Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 319 (2015) 67–82

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Presheaf Model of Parametric Type Theory

Jean-Philippe Bernardy Thierry Coquand Guilhem Moulin

Chalmers University of Technology and University of Gothenburg
{bernardy,coquand,mouling}@chalmers.se

Abstract

We extend Martin-Löf's Logical Framework with special constructions and typing rules providing internalized parametricity. Compared to previous similar proposals, this version comes with a denotational semantics which is a refinement of the standard presheaf semantics of dependent type theory. Further, this presheaf semantics is a refinement of the one used to interpret nominal sets with restrictions. The present calculus is a candidate for the core of a proof assistant with internalized parametricity.

Keywords: Parametricity, Presheaf semantics, Type theory

1 Introduction

Reynolds [17] proved a general *abstraction theorem* (sometimes called *parametricity theorem*) about polymorphic functions. His argument is about a set theoretic semantic. As he stated it, *the underlying idea is that the meanings of an expression in "related" environments will be "related" values.* For instance, he proves that if t_X is a term of type $X \to X$ and if we consider two sets A_0, A_1 and a relation $R \subseteq A_0 \times A_1$, then we have $R([t_X]_{X=A_0}(a_0), [t_X]_{X=A_1}(a_1))$ whenever $R(a_0, a_1)$, where $[t_X]_{X=A}$ denotes the meaning of the expression t_X where X is interpreted by the set A. As he noted, one can replace binary relations by n-ary relations in this statement, and in particular unary relations (predicates). In the latter case, the statement is the following: if A is a set and P is a predicate on A, then we have $P([t_X]_{X=A}(a))$ whenever $P(a)$ holds. Wadler [18] illustrates by many examples how this result is useful for reasoning about functional programs.

The argument and result of Reynolds are model-theoretic in nature. In the Logical Framework, it is possible to state such an abstraction result in a purely syntactical way. One states for example that if a function f has type $(A : U) \to A \to A$ — the type of the polymorphic identity — then $f\,A\,x$ is Leibniz-equal to x, *i.e.*, the following proposition holds:

$$(A : U) \to (P : A \to U) \to (x : A) \to P\,x \to P(f\,A\,x)$$

http://dx.doi.org/10.1016/j.entcs.2015.12.006

Indeed Bernardy et al. [9] prove such a result as a (syntactical) meta-theorem about type systems. However this result is not provable internally, *i.e.*, the following proposition is not provable:

$$(f : (A : U) \to A \to A) \to (A : U) \to (P : A \to U) \to (x : A) \to P\,x \to P(f\,A\,x) \quad (\star)$$

Therefore users relying on the parametricity conditions have postulated the parametricity axiom [3, 11, 16]. However, because postulates do not have computational interpretations, such parametricity conditions can only be used in computationally-irrelevant positions.

Instead, one would like to be able to rely on parametricity conditions within the theory itself. Several attempts have been made [6, 7] — or are currently developed [2] — for designing an extension of dependent type theory in which such an internal form of parametricity holds. We propose another such system here. Our technical contributions are as follows:

- We present an extension of Martin-Löf's Logical Framework (Section 2) which internalizes parametricity (as we show in Section Section 3) and can be seen as a simplification and generalization of the systems of Bernardy and Moulin [6, 7]. In particular, we have a special construction $(a,_i\,p)$ which pairs a term a with its parametricity proof p, as well as special projections to extract the proof. As we will show in Section 3.3, these new constructions enable us to prove the proposition (Equation $\star$) *internally.* (This is not possible with usual pairs and projections since the first projection does not commute with application.) The name i in the above construction is what we call a "color"; we want internalized parametricity not only for LF but also for the extended calculus, and as explained in [7], colors enable nested parametricity by keeping track of the different uses (this is analogous to building hypercubes and accessing their vertices as in [6]). However, unlike previous type theories with internalized parametricity [6, 7], the system presented here does not *compute* parametricity types: for instance, parametricity conditions are *isomorphic to* functions, rather than functions themselves. (As shown in Section 3, this does not appear to be an issue in practice.)
- We provide a *denotational* semantics, in the form of a presheaf model, for this type theory (Section 4). This model is a refinement of the presheaf semantics used to interpret nominal sets with restrictions [10, 15].

We conjecture that conversion and type-checking are decidable for this system.

2 Syntax

In this section we define the syntax and typing rules of our parametric type theory, as well as the equality judgment.

We assume a special symbol '0', and a countably infinite set $\mathbb{I}$ of other symbols, called *colors*. The metasyntactic variables $i, j, \ldots$ range over colors, while φ range over $\mathbb{I} \cup \{0\}$. We further assume a fixed function $\mathsf{fresh}(\cdot)$ such that $\mathsf{fresh}(I) \in \mathbb{I}\backslash I$

for any finite color set I. The main innovation of the type theory presented here is that terms may depend on (a finite number of) colors. For any term a, we note $\mathsf{supp}(a)$ the set of free colors in a.

We do not attempt to explain what lead us to consider a colored type theory; for that we refer to [7] instead.

Definition 2.1 [Syntax of terms and contexts]

$$
\begin{array}{rlr}
A, B, P, T, a, p, t, u \coloneqq & x & \text{variable} \\
& \mid t\ u & \text{application} \\
& \mid \lambda x : A.t & \text{abstraction} \\
& \mid (x : A) \to B & \text{product} \\
& \mid |A| & \text{code} \\
& \mid \mathrm{El}(A) & \text{decode} \\
& \mid U & \text{universe} \\
& \mid (a,_i\, p) & \text{colored pair} \\
& \mid (x : A) \times_i P & \text{colored type pair} \\
& \mid \langle t,_i\, u\rangle & \text{colored function pair} \\
& \mid A \ni_i a & \text{parametricity type} \\
& \mid a \cdot i & \text{parametricity proof} \\
\Gamma, \Delta \coloneqq & () \mid \Gamma, x : A \mid \Gamma, i : \mathbb{I} &
\end{array}
$$

We give a few intuitions to interpret the novel syntax, before formally giving the typing rules of the system.

(i) Reynolds associates each type with a predicate. Here, each type is associated not with a single predicate, but many: one for every color. These multiple predicates are essential to interpret parametricity when it is nested. Indeed, using a single predicate yields inconsistencies. Furthermore these predicates are definable in the logic: the type $A \ni_i a$ expresses that a satisfies the parametricity predicate associated with the type A on color i. For each term a and color i, the term $a(i\,0)$ is the erasure of i in a. It is defined by induction on a (Definition 2.2) and can be understood as a realizer [5] of a.

(ii) The term $a \cdot i$ yields a proof of $A \ni_i a(i\,0)$.

(iii) The forms $(a,_i\, p)$, $(x : A) \times_i P$ and $\langle t,_i\, u\rangle$ allow to locally associate parametricity proofs with a given realizer.

Definition 2.2 [Color renaming and erasure] We consider a color $i \in \mathbb{I}$ and $\varphi \in \mathbb{I} \cup \{0\}$, and define the term $a(i\,\varphi)$ by induction on a.

$$
\begin{aligned}
x(i\,\varphi) &= x \\
(t\,u)(i\,\varphi) &= (t(i\,\varphi))\,(u(i\,\varphi)) \\
(\lambda(x : A).t)(i\,\varphi) &= \lambda(x : A(i\,\varphi)).t(i\,\varphi) \\
((x : A) \to B)(i\,\varphi) &= (x : A(i\,\varphi)) \to (B(i\,\varphi))
\end{aligned}
$$

$$\begin{aligned}
|A|(i\,\varphi) &= |A(i\,\varphi)| \\
\mathrm{El}(A)(i\,\varphi) &= \mathrm{El}(A(i\,\varphi)) \\
U(i\,\varphi) &= U \\
(a,_i\,p)(i\,0) &= a \\
(a,_i\,p)(i\,j) &= (a,_j\,p) \\
(a,_j\,p)(i\,\varphi) &= (a(i\,\varphi),_j\,p(i\,\varphi)) && \text{if } i \neq j \\
((x:A)\times_i P)(i\,0) &= A \\
((x:A)\times_i P)(i\,j) &= (x:A)\times_j P \\
((x:A)\times_j P)(i\,\varphi) &= (x:A(i\,\varphi))\times_j P(i\,\varphi) && \text{if } i \neq j \\
\langle t,_i\,u\rangle(i\,0) &= t \\
\langle t,_i\,u\rangle(i\,j) &= \langle t,_j\,u\rangle \\
\langle t,_j\,u\rangle(i\,\varphi) &= \langle t(i\,\varphi),_j\,u(i\,\varphi)\rangle && \text{if } i \neq j \\
(A \ni_i a)(i\,\varphi) &= A(i\,j)(i\,\varphi) \ni_j a(i\,\varphi) && \text{where } j = \mathsf{fresh}(\mathsf{supp}(A)) \\
(A \ni_j a)(i\,\varphi) &= (A(i\,\varphi)) \ni_j (a(i\,\varphi)) && \text{if } i \neq j \\
(a\cdot i)(i\,\varphi) &= a(i\,j)(i\,\varphi)\cdot j && \text{where } j = \mathsf{fresh}(\mathsf{supp}(a)) \\
(a\cdot j)(i\,\varphi) &= a(i\,\varphi)\cdot j && \text{if } i \neq j
\end{aligned}$$

Definition 2.3 [Typing judgements — à la Tarski]

$\boxed{\Gamma \vdash}$

Empty
$$\frac{}{() \vdash}$$

NewVar
$$\frac{\Gamma \vdash \qquad \Gamma \vdash A}{\Gamma, x : A \vdash}$$

NewCol
$$\frac{\Gamma \vdash}{\Gamma, i : \mathbb{I} \vdash}$$

$\boxed{\Gamma \vdash A}$

Universe
$$\frac{}{\Gamma \vdash U}$$

Decode
$$\frac{\Gamma \vdash A : U}{\Gamma \vdash \mathrm{El}(A)}$$

Swap
$$\frac{\Gamma, i : \mathbb{I}, j : \mathbb{I}, \Delta \vdash A}{\Gamma, j : \mathbb{I}, i : \mathbb{I}, \Delta \vdash A}$$

Pi
$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash (x : A) \to B}$$

Out
$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma \vdash a : A(i\,0)}{\Gamma \vdash A \ni_i a}$$

In-Pred
$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash P}{\Gamma, i : \mathbb{I} \vdash (x : A) \times_i P}$$

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

Conv
$$\frac{\Gamma \vdash t : A \qquad A = B}{\Gamma \vdash t : B}$$

Var
$$\frac{\Gamma \vdash \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Code
$$\frac{\Gamma \vdash A}{\Gamma \vdash |A| : U}$$

Swap
$$\frac{\Gamma, i : \mathbb{I}, j : \mathbb{I}, \Delta \vdash a : A}{\Gamma, j : \mathbb{I}, i : \mathbb{I}, \Delta \vdash a : A}$$

Lam
$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : (x : A) \to B}$$

App
$$\frac{\Gamma \vdash t : (x : A) \to B[x] \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

$$\textsc{In-Abs}\quad \frac{\Gamma \vdash a : A(i\,0) \qquad \Gamma \vdash p : A \ni_i a}{\Gamma, i : \mathbb{I} \vdash (a ,_i p) : A}$$

$$\textsc{In-Fun}\quad \frac{\Gamma \vdash t : ((x : A) \to P[x])(i\,0) \qquad \Gamma \vdash u : (x : A(i\,0)) \to (x' : A \ni_i x) \to P[(x ,_i x')] \ni_i tx}{\Gamma, i : \mathbb{I} \vdash \langle t ,_i u\rangle : (x : A) \to P[x]}$$

$$\textsc{Color-Elim}\quad \frac{\Gamma, i : \mathbb{I} \vdash a : A}{\Gamma \vdash a{\cdot}i : A \ni_i a(i\,0)}$$

The parametricity constructions ($\cdot$ and $\ni$) are color binders (they bring colors into scope), while the pairing constructs remove colors from scope. The equality relation used in the Conv rule is detailed below in Definition 2.5. The Swap rules allow us to use Out and Color-Elim with any free color, provided that no variable was introduced after that color (see *e.g.*, Theorem 3.6).

Additionally, for the above system to be well-founded, we need to distinguish small and big types, and allow only small types to be encoded in U. Small types are closed under product, $\times_i$ and $\ni_i$. The distinction between big and small types being standard, and to keep the presentation concise, we leave it implicit in the syntax[1] .

Theorem 2.4 (Color erasure and substitution preserve typing) *If* $\Gamma, i : \mathbb{I} \vdash a : A$ *then the terms* $a(i\,\varphi)$ *and* $A(i\,\varphi)$ *are defined and*

- $\Gamma \vdash a(i\,0) : A(i\,0)$, *and*
- $\Gamma, j : \mathbb{I} \vdash a(i\,j) : A(i\,j)$.

Proof. By induction on the typing judgment. □

Definition 2.5 [Conversion] The convertibility of types used in the Conv rule and written simply ($=$) is defined as the smallest reflexive-symmetric-transitive congruence containing the following rules.

$$\textsc{Pair-Param}\quad (a ,_i p){\cdot}i = p \qquad \textsc{Pair-App}\quad \langle t ,_i u\rangle\, a = (t\,a(i\,0) ,_i u\,a(i\,0)\,(a{\cdot}i)) \qquad \textsc{Pair-Pred}\quad ((x : A) \times_i P[x]) \ni_i a = P[a]$$

$$\textsc{Surj-Param}\quad t = (t(i\,0) ,_i t{\cdot}i) \qquad \textsc{Surj-Typ}\quad T = (x : T(i\,0)) \times_i (T(i\,j) \ni_j x)$$

$$\mathrm{El}(|A|) = A \qquad |\mathrm{El}(A)| = A \qquad \overset{\beta}{(\lambda x : A.t[x])u = t[u]} \qquad \overset{\eta}{\frac{t\ x = u}{t = \lambda x : A.u}}$$

Corollary 2.6 (Surj-Fun) $t = \langle t(i\,0) ,_i \lambda x x'.(t(x ,_i x')){\cdot}i\rangle$

Remark 2.7 In order to be well-typed, any context for the conclusion of the Pair-App, Surj-Param, Surj-Fun and Surj-Typ rules needs to end with a color binding.

[1] Our rules are semantically justified in Section 4; the use of codes enables a presentation à la Tarski, while avoiding us to split each constructor in two flavors, one for small types and one for large ones.

Remark 2.8 Although it looks as if $\langle t,_i u\rangle$ can be definable as $\lambda x.(t\,x,_i\,u\,x\,x\cdot i)$, the latter rebinds i, and does not allow us to prove parametricity for the Church-encoded naturals (Example 3.4) for instance.

Our conversion relation is intensional for functions, but extensional when it comes to dependencies on colors. Because there is at any point only a finite number of colors to consider, we conjecture that our conversion relation is decidable.

3 Parametricity

In this section we prove that our system properly internalizes unary parametricity; it could naturally be extended to the n-ary case by using further special symbols $1, \ldots, n-1$. We also illustrate the system by giving a few simple proofs relying on parametricity (including iterated parametricity). For the sake of readability, we leave out the distinction between types and their codes, which plays no role here.

Unlike previous type theories with internalized parametricity [6, 7], the system presented here lacks equalities which allow to compute parametricity types. Expressed in our syntax, those equalities would become the conversion rules:

$$\begin{aligned} U \ni_i A &= A \to U, \text{ and} \\ ((x:A) \to B[x]) \ni_i f &= (x:A) \to (x' : A \ni_i x) \to B[(x,_i x')] \ni_i (f x). \end{aligned}$$

The absence of the above equalities allows for a simpler system, but how can we ensure that all parametricity theorems hold? The answer is that the above relationships hold as isomorphisms. We say that A is *isomorphic to* B iff.

(i) there exist $f : A \to B$,
(ii) there exist $g : B \to A$,
(iii) for any x, $f\,(g\,x) = x$, and
(iv) for any x, $g\,(f\,x) = x$.

This notion of isomorphism is quite strong, because the equality used in its definition is the conversion relation (Definition 2.5).

Theorem 3.1 *$U \ni_i A$ is isomorphic to $A \to U$.*

Proof.

(i) $f : (Q : U \ni_i A) \to A \to U$

$f\,Q\,x = (A,_i Q) \ni_i x$

(ii) $g : (P : A \to U) \to U \ni_i A$

$g\,P = ((x:A) \times_i (Px))\cdot i$

(iii) $(A,_i ((y:A) \times_i (Py))\cdot i) \ni_i x = ((y:A) \times_i (Py)) \ni_i x = Px$ by Pair-Param then Pair-Pred, and we conclude by η-contraction.

(iv) $((x:A) \times_i (A,_i Q) \ni_i x)\cdot i = (A,_i Q)\cdot i = Q$ by Surj-Typ (indeed $(x:A) \times_i (A,_i Q) \ni_i x$ is typed in a context ending with $i : \mathbb{I}$) and Pair-Pred. □

Theorem 3.2 $((x:A) \to B[x]) \ni_i f$ *is isomorphic to*

$$(x:A) \to (x' : A \ni_i x) \to B[(x,_i x')] \ni_i (f\,x)$$

Proof.

(i) $f : (q : ((x : A) \to B[x]) \ni_i f) \to (x : A) \to (x' : A \ni_i x) \to B[(x,_i x')] \ni_i (f x)$

$f\, q\, x\, x' = ((f,_i q)(x,_i x'))\cdot i$

(ii) $g : ((x : A) \to (x' : A \ni_i x) \to B[(x,_i x')] \ni_i (f\, x)) \to ((x : A) \to B[x]) \ni_i f$

$g\, p = \langle f,_i p\rangle \cdot i$

(iii) $((f,_i \langle f,_i p\rangle \cdot i)\,(x,_i x)')\cdot i = (\langle f,_i p\rangle\,(x,_i x'))\cdot i = (f\,x,_i p\,x\,x')\cdot i = p\,x\,x'$ by Surj-Param then Pair-App (indeed $\langle f,_i p\rangle$ and $\langle f,_i p\rangle\,(x,_i x')$ are typed in a context ending with $i : \mathbb{I}$) and we conclude by Pair-Param.

(iv) $\langle f,_i \lambda x x'.((f,_i q)\,(x,_i x'))\cdot i\rangle\cdot i = (f,_i q)\cdot i = q$ by Surj-Fun (indeed $(f,_i q)$ is typed in a context ending with $i : \mathbb{I}$) and we conclude by Pair-Param. □

In practice however, when carrying out parametricity proofs, many of the steps of the above isomorphisms cancel each other and one obtains a simpler proof. This behaviour is illustrated by the following examples: parametricity for the polymorphic identity and Church-encoded natural numbers.

Example 3.3 Any function $f : (X : U) \to X \to X$ is the polymorphic identity, *i.e.*, its output is Leibniz-equal to its second input. Assume a context

$$\Gamma = (f : (X : U) \to X \to X,\;\; A : U,\;\; P : A \to U,\;\; a : A,\;\; p : P\,a).$$

Then $\Gamma, i : \mathbb{I} \vdash (x : A) \times_i (P\,x)$ and by Pair-Pred $\Gamma, i : \mathbb{I} \vdash (a,_i p) : (x : A) \times_i (P\,x)$, thus $\Gamma, i : \mathbb{I} \vdash f\,((x : A) \times_i (P\,x))\,(a,_i p) : (x : A) \times_i (P\,x)$ and finally

$$\begin{aligned}\Gamma \vdash (f\,((x : A) \times_i (P\,x))\,(a,_i p))\cdot i : ((x : A) \times_i (P\,x)) \ni_i (f\,((x : A) \times_i (P\,x))\,(a,_i p))(i\,0)\\ = P\,(f\,((x : A) \times_i (P\,x))\,(a,_i p))(i\,0) = P\,(f\,A\,a)\end{aligned}$$

Example 3.4 Let $N = (X : U) \to X \to (X \to X) \to X$. Proving (unary) parametricity for N means that, assuming a context Γ

$$f : N,\;\; A : U,\;\; P : A \to U,\;\; z : A,\;\; z' : P\,z,\;\; s : A \to A,\;\; s' : (x : A) \to P\,x \to P\,(s\,x),$$

we can prove $P\,(f\,A\,z\,s)$.

Indeed $\Gamma, i : \mathbb{I} \vdash (x : A) \times_i (P\,x)$, and by Pair-Pred $\Gamma, i : \mathbb{I} \vdash (z,_i z') : (x : A) \times_i (P\,x)$ and $\Gamma, i : \mathbb{I} \vdash \langle s,_i s'\rangle : (x : A) \times_i (P\,x) \to (x : A) \times_i (P\,x)$, thus

$\Gamma, i : \mathbb{I} \vdash f\,((x : A) \times_i (P\,x))\,(z,_i z')\,\langle s,_i s'\rangle : (x : A) \times_i (P\,x)$, and finally
$\Gamma \vdash (f\,((x : A) \times_i (P\,x))\,(z,_i z')\,\langle s,_i s'\rangle)\cdot i : ((x : A) \times_i (P\,x)) \ni_i (f\,A\,z\,s) = P\,(f\,A\,z\,s)$

As seen in Example 3.4, one needs to use $\langle t,_i u\rangle$ to pair a function with the parametricity proof of its type if one wants to apply that pair to some argument and reduce the application. This is because as noted above, our system does not support direct computation of free theorems: in particular $(A \to B) \ni_i a$ does not reduce.

At this point one may wonder, since a new syntactic construction was introduced for function types, whether yet another construction is required for higher order functions. This objection was preemptively refuted by Theorem 3.2: it turns out that $\langle t,_i u\rangle$ can be combined with $(a,_i p)$ to pair higher order functions with the parametricity proof of their type. The following example illustrates this technique:

Example 3.5 Let $F = (X : U) \to ((X \to X) \to X) \to X$. Proving (unary) parametricity for F means that, assuming a context $\Gamma = f : F,\ A : U,\ P : A \to U,\ g : (A \to A) \to A,\ g' : (h : A \to A) \to ((x : A) \to P\,x \to P\,(h\,x)) \to P\,(g\,h)$, we can prove $P\,(f\,A\,g)$.

Let $T = (x : A) \times_i (P\,x)$. We have $\Gamma, i : \mathbb{I} \vdash T$ and

$$\begin{aligned}
&\Gamma, h : A \to A, h' : (T \to T) \ni_i h, x : A, x' : P\,x, i : \mathbb{I} \vdash (h,_i h') : T \to T, \text{ hence}\\
&\Gamma, h : A \to A, h' : (T \to T) \ni_i h, x : A, x' : P\,x \vdash ((h,_i h')\,(x,_i x'))\cdot i : T \ni_i h\,x = P\,(h\,x)\\
&\Gamma, h : A \to A, h' : (T \to T) \ni_i h \vdash g'\,h\,(\lambda(x : A).\,\lambda(x' : P\,x).\,((h,_i h')\,(x,_i x'))\cdot i) : P\,(g\,h)
\end{aligned}$$

Let $g'' = \lambda h.\,\lambda h'.\,g'\,h\,\lambda(x : A).\,\lambda(x' : P\,x).\,((h,_i h')\,(x,_i x'))\cdot i$. Since we have $\Gamma \vdash g'' : (h : A \to A) \to (T \to T) \ni_i h \to P\,(g\,h)$ we can pair it with g and $\Gamma, i : \mathbb{I} \vdash \langle g,_i g''\rangle : (T \to T) \to T$. We can finally conclude as before, that $\Gamma \vdash (f\,T\,\langle g,_i g''\rangle)\cdot i : P\,(f\,A\,g)$.

3.1 Iterating Parametricity

In our system, one can use parametricity generically as follows:

$$\begin{aligned}
&p\colon (X : U) \to (x : X) \to X \ni_i x\\
&p\,X\,x = x\cdot i
\end{aligned}$$

We have already seen that $A \ni_i$ corresponds to a parametricity predicate for the type A. As we hinted at in the introduction, the color index i allows us to distinguish each application of parametricity. (As a side remark, since the COLOR-ELIM rule introduces a color, limiting the depth of nested applications of parametricity can trivially be enforced in our system by limiting the number of free colors in the context.) We can iterate the operator $A \ni_.$ to construct relations between parametricity witnesses. That is, given a context with

$$x : A,\ \ y : A \ni_j x,\ \ z : A \ni_i x,$$

the type $A \ni_i (x,_j y) \ni_j z$ is well formed ($\ni$ is left associative), and can be understood as a binary relation between the parametricity *proofs* y and z. The following results about this relation illustrate the expressivity of our system.

Theorem 3.6 *If the type A does not depend on either i or j, the relation $\lambda yz.A \ni_i (x,_j y) \ni_j z$ is symmetric.*

Proof. We first construct the proof term:

$$\begin{aligned}
&\sigma_1 : (x : A) \to (y : A \ni_i x) \to (z : A \ni_i x) \to A \ni_i (x,_j y) \ni_j z \to A \ni_j (x,_i z) \ni_i y\\
&\sigma_1\,x\,y\,z\,w = ((x,_j y),_i (z,_j w))\cdot j\cdot i
\end{aligned}$$

And, by α-equivalence on colors, $A \ni_j (x,_i z) \ni_i y = A \ni_i (x,_j z) \ni_j y$. □

Theorem 3.7 *If the type A does not depend on either i or j, then the types $A \ni_i (x,_j y) \ni_j z$ and $A \ni_j (x,_i z) \ni_i y$ are isomorphic.*

Proof. We show that $\sigma_1\, y\, x\, z\, (\sigma_1\, x\, y\, z\, w) = w$. Let $t = ((x,_j\, y),_i\, (z,_j\, w))$, $w' = t{\cdot}j{\cdot}i$, $t' = ((x,_i\, z),_j\, (y,_i\, w'))$. Then $t'(i\,0) = (x,_j\, y) = t(i\,0)$, $t'(j\,0) = (x,_i\, z) = t(j\,0)$, and $(t{\cdot}j)(i\,0) = y$. We now continue to reason by deduction:

$$\begin{aligned} w' &= t{\cdot}j{\cdot}i && \text{by def.}\\ (y,_i\, w') &= t{\cdot}j && \text{because } (t{\cdot}j)(i\,0) = y\\ t'{\cdot}j &= t{\cdot}j && \text{by def.}\\ t' &= t && \text{because } t'(j\,0) = t(j\,0)\\ t' &= ((x,_j\, y),_i\, (z,_j\, w)) && \text{by def.}\\ t'{\cdot}i &= (z,_j\, w) &&\\ t'{\cdot}i{\cdot}j &= w && \end{aligned}$$

□

Remark 3.8 At this point one may wonder if the system could have been set up to have $t{\cdot}i{\cdot}j = t{\cdot}j{\cdot}i$, and the equality between $A \ni_i (x,_j\, y) \ni_j z$ and $A \ni_j (x,_i\, z) \ni_i y$ rather than an isomorphism. The answer is that the equation

$$A \ni_i (x,_j\, y) \ni_j z = A \ni_j (x,_i\, z) \ni_i y$$

is inconsistent: in particular for $A = U$ one gets

$$U \ni_i (X,_j\, P) \ni_j Q = U \ni_j (X,_i\, Q) \ni_i P$$

for arbitrary P and Q of type $U \ni_i X$. The above equality in turn implies

$$(x : X) \to P\,x \to Q\,x \to U = (x : X) \to Q\,x \to P\,x \to U$$

for arbitrary predicates P and Q over X, which is obviously inconsistent.

Theorem 3.9 *If the type A and the term a do not depend on either i or j, and $a' : A \ni_i a$ (not depending on i or j either), then $A \ni_i (a,_j\, a{\cdot}i) \ni_j a'$.*

Proof. We can construct the following closed term:

$$\begin{aligned} &q : (A : U) \to (x : A) \to (x' : A \ni_i x) \to A \ni_i (x,_j\, x{\cdot}i) \ni_j x' &&\\ &q : (A : U) \to (x : A) \to (x' : A \ni_i x) \to A \ni_i x \ni_j x' && \text{by S{\scriptsize URJ}-P{\scriptsize ARAM}}\\ &q\,A\,x\,x' = x'{\cdot}j && \end{aligned}$$

The result is then obtained by substituting a for x and a' for x'. □

To conclude the section we note that by iterating parametricity n times, one creates n-ary relations between proofs of relations of arity $n-1$. Furthermore, the above results carry over to the n-ary case. That is, for each $k < n$, one can construct a function σ_k, which exchanges the arguments k and $k+1$ of a relation. Furthermore, these functions satisfy the laws of the generators of the symmetric group.

4 Presheaf model

In this section we show how to interpret our type theory by a presheaf model.

Definition 4.1 If I and J are two *finite* subsets of $\mathbb{I}$, we call a *color map* any function $f : I \to J \cup \{0\}$ such that $i_1 = i_2$ for any $i_1, i_2 \in I$ with $f(i_1) = f(i_2) \in J$.

Definition 4.2 [Category **pI**] Let objects be finite color sets and morphisms be color maps (a.k.a. partial injections; the Hom-set $I \to J$ denotes functions $I \to J \cup \{0\}$). If $f : I \to J$ and $g : J \to K$, we define the composition as the Kleisli one: $fg : I \to K$ as $fg(i) = 0$ if $f(i) = 0$ and $fg(i) = g(f(i))$ if $f(i) \in J$. We write $1_I : I \to I$ for the identity map. It is easy to check that **pI** is a category (see [14, ex. 9.7 p. 176] for another description of this category).

If $f : I \to J$, $i \notin I$ and $j \notin J$, let $(f, i = j) : I, i \to J, j$ (where I, i is a shorthand for $I \cup \{i\}$) denote the map defined by $(f, i = j)(i) = j$ and $(f, i = j)(k) = f(k)$ for every $k \in I$.

If $f : I, i \to J$ (resp. $f : I, i \to J, j$) is such that $f(i) = 0$ (resp. $f(i) = j$), let $f - i : I \to J$ denote the map defined by $(f - i)(k) = f(k)$ for every $k \in I$.

For any object I and $i \notin I$, let $\iota_i : I \to I, i$ denote the inclusion map, defined by $\iota_i(k) = k$ for every $k \in I$.

Definition 4.3 [Projection] We say that a morphism $\alpha : I \to I_\alpha$ is a *projection* if $I_\alpha \subseteq I$, $\alpha(i) = 0$ for each $i \in I \backslash I_\alpha$, and $\alpha(i) = i$ for each $i \in I_\alpha$.

Definition 4.4 [Total maps] We say that a morphism $h : I \to J$ is *total*, and note $h : I \rightarrowtail J$, if it is injective, *i.e.*, if $h(i) \neq 0$ for each $i \in I$.

Remark 4.5 [Morphism decomposition] Any morphism $f : I \to J$ has a unique decomposition into a projection map $\alpha : I \to I_\alpha$ and a total map $h : I_\alpha \rightarrowtail J$.

Definition 4.6 [I-set] Let an I-element be any tuple indexed by the subsets of I: $(u_J)_{J \subseteq I}$. An I-set is a set of I-elements. For instance, the elements of an $\{i, j\}$-set are of the form $u = (u_\varnothing, u_i, u_j, u_{i,j})$. Alternatively, such an element can be seen as a tuple (u_α) indexed by the projections $\alpha : I \to I_\alpha$.

If a, b are I-elements and $j \notin I$, we define the (I, j)-element $(a,_j b)$ as $(a,_j b)_J \coloneqq a_J$ if $j \notin J$ and $(a,_j b)_{J,j} \coloneqq b_J$. Any (I, i)-element can be written $u = (u_J)_{J \subseteq I,i} = (u_J)_{J \subseteq I} \cup (u_{J,i})_{J \subseteq I}$; We can therefore define the I-elements $u(i\,0) \coloneqq (u_J)_{J \subseteq I}$ and $u \cdot i \coloneqq (u_{J,i})_{J \subseteq I}$. (Hence by definition $u = (u(i\,0),_i u \cdot i)$.)

Recall that a *presheaf* F on $\mathbf{pI}^{\mathrm{op}}$ is given by a family of sets $F(I)$ together with restriction maps $F(I) \to F(J)$, $u \mapsto uf$ for $f : I \to J$ satisfying $u1 = u$ and $(uf)g = u(fg)$. (Note that the category of presheaves on $\mathbf{pI}^{\mathrm{op}}$ is equivalent to the category **Res** of nominal restriction sets [14, rem. 9.9 p. 161].) We use a refined presheaf on $\mathbf{pI}^{\mathrm{op}}$ by requiring two further conditions:

(i) for any object I, $F(I)$ is an I-set; and

(ii) for any projection map $\alpha : I \to I_\alpha$, the restriction map $F(I) \to F(I_\alpha)$, $u \mapsto u\alpha$ is the projection operation, *i.e.*, $u\alpha_J = u_J$ for any $J \subseteq I$ (alternatively, seeing I-elements as tuples indexed by projection maps, $(u\alpha)_\beta = u_{\alpha\beta}$).

Unless written otherwise, any presheaf in the remainder of this section is assumed to satisfy these conditions. The refinement is necessary for the interpretation of some of our syntactic constructions. Indeed, without it, it is not clear how to validate the equality PAIR-PRED: $((x : A) \times_i P[x]) \ni_i a = P[a]$.

A context $\Gamma \vdash$ is interpreted by a (non-refined) presheaf on $\mathbf{pI}^{\mathrm{op}}$, *i.e.*, by a family of sets $\Gamma(I)$ for each object I, together with restriction maps $\Gamma(I) \to \Gamma(J)$, $\rho \mapsto \rho f$ for $f : I \to J$ satisfying the conditions $\rho 1 = \rho$ and $(\rho f)g = \rho(fg)$.

A type $\Gamma \vdash A$ is interpreted by an I-set $A\rho$ for each object I and $\rho \in \Gamma(I)$, together with restriction maps $A\rho \to A(\rho f)$, $u \mapsto uf$ if $f : I \to J$ satisfying $u1 = u$ and $(uf)g = u(fg)$ for any $g : J \to K$. Furthermore the map $A\rho \to A(\rho\alpha)$, $u \mapsto u\alpha$ is the projection operation.

A term $\Gamma \vdash a : A$ is interpreted by an I-element $a\rho \in A\rho$ for each object I and $\rho \in \Gamma(I)$, such that $a\rho f = a(\rho f)$ for any $f : I \to J$.

If $\Gamma \vdash$ and $\Gamma \vdash A$ we define the interpretation of $\Delta = (\Gamma, x : A)$ by taking $\langle \rho, x = u\rangle \in \Delta(I)$ to mean $\rho \in \Gamma(I)$ and $u \in A\rho$. The restriction map is defined by $\langle \rho, x = u\rangle f = \langle \rho f, x = uf\rangle$.

If $\Gamma \vdash$ we define the interpretation of $\Delta = (\Gamma, i : \mathbb{I})$ by taking $[\rho, i = \varphi] \in \Delta(I)$ to mean either $\varphi = 0$ and $\rho \in \Gamma(I)$, or $\varphi = j \in I$ and $\rho \in \Gamma(I\backslash\{j\})$. The restriction map is defined by $[\rho, i = 0]f = [\rho f, i = 0]$ and $[\rho, i = j]f = [\rho(f - j), i = f(j)]$.

Remark 4.7 In other words, $\Gamma, x : A \vdash$ is interpreted by the cartesian product $(\rho \in \Gamma) \times A\rho$ of the interpretations of $\Gamma \vdash$ and $\Gamma \vdash A$, while $\Gamma, i : \mathbb{I} \vdash$ is interpreted by the separated product [14, sec. 3.4 p. 54] $\Gamma * \mathbb{I}$ of the interpretation of $\Gamma \vdash$ and $\mathbb{I} \cup \{0\}$:

$$\Gamma * \mathbb{I}(I) = \{[\rho, i = 0] \mid \rho \in \Gamma(I)\} \cup \{[\rho, i = j] \mid j \in I, \rho \in \Gamma(I\backslash\{j\})\}$$

We also note that $\Gamma, i : \mathbb{I}, j : \mathbb{I} \vdash$ and $\Gamma, j : \mathbb{I}, i : \mathbb{I} \vdash$ are respectively interpreted as the sets of $[\rho, i = \varphi, j = \varphi']$ and $[\rho, j = \varphi, i = \varphi']$, which are trivially isomorphic.

The semantics we define satisfy the substitution law. That is, if $\Gamma, x : A \vdash B$ and $\Gamma \vdash a : A$ then for any $\rho \in \Gamma(I)$ we have $B[a]\rho = B\langle \rho, x = a\rho\rangle$. It also satisfies the substitution law on colors, *i.e.*, if $\Gamma, i : \mathbb{I} \vdash A$ then for any $\rho \in \Gamma(I)$ and $j \notin I$ we have $A(i0)\rho = A[\rho, i = 0] = A[\rho, i = j](j\,0)$. (Since $[\rho, i = 0] \in \Gamma * \mathbb{I}(I)$ and $[\rho, i = j] \in \Gamma * \mathbb{I}(I, j)$, $A(i\,0)\rho$ and $A[\rho, i = 0]$ are I-sets while $A[\rho, i = j]$ is a (I, j)-set.) For establishing these properties, we proceed as Aczel [1].

We proceed to interpret each type construction.

PI. Assume $\rho \in \Gamma(I)$. We define $((x : A) \to B)\rho$ as a I-set. An I-element of $((x : A) \to B)\rho$ is defined as a tuple $\lambda = (\lambda_\alpha)$, where each λ_α is a family of elements indexed by a total map $f : I_\alpha \rightarrowtail J$:

$$\lambda_{\alpha f} \in \prod_{u \in A(\rho\alpha f)} B\langle \rho\alpha f, x = u\rangle$$

such that $\mathsf{app}(\lambda_{\alpha f}, u)g = \mathsf{app}(\lambda_{\alpha f g}, ug)$ for $f : I_\alpha \rightarrowtail J$ total and for *any* $g : J \to K$ (where app is the semantic application). Because any map $I \to J$

has an unique decomposition as a projection and a total map, we can consider λ_f for an arbitrary map $f : I \to J$.

If $f : I \to J$ is an arbitrary map, we define λf to be the tuple (λf_β) where λf_β is the family $\lambda f_{\beta g} = \lambda_{f\beta g}$. With this definition, we directly have $\lambda\alpha_\beta = \lambda_{\alpha\beta}$.

This is similar to the usual interpretation of dependent product in presheaf models [10, 12]; but to satisfy our first extra condition on presheaves we present each element as a tuple, which can be done naturally by repartitioning the family as follows: $(\lambda_f)_{f:I\to J} = (\lambda_{\alpha g})_{I_\alpha\subseteq I, g:I_\alpha\rightarrowtail J} \cong ((\lambda_{\alpha g})_{g:I_\alpha\rightarrowtail J})_{I_\alpha\subseteq I}$.

Universe. The universe U is interpreted as a presheaf over **pI**. An element A of $U(I)$ is a tuple (A_α) where each A_α is a family $(A_{\alpha f})$ of $\mathcal{U}$-small sets (where $\mathcal{U}$ is a fixed Grothendieck universe) indexed by $f : I_\alpha \rightarrowtail J$ *total* together with restriction maps $A_{\alpha f} \to A_{\alpha f g}$, $u \mapsto ug$ for $f : I_\alpha \rightarrowtail J$ total and $g : J \to K$ arbitrary, such that $u1 = u$ and $(ug)h = u(gh)$.

As before, such data define a set A_f for an arbitrary map $f : I \to J$ with restriction maps $A_f \to A_{fg}$ if $g : J \to K$.

If $f : I \to J$ is an arbitrary map, we define Af by taking $Af_{\beta g}$ to be the set $A_{f\beta g}$, together with restriction maps $Af_{\beta g} \to Af_{\beta gh}$ defined as the given maps $A_{f\beta g} \to A_{f\beta gh}$. We can then check, as before, that we have $A\alpha_\beta = A_{\alpha\beta}$.

As before, this is similar to the usual interpretation of universe in presheaf models, where each element is presented as a tuple.

Out. Assume $\rho \in \Gamma(I)$. We need to define the I-set $(A \ni_i a)\rho$. Let $j = \mathsf{fresh}(I)$. We get a (I, j)-set $A[\rho, i = j]$, and the I-element $a\rho$ belongs to $A(i\,0)\rho = A[\rho, i = 0] = A[\rho, i = j](j0)$.

We define $(A \ni_i a)\rho$ to be the set of I-elements v such that $(a\rho,_j v) \in A[\rho, i = j]$. If v is such an element and $f : I \to J$ and $k = \mathsf{fresh}(J)$, then vf is defined by the equation $(a\rho f,_k vf) = (a\rho,_j v)(f, j = k)$.

In-Pred. Assume $[\rho, i = \varphi] \in \Gamma * \mathbb{I}(I)$. We define the I-set $((x : A) \times_i P)[\rho, i = \varphi]$ by case analysis on $\varphi \in I \cup \{0\}$. If $\varphi = 0$ then $\rho \in \Gamma(I)$, and we define $((x : A) \times_i P)[\rho, i = 0]$ as the I-set $A\rho$. If $\varphi = j \in I$ then $\rho \in \Gamma(I\backslash\{j\})$, and we define $((x : A) \times_i P)[\rho, i = j]$ as the I-set of $(u,_j v)$ where $u \in A\rho$ and $v \in P\langle\rho, x = u\rangle$.

Decode. Assume $\rho \in \Gamma(I)$. We have $A\rho \in U(I)$ and we define $\mathrm{El}(A)\rho$ to be the set $A\rho_1$. The restriction map $\mathrm{El}(A)\rho \to \mathrm{El}(A)\rho f$, $u \mapsto uf$ is defined using the restriction map $A\rho_1 \to A\rho_f$ and the fact that we have $A\rho_f = A(\rho f)_1$.

Remark 4.8 Our calculus does not have any base types, but they could be interpreted by modifying their usual interpretation as a constant presheaf into an isomorphic I-set. For instance, the base type of natural numbers would be interpreted as the I-set of $(n_J)_{J\subseteq I}$ where $n_\varnothing \in \mathbb{N}$ and $n_J = \varnothing$ for any non-empty $J \subseteq I$.

We now describe how to interpret terms.

Var. We define $x\langle\rho, y = u\rangle$ to be u if $x = y$, and $x\rho$ otherwise. We define $x[\rho, i = \varphi]$ to be $x\rho$ if $\varphi = 0$, and $x(\rho\iota_j)$ if $\varphi = j$.

Lam. We define $\mathsf{app}((\lambda x : A.t)\rho_f, u)$ to be $t\langle\rho f, x = u\rangle$

App. We define $(t\,u)\rho$ to be $\mathsf{app}(t\rho_1, u\rho)$

In-Abs. Assume $[\rho, i = \varphi] \in \Gamma * \mathbb{I}(I)$. We define the I-element $(a,_i p)[\rho, i = \varphi]$ by case analysis on $\varphi \in I \cup \{0\}$. If $\varphi = 0$ then $\rho \in \Gamma(I)$, and we take $(a,_i p)[\rho, i = 0]$ to be $a\rho \in A(i\,0)\rho = A[\rho, i = 0]$. If $\varphi = j \in I$ then $\rho \in \Gamma(I\backslash\{j\})$, and we take $(a,_i p)[\rho, i = j]$ to be $(a\rho,_j p\rho)$.

In-Fun. Assume $[\rho, i = \varphi] \in \Gamma * \mathbb{I}(I)$. We define the J-element $\langle t,_i u\rangle[\rho, i = \varphi]_f$ by case analysis. If $\varphi = 0$, then $\rho \in \Gamma(I)$ and $\rho f \in \Gamma(J)$; we define $w = \langle t,_i u\rangle[\rho, i = 0]_f$ by $\mathsf{app}(w, a) = \mathsf{app}(t\rho_f, a)$. If $\varphi = j \in I$ and $f(j) = 0$, then $\rho \in \Gamma(I\backslash\{j\})$ and $\rho(f - j) \in \Gamma(J)$; we define $w = \langle t,_i u\rangle[\rho, i = j]_f$ by $\mathsf{app}(w, a) = \mathsf{app}(t\rho_{f-j}, a)$. If $\varphi = j \in I$ and $f(j) = k \in J$, then $\rho \in \Gamma(I\backslash\{j\})$ and $\rho(f - j) \in \Gamma(J\backslash\{k\})$; we define $w = \langle t,_i u\rangle[\rho, i = j]_f$ by $\mathsf{app}(w, (a,_k b)) = (\mathsf{app}(t\rho_{f-j}, a),_k \mathsf{app}(\mathsf{app}(u\rho_{f-j}, a), b))$.

Color-Elim. Assume $\rho \in \Gamma(I)$. We define $(a\!\cdot\! i)\rho$ as $a[\rho, i = j]\cdot j$ where $j = \mathsf{fresh}(J)$.

Theorem 4.9 (Convertible terms are semantically equal)

- *If $\Gamma \vdash A_1$ and $\Gamma \vdash A_2$ with $A_1 = A_2$, then $A_1\rho = A_2\rho$ for any $\rho \in \Gamma(I)$.*
- *If $\Gamma \vdash a_1 : A$ and $\Gamma \vdash a_2 : A$ with $a_1 = a_2$, then $a_1\rho = a_2\rho$ for any $\rho \in \Gamma(I)$.*

Proof. By simultaneous induction on the derivation. We only show the conversion rules Pair-Param, Pair-Pred and Surj-Param here; other rules involving colors can be proven in a similar fashion, while β and η can be proven in the usual way.

Pair-Param. Let $\rho \in \Gamma(I)$ and $j = \mathsf{fresh}(I)$. We have

$$\begin{aligned} & v \in (((x : A) \times_i P) \ni_i a)\rho \\ \text{iff. } & (a\rho,_j v) \in ((x : A) \times_i P)[\rho, i = j] \\ \text{iff. } & (a\rho,_j v) \in \{(u,_j w) \mid u \in A\rho, w \in P\langle\rho, x = u\rangle\} \\ \text{iff. } & v \in P\langle\rho, x = a\rho\rangle \\ \text{iff. } & v \in P[a]\rho \end{aligned}$$

Pair-Pred. Let $\rho \in \Gamma(I)$ and $j = \mathsf{fresh}(I)$. We have $((a,_i p)\cdot i)\rho = (a,_i p)[\rho, i = j] \cdot j = (a\rho,_j p\rho) \cdot j = p\rho$

Surj-Param. For each $\rho \in \Gamma(I)$ we have $(t(i\,0),_i t \cdot i)[\rho, i = 0] = t(i\,0)\rho = t[\rho, i = 0]$, and if $j \notin I$ then $(t(i\,0),_i t \cdot i)[\rho, i = j] = (t(i\,0)\rho,_j (t \cdot i)\rho) = (t[\rho, i = j](j\,0),_j t[\rho, i = j]\cdot j) = t[\rho, i = j]$. Hence $(t(i\,0),_i t\cdot i)\rho = t\rho$ for any $\rho \in \Gamma * \mathbb{I}(I)$. □

Remark 4.10 As noted earlier, the types $U \ni_i (X,_j P) \ni_j Q$ and $U \ni_j (X,_i Q) \ni_i P$ are not convertible. Their semantic interpretations are not equal either. Indeed taking $\rho \in \Gamma(I)$, $k = \mathsf{fresh}(I)$ and $l = \mathsf{fresh}(I, k)$, we have (leaving out the context interpretation ρ for the sake of readability) on the one hand

$$\begin{aligned} & v \in (U \ni_i (X,_j P) \ni_j Q)\rho \\ \text{iff. } & (Q\rho,_k v) \in (U \ni_i (X,_j P))[\rho, j = k] \\ \text{iff. } & ((X,_j P)[\rho, j = k],_l (Q\rho,_k v)) \in U(l, k) \\ \text{iff. } & ((X\rho,_k P\rho),_l (Q\rho,_k v)) \in U(l, k) \end{aligned}$$

while on the other hand

$$\begin{aligned} & v \in (U \ni_j (X,_i Q) \ni_i P)\rho \\ \text{iff. } & (P\rho,_k v) \in (U \ni_j (X,_i Q))[\rho, i = k] \\ \text{iff. } & ((X,_i Q)[\rho, i = k],_l (P\rho,_k v)) \in U(k,l) \\ \text{iff. } & ((X\rho,_k Q\rho),_l (P\rho,_k v)) \in U(k,l) \end{aligned}$$

hence $(U \ni_i (X,_j P) \ni_j Q)\rho \neq (U \ni_j (X,_i Q) \ni_i P)\rho$ since the map $U(l,k) \to U(k,l)$, $u \mapsto ug$ where $g(k) = l$ and $g(l) = k$ is not the identity.

Theorem 4.11 (Validity) *If* $\Gamma \vdash a : A$ *then* $a\rho \in A\rho$ *for any* $\rho \in \Gamma(I)$.

Proof. By induction on the typing judgment. We only show the cases In-Abs and Color-Elim. In-Fun is similar to the former, and the other cases match the usual proof (using Theorem 4.9 for Conv).

In-Abs. Assume $[\rho, i = \varphi] \in \Gamma * \mathbb{I}(I)$. We proceed by case analysis on $\varphi \in I \cup \{0\}$. If $\varphi = 0$ then $\rho \in \Gamma(I)$, and we have $(a,_i p)[\rho, i = 0] = a\rho \in A(i\,0)\rho = A[\rho, i = 0]$. If $\varphi = j \in I$ then $\rho \in \Gamma(I \backslash \{j\})$, and we have $(a,_i p)[\rho, i = j] = (a\rho,_j p\rho)$; Since by induction hypothesis $p\rho \in (A \ni_i a)\rho$, we conclude by definition that $(a\rho,_j p\rho) \in A[\rho, i = j]$.

Color-Elim. Assume $\rho \in \Gamma(I)$. We need to show that $(a \cdot i)\rho \in (A \ni_i a(i\,0))\rho$, *i.e.*, that $(a(i\,0)\rho,_j (a \cdot i)\rho) \in A[\rho, i = j]$ where $j = \mathsf{fresh}(I)$. By induction hypothesis $a[\rho, i = j] \in A[\rho, i = j]$, hence we have $(a(i\,0)\rho,_j (a \cdot i)\rho) = (a[\rho, i = j](j\,0),_j a[\rho, i = j] \cdot j) = a \in A[\rho, i = j]$. □

5 Related Work

Our own line of work

This work continues a line of work aiming at a smooth integration of parametricity with dependent types [5–9]. The present work offers two improvements over previous publications: 1. a denotational semantics, and 2. a much simplified syntax, suitable as the basis of a proof assistant.

The simplification of syntax is allowed by not requiring the preservation of functions by parametricity. We call preservation of functions by parametricity the property that if f were a function, then the canonical proof that f is parametric (denoted $f \cdot i$ here) is also a function. To our knowledge, following Reynolds [17], all parametric *models* of parametricity (both syntactical and semantical ones) have this property. However, having this property in the *syntax* implies that certain function arguments must be swapped when performing the substitution of beta reduction, as identified by Bernardy and Moulin [6]. In the present system, the parametric interpretation of functions is instead merely isomorphic to a function, thanks to the In-Fun rule (Theorem 3.2). This isomorphism (rather than equality) means on the one hand that the swapping of arguments is handled by the usual rules of logic, instead of special-purpose ones. On the other hand, obtaining the usual parametric interpretation of types requires some purely mechanical work by the user of the logic.

Parametric Models of Type Theory vs. Parametric Type Theories

Two pieces of work propose alternative parametric models of type theory [4, 13], but do not integrate parametricity in the syntax of the calculus. This means that, while certain consequences of parametricity can be made available in the logic (*e.g.*, via constants validated by the model), parametricity itself is not available. In this paper, we not only propose a parametric model, but also show how it can be used to interpret parametricity in the syntax of the type theory.

Various kinds of models

Another characterizing feature of proposals for parametricity is the kind of model underlying the semantics. Krishnaswami and Dreyer [13] propose a model based on Q-PER. Atkey et al. [4] propose a model based on reflexive graphs. The model that we use is based on cubes (functions from subsets of colors). In Bernardy and Moulin [6] the cubes were reified as syntax in an underlying calculus, while in the present work they refine a presheaf structure.

Presheaf models

The presheaf construction used in this paper follows a known template, used for example by Bezem et al. [10] and Pitts [15] to model univalence in type theory. Not only do both models use a presheaf, but they also use a category closely connected to the underlying category **pI**. This means that all these models have an additional cubical structure. We think that it is remarkable that cubical structures are useful for modeling both parametricity and univalence. Altenkirch and Kaposi [2] give a syntax for Bezem et al.'s Cubical Type Theory, effectively modelling univalence by internalization of their model. The present work further refines the model by interpreting terms as I-elements, which is essential to interpret our special-purpose pairing constructions.

6 Future work and conclusion

We have defined a new type theory with internalized parametricity. Thanks to our model construction, we have proved the consistency of the system. The missing piece to construct a type-checker is a decision algorithm for the conversion relation. This checker could then be used as a minimal proof assistant for a type theory with parametricity.

Acknowledgment

The fact that the category of partial bijections **pI** should be relevant for internalization of parametricity became apparent through discussions between Thorsten Altenkirch and the second author about the paper [6].

We are extremely grateful to Simon Huber for finding a flaw in an earlier version of this paper, and helping us to fix it. We also thank Peter Dybjer, Patrik Jansson, Andrea Vezzosi and the anonymous referees for valuable feedback and discussion.

References

[1] P. Aczel. On relating type theories and set theories. In *Proceedings of TYPES'98, volume 1657 of Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 1998.

[2] T. Altenkirch and A. Kaposi. A syntax for cubical type theory. 2014. URL http://www.cs.nott.ac.uk/~txa/publ/ctt.pdf. Draft.

[3] R. Atkey, S. Lindley, and J. Yallop. Unembedding domain-specific languages. In *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*, pages 37–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-508-6. doi: http://doi.acm.org/10.1145/1596638.1596644. URL http://doi.acm.org/10.1145/1596638.1596644.

[4] R. Atkey, N. Ghani, and P. Johann. A relationally parametric model of dependent type theory. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 503–516, 2014. doi: 10.1145/2535838.2535852. URL http://doi.acm.org/10.1145/2535838.2535852.

[5] J.-P. Bernardy and M. Lasson. Realizability and parametricity in Pure Type Systems. In M. Hofmann, editor, *Foundations Of Software Science And Computational Structures*, volume 6604 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2011.

[6] J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *LICS*. IEEE Computer Society, 2012.

[7] J.-P. Bernardy and G. Moulin. Type-theory in color. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional Programming*, pages 61–72, 2013.

[8] J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 345–356, Baltimore, Maryland, 2010. ACM. doi: 10.1145/1863543.1863592.

[9] J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012. doi: 10.1017/S0956796812000056.

[10] M. Bezem, T. Coquand, and S. Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, 2014.

[11] A. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the 13th ACM SIGPLAN international conference on Functional Programming*, pages 143–156, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-919-7. doi: 10.1145/1411204.1411226.

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

[13] N. R. Krishnaswami and D. Dreyer. Internalizing relational parametricity in the extensional calculus of constructions. In *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, pages 432–451, 2013. doi: 10.4230/LIPIcs.CSL.2013.432. URL http://dx.doi.org/10.4230/LIPIcs.CSL.2013.432.

[14] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013. ISBN 9781107017788.

[15] A. M. Pitts. An equivalent presentation of the Bezem-Coquand-Huber category of cubical sets. *CoRR*, abs/1401.7807, 2014. URL http://arxiv.org/abs/1401.7807.

[16] N. Pouillard. Nameless, painless. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional Programming*, ICFP '11, pages 320–332, New York, NY, USA, 2011. ACM. to appear.

[17] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information processing*, 83(1): 513–523, 1983.

[18] P. Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359, Imperial College, London, United Kingdom, 1989. ACM. ISBN 0-89791-328-0. doi: 10.1145/99370.99404. URL http://portal.acm.org/citation.cfm?id=99404.