because inserting different values of n into function applications $f\,n^\sharp$ or existential pairs $(n^\sharp, t)$ yields path-connected and hence equal values. The reasoning for (b) is similar. The isomorphisms (c) involve trivial swapping of arguments or components. In the special case where $T = \mathsf{Bool}$, we get (d) modulo a trivial isomorphism. In the isomorphism (e), the first component is constant by (a) and can be extracted from the quantifier. Specializing to Bool again, we arrive at (f). To the right of (e), we would like to write that $\exists n.\Pi(x:T).B\,x\,n$ were isomorphic to $\Pi(x:T).\exists n.B\,x\,n$. However, this would require an operator that joins T many sizes into a single size $n_\sqcup$, and moreover a way to transport from various types $B\,x\,n$ to the type $B\,x\,n_\sqcup$. This is possible when $T = \mathsf{Bool}$ by using the maximum operator ($\sqcup$), provided that $B\,x\,n$ is covariant in n. This covariance is clear if $B\,x$ is of the form $C^<$, in which case we can use (b) to extract C on the left. This yields (g).

Example 4.5. We build a fixpoint for the functor $\top + (A \times \llcorner)$, assuming that we already have the sized initial algebra $\mathsf{SList}\,A : \mathsf{Size} \to \mathcal{U}$ such that $\mathsf{SList}\,A\,n \cong \top + A \times (\exists m < n.\mathsf{SList}\,A\,m)$. We have

$$\exists n.\mathsf{SList}\,A\,n \cong \exists n.(\top + A \times (\exists m < n.\mathsf{SList}\,A\,m)) \cong_{\mathrm{d}} (\exists n.\top) + \exists n.(A \times (\exists m < n.\mathsf{SList}\,A\,m))$$
$$\cong_{\mathrm{a,c}} \top + A \times \exists n.\exists m < n.\mathsf{SList}\,A\,m \cong_{\mathrm{b}} \top + A \times \exists n.\mathsf{SList}\,A\,n$$

4.2.2 Sized Initial Algebras. In this section we fix a universe level ℓ, a context Γ and a type $\sharp \setminus \Gamma \vdash Z : \mathcal{U}_\ell$ and take an Z-indexed functor to be a pair of terms

$$\sharp \setminus \Gamma \vdash F : (Z \to \mathcal{U}_\ell) \to (Z \to \mathcal{U}_\ell)$$
$$\Gamma \vdash \vec{F} : \forall(A, B : Z \to \mathcal{U}_\ell).\,(\Pi(z : Z).\,A\,z \to B\,z) \to \Pi(z : Z).\,F\,A\,z \to F\,B\,z$$

such that $\vec{F}$ satisfies the identity and composition laws propositionally. We omit parametric type arguments such as the first two arguments to $\vec{F}$. Write $(A \Rightarrow B)$ for $(\forall n.\Pi(z : Z).\,A\,n\,z \to B\,n\,z)$. We lift the functor F to a functor $\hat{F}$ on the category of sized indexed types as follows (using some syntactic sugar for eliminating pairs):

$$\sharp \setminus \Gamma \vdash \hat{F} :\equiv \lambda A.\lambda n.F(\lambda z.\exists m < n.A\,m\,z) : (\mathsf{Size} \to Z \to \mathcal{U}_\ell) \to (\mathsf{Size} \to Z \to \mathcal{U}_\ell)$$
$$\Gamma \vdash \vec{\hat{F}} :\equiv \lambda A^\sharp.\lambda B^\sharp.\lambda f.\lambda n^\sharp.\vec{F}(\lambda z.\lambda(m^\sharp, e, a).(m^\sharp, e, f\ m^\sharp\ z\ a)) : \forall A, B.\,(A \Rightarrow B) \to \hat{F}\,A \Rightarrow \hat{F}\,B$$

which can be shown to also respect the functor laws. We then define the sized initial algebra $\widehat{\mathsf{Mu}}$ as the unique fixpoint of $\hat{F}$, using well-founded induction on Size:

$$\sharp \setminus \Gamma \vdash \widehat{\mathsf{Mu}} :\equiv \mathsf{fix}\,(\lambda n.\lambda \mathsf{Mu}'.F(\lambda z.\exists(m : \mathsf{Size}).\Sigma(e : m < n).\mathsf{Mu}'\,m\,e\,z)) : \mathsf{Size} \to Z \to \mathcal{U}_\ell.$$

From $\mathsf{fix}_=$ we obtain the propositional equality $\widehat{\mathsf{Mu}} = \hat{F}\,\widehat{\mathsf{Mu}}$ which gives us the (invertible) algebra structure $\Gamma \vdash \mathsf{mk}\widehat{\mathsf{Mu}} : \hat{F}\,\widehat{\mathsf{Mu}} \Rightarrow \widehat{\mathsf{Mu}}$, while the initiality is given by the following $\widehat{\mathsf{fold}}$ function:

$$\Gamma \vdash \widehat{\mathsf{fold}} : \forall A.(\hat{F}\,A \Rightarrow A) \to (\widehat{\mathsf{Mu}} \Rightarrow A), \qquad \widehat{\mathsf{fold}}\,A^\sharp\,\mathsf{mk}A :\equiv$$
$$\mathsf{fix}^\sharp\Big(\lambda n^\sharp.\lambda \mathsf{fold}'.\lambda z.\lambda mu.\mathsf{mk}A\,n^\sharp\,z\,\Big(\vec{F}\,(\lambda z'.\lambda(m^\sharp, e, mu').(m^\sharp, e, \mathsf{fold}'\,m^\sharp\,e\,z'\,mu'))\,z\,(\mathsf{mk}\widehat{\mathsf{Mu}}^{-1}\,mu)\Big)\Big).$$

Uniqueness is obtained from well-founded induction on sizes and the functor laws for F.

4.2.3 Initial Algebras. Finally we define the initial algebra as $\mathsf{Mu} :\equiv \lambda z.\exists n.\widehat{\mathsf{Mu}}\,n\,z$. However, in order to define the algebra map $\mathsf{mkMu} : \Pi(z : Z).F\,\mathsf{Mu}\,z \to \mathsf{Mu}\,z$ we need an extra property of F: we say that F *weakly commutes with* $\exists(n : \mathsf{Size})$ if the canonical map of type $\Pi(z : Z).(\exists n.\hat{F}\,A\,n\,z) \to F(\lambda z'.\exists n.A\,n\,z')\,z$ is an isomorphism for every A. All finitely branching indexed containers [Altenkirch et al. 2006], i.e. functors of the form $F\,X\,z \equiv \Sigma(c : C\,z).(\Pi(b : B\,z\,c).X\,(r\,z\,c\,b))$ for some $r : \Pi(z : Z).\Pi(c : C\,z).B\,z\,c \to Z$ with $B\,z\,c$ finite, satisfy this property. If F weakly commutes with $\exists(n : \mathsf{Size})$, then $F\,\mathsf{Mu} \cong \mathsf{Mu}$, which gives us the algebra map as anticipated. The unique algebra morphism fold is defined using $\widehat{\mathsf{fold}}$ and its uniqueness follows from the fact that there is an isomorphism between algebra morphisms from $(\mathsf{Mu}, \mathsf{mkMu})$ and algebra morphisms from $(\widehat{\mathsf{Mu}}, \mathsf{mk}\widehat{\mathsf{Mu}})$.

4.2.4 Final Co-algebras. The entire construction can be dualized to obtain final co-algebras, here we give only the main definitions:

$$\hat{F} :\equiv \lambda A.\lambda n.F\,(\lambda z'.\forall m < n.A\,m\,z),$$
$$\widehat{\mathsf{Nu}} :\equiv \mathsf{fix}\,(\lambda n.\lambda \mathsf{Nu}'.F(\lambda z'.\forall(m : \mathsf{Size}).\Pi(e : m < n).\mathsf{Nu}'\,m\,e\,y),$$
$$\mathsf{Nu} :\equiv \lambda z.\forall n.\widehat{\mathsf{Nu}}\,n\,z.$$

Then Nu is the final co-algebra if F weakly commutes with $\forall(n : \mathsf{Size})$, i.e. if the canonical map of type $\Pi(z : Z).F(\lambda z'.\forall(n : \mathsf{Size}).A\,n\,z')\,z \to \forall n.\hat{F}\,A\,n\,z$ is an isomorphism for every A. This property is satisfied by all containers.

5 SOUNDNESS: AN OVERVIEW OF THE PRESHEAF MODEL

In this section we give a high-level overview of the presheaf model that we constructed to prove the soundness of ParamDTT. This section can be safely skipped by readers who are primarily interested in the type theory side of the story. No prior knowledge of presheaves is assumed.

We start with a brief review of the set model of MLTT and show why it cannot model contexts containing continuous or parametric interval variables and hence also fails to model bridges and paths. We then explain how the general presheaf model of dependent type theory as treated by Hofmann [1997], can be seen as a method for manually adding non-setlike 'primitive' contexts to the set model. We point out existing presheaf models that support some form of interval: the reflexive graph model [Atkey et al. 2014] and cubical models [Bernardy et al. 2015; Bezem et al. 2014; Cohen et al. 2016]. Finally, we construct a specific presheaf model by manually adding *bridge/path cubes* — contexts consisting solely of continuous and parametric interval variables — and give a high-level overview of the semantics of ParamDTT in that model.

For the sake of the exposition, we ignore universe level issues. Most of the time, we do not use an interpretation function but instead give judgements a direct meaning in the model. A more formal treatment of the model is found in [Nuyts 2017].

5.1 The Set Model of Dependent Type Theory

In this model, a closed type is a set, whose elements are precisely its semantic closed terms. So $\vdash T$ type means that T is a set, and $\vdash t : T$ means that $t \in T$. A context Γ is modelled as a set, whose elements are vectors of semantic closed terms that give meaning to all of the variables in Γ:

$$() = \{()\}, \qquad (\Gamma, t : T) = \{(\gamma, t) \mid \gamma \in \Gamma, t \in T[\gamma]\}.$$

So $\Gamma \vdash \mathrm{Ctx}$ means that Γ is a set. From the notation $T[\gamma]$, it is already clear that an open type $\Gamma \vdash T$ type is a function that assigns to every $\gamma \in \Gamma$ a set $T[\gamma]$. A term $\Gamma \vdash t : T$ is a function that maps $\gamma \in \Gamma$ to $t[\gamma] \in T[\gamma]$. Note that elements of a context Γ correspond precisely to semantic substitutions $\gamma : () \to \Gamma$ from the empty context. More generally, a substitution $\sigma : \Delta \to \Gamma$ is a function from Δ to Γ. Substitutions of types and terms are given by $T[\sigma][\delta] = T[\sigma(\delta)]$ and $t[\sigma][\delta] = t[\sigma(\delta)]$. Definitional equality is modelled as equality of mathematical objects.

So to wrap up, we have picked a category (namely the category of sets) whose objects model contexts and whose morphisms model substitutions. We have defined for every object Γ what it means to be a type $\Gamma \vdash T$ type and how we define $T[\sigma]$, and finally we have done the same for terms. We have explained how to extend a context with a type. A few additional features will give this setup the structure of a *category with families* [CwF, Dybjer 1996], and we can prove soundness of MLTT with any extensions of interest by giving every inference rule a meaning in the model in such a way that some contradictory type has no semantic terms.

The set model is insufficient to model ParamDTT. Indeed, how do we interpret the closed type $\mathbb{I}$ or the contexts $(i : \mathbb{I})$ and $(i^\sharp : \mathbb{I})$ as a set? The terms $\vdash 0, 1 : \mathbb{I}$ show that these have at least two elements, and $(0 \doteq 1) \equiv \bot$ shows that they are distinct. But there is nothing that allows us to relate them, which is necessary if we want to model the path degeneracy axiom. The interval is, in short, not a set.

5.2 Presheaf Models

The problem of contexts that cannot be modelled by sets, can be overcome by adding them explicitly to the model. This is what presheaf models do. The first step in constructing a presheaf model, is to identify a set of *primitive contexts* (also called levels, shapes, worlds or simply objects) that 'generate' the problem. Secondly, for every two primitive contexts V, W, we must explicitly provide the set of all substitutions $\varphi : V \xrightarrow{\mathrm{P}} W$ that we want to exist between V and W; we will call these

primitive substitutions (more commonly, they are called face or restriction maps). Together, these must form a category $\mathcal{W}$ (the base category).

Now, we define contexts Γ not in relation to the empty context by providing the set of all substitutions $() \to \Gamma$, but in relation to all primitive contexts. So in order to define Γ, we need to provide for every primitive context W the set of all *defining substitutions* $\gamma : W \xrightarrow{\mathrm{D}} \Gamma$ *and* for every primitive substitution $\varphi : V \xrightarrow{\mathrm{P}} W$ a composition operator $\llcorner \circ \varphi : (W \xrightarrow{\mathrm{D}} \Gamma) \to (V \xrightarrow{\mathrm{D}} \Gamma)$. In other words, a context Γ is a functor $(\llcorner \xrightarrow{\mathrm{D}} \Gamma) : \mathcal{W}^{\mathrm{op}} \to \mathrm{Set}$ (also called a presheaf over $\mathcal{W}$). A substitution $\sigma : \Delta \to \Gamma$ (also: presheaf map/morphism) is then defined by saying how it composes with the defining substitutions of Δ, i.e. we have to give an operator $\sigma\circ : (W \to \Delta) \to (W \to \Gamma)$ that is natural in W. In other words, substitutions are natural transformations, and our category of contexts is the functor space $\mathrm{Set}^{\mathcal{W}^{\mathrm{op}}}$, also called $\widehat{\mathcal{W}}$.

Note that we can view a primitive context W as a context by setting $(V \xrightarrow{\mathrm{D}} W) := (V \xrightarrow{\mathrm{P}} W)$, and a primitive substitution $\varphi : V \xrightarrow{\mathrm{P}} W$ as a substitution by defining $\varphi \circ \llcorner$ as composition with φ in $\mathcal{W}$. This defines a functor $\mathbf{y} : \mathcal{W} \to \widehat{\mathcal{W}}$ called the *Yoneda-embedding*, which is fully faithful, meaning that $(V \xrightarrow{\mathrm{P}} W) \cong (V \to W)$, i.e. every substitution between primitive contexts is in fact a primitive substitution. Similarly, we have $(W \xrightarrow{\mathrm{D}} \Gamma) \cong (W \to \Gamma)$.

So $\Gamma \vdash \mathrm{Ctx}$ means that Γ is a presheaf, and a substitution $\sigma : \Delta \to \Gamma$ is a presheaf map. We define a type $\Gamma \vdash T$ type by giving for every W and every $\gamma : W \xrightarrow{\mathrm{D}} \Gamma$ its set of *defining terms* $W \vdash^{\mathrm{D}} t : T[\gamma]$, plus for every $\varphi : V \xrightarrow{\mathrm{P}} W$ a substitution operator $\llcorner[\varphi]$ that takes $W \vdash^{\mathrm{D}} t : T[\gamma]$ to $V \vdash^{\mathrm{D}} t[\varphi] : T[\gamma \circ \varphi]$. A term $\Gamma \vdash t : T$ is then a thing that maps defining substitutions $\gamma : W \xrightarrow{\mathrm{D}} \Gamma$ to defining terms $W \vdash^{\mathrm{D}} t[\gamma] : T[\gamma]$ in such a way that $t[\gamma][\varphi] = t[\gamma \circ \varphi]$. Again, one can show that defining terms $W \vdash^{\mathrm{D}} t : T[\mathrm{id}]$ are in bijection to terms $W \vdash t : T$. We extend contexts as follows:

$$W \xrightarrow{\mathrm{D}} (\Gamma, t : T) = \left\{ (\gamma, t) \,\middle|\, \gamma : W \xrightarrow{\mathrm{D}} \Gamma \text{ and } W \vdash^{\mathrm{D}} t : T[\gamma] \right\}, \qquad (\gamma, t) \circ \varphi = (\gamma \circ \varphi, t[\varphi]).$$

One can show that every presheaf category constitutes a CwF [Hofmann 1997] that supports universes if the metatheory does [Hofmann and Streicher 1997].

The category of reflexive graphs. In order to model parametricity for a dependent type system without modalities, we can pick as base category the category RG with just two primitive contexts $()$ and $(i : \mathbb{I})$, and where primitive substitutions are (non-freely) generated by $(0/i), (1/i) : () \to (i : \mathbb{I})$ and $() : (i : \mathbb{I}) \to ()$. A presheaf Γ over RG is then a reflexive graph, with a set of nodes $() \xrightarrow{\mathrm{D}} \Gamma$ and a set of edges $(i : \mathbb{I}) \xrightarrow{\mathrm{D}} \Gamma$. Precomposition with $(0/i)$ and $(1/i)$ determines the source and target of an edge, and precomposition with $()$ determines the reflexive edge at a node. This corresponds to the reflexive graph model as treated by Atkey et al. [2014], although they use a non-standard universe to model identity extension (IEL). This model does not support iterated parametricity; hence it does not support internal parametricity operators which, in absence of modalities, can be self-applied. The non-standard universe supports IEL, but not in combination with proof-relevant relations.

The category of cubical sets. In order to support iterated parametricity or to have identity extension in the presence of proof-relevant relations, we need a way to express relatedness of relations. In other words, we need a notion of edges between edges (squares), edges between squares (cubes), etc. Although contexts like $(i : \mathbb{I}, j : \mathbb{I})$ exist in the category of reflexive graphs, these are still just graphs that do not contain data to express that the faces $(0/j), (1/j) : (i : \mathbb{I}) \xrightarrow{\mathrm{D}} (i : \mathbb{I}, j : \mathbb{I})$ are related.

In the cubical model, we add cubes explicitly. As base category, we pick the *cube category* Cube with primitive contexts $(\vec{i} : \mathbb{I}^n)$ for $n \in \mathbb{N}$ and primitive substitutions $(\vec{i} : \mathbb{I}^m) \xrightarrow{\mathrm{P}} (\vec{i} : \mathbb{I}^n)$ that substitute every variable of the codomain with either 0, 1 or a variable from the domain. A presheaf Γ over Cube is then a so-called cubical set, consisting of, for every number n, a set of n-dimensional

cubes $(\vec{i} : \mathbb{I}^n) \xrightarrow{\mathrm{D}} \Gamma$ and a composition operator with primitive substitutions (face maps) that allow us to extract faces, diagonals and reflexive cubes from a given cube. This model is close to the (binary version of) the one used by Bernardy et al. [2015]. Cubical type theory [Bezem et al. 2014; Cohen et al. 2016] uses a similar model to model univalence, but they have additional operators $\vee$, $\wedge$ and $\neg$ on the interval, resulting in a base category with the same objects but more primitive substitutions.

The category of bridge/path cubical sets. We need to make just one modification to the base category to obtain a presheaf category of our system: we annotate the interval variables in primitive contexts with either **id** or $\sharp$ (as $(i^{\P} : \mathbb{I})$ really just contains two points 0 and 1). So as base category, we pick the category BPCube whose objects are *bridge/path cubes* $(\vec{j} : \mathbb{I}^m, \vec{i}^{\sharp} : \mathbb{I}^n)$, where $m, n \in \mathbb{N}$. Primitive substitutions $\varphi : V \to W$ substitute every continuous 'bridge' variable from W with 0, 1 or a bridge variable from V, and every parametric 'path' variable from W with 0, 1 or any variable from V (Fig. 7). A presheaf Γ is then a bridge/path cubical set that contains for any $m, n \in \mathbb{N}$ a set of cubes with m bridge dimensions and n path dimensions. The defining composition operator of Γ allows us to extract faces, diagonals and reflexive cubes from a given cube, as well as to weaken path dimensions to bridge dimensions.

The standard presheaf model over BPCube supports iterated parametricity as it is essentially just the cubical set model extended with an arbitrary distinction between bridge and path dimensions. However, on top of this model we will build a machinery of modalities that adds the path degeneracy axiom, but loses full internal iterated parametricity. We still need the cubical structure in order to support the path degeneracy axiom in the presence of proof-relevant relations.

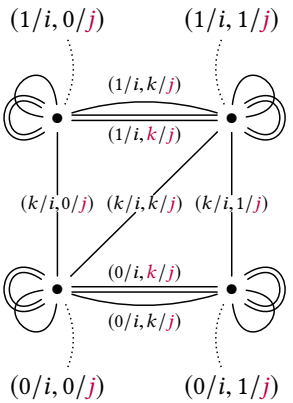

Fig. 7. All points, bridges (single line) and paths (double line) of the bridge/path cube $(i : \mathbb{I}, j^{\sharp} : \mathbb{I})$. The labels are the corresponding primitive substitutions with domains $()$, $(k : \mathbb{I})$ and $(k^{\sharp} : \mathbb{I})$ respectively.

5.3 The Cohesive Structure of $\widehat{\text{BPCube}}$

A bridge/path cubical set $\Gamma \in \widehat{\text{BPCube}}$ can be seen as an ordinary cubical set (namely the cubical set of its bridges) equipped with a notion of *cohesion* in the sense of Licata and Shulman [2016] expressed by its paths. This is formalized by a *forgetful* functor $\sqcup : \widehat{\text{BPCube}} \to \widehat{\text{Cube}}$ that forgets paths. This functor can be shown to be a morphism of CwFs, i.e. it extends to types and terms in a well-behaved manner. It is part of a chain of at least five adjoint functors (Fig. 8):

$$\sqcap \dashv \triangle \dashv \sqcup \dashv \nabla \dashv \boxminus, \qquad \sqcap, \sqcup, \boxminus : \widehat{\text{BPCube}} \to \widehat{\text{Cube}}, \qquad \triangle, \nabla : \widehat{\text{Cube}} \to \widehat{\text{BPCube}}$$

of which only $\sqcap$ is *not* a morphism of CwFs. The *discrete* functor $\triangle$ takes a cubical set with only bridges and equips it with a path relation defined as the equality relation — the strictest path relation possible. The *codiscrete* functor ∇ on the other hand defines the path relation as the bridge relation — the most liberal possibility. The functor $\boxminus$ is a forgetful functor at another level: it maps a bridge/path cubical set to its cubical set of paths and forgets the bridge structure. From this perspective, ∇ takes a cubical set with only paths and equips it with a bridge relation defined as the path relation — the strictest possibility. Finally, the functor $\sqcap$ identifies all path-connected objects, producing a cubical set with only a bridge relation. This functor is not a morphism of CwFs as it would not respect substitution.

Composing each adjoint pair to a (co)monad $\widehat{\text{BPCube}} \to \widehat{\text{BPCube}}$, we get a chain of four adjoint endofunctors on $\widehat{\text{BPCube}}$,

$$\int \dashv \flat \dashv \sharp \dashv \P : \widehat{\mathrm{BPCube}} \to \widehat{\mathrm{BPCube}}.$$

The *shape* monad $\int = \triangle\sqcap$, again the only one that is not a morphism of CwFs, identifies path-connected objects and then reintroduces a discrete path relation. The *flat* co-monad $\flat = \triangle\sqcup$ redefines the path relation discretely as the equality relation. The *sharp* monad $\sharp = \triangledown\sqcup$ redefines the path relation codiscretely as the bridge relation, and the *pointwise* co-monad $\P = \triangledown\boxminus$ redefines the bridge relation as the path relation. Note that a presheaf map $\sharp\Delta \to \Gamma$ maps bridges in Δ to paths in Γ, as does a parametric function, and that a presheaf map $\P\Delta \to \Gamma$ maps paths in Δ to paths in Γ, but does not act on bridges in Δ, just like a pointwise function. The functors $\P$, Id and $\sharp$ satisfy the same composition rules that we have for modalities, and we have the co-unit $\vartheta : \P \to \mathrm{Id}$ and the unit $\iota : \mathrm{Id} \to \sharp$ to interpret the order relation on modalities.

5.4 Meaning of Judgements

In this section, we write internal judgements between interpretation brackets $[\![\ldots]\!]$, in order to tell them apart from mathematical statements about the standard presheaf model from Section 5.2 which we also write in judgement-style.

Contexts. As usual, $[\![\Gamma \vdash \mathrm{Ctx}]\!]$ means that $[\![\Gamma]\!]$ is a presheaf over BPCube, i.e. $[\![\Gamma]\!] \vdash \mathrm{Ctx}$.

Types. Unusually, $[\![\Gamma \vdash T\ \mathrm{type}]\!]$ means that $[\![T]\!]$ is a type over the presheaf $\sharp[\![\Gamma]\!]$, i.e. $\sharp[\![\Gamma]\!] \vdash [\![T]\!]\ \mathrm{type}$, that is moreover *discrete*. The fact that it lives in $\sharp[\![\Gamma]\!]$ means that when there is a bridge in $[\![\Gamma]\!]$, then $[\![T]\!]$ provides a notion of paths over it. This is necessary if we want to add $x^{\P} : T$ to the context, because a bridge in $\P[\![T]\!]$ is a path in $[\![T]\!]$. Of course we have $[\![\Gamma]\!] \vdash [\![T]\!][\iota]\ \mathrm{type}$. We say that a type $\Delta \vdash A\ \mathrm{type}$ is *discrete* if every path in A that lives above a constant path in Δ, is also constant. More formally, if we have a primitive context $(W, k^{\sharp} : \mathbb{I})$ and a defining substitution $\delta : (W, k^{\sharp} : \mathbb{I}) \xrightarrow{\mathrm{D}} \Delta$ that is constant in k, i.e. it factors as $(W, k^{\sharp} : \mathbb{I}) \xrightarrow{()} W \xrightarrow{\delta'} \Delta$, then every defining term $(W, k^{\sharp} : \mathbb{I}) \vdash^{\mathrm{D}} a : A[\delta]$ is also constant in k, i.e. it is a weakening of some $W \vdash^{\mathrm{D}} a' : A[\delta']$. This is essentially the statement that non-dependent paths in A are constant, i.e. that A satisfies the path degeneracy axiom. So $[\![\Gamma \vdash T\ \mathrm{type}]\!]$ means that $[\![T]\!]$ is a type in context $\sharp[\![\Gamma]\!]$ that satisfies the path degeneracy axiom.

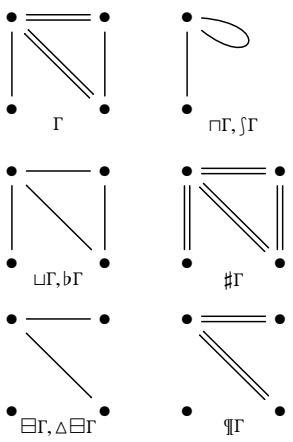

Fig. 8. Example of a bridge/path cubical set Γ and its images for various functors. A single line denotes an edge (for edge cubical sets) or a bridge, a double line is a path. Implied content, such as constant lines and the bridge under a path, are not shown. Hence, applying $\triangle$ preserves the drawing.

Modal context extension. If $[\![\Gamma \vdash T\ \mathrm{type}]\!]$ holds, then we need to be able to extend $[\![\Gamma]\!]$ with variables of type $[\![T]\!]$ in any modality μ. We already know that μ corresponds to an endomorphism of CwFs on $\widehat{\mathrm{BPCube}}$. The laws of a morphism of CwFs allow us to apply μ to both sides of a typing judgement, obtaining $\mu\sharp[\![\Gamma]\!] \vdash \mu[\![T]\!]\ \mathrm{type}$. Now $\mu\sharp = \sharp$ by the composition table of modalities so that we have $[\![\Gamma]\!] \vdash (\mu[\![T]\!])[\iota]\ \mathrm{type}$ (note how this would not work if $[\![T]\!]$ lived in $[\![\Gamma]\!]$ and $\mu = \P$). We now interpret $[\![\Gamma, x^{\mu} : T]\!]$ as $([\![\Gamma]\!], x : (\mu[\![T]\!])[\iota])$. Then one can show that $[\![\sharp \setminus \Gamma]\!] \cong \flat[\![\Gamma]\!]$ and $[\![\P \setminus \Gamma]\!] = \sharp[\![\Gamma]\!]$, so that left dividing a context by a modality corresponds to applying the modality's left adjoint.

Terms. A term $[\![\Gamma \vdash t : T]\!]$ is now interpreted as $[\![\Gamma]\!] \vdash [\![t]\!] : [\![T]\!][\iota]$.

5.5 Some Remarkable Interpretations of Types

The universe. For simplicity, we ignore universe levels — a more formal exposition can be found in [Nuyts 2017]. The standard presheaf universe $\mathcal{U}^{\mathrm{Psh}}$ as described by Hofmann and Streicher

[1997] has as its defining terms $W \vdash^{\mathrm{D}} A : \mathcal{U}^{\mathrm{Psh}}$ the types $W \vdash A\,\mathrm{type}$ over the Yoneda-embedded primitive context W. This universe is problematic for our purposes in three ways. First, the types it contains are not all discrete. Secondly, $\mathcal{U}^{\mathrm{Psh}}$ is not discrete itself and thus cannot satisfy the path degeneracy axiom. Thirdly, an encoded type $\Gamma \vdash A : \mathcal{U}$ should admit terms $\sharp\Gamma \vdash a : A$; indeed, the $\forall$-quantifier admits parametric functions in non-parametric types. The universe $\mathcal{U}^{\mathrm{Psh}}$ does not satisfy this requirement.

The first issue is easiest to resolve: we define a non-discrete universe of discrete types $\mathcal{U}^{\mathrm{NDD}}$ by taking as defining terms $W \vdash^{\mathrm{D}} A : \mathcal{U}^{\mathrm{NDD}}$ precisely the *discrete* types $W \vdash A\,\mathrm{type}$.

The other two issues can be solved together. A closed type such as the universe that we want to model, is discrete if and only if its path relation is the equality relation. Meanwhile, a bridge $(i : \mathbb{I}) \vdash^{\mathrm{D}} A : \mathcal{U}$ should admit paths $(i^\sharp : \mathbb{I}) \vdash^{\mathrm{D}} a : A$. Both of these matters are resolved if we take as defining terms $(\vec{j} : \mathbb{I}^m, \vec{i}^\sharp : \mathbb{I}^n) \vdash^{\mathrm{D}} A : \mathcal{U}$ precisely the discrete types $(\vec{j}^\sharp : \mathbb{I}^m) \vdash A\,\mathrm{type}$, or equivalently terms $(\vec{j}^\sharp : \mathbb{I}^m) \vdash^{\mathrm{D}} A : \mathcal{U}^{\mathrm{NDD}}$. So the paths of $\mathcal{U}$ are constant and the bridges of $\mathcal{U}$ are the paths of $\mathcal{U}^{\mathrm{NDD}}$; this amounts to saying that $\mathcal{U} = \flat\P\mathcal{U}^{\mathrm{NDD}} = \triangle\boxminus\mathcal{U}^{\mathrm{NDD}}$. Indeed: $\P$ redefines the bridge relation as the path relation, and $\flat$ subsequently redefines the path relation as equality.

Now if we have $[\![\sharp \setminus \Gamma \vdash A : \mathcal{U}]\!]$, i.e. (up to isomorphism) $\flat[\![\Gamma]\!] \vdash [\![A]\!] : \flat\P\mathcal{U}^{\mathrm{NDD}}$, then applying $\sharp$ and using that $\sharp\flat = \sharp$ and $\sharp\P = \P$ yields a term $\sharp[\![\Gamma]\!] \vdash {}^\sharp[\![A]\!] : \P\mathcal{U}^{\mathrm{NDD}}$. Finally we can use $\vartheta : \P \to \mathrm{Id}$ to obtain $\sharp[\![\Gamma]\!] \vdash \vartheta({}^\sharp[\![A]\!]) : \mathcal{U}^{\mathrm{NDD}}$, which encodes a discrete type $\sharp[\![\Gamma]\!] \vdash \vartheta({}^\sharp[\![A]\!])\,\mathrm{type}$, i.e. an interpretation for $[\![\Gamma \vdash A\,\mathrm{type}]\!]$. We will write $\mathsf{El}\,T$ for $\vartheta({}^\sharp T)$.

An important challenge is now to establish that all type formers work on elements of the discrete universe $\mathcal{U} = \flat\P\mathcal{U}^{\mathrm{NDD}}$. The general approach is the following: We move the functor $\flat\P$ to the left of the turnstile ($\vdash$) in the form of its left adjoint $\sharp\int$. Then we apply existing type formers for $\mathcal{U}^{\mathrm{NDD}}$ in the context $\sharp\int\Gamma$ and finally we move the functor back to the right. Great care is needed where we have interactions between terms and types, i.e. when modelling dependent quantifiers, identity types and Glue/Weld types.

Identity types. The standard presheaf identity type has a single defining term $W \vdash^{\mathrm{D}} \star : (a =_A b)[\gamma]$ when $a[\gamma] = b[\gamma]$ and no defining terms for γ otherwise. In other words, it represents definitional, proof-irrelevant equality. Now suppose we have $[\![\Gamma \vdash T : \mathcal{U}]\!]$ and $[\![\Gamma \vdash s, t : T]\!]$. The former judgement can be unfolded to $\sharp\int[\![\Gamma]\!] \vdash T'\,\mathrm{type}$. Using the fact that T is discrete, we are allowed to disregard the $\int$ functor, so by applying $\sharp$ to $[\![s]\!]$ and $[\![t]\!]$, we obtain terms $\sharp\int[\![\Gamma]\!] \vdash s', t' : \sharp T'$. Then we can construct $\sharp\int[\![\Gamma]\!] \vdash s' =_{\sharp T'} t'\,\mathrm{type}$, which is discrete as it is a proposition, and finally we can go back to $[\![\mathcal{U}]\!]$. It is noteworthy that the interpretation of the identity type is not over T', but over $\sharp T'$. This also allows us to model parametricity of the reflexivity constructor, without damaging the power of the J-rule.

Function types. Assume we have $[\![\Gamma \vdash A : \mathcal{U}]\!]$ and $[\![\Gamma, x^{\sharp\setminus\mu} : A \vdash B : \mathcal{U}]\!]$, i.e. $[\![\Gamma]\!] \vdash [\![A]\!] : \flat\P\mathcal{U}^{\mathrm{NDD}}$ and $[\![\Gamma], x : ((\sharp \setminus \mu)\mathsf{El}\,A)[\iota] \vdash B : \flat\P\mathcal{U}^{\mathrm{NDD}}$. Again, we push $\flat\P$ to the left in the form of $\sharp\int$. Remember that $\int$ is a monad that we can map *into*; this allows us to extract the variable x out of it. Meanwhile, we have a natural transformation $\mu \to \sharp \circ (\sharp \setminus \mu)$. All of this allows us to derive $\sharp\int[\![\Gamma]\!] \vdash A' : \mathcal{U}^{\mathrm{NDD}}$ and $\sharp\int[\![\Gamma]\!], x : \mu A' \vdash B' : \mathcal{U}^{\mathrm{NDD}}$, whence $\sharp\int[\![\Gamma]\!] \vdash \Pi(x : \mu A').B' : \mathcal{U}^{\mathrm{NDD}}$ and finally $[\![\Gamma]\!] \vdash [\![\Pi^\mu(x : A).B]\!] : \flat\P\mathcal{U}^{\mathrm{NDD}}$.

Pair types. The pointwise and continuous pair types are formed in the same way as the function types. For the parametric pair type $\exists$, a problem occurs because $\Sigma(x : \sharp A').B'$ is not generally discrete. We resolve this by applying a type-level shape operator $\oint$ that identifies all terms connected by non-dependent paths, obtaining $\sharp\int[\![\Gamma]\!] \vdash \oint\Sigma(x : \sharp A').B' : \mathcal{U}^{\mathrm{NDD}}$. Pushing $\sharp\int$ to the right again, we find $[\![\Gamma]\!] \vdash [\![\exists(x : A).B]\!] : \flat\P\mathcal{U}^{\mathrm{NDD}}$.

Face predicates. Similar to $\mathcal{U}^{\mathsf{Psh}}$, there is a standard presheaf universe of propositions $\mathsf{Prop}^{\mathsf{Psh}}$. The defining terms $W \vdash^{\mathrm{D}} P : \mathsf{Prop}^{\mathsf{Psh}}$ are the *proof-irrelevant* types $W \vdash P$ type over the Yoneda-embedded primitive context W; these are types for which all defining terms are equal to $\star$. We define $[\![\mathbb{F}]\!] = \flat\P(\sharp\mathsf{Prop}^{\mathsf{Psh}}) = \flat\mathsf{Prop}^{\mathsf{Psh}}$ (so $\sharp\mathsf{Prop}^{\mathsf{Psh}}$ takes the role that $\mathcal{U}^{\mathsf{Psh}}$ took before) and interpret context extension similar to the way we did for types. See [Nuyts 2017] for details.

Glueing and welding. We first define these types in the standard presheaf model. Write $W = \mathsf{Weld}\{A \to (P\,?\,T, f)\}$ and $G = \mathsf{Glue}\{A \leftarrow (P\,?\,T, h)\}$. We want these to be types: $\Gamma \vdash W, G$ type. We define W as follows: the defining terms $V \vdash^{\mathrm{D}} w : W[\gamma]$ are defining terms $V \vdash^{\mathrm{D}} w : T[(\gamma, \star)]$ if $P[\gamma] = \top$; otherwise, they are defining terms $V \vdash^{\mathrm{D}} w : A[\gamma]$. The primitive substitutions $w[\varphi]$ of w are defined using some case distinctions and the function f. Similarly, a defining term $V \vdash^{\mathrm{D}} g : G[\gamma]$ is a defining term $V \vdash^{\mathrm{D}} g : T[(\gamma, \star)]$ if $P[\gamma] = \top$. Otherwise, it is a pair (a, t) with $V \vdash^{\mathrm{D}} a : A[\gamma]$ and $V, p : P \vdash t : T[(\gamma, p)]$, such that h maps t to a. Again, primitive substitutions are defined using case distinction and the function h.

The next step is to internalize these types. We have $[\![A]\!]$ and $[\![T]\!]$ living in $\flat\P\mathcal{U}^{\mathsf{NDD}}$ and $[\![P]\!]$ living in $\flat\P\sharp\mathsf{Prop}^{\mathsf{Psh}}$, all in context $[\![\Gamma]\!]$. In each of these cases, we can push $\flat\P$ to the left, arriving in context $\sharp\int[\![\Gamma]\!]$. Further unpacking is needed for $[\![P]\!]$. Meanwhile, $[\![f]\!]$ lives in context $[\![\P \setminus \Gamma]\!] \cong \sharp[\![\Gamma]\!]$. The fact that the type of f is discrete and lives in a discrete universe, allows us to disregard $\int$, so that f too lives in $\sharp\int[\![\Gamma]\!]$. Now we can apply the presheaf Glue or Weld type formers and push the functors to the right again.

Size. We define a closed type Size in the presheaf model as a type of naturals with codiscrete bridge structure and discrete path structure: a defining term $(\vec{j} : \mathbb{I}^m, \vec{i}^\sharp : \mathbb{I}^n) \vdash^{\mathrm{D}} s : \mathsf{Size}$ consists of 2^m natural numbers, one for each vertex of the bridge cube. So we relate any two numbers with a bridge, while paths are required to be constant. The type $\sharp\mathsf{Size}$ then has defining terms $(\vec{j} : \mathbb{I}^m, \vec{i}^\sharp : \mathbb{I}^n) \vdash^{\mathrm{D}} s : \sharp\mathsf{Size}$ consisting of 2^{m+n} natural numbers. Given $\Gamma \vdash s, t : \sharp\mathsf{Size}$, we define a proposition $\Gamma \vdash s \leq t$ type, where we have a defining term $W \vdash^{\mathrm{D}} \star : (s \leq t)[\gamma]$ if and only if $s[\gamma] \leq t[\gamma]$ vertexwise. The internal type $[\![s \leq t]\!]$ is then defined similarly to the identity type. Finally, the fixpoint operator on Size is defined by induction on the greatest vertex of a given cube.

6 RELATED AND FUTURE WORK

Parametricity in and of dependent type theory. Figure 9 provides an overview of related work. Reynolds's original formulation [1983] is in terms of a (partly ill-conjectured [Reynolds 1984]) set-theoretic semantics, but others have shown that parametricity of System F can be formulated in a predicate logic on System F. Such frameworks can be seen as 'internal' because the parametricity proof that we obtain for a specific program, can be constructed as a proof term internally in the predicate logic. However, as the 'journey' column emphasizes, the fact that this program-to-proof translation works for *any* program, is proven externally. IEL holds in each of the cited frameworks for System F.

The second chunk of Fig. 9 contains frameworks for parametricity of dependent type theory. In this class of work, IEL is only shown to hold for small types. In fact, the function leak from Section 1 would be ruled out by IEL; hence, IEL cannot hold in general. In our terminology, work like Bernardy et al.'s [2012] which omits IEL altogether, shows that MLTT is *continuous* (i.e. all functions respect relations) analogous to Lemma 3.5 where μ is the continuous modality.

Atkey et al. [2014] prove their results in a reflexive graph model, following related models for simpler type systems such as those by Robinson and Rosolini [1994], Hasegawa [1994], and Atkey [2012]. This model has been enhanced by Bernardy et al. [2015] to a (unary, but generalizable) model in terms of cubical sets (iterated reflexive graphs) that supports iterated parametricity. Our work builds further on that. Bernardy et al.'s type system provides operators that allow us to map

citation	source	target	journey	model	IEL proof
[Reynolds 1983]	System F			conject. set model	yes
[Abadi et al. 1993]	System F	System $\mathscr{R}$	external		yes
[Plotkin and Abadi 1993]	System F	System F + logic	external		yes
[Wadler 2007]	System F	System F + logic	external		yes
[Takeuti 2001]	$\mathscr{X} \in \lambda$-cube	$\mathscr{Y} \in \lambda$-cube	external		for small types
[Bernardy et al. 2012]	any PTS	other PTS	external		no
[Krishnaswami and Dreyer 2013]	dependent types			PER-model	only some corollaries
[Atkey et al. 2014]	dependent types			presheaves: reflexive graphs	for small types
[Bernardy et al. 2015]	dependent types + param. operators	same as source	internal	presheaves: (unary) cubical sets	no
This work	dependent types + Glue, Weld, $\forall$, $\exists$	same as source	internal	presheaves: bridge/path cubical sets	yes: (semantics of) degax

Fig. 9. Classification of important related frameworks that prove parametricity. The 'source' column lists the type system that parametricity is proven of. If parametricity is formulated in some type system, it is listed under 'target' and the 'journey' column lists whether the translation from program to parametricity proof, takes place internal to the type system, or externally in the metatheory. If a metatheoretic model is (also) used, it is listed under 'model'. The last column lists whether the identity extension lemma (IEL) is proven.

programs to their parametricity proofs *internal* to the type system. Modulo some technical issues that can be overcome, their operators could be plugged into our presheaf model and supplementing them with the path degeneracy axiom would have given our system similar power. However, just as in their system, we would have had non-duplicable interval variables, severely complicating implementation as an extension of Agda. We overcome this, somewhat experimentally, by choosing instead to use the more indirect Glue and Weld types, which exist in any presheaf model and are in this sense also more robust against future reworkings of the model. This decision is unrelated to the appearance of the pointwise modality (¶). The graph type $/f\backslash$ from Section 3.2.1 with its push and pull functions, is analogous to the graph relation in System $\mathscr{R}$ [Abadi et al. 1993].

Modalities. Although the use of modalities for keeping track of parametricity is to our knowledge new, parametricity is just one addition to a large list of applications of modalities, including (eponymously) modal logic [Pfenning and Davies 2001], variance of functorial dependencies [Abel 2006, 2008; Licata and Harper 2011], irrelevance [Abel and Scherer 2012; Reed 2003], erasure [Mishra-Linger and Sheard 2008], intensionality vs. extensionality [Pfenning 2001]. Licata and Shulman's modality system for axiomatic cohesion [2016] is an important ingredient of our model. The syntactic treatment in terms of order, composition and left division has been developed by Pfenning [2001] and Abel [2006, 2008], and was already implemented in Agda as the basis for its irrelevance modality, facilitating the implementation of the ParamDTT extension of Agda.

Parametricity versus irrelevance. Notions closely related to parametricity, especially in non-dependently-typed systems, are irrelevance and erasability. The meanings of these words seem to shift somewhat throughout the literature, so we start by defining the terminology we will use. By a *parametric* dependency, as in the rest of the paper, we mean a dependency that maps related inputs to (heterogeneously) equal outputs. This includes the identity function $\sharp\mathsf{Nat} \to \mathsf{Nat}$ defined in Eq. (2). By an *erasable* dependency, we mean a dependency that can be erased after type-checking, at compile time, while preserving the operational semantics of a program. By an *irrelevant* dependency, we mean a dependency that can be erased already during type-checking, implying that terms can be converted between types that are equal up to their irrelevant parts. It is intuitively clear that irrelevance is stronger than erasability, which in turn is stronger than parametricity.

Mishra-Linger and Sheard's EPTS [2008] and Barras and Bernardo's ICC* [2008] are type systems with quantifiers for erasable dependencies based on Miquel's implicit calculus of constructions (ICC) [2001a; 2001b]. Both propose a conversion rule that erases at type-checking time, making their quantifiers irrelevant. If we allow conversion only between β-equal types, as Mishra-Linger and Sheard also suggest, both systems embed into ours. Abel and Scherer [2012] show that it is problematic to view the quantifiers of EPTS and ICC* as irrelevant. The problem can be neatly formulated in terms of our bridges and paths: if a function is to be irrelevant, then surely it must map any pair of inputs to equal outputs. However, both type systems consider irrelevant functions $f : \Pi^{\mathsf{irr}}(x : A).B$ with merely 'continuous' codomain $B : A \to \mathcal{U}$. In our system, this means that we would require a path between any two values $f\ a_1^{\mathsf{irr}}$ and $f\ a_2^{\mathsf{irr}}$ without necessarily providing a *notion* of paths between $B\,a_1$ and $B\,a_2$. The result is unclarity about how to check cross-type equality.

Reed [2003] and Abel and Scherer [2012] present similar type systems with irrelevant quantifiers in which this problem does not arise as they require the codomain $B : \mathsf{irr}\ A \to \mathcal{U}$ to be irrelevant as well. We can conclude that each of the concepts mentioned above has its own virtues. Irrelevance admits erasure at type-checking time, but we cannot consider irrelevant functions for an arbitrary dependent codomain. Erasability does allow arbitrary codomains, but admits erasure only at compile time. Parametricity does not admit erasure whatsoever, but it does admit pattern matching eliminators while still producing free theorems.

Cubical type theory and HoTT. Cubical type theory [Bezem et al. 2014; Cohen et al. 2016] uses a cubical set model, similar to ours, to model the univalence axiom from homotopy type theory (HoTT) and consequently, function extensionality. The cubical type system and ParamDTT have in common that equalities can be expressed using functions from the interval, and that types varying over the interval can be constructed using variations of Glue. We expect that both systems can be merged into a system for parametric HoTT. Voevodsky's homotopy type system [HTS, 2013] and Altenkirch et al.'s 2-level type system [2016], which contain both a fibrant path type and a non-fibrant strict equality type, may play well with such a system for parametric HoTT, where types live in a different context as their terms and hence need not be fibrant in their terms' context.

Iterated bridges. It is regrettable that we have lost internal iterated parametricity. This issue is directly related to the fact that in ParamDTT there is no way to provide, between two types A_0 and A_1, a notion of heterogeneous bridges without also providing a notion of heterogeneous paths. Indeed, if we have a bridge $A : \mathbb{I} \to \mathcal{U}$ from A_0 to A_1, then we can consider both bridges $\Pi(i : \mathbb{I}).A\,i$ and paths $\forall(i : \mathbb{I}).A\,i$. However, if we have a bridge between functions f and g, then a heterogeneous bridge from $a : \mathrm{Glue}\{A \leftarrow (P\,?\,T, f)\}$ to $b : \mathrm{Glue}\{A \leftarrow (P\,?\,T, g)\}$ has meaning in the model, whereas a path does not. This suggests that we should add to our model a weaker connection of *pro-bridges*, such that a pro-bridge between types expresses a notion of bridges, but not paths. This will then immediately ask for the addition of pro-pro-bridges, etc. It seems that a system for iterated bridge/path parametricity needs to be modelled in iterated bridge/path cubical sets which contain ever weaker notions of edges. On the syntax side, the consequence would be that the ¶ modality is lossless and we can have $\sharp \circ ¶ = \mathbf{id}$, which would stop the propagation of the ¶ modality that precludes iterated parametricity. A possibly related feature that ParamDTT lost with respect to both cubical type theory and Bernardy et al.'s work, is that our bridge and path types are not indexed by their endpoints; rather, they look like ordinary function spaces. The reason is that the interaction between modalities and indexed function types poses very subtle problems and we were able to achieve good results without. We believe that iterated bridge/path cubical sets could create clarity on this issue as well.

7 ACKNOWLEDGEMENTS

We want to thank Andreas Abel, Paolo Capriotti, Jesper Cockx, Dan Licata and Sandro Stucki for many fruitful discussions, and Andreas Abel, Sandro Stucki, Philip Wadler and the anonymous reviewers for their valuable feedback on the paper. Andreas Nuyts and Dominique Devriese hold a Ph.D. Fellowship and a Postdoctoral Mandate (resp.) from the Research Foundation - Flanders (FWO).

REFERENCES

Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. 1993. Formal parametric polymorphism. *Theoretical Computer Science* 121, 1 (1993), 9 – 58. DOI: http://dx.doi.org/10.1016/0304-3975(93)90082-5

Andreas Abel. 2006. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types.* Ph.D. Dissertation. Ludwig-Maximilians-Universität München.

Andreas Abel. 2008. Polarised subtyping for sized types. *Mathematical Structures in Computer Science* 18, 5 (2008), 797–822. DOI: http://dx.doi.org/10.1017/S0960129508006853

Andreas Abel and Gabriel Scherer. 2012. On Irrelevance and Algorithmic Equality in Predicative Type Theory. *Logical Methods in Computer Science* 8, 1 (2012), 1–36. DOI: http://dx.doi.org/10.2168/LMCS-8(1:29)2012 TYPES'10 special issue.

Andreas M. Abel and Brigitte Pientka. 2013. Wellfounded Recursion with Copatterns: A Unified Approach to Termination and Productivity. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP '13)*. ACM, New York, NY, USA, 185–196. DOI: http://dx.doi.org/10.1145/2500365.2500591

Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. 2016. Extending Homotopy Type Theory with Strict Equality. *CoRR* abs/1604.03799 (2016). http://arxiv.org/abs/1604.03799

Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. 2006. Indexed Containers. Manuscript, available online. (February 2006).

Robert Atkey. 2012. Relational Parametricity for Higher Kinds. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 16. 46–61. DOI: http://dx.doi.org/10.4230/LIPIcs.CSL.2012.46

Robert Atkey, Neil Ghani, and Patricia Johann. 2014. A Relationally Parametric Model of Dependent Type Theory. In *Principles of Programming Languages*. DOI: http://dx.doi.org/10.1145/2535838.2535852

Bruno Barras and Bruno Bernardo. 2008. *The Implicit Calculus of Constructions as a Programming Language with Dependent Types.* Springer Berlin Heidelberg, Berlin, Heidelberg, 365–379. DOI: http://dx.doi.org/10.1007/978-3-540-78499-9_26

Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. 2015. A Presheaf Model of Parametric Type Theory. *Electron. Notes in Theor. Comput. Sci.* 319 (2015), 67 – 82. DOI: http://dx.doi.org/10.1016/j.entcs.2015.12.006

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for Free — Parametricity for Dependent Types. *Journal of Functional Programming* 22, 02 (2012), 107–152. DOI: http://dx.doi.org/10.1017/S0956796812000056

Marc Bezem, Thierry Coquand, and Simon Huber. 2014. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 26. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 107–128. DOI: http://dx.doi.org/10.4230/LIPIcs.TYPES.2013.107

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2016. Cubical Type Theory: a constructive interpretation of the univalence axiom. *CoRR* abs/1611.02108 (2016). http://arxiv.org/abs/1611.02108

Peter Dybjer. 1996. *Internal type theory.* Springer Berlin Heidelberg, Berlin, Heidelberg, 120–134. DOI: http://dx.doi.org/10.1007/3-540-61780-9_66

Ryu Hasegawa. 1994. Relational limits in general polymorphism. *Publications of the Research Institute for Mathematical Sciences* 30 (1994), 535–576.

Martin Hofmann. 1997. *Syntax and semantics of dependent types.* Springer London, London, Chapter 4, 13–54. DOI: http://dx.doi.org/10.1007/978-1-4471-0963-1_2

Martin Hofmann and Thomas Streicher. 1997. Lifting Grothendieck Universes. Unpublished note. (1997).

Neelakantan R. Krishnaswami and Derek Dreyer. 2013. Internalizing Relational Parametricity in the Extensional Calculus of Constructions. In *Computer Science Logic 2013 (CSL 2013) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 432–451. DOI: http://dx.doi.org/10.4230/LIPIcs.CSL.2013.432

Daniel Leivant. 1991. Finitely stratified polymorphism. *Information and Computation* 93, 1 (1991), 93 – 113. DOI: http://dx.doi.org/10.1016/0890-5401(91)90053-5

Daniel R. Licata and Robert Harper. 2011. 2-Dimensional Directed Type Theory. *Electronic Notes in Theoretical Computer Science* 276 (2011), 263 – 289. DOI: http://dx.doi.org/10.1016/j.entcs.2011.09.026 27th Conference on the Mathematical

Foundations of Programming Semantics.

Daniel R. Licata and Michael Shulman. 2016. *Adjoint Logic with a 2-Category of Modes.* Springer International Publishing, 219–235. DOI: http://dx.doi.org/10.1007/978-3-319-27683-0_16

Alexandre Miquel. 2001a. The Implicit Calculus of Constructions: Extending Pure Type Systems with an Intersection Type Binder and Subtyping. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*. Springer-Verlag, Berlin, Heidelberg, 344–359. http://dl.acm.org/citation.cfm?id=1754621.1754650

Alexandre Miquel. 2001b. *Le Calcul des Constructions Implicite: Syntaxe et Sémantique.* Ph.D. Dissertation. Université Paris 7.

Nathan Mishra-Linger and Tim Sheard. 2008. *Erasure and Polymorphism in Pure Type Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 350–364. DOI: http://dx.doi.org/10.1007/978-3-540-78499-9_25

Andreas Nuyts. 2017. *A Model of Parametric Dependent Type Theory in Bridge/Path Cubical Sets.* Technical Report. KU Leuven, Belgium. https://arxiv.org/abs/1706.04383

Frank Pfenning. 2001. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*. 221–230. DOI: http://dx.doi.org/10.1109/LICS.2001.932499

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

Benjamin C. Pierce. 2002. *Types and programming languages.* MIT Press.

Gordon Plotkin and Martín Abadi. 1993. *A logic for parametric polymorphism.* Springer Berlin Heidelberg, Berlin, Heidelberg, 361–375. DOI: http://dx.doi.org/10.1007/BFb0037118

Jason Reed. 2003. Extending Higher-Order Unification to Support Proof Irrelevance. In *Theorem Proving in Higher Order Logics: 16th International Conference, TPHOLs 2003, Rome, Italy, September 8-12, 2003. Proceedings*, David Basin and Burkhart Wolff (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 238–252. DOI: http://dx.doi.org/10.1007/10930755_16

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism.. In *IFIP Congress*. 513–523.

John C. Reynolds. 1984. *Polymorphism is not set-theoretic.* Research Report RR-0296. INRIA. https://hal.inria.fr/inria-00076261

Edmund P. Robinson and Giuseppe Rosolini. 1994. Reflexive graphs and parametric polymorphism. In *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*. 364–371. DOI: http://dx.doi.org/10.1109/LICS.1994.316053

Izumi Takeuti. 2001. *The Theory of Parametricity in Lambda Cube.* Technical Report 1217. Kyoto University.

Vladimir Voevodsky. 2013. A simple type system with two identity types. (2013). https://ncatlab.org/homotopytypetheory/files/HTS.pdf unpublished note.

Philip Wadler. 1989. Theorems for Free!. In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture (FPCA '89)*. ACM, New York, NY, USA, 347–359. DOI: http://dx.doi.org/10.1145/99370.99404

Philip Wadler. 2007. The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science* 375, 1 (2007), 201 – 226. DOI: http://dx.doi.org/10.1016/j.tcs.2006.12.042