Thesis for the Degree of Doctor of Philosophy

Internalizing Parametricity

Guilhem Moulin

CHALMERS

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2016

Internalizing Parametricity
Guilhem Moulin
ISBN 978-91-7597-384-5

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4065
ISSN 0346-718X

Technical Report 126D
Department of Computer Science and Engineering
Programming Logic Research Group

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Printed at Chalmers
Göteborg, Sweden 2016

Abstract

Parametricity results have recently been proved for dependently-typed calculi such as the Calculus of Constructions. However these results are meta theorems, and although the theorems can be stated as internal propositions, they cannot be proved internally. In this thesis, we develop a dependent type-theory in which each instance of the parametricity theorem, including those for open terms, can be proved internally. For instance we can prove *inside the system* that each term of type $(X : \star) \to X \to X$ is an identity.

We show three successive proposals for a solution to this problem, each an improvement of the previous one. In the first one we introduce a dependent type theory with special syntax for hypercubes. In the second proposal we outline a colored type theory, which provides a simplification of the type theory with hypercubes. In the third and final proposal, we give a more definite presentation of the colored type theory. We also prove its consistency by a modification of the presheaf model of dependent type theory.

We believe that our final colored type theory is simple enough to provide a basis for a proof assistant where proofs relying on parametricity can be performed internally.

Keywords: Polymorphism, Parametricity, Type structure, Lambda Calculus, Presheaf Model.

This thesis is based on the following publications:

1. J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science*, pages 135–144, 2012

2. J.-P. Bernardy and G. Moulin. Type theory in color. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 61–72. ACM, 2013

3. J.-P. Bernardy, T. Coquand, and G. Moulin. A presheaf model of parametric type theory. volume 319, pages 67–82, 2015. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)

Contents

Acknowledgements

I wish to thank my advisors Peter Dybjer and Jean-Philippe Bernardy for their patience and valuable guidance along the way. Special thanks to Peter for his tireless readings and numerous corrections and other improvements on earlier versions of this thesis; and to Jean-Philippe whose intuition gave birth to the papers included here. Thanks also to Thierry Coquand for stepping in as co-supervisor during the last few months, after Jean-Philippe left the department.

I am also grateful to Tarmo Uustalu for accepting the role of opponent for my defense; and to Thorsten Altenkirch, Patrik Jansson, Rasmus Ejlers Møgelberg and Andreas Abel for accepting to sit in the grading committee.

Thanks to my colleagues and students, who make the CSE department such a great place to work at, especially to my office mates and friends Simon Huber, Bassel Mannaa, Anders Mörtberg and Fabian Ruch. Last but not least, our office's sofa deserves my thanks for its flawless support along the years, as well the poor coffee machine in the lunch room for working most of the time despite the heavy load I put on it.

Introduction

Reynolds [1983] proved a general *abstraction theorem* (sometimes also called *parametricity theorem*) about polymorphic functions in System F. For instance, he proved that if f is a term of type $\forall X.X \to X$ (the type of the polymorphic identity), then for any two sets S_1 and S_2, any relation $R \subseteq S_1 \times S_2$ and any $x_1 \in S_1$ and $x_2 \in S_2$, one has $R([f]_{S_1}(x_1), [f]_{S_2}(x_2))$ whenever $(x_1, x_2) \in R$, where the set-theoretic function $[f]_S \in S^S$ denotes the meaning of f instantiated with the set assignment S. As he puts it, *in essence, the semantics of any expression maps related environments into related values.*

Furthermore, Reynolds noted that one can replace binary relations by n-ary relations in the abstraction theorem, in particular by unary relations (predicates). In the previous example, one obtains that for any set S, any predicate $P \subseteq S$ and any $x \in S$, $P([f]_S(x))$ holds whenever $P(x)$ does. In other words $[f]_S(x)$ is Leibniz-equal to x, *i.e.*, $[f]_S$ must be the identity function on S.

Wadler [1989] illustrated by many examples how this result is useful for reasoning about functional programs. Indeed, the parametricity theorem is used to justify equalities between syntactic terms. For instance, in a total language with Polymorphism à la ML, one can deduce that for any program $f : X \to X$ (where the type variable X is implicitly universally quantified at the top level) and type A and $a : A$, one has $f\,a = a$ (where $=$ denotes the Leibniz equality). Or perhaps more interestingly, if r_X is a polymorphic program of type $X^\star \to X^\star$, where $X^\star$ is the type of lists of type X, then for any two types A and B and function $f : A \to B$, one has that $f^\star \circ r_A = r_B \circ f^\star$ (where $f^\star : A^\star \to B^\star$ is obtained by applying f on each element of the input list). These equalities are in turn used to justify rewrite rules in functional compiler optimizations, and merge multiple function calls into one, or remove intermediate structures [Johann, 2002].

In a system with dependent function types and a universe $\mathcal{U}$, such as the Logical Framework [Nordström et al., 1990], it is possible to state such an abstraction result in a purely syntactic way [Bernardy et al., 2012]. For

instance, if a function f has type

$$(X : \mathcal{U}) \to X \to X,$$

then each instance of f is Leibniz equal to the identity function, *i.e.*, the following type is inhabited:

$$(X : \mathcal{U}) \to (P : X \to \mathcal{U}) \to (x : X) \to P\,x \to P(f\,X\,x)$$

Bernardy et al. [2012] showed that for any term $a : A$ in (strong enough) dependent type theories the conclusion and proof of Reynolds' meta-theorem can respectively be expressed as a syntactic type and term of that theory. They defined a meta-operator $[\![\cdot]\!]$ (called the *parametric interpretation*) by induction on the raw syntax, and proved that $[\![a]\!] : [\![A]\!]\,a$ whenever $a : A$, where $[\![A]\!]$ expands to the (unary) parametricity theorem for the type A. For instance, a function $\lambda(x : A).b$ of type $(x : A) \to B$ is mapped via $[\![\cdot]\!]$ to $\lambda(x : A).\lambda(\dot{x} : [\![A]\!]\,x).[\![b]\!]$, which proves the parametricity theorem $(x : A) \to (\dot{x} : [\![A]\!]\,x) \to [\![B]\!]\,b$. And if f is a function of type $(X : \mathcal{U}) \to X \to X$, then $[\![f]\!]$ is of type

$$\begin{aligned}
&[\![(X : \mathcal{U}) \to X \to X]\!]f \\
&\quad = (X : \mathcal{U}) \to (\dot{X} : [\![\mathcal{U}]\!]\,X) \to [\![X \to X]\!]\,(f\,X) \\
&\quad = (X : \mathcal{U}) \to (\dot{X} : [\![\mathcal{U}]\!]\,X) \to (x : X) \to (\dot{x} : [\![X]\!]\,x) \to [\![X]\!]\,(f\,X\,x) \\
&\quad = (X : \mathcal{U}) \to (\dot{X} : [\![\mathcal{U}]\!]\,X) \to (x : X) \to (\dot{x} : \dot{X}\,x) \to \dot{X}\,(f\,X\,x)
\end{aligned}$$

which, since $[\![\mathcal{U}]\!]\,X \stackrel{\text{def}}{=} X \to \mathcal{U}$ (the type of predicates on X), proves that $f\,X\,x$ is Leibniz-equal to its second input x.

While the conclusion of Reynolds' abstraction theorem has been internalized in the above statement, the whole theorem is still a meta implication. What happens if we try to internalize this meta implication and replace it with an object implication? The parametricity theorem of the polymorphic identity becomes the following syntactic type:

$$\begin{aligned}
&(f : (X : \mathcal{U}) \to X \to X) \to \\
&\qquad (X : \mathcal{U}) \to (\dot{X} : X \to \mathcal{U}) \to (x : X) \to \dot{X}\,x \to \dot{X}\,(f\,X\,x) \qquad (\dagger)
\end{aligned}$$

However this type is not inhabited.

Therefore users relying on the parametricity conditions have postulated it as an axiom instead [Chlipala, 2008, Atkey et al., 2009, Pouillard, 2011]. However, because proofs of postulates are constants, they do not have a computational interpretation, thus the parametricity conditions can only be used in computationally-irrelevant positions.

The question we address in this thesis is whether the type theory can be extended so that the type corresponding to each instance of the parametricity theorem is indeed inhabited. In particular, we are trying to develop a type theory where the type (†) above is inhabited.

A property of the above parametricity interpretation is that each bound variable x is duplicated into a pair $(x, \dot{x})$ of x itself and its parametricity proof $\dot{x}$. In other words, the parametricity proof of bound variables is always in scope. Thus defining $[\![x]\!] \stackrel{\text{def}}{=} \dot{x}$ in the variable case of the induction always leads to well-scoped terms.

Unfortunately this definition does not work in the extension we are envisioning, because the type expressing the parametricity theorem in its general form, namely $T \stackrel{\text{def}}{=} (X : \mathcal{U}) \to (x : X) \to [\![X]\!]\, x$, would not be well-scoped ($[\![X]\!] = \dot{X}$ is not in scope). However we know that any *closed* type A is parametric, and that $[\![A]\!]$ proves its parametricity theorem. Hence each instance $T\,A$ is a valid type (although not necessarily inhabited, as shown by the above example (†)).

A way around the problem, explored by Bernardy and Moulin [2011], is to change the definition of the parametric interpretation $[\![\cdot]\!]$ on free variables. In this work, they introduced a new syntactic construction $\llbracket\cdot\rrbracket$ and defined $[\![x]\!]$ to be the term $\llbracket x \rrbracket$ when x is free, and the variable $\dot{x}$ otherwise. The computation of $[\![\cdot]\!]$ is later resumed when the variable is substituted for a *concrete* term: $\llbracket x \rrbracket[x \mapsto u] \stackrel{\text{def}}{=} [\![u]\!]$. This coincides with the substitution behavior on bound variables: indeed for a bound variable x one has $[\![x[x \mapsto u]]\!] = [\![x]\!][x \mapsto u][\dot{x} \mapsto [\![u]\!]] = \dot{x}[\dot{x} \mapsto [\![u]\!]] = [\![u]\!]$. Since $\llbracket x \rrbracket$ is the parametricity proof associated with x, the new constructor comes with the following typing rule:

$$\textsc{Param}\ \frac{\Gamma \vdash A : \mathcal{U}}{\Gamma, x : A \vdash \llbracket x \rrbracket : [\![A]\!]\, x}$$

from which one can derive that

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash [\![a]\!] : [\![A]\!]\, a}$$

This effectively internalizes the parametricity theorem, since an inhabitant of $(X : \mathcal{U}) \to (x : X) \to [\![X]\!]\, x$ is given by

$$\lambda(X : \mathcal{U}).\lambda(x : X).\llbracket x \rrbracket$$

In particular, an inhabitant of (†) is given by

$$\lambda(f : (X : \mathcal{U}) \to X \to X).\ \lambda(X : \mathcal{U}).\lambda(\dot{X} : X \to \star).\lambda(x : X).\lambda(\dot{x} : \dot{X}\,x).\ \llbracket f \rrbracket\, X\, \dot{X}\, x\, \dot{x}$$

Considering the Calculus of Constructions with a universe $\mathcal{U}$ of types and another universe $\star$ of propositions, $[\![\mathcal{U}]\!]\, x \stackrel{\text{def}}{=} x \to \star$ is defined as the type of predicates on $x : \mathcal{U}$. One may remark that this definition forbids nested parametricity. Indeed if $A : \mathcal{U}$ and $x : A$ then $[\![A]\!]\, x : \star$ hence the system does not permit the application of the Param rule on the $[\![A]\!]$ itself.

However by the Curry-Howard isomorphism parametricity proofs are also programs, hence one might want them to compute, or even derive the parametricity condition of their type, which is not permitted by the above solution.

In a system with fully internalized parametricity there is nothing preventing nested uses of the parametricity theorem. However as shown by Bernardy and Moulin [2012] and as hinted at below, this yields technical difficulties. For instance one could use two instances of the above Param rule to type the term $[\![\lambda(x : A).[\![x]\!]]\!]$ (which proves that parametricity itself is parametric). On the one hand since one has $\lambda(x : A).[\![x]\!] : (x : A) \to [\![A]\!]\, x$, one obtains the type of $[\![\lambda(x : A).[\![x]\!]]\!]$ by application of the Param rule:

$$[\![(x : A) \to [\![A]\!]\, x]\!]\, [\![x]\!] \quad = \quad (x : A) \to (\dot{x} : [\![A]\!]\, x) \to [\![[\![A]\!]]\!]\, x\, \dot{x}\, [\![x]\!]$$

(using the fact that the parametric interpretation duplicates bindings and applications). On the other hand, the term $[\![\lambda(x : A).[\![x]\!]]\!]$ is not in normal form, and naively reduces to $\lambda(x : A).\lambda(\dot{x} : [\![A]\!]\, x).[\![[\![x]\!]]\!] = \lambda(x : A).\lambda(\dot{x} : [\![A]\!]\, x).[\![\dot{x}]\!]$, which is of type

$$(x : A) \to (\dot{x} : [\![A]\!]\, x) \to [\![[\![A]\!]]\!]\, x\, [\![x]\!]\, \dot{x}.$$

(Indeed the body of the function is a parametricity witness of level 2, so it is natural to apply $[\![\cdot]\!]$ on $\dot{x}$ which is a parametricity witness of level 1; moreover since $\dot{x} : [\![A]\!]\, x$, Param yields $[\![\dot{x}]\!] : [\![[\![A]\!]\, x]\!]\, \dot{x} = [\![[\![A]\!]]\!]\, x\, [\![x]\!]\, \dot{x}$.)

Hence the previous term has two types, which differ in that the iterated parametricity type $[\![[\![A]\!]]\!]$ is used with its second and third arguments swapped. As shown by Bernardy and Moulin [2012], the problem comes from the fact that the syntax does not support a way to associate x with $\dot{x}$ and "remember" that the latter is a parametricity proof of the former. One could bind them as a pair $(x, \dot{x}) : (x : A) \times ([\![A]\!]\, x)$ instead, using $(x : A) \times B$ for the dependent sum $\Sigma x : A.B$. For nested parametricity, it would then be tempting to use a tuple

$$(x, \dot{x}, \ddot{x}, \ldots) : (x : A) \times (\dot{x} : [\![A]\!]\, x) \times (\ddot{x} : [\![[\![A]\!]]\!]\, (x, \dot{x}, \dot{x})) \times \ldots$$

(the length of which being the nesting level of applications of parametricity). However, as hinted at above, this does not work due to the swapping problem; one has to nest pairs as well, leading to an exponential

structure and a combinatorial explosion:

$$\begin{aligned}
(x_0, x_1) &: (x_0 : A) \times (\llbracket A \rrbracket\, x_0) \\
((x_{00}, x_{01}), (x_{10}, x_{11})) &: (x_0 : (x_{00} : A) \times (\llbracket A \rrbracket\, x_{00})) \times \\
&\qquad ((x_{10} : \llbracket A \rrbracket\, (\pi_1 x_0)) \times (\llbracket \llbracket A \rrbracket \rrbracket\, (x_0, x_{10}))) \\
&\vdots
\end{aligned}$$

Since the nested pairs are well-balanced, at level n the left-hand side can be seen as a hypercube of dimension n. (Each application of parametricity adds a dimension to the cube, doubling the number of its vertices.) Bernardy and Moulin [2012] have developed such a system with hypercube bindings and applications, as well as a new operator swapping the vertices. However the manipulation of hypercubes is cumbersome, feels too ad-hoc, and makes the system too technical to be suitable for the internal language of a proof assistant. A first attempt to simplify it has been made by Bernardy and Moulin [2013]: instead of using a single parametricity operation $\llbracket \cdot \rrbracket$, in this system each occurrence of parametricity is annotated with a fresh *color*, which essentially corresponds to a name for the new dimension added to the hypercube (in other words to the outermost pair, as shown above). Moreover a *color erasure* operator allows "extraction" of terms and types of a given color, which corresponds to accessing faces or vertices of hypercubes. The Param rule is modified accordingly to enforce the freshness condition. The color annotations allow the system to distinguish between multiple applications of parametricity. For instance, considering

$$\begin{aligned}
p &: (x : A) \to \llbracket A \rrbracket\, x \\
p &\stackrel{\text{def}}{=} \lambda (x : A).\, \llbracket x \rrbracket
\end{aligned}$$

an application of the Param rule yields

$$\begin{aligned}
\llbracket p \rrbracket &: \llbracket (x : A) \to \llbracket A \rrbracket\, x \rrbracket\, p \\
&= (x : A) \to (\dot{x} : \llbracket A \rrbracket\, x) \to \llbracket \llbracket A \rrbracket\, x \rrbracket\, (p\, x) \\
&= (x : A) \to (\dot{x} : \llbracket A \rrbracket\, x) \to \llbracket \llbracket A \rrbracket \rrbracket\, x\, \dot{x}\, \llbracket x \rrbracket
\end{aligned}$$

while one the other hand one can reduce $\llbracket \cdot \rrbracket$ to get

$$\begin{aligned}
\llbracket p \rrbracket &= \lambda (x : A).\, \lambda (\dot{x} : \llbracket A \rrbracket\, x).\, \llbracket \llbracket x \rrbracket \rrbracket \\
&= \lambda (x : A).\, \lambda (\dot{x} : \llbracket A \rrbracket\, x).\, \llbracket \dot{x} \rrbracket
\end{aligned}$$

thus by two applications of the Param rule one obtains

$$\begin{aligned}
\llbracket p \rrbracket &: (x : A) \to (\dot{x} : \llbracket A \rrbracket\, x) \to \llbracket \llbracket A \rrbracket\, x \rrbracket\, \dot{x} \\
&= (x : A) \to (\dot{x} : \llbracket A \rrbracket\, x) \to \llbracket \llbracket A \rrbracket \rrbracket\, x\, \llbracket x \rrbracket\, \dot{x}
\end{aligned}$$

Unlike for the non annotated version, the conversion relation can be defined in such a way that the types $\llbracket \llbracket A \rrbracket \rrbracket\, x\, \dot{x}\, \llbracket x \rrbracket$ and $\llbracket \llbracket A \rrbracket \rrbracket\, x\, \llbracket x \rrbracket\, \dot{x}$ are convertible.

Outline

This thesis consists of three chapters, each representing a different stage in our work aiming at a smooth integration of parametricity with dependent types. (The initial stage [Bernardy and Moulin, 2011] of our work, which can be seen as an intermediate step between [Bernardy et al., 2012] and chapter 1, is not included in this thesis.) In particular, chapters 1 and 2 describe complex and preliminary calculi which are subsumed by chapter 3. Indeed, as we explain below, the calculus from chapter 1 feels too ad-hoc and technical, and its meta-theory is not satisfactory; while chapter 2 attempts to simplify the syntax, the meta-theory remains largely unchanged.

Chapter 1 is a copy (omitting the abstract and acknowledgements, and with minor layout changes) of my Licentiate Thesis [Moulin, 2013], which in turn is an extended version of [Bernardy and Moulin, 2012]. This thesis was defended on January 30, 2014 with Thorsten Altenkirch as discussion leader. In this chapter we show how to extend any strong enough Pure Type System (such as the Calculus of Constructions) with new rules, including a *Parametricity* Rule, which all have computational content. More specifically, we precisely define the system outlined above, where the syntax supports a notion of hypercubic structure. Then we prove that all instances of the abstraction theorem can be both expressed and proved in the calculus itself. Finally, by defining a reduction-preserving interpretation from our system to the underlying PTS, we show how to derive desired meta properties such as Church-Rosser and strong normalization.

In chapter 2, a copy (with minor layout changes) of [Bernardy and Moulin, 2013], we propose to extend the Calculus of Constructions with a notion of *colors* and a *color erasure* operator. We show how the result is a more powerful type theory: some definitions and proofs may be omitted as they become trivial; it becomes easier to program with precise types; and some propositions that were impossible to prove now become provable thanks to the erasure operator which reveals some structural invariants. We further extend the system with a Param rule; which yields the system from chapter 1 where hypercubes are kept implicit and their dimensions are being given color names instead. Finally as with chapter 1, we show via a reduction-preserving interpretation that some important properties of our extension, such as Church-Rosser and strong normalization, are inherited from that of the underlying system. We also conjecture that our extension and its properties apply to the Calculus of Inductive Constructions as well.

Chapter 3 is a major revision of [Bernardy et al., 2015]. There, we consider Martin-Löf's Logical Framework [Nordström et al., 1990] as the un-

derlying type theory, and extend it with *nominal* and *parametric* rules. Although this theory has a syntactic notion of color (which we use to distinguish between multiple iterations of parametricity) and color erasure, it differs from the extension presented in chapter 2 in that binding and typing judgments are not indexed with the set of color names in scope: instead, color names are only present in the raw syntax. This allows for a simpler system than the one presented in chapter 2. We finally equip our system with presheaf semantics following Bezem et al. [2013]; unlike the syntactic models described above, this model is compositional and provides denotational semantics.

Statement of personal contribution

As written above, this thesis is largely based on three co-authored papers. My technical contributions are as follows:

A Computational Interpretation of Parametricity

I worked together with Jean-Philippe Bernardy on the definition of the syntax, including the two novel features of the calculus, namely the hypercubes and the cube rotation operator. Proofs of confluence and the substitution lemma are mine, as well as the construction of the syntactic model. On the other hand the examples, as well as the proof of generation and subject reduction properties, are due to Jean-Philippe Bernardy only. I am also the author of the material which is not in the LICS paper but included in my Licentiate Thesis, such as the details in the proof of the abstraction theorem.

Type Theory in Color

The calculus itself and the type-checking algorithm are due to Jean-Philippe Bernardy. I am responsible for the normalization proof and am the sole author of the fork of the Agda proof assistant to include colors in its core theory.

A Presheaf Model of Parametric Type Theory

The definition of the syntax is due to Jean-Philippe Bernardy and me together, after he made the crucial remark that not requiring parametricity types to compute leads to a simpler system. While I wrote the proofs of the validity results, the idea behind the model itself, especially the I-sets and the category of colors and partial injections, are due to Thierry Coquand. I am also the author of the material which is not in the MFPS

paper but included in this chapter, in particular of the syntax modification (such as the introduction of rays) and of the proof details for the validity results.

Chapter 1

Pure type systems with an internalized parametricity theorem

Introduction

Parametricity, as formally stated by Reynolds [1983], expresses that polymorphic functions must behave *uniformly*. This is done by interpreting a type A as a relation $[\![A]\!] : A \to A \to \star$ such that $[\![A]\!]\, a\, a$ for every $a : A$. In other words this result, known as the *abstraction* theorem, says that every type gives a theorem which holds for any of its inhabitants.

The study of parametricity, starting with Reynolds' work, is typically semantic: the abstraction theorem was originally proved for types of system F, and the concern was then to construct a model capturing its polymorphic character. Later Mairson [1991], followed by Abadi et al. [1993], developed a more syntactic approach: types were interpreted in another calculus (of proofs and propositions), and for each proof term, they showed how to construct a proof term inhabiting the relational interpretation of its types.

Bernardy et al. [2010], Bernardy and Lasson [2011] have more recently shown how to extend the relational interpretation to some dependent type theories, such as the Calculus of Constructions [Coquand and Huet, 1988] or Martin-Löf's Intuitionistic Type Theory [1984]. They also show how terms, types and their relational interpretations as proofs and propositions can all be expressed in the same calculus.

For instance, in the Calculus of Constructions, the interpretation of any $f : \forall A : \star. A \to A$ gives that it must be an identity, in other words that x

is Leibniz-equal to $f\,A\,x$ for each $x : A$:

$$\forall A : \star.\, \forall x : A.\, x \equiv_A f\,A\,x \tag{1}$$

where the Leibniz equality $x \equiv_A y$ is defined (for $A : \star$ and $x, y : A$) as $\forall P : (A \to \star).\, P\,x \to P\,y$.

The notion of parametricity, in particular the abstraction theorem, is used in numerous applications when reasoning about functional programs [Wadler, 1989], for instance to prove the correctness of short-cut fusion [Gill et al., 1993, Johann, 2002]. One also needs to rely on parametricity conditions when using Church-encoding to represent datatypes [Plotkin and Abadi, 1993].

Parametricity theorems have also been used for richer calculi such as the Calculus of Inductive Constructions [Pfenning and Paulin-Mohring, 1990, Keller and Lasson, 2012], for instance to prove the correctness of well-scoped representations of λ-terms [Chlipala, 2008, Pouillard, 2011]. Indeed, an informal justification of the fact that all inhabitants of the following inductive definition (here in Agda syntax, and due to Pouillard [2011]) are well-scoped lies in the fact that *the index V is abstract*, hence the only way to introduce new variables is by abstraction.

```
data Term (V : ⋆) : ⋆ where
  var : V → Term V
  app : Term V → Term V → Term V
  abs : Term (Maybe V) → Term V
```

However, the *parametricity property*, *i.e.*, the fact that every term satisfies the parametric interpretation of its type, has not been known to be provable in the system in which the type is expressed. In particular, the following property cannot be proved in the Calculus of Constructions or Martin-Löf's Intuitionistic Type Theory, hence cannot be proved either in existing proof assistants based on these systems, such as Coq [The Coq development team, 2016] or Agda [Norell, 2007].

$$\begin{array}{l} f : \forall (A : \star).\, A \to A \\ A : \star \\ x : A \\ \hline f\,A\,x \equiv_A x \end{array}$$

(Were $\equiv$ stands for the Leibniz equality defined above.)

On the other hand, one may notice that the parametricity condition associated with the polymorphic identity is the missing assumption to prove this property.

In fact, users relying on the parametricity conditions have postulated the parametricity axiom [Chlipala, 2008, Atkey et al., 2009, Pouillard, 2011]. However, this approach has a fundamental drawback: because the postulate does not have a computational interpretation, parametricity conditions can only be used in computationally-irrelevant positions. Also, Wadler [2007] has shown that, given extensionality, induction schemes associated with datatypes can be deduced directly from their Church-encoding. However, to conveniently program with these encodings one needs to use the parametricity conditions in computationally relevant positions. For instance the natural numbers can be encoded in the Calculus of Constructions as the polymorphic type

$$\mathbf{N} \stackrel{\text{def}}{=} \forall X : \star.\, X \to (X \to X) \to X,$$

since any inhabitant n of this type cannot inspect the parameter X. The free theorem associated with any such Church Numeral $n : \mathbf{N}$ is:

$$\begin{array}{l}\forall X : \star.\, \forall P : (X \to \star).\\ \forall z : X.\, P\, z \to\\ \forall s : X \to X.\, (\forall x : X.\, P\, x \to P\, (s\, x)) \to\\ P\, (n\, z\, s)\end{array}$$

which, in extensional theories, can be used to derive the usual induction principle for $n : \mathbb{N}$ [Wadler, 2007]:

$$\forall P : (\mathbb{N} \to \star).\, (\forall m : \mathbb{N}.\, P\, m \to P\, (\mathsf{succ}\, m)) \to P\, n$$

Related work

This chapter is an extended version of [Bernardy and Moulin, 2012], and does not reflect the state of the art on Internalized Parametricity. Several related papers have been written since our paper was published in the 2012 LICS Proceedings; we briefly present a few of them below:

Keller and Lasson [2012] extended relational parametricity to the Calculus of Inductive Constructions (CIC). They added a new, non-informative, sort hierarchy inhabited by the codomain of parametric relations, which forbids nested application of parametricity. They also prove the Abstraction Theorem for CIC, and provide a Coq tactic for constructing proof terms by parametricity.

Bernardy and Moulin [2013] developed an alternative presentation of the calculus presented in this chapter, in which hypercubes are kept implicit and their dimensions (called *colors*) are *named*; the ability to abstract

over dimensions removes the need for some of the technicalities we developed earlier in [Bernardy and Moulin, 2012]. In addition, an *erasure* operator reveals some structural invariants, hence some definitions and proofs may be omitted as they become trivial. In fact, the latter paper was an attempt to make the calculus presented here easier to use as a programming language.

Krishnaswami and Dreyer [2013] built a parametric model of the Extensional Calculus of Constructions. They focus on soundness properties and show how to derive equality results (such as some common postulates on Church-encoded data) from parametricity conditions. However, the fact that parametricity is modelled in an extensional theory makes it impractical to use their model to build a programming language.

Atkey et al. [2014] describe parametric models of predicative and impredicative Dependent Type Theories in reflexive graphs, which are in turn seen as Categories with Families. In the impredicative case, they show how to take advantage of parametricity to derive the existence of initial algebras for all indexed functors.

Outline

After recalling previous work by Bernardy et al. [2010] in section 1, we show in section 2 how to extend any strong enough Pure Type System $\mathcal{O}$ (such as the Calculus of Constructions) with new rules, including a *Parametricity* Rule, which all have a computational content. More specifically, we describe a new system and show how to adapt the previous result in order to achieve internalization. In the latter subsections, we expose some of the technical problems encountered, and the solutions we found, namely the introduction of hypercubes. In section 3 we give a formal presentation of our calculus, and state and prove important meta-properties of our system. In particular, we prove that all instances of the abstraction theorem can be both expressed and proved in the calculus itself. Finally, by defining a reduction-preserving interpretation from our system to the underlying PTS $\mathcal{O}$ in section 3.6, we show how to derive desired meta properties properties such as Church-Rosser and strong normalization.

1 Proofs for free

This section is a reminder and synthesis of previous work by Bernardy et al. [2010] and Bernardy and Lasson [2011], which the present work is largely based on.

1.1 Pure type systems

Pure Type Systems (PTSs) are a family of λ-calculi, parameterized by a set of sorts $\mathcal{S}$, a set of axioms $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ and set of rules $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$. The various syntactic forms of quantifications (and corresponding abstraction and application) are syntactically unified, and one needs to inspect sorts to identify which form is meant. The axioms $\mathcal{A}$ give the typing rules for sorts, and $\mathcal{R}$ determines which forms of quantification exist in the system. Many systems (*e.g.*, the Calculus of Constructions or System F) are examples of PTSs.

The syntax of PTS terms is the following:

$$\begin{array}{lllll} \mathsf{Term} \quad \ni A, \ldots, U & \stackrel{\text{def}}{=} & s & \textit{sort} \\ & | & x & \textit{variable} \\ & | & A\,B & \textit{application} \\ & | & \lambda x : A.\, B & \textit{abstraction} \\ & | & \forall x : A.\, B & \textit{product} \end{array}$$

The product $\forall x : A.\, B$ may also be written $A \to B$ when x does not occur free in B. In the rest of this document we assume a given PTS specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, and we name the calculus arising from that specification $\mathcal{O}$. In particular, a suitable $\mathcal{O}$ is the Calculus of Constructions, the typing rules of which can be expressed in a PTS fashion by choosing the following specification: the general definition for PTSs below.

$$\begin{aligned} \mathcal{S} &= \{\star, \Box\} \\ \mathcal{A} &= \{(\star, \Box)\} \\ \mathcal{R} &= \{(\star, \star, \star),\ (\star, \Box, \Box),\ (\Box, \star, \star),\ (\Box, \Box, \Box)\} \end{aligned}$$

1.2 Logical relations, from PTS to PTS

In this section we recall the relational interpretation of terms and types of the PTS $\mathcal{O}$ into another PTS, here called $[\![\mathcal{O}]\!]$, following the construction of [Bernardy et al., 2010, Bernardy and Lasson, 2011].

In any PTS, types and terms in $\mathcal{O}$ can respectively be interpreted in $[\![\mathcal{O}]\!]$ as predicates and proofs that the terms satisfy the predicates. Each *type* can be interpreted as a predicate that its inhabitants satisfy; and each *term* can be turned into a proof that it satisfies the predicate of its type. Usual presentations of parametricity use binary relations, but for simplicity of notation we present here a unary version. The generalization to arbitrary arity is straightforward, as shown by Bernardy and Lasson [2011].

In the following we define what it means for a term C to satisfy the predicate generated by a type T (which we write $C \in [\![T]\!]$); and the translation from a program C of type T to a proof $[\![C]\!]$ that C satisfies the predicate.

$$\frac{}{\vdash s_1 : s_2}\ (s_1, s_2) \in \mathscr{A}$$
Axiom

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$$
Weakening

$$\frac{\Gamma \vdash F : (\forall x : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B[a/x]}$$
Application

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\forall x : A.\, B) : s}{\Gamma \vdash (\lambda x : A.\, b) : (\forall x : A.\, B)}$$
Abstraction

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\forall x : A.\, B) : s_3}$$
Product $(s_1, s_2, s_3) \in \mathscr{R}$

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$$
Conversion

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$$
Start

Figure 1: Typing rules of the Pure Type System specified by $(\mathscr{S}, \mathscr{A}, \mathscr{R})$

More precisely, we define (Definition 4) two mutually recursive functions $T \mapsto [\![T]\!]$ and $T \mapsto (\cdot \in [\![T]\!])$, by induction on the structure of the raw term T. These interpretations respectively take terms and types in $\mathcal{O}$, and return proofs and propositions in $[\![\mathcal{O}]\!]$. Let $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ be the specifications of the PTS $\mathcal{O}$; then those of $[\![\mathcal{O}]\!]$ are

$$\begin{aligned}\mathcal{S}' &= \mathcal{S}\\ \mathcal{A}' &= \mathcal{A}\\ \mathcal{R}' &= \{(s_1, s_2, s_2) \mid (s_1, s_2, s_3) \in \mathcal{R}\} \cup \mathcal{R}\end{aligned} \tag{2}$$

Before we formally define the interpretation, let us begin by stating the *abstraction* theorem for $\mathcal{O}$ and $[\![\mathcal{O}]\!]$: Any well-typed term of $\mathcal{O}$ is interpreted as a proof that it satisfies the parametricity condition of its type [Bernardy and Lasson, 2011].

Theorem 1 (Abstraction). *If* $\Gamma \vdash_{\mathcal{O}} A : B : s$*, then*

$$[\![\Gamma]\!] \vdash_{[\![\mathcal{O}]\!]} [\![A]\!] : (\{A\} \in [\![B]\!]) : s$$

(Where $\{A\}$ *is* A *in which each free variable* y *in* Γ *is renamed to* y_0 *in the extended context* $[\![\Gamma]\!]$*, defined below.)*

Furthermore, if $\mathcal{O}$ *is consistent, for instance if* $\mathcal{O}$ *is the Calculus of Constructions, then so is* $[\![\mathcal{O}]\!]$ *[Bernardy and Lasson, 2011].*

A property of the translation $[\![\cdot]\!]$ is that whenever $x : A$ is free in T, there are two variables x_0 and x_1 in $[\![T]\!]$, where x_1 witnesses that x_0 satisfies the parametricity condition of its type $(x_1 : x_0 \in [\![A]\!])$. This means that the translation needs to be extended to contexts, as follows:

$$\begin{aligned}[\![\Diamond]\!] &= \Diamond\\ [\![\Gamma, x : A]\!] &= [\![\Gamma]\!], x_0 : \{A\}, x_1 : x_0 \in [\![A]\!]\end{aligned}$$

It is important to notice that this definition assumes a *global* renaming from each variable x to fresh variables x_0 and x_1. (The renaming will be made local in further sections.)

A raw term T in $\mathcal{O}$ is syntactically translated, by mutual induction on its structure, to both a proof term $[\![T]\!]$ in $[\![\mathcal{O}]\!]$, and to a predicate $C \mapsto (C \in [\![T]\!])$. We separate these two interpretations in the presentation below.

- The translation of a variable is done by looking up the corresponding parametric witness in the context.

$$[\![x]\!] = x_1$$

- The case for abstraction adds a witness that the input satisfies the relational interpretation of its type and returns the relational interpretation of the body.

$$[\![\lambda x : A.\, B]\!] = \lambda x_0 : \{A\}.\, \lambda x_1 : x_0 \in [\![A]\!].\, [\![B]\!]$$

- The application follows the same pattern: the function is passed a witness that the argument satisfies the interpretation of its type.

$$[\![A\ B]\!] = [\![A]\!]\ \{B\}\ [\![B]\!]$$

- If the term has another syntactic form, namely a product or a sort, then it is a type (T). Thus we can use λ-abstraction to create a predicate and check that the abstracted variable z satisfies the relational interpretation of the type in the body ($z \in [\![T]\!]$).

$$\begin{aligned}
[\![s]\!] &= \lambda z : s.\, z \to s \\
[\![\forall x : A.\, B]\!] &= \lambda z : (\forall x_0 : \{A\}.\, \{B\}).\\
&\qquad \lambda x_0 : \{A\}.\, \lambda x_1 : x_0 \in [\![A]\!].\\
&\qquad (z\, x) \in [\![B]\!]
\end{aligned}$$

We now need to define the proposition $C \in [\![T]\!]$ which, as it can be seen in Theorem 1, is the type of $[\![C]\!]$ for any well-typed $C : T$.

- Because types in a PTS are abstract, no predicate can discriminate between them, hence any predicate over a type C can be used to witness that C satisfies the relational interpretation of its sort s.

$$C \in [\![s]\!] = C \to s$$

- If the type is a product ($\forall x : A.\, B$), then C must be a function, and it satisfies the relational interpretation of its type if and only if it maps satisfying inputs to satisfying outputs.

$$C \in [\![\forall x : A.\, B]\!] = \forall x_0 : \{A\}.\, \forall x_1 : x_0 \in [\![A]\!].\, (C\, x_0) \in [\![B]\!]$$

- For any other syntactic form for a type T, namely a variable, an application or a lambda, $C \in [\![T]\!]$ is defined using the interpretation $[\![\cdot]\!]$ given above: $C \in [\![T]\!] = [\![T]\!]\ \{C\}$.

$$\begin{aligned}
C \in [\![x]\!] &= x_1\ \{C\} \\
C \in [\![A\ B]\!] &= [\![A]\!]\ \{B\}\ [\![B]\!]\ \{C\} \\
C \in [\![\lambda x : A.\, B]\!] &= \lambda x_1 : \{C\} \in [\![A]\!].\, [\![B]\!]
\end{aligned}$$

A direct reading of Theorem 1 is as a typing judgment about translated terms: if A has type B, then $[\![A]\!]$ has type $\{A\} \in [\![B]\!]$. However, it can

also be understood as an abstraction theorem for $\mathcal{O}$: if a program A has type B in Γ, then A satisfies the relational interpretation of its type ($\{A\} \in [\![B]\!]$). Remember that $\{A\}$ is merely the term A, but using variables in $[\![\Gamma]\!]$ instead of Γ. In particular, if A is closed then $\{A\} = A$. If we were to study binary parametricity, $[\![\Gamma]\!]$ would contain two related environments (and witnesses that they are properly related). Therefore A would have two possible interpretations $\{A\}$, each obtained by picking variables out of each copy of the environment, and $[\![A]\!]$ would be a proof that the two possible interpretations of A are related.

One can show by induction on raw terms that whenever $C : T : s$, we have:

$$[\![T]\!] : \{T\} \to s \qquad C \in [\![T]\!] =_\beta [\![T]\!]\,\{C\} : s \qquad \{C\} : \{T\}$$

One may wonder why we mutually define two interpretations, since instead one could be defined from the other using the above equality. The advantage of the distinction, as described by Bernardy and Lasson [2011], is that it makes derivations in $[\![\mathcal{O}]\!]$ follow the same structure as those in $\mathcal{O}$. Indeed, if we were using the same interpretation both for types and terms, derivations in $[\![\mathcal{O}]\!]$ would be cluttered by extra uses of the conversion rule, as it was earlier presented by Bernardy et al. [2010]. Furthermore, preserving cuts makes the congruence of our model (Lemma 16) trivial.

In general the PTS $[\![\mathcal{O}]\!]$, where parametricity conditions are expressed, extends the source system $\mathcal{O}$. However, for rich enough systems, such as the calculus of constructions, they can be identical [Bernardy et al., 2010, Bernardy and Lasson, 2011]. Indeed, the PTS specifications are then closed under the parametric interpretation, presented at the beginning of this section. We now show how to extend such a system $\mathcal{O}$ to a new calculus $\mathcal{P}$ with internalized parametricity.

2 Towards internalizing parametricity

In this section we describe and motivate our system step by step, starting from a Pure Type System (such as the Calculus of Constructions) and extending it with our new constructions. In this section we gradually motivate and informally describe the system we envision. The full specification of our calculus can be found in definitions 3 to 8.

2.1 Aim and example

Let us assume a PTS $\mathcal{Q}$ satisfying (2) (*i.e.*, $\mathcal{Q} = [\![\mathcal{Q}]\!]$), such as the Calculus of Constructions. This means that both types and their parametric-

ity conditions can be expressed in $\mathcal{Q}$. Thus one can hope that for every term A of type B, we can get a witness $[\![A]\!]$ that it is parametric ($\{A\} \in [\![B]\!]$). Even though this holds for closed terms, it is not so for open terms, because the context where $[\![A]\!]$ is meaningful is "bigger" than that where A is: for each free variable $x : A$ in Γ, we need a variable $x_1 : x \in [\![A]\!]$ in $[\![\Gamma]\!]$. In other words, given $\Gamma \vdash_{\mathcal{Q}} A : B$ we have $[\![\Gamma]\!] \vdash_{\mathcal{Q}} [\![A]\!] : \{A\} \in [\![B]\!]$. However if we are to use this judgment inside a proof or a program, we are bound to the context encountered, hence we cannot extend it with explicit parametric witnesses for each free variable.

What we really want is to *derive* each free theorem rather than postulating the precise instances, and to be able to rely on parametricity conditions *in the same context*. Therefore, we need the following judgment to be valid:

$$\Gamma \vdash_{\mathcal{Q}} [\![A]\!] : A \in [\![B]\!].$$

The aim of this work is to find a system $\mathscr{P}$ such that the following proposition is verified.

Proposition 1 (Internal Parametricity)**.** *If* $\Gamma \vdash_{\mathscr{P}} A : B$*, then*

$$\Gamma \vdash_{\mathscr{P}} [\![A]\!] : A \in [\![B]\!]$$

That is, the free theorem associated with each inhabited type B can be proved in the system $\mathscr{P}$ *itself, regardless of whether B is closed or not.*

In that case, for any term A, terms of $\mathscr{P}$ can invoke the fact that A is parametric, by writing $[\![A]\!]$. The notations $[\![A]\!]$ and $A \in [\![B]\!]$ for $\mathscr{P}$ will be defined later in this section, following and extending their homonyms in $\mathcal{O}$.

Such a system would allow a full internalization of Reynolds' abstraction theorem seen in the introduction, in the sense that variables and implication no longer need to be expressed at the meta-level:

Example 1. *Assume that* $\mathscr{P}$ *extends the Calculus of Constructions. Let us consider the following instance of Internal Parametricity:*

$$\Gamma \stackrel{\text{def}}{=} f : (\forall a : \star . a \to a), a : \star, x : a \qquad A \stackrel{\text{def}}{=} f \qquad B \stackrel{\text{def}}{=} \forall a : \star . a \to a$$

Then applying internal parametricity gives:

$$\begin{aligned} f : (\forall a : \star . a \to a), a : \star, x : a \vdash_{\mathscr{P}} f : \forall a : \star . a \to a \quad \Longrightarrow \\ f : (\forall a : \star . a \to a), a : \star, x : a \vdash_{\mathscr{P}} [\![f]\!] : \forall a : \star . \forall P : a \to \star . \\ \forall x : a . P\, x \to \\ P\, (f\, a\, x) \end{aligned}$$

We are thus able to prove that any function of type $\forall a : \star . a \to a$ is an identity, as we hinted at in the introduction. The formulation of the theorem within $\mathscr{P}$ and its proof term are as follows.

$$\begin{aligned} &\mathit{identities} : \forall f : (\forall a : \star . a \to a) . \forall a : \star . \forall x : a . f\, a\, x \equiv x \\ &\mathit{identities} = \lambda f . \lambda a . \lambda x . [\![f]\!]\, a\, (\cdot \equiv x)\, x\, (\mathit{refl}\, a\, x) \end{aligned}$$

where the infix $\equiv$ stands for Leibniz equality described in the introduction, and for $a : \star$ and $x : a$, $(\cdot \equiv x)$ denotes the predicate of terms Leibniz-equal to x: $(\cdot \equiv x) \stackrel{\text{def}}{=} \forall y : a . y \equiv x$; $\mathit{refl}\, a\, x : x \equiv x$ is merely the identity function $\lambda P : (\forall x : a . \star) . \lambda p : P\, x . p$

If identities *is applied to a "concrete" identity function, such as $f = \lambda a : \star . \lambda x : a . x$, then $f\ a\ x$ reduces to x, and the theorem specializes to reflexivity of equality:*

$$\mathit{identities}\, f \ : \ \forall a : \star . \forall x : a . x \equiv x$$

After reduction, the proof no longer mentions $[\![\cdot]\!]$:

$$\begin{array}{rcl} \mathit{identities}\ i & \to_\beta & \lambda a . \lambda x . [\![f]\!] [\lambda a : \star . \lambda x : a . x / f]\ a\, (\cdot \equiv x)\, x (\mathit{refl}\, a\, x) \\ & = & \lambda a . \lambda x . [\![\lambda a : \star . \lambda x : a . x]\!]\ a\, (\cdot \equiv x)\, x (\mathit{refl}\, a\, x) \\ & = & \lambda a . \lambda x . (\lambda a . \lambda a_1 . \lambda x . \lambda x_1 . x_1)\ a\, (\cdot \equiv x)\, x (\mathit{refl}\, a\, x) \\ & \to_\beta & \lambda a . \lambda x . \mathit{refl}\, a\, x \end{array}$$

(Where $\to_\beta$ stands for the β reduction in $\mathscr{P}$, which we will define below in Definition 6.)

It is to be noted that when applying Theorem 1 instead of Internal Parametricity to the above instance, the context Γ is extended to

$$\begin{array}{ll} f : (\forall A : \star . A \to A), & f_1 : \forall A : \star . \forall P : A \to \star . \forall x : A . P\, x \to P\, (f\, A\, x), \\ A : \star, & P : A \to \star, \\ x : A, & p : P\, x \end{array}$$

Hence in $\mathit{identities}$, one cannot rely on a parametricity witness for f without asserting it. And in a proof assistant, free variables will only ever be instantiated by λ-terms, which are known to be parametric by Theorem 1.

However Theorem 1 and internal parametricity coincide on closed instances (i.e., when Γ is the empty context).

2.2 Internalization

We will now give an overview of our system $\mathscr{P}$. This system is obtained by starting from a PTS $\mathscr{O}$ such that $\mathscr{O} = [\![\mathscr{O}]\!]$, for instance the Calculus of Constructions, and adding several constructions. In the present section we give motivations for the new constructions, and present the precise

syntax and inference rules of system $\mathscr{P}$ later in definitions 3 to 8. We emphasize that the motivations given in this section are informal, and consistency of the system and other fundamental properties are proved later in sections 3.3 to 3.6.

We have seen that the abstraction theorem (Theorem 1) for PTSs gives us something very close to internal parametricity, except that for each free variable $x : A$ in Γ, we need an explicit witness that x is parametric ($x_1 : x \in [\![A]\!]$) in the environment.

However, we know that every closed term is parametric. Therefore, ultimately, we know that for each possible *concrete* term a that can be substituted for a free variable x, it is possible to construct a concrete term $[\![a]\!]$ to substitute for x_1. This means that the witness of parametricity for x does not need to be given explicitly (if x is bound). Therefore we allow to access such a witness via the new *syntactic* form $[\![x]\!]$. This intuition justifies the addition of the substitution rule

$$[\![x]\!][a/x] = [\![a]\!]$$

as well as the following typing rule, expressing that if x is found in the context, then it is valid to use $[\![x]\!]$, which witnesses that x satisfies the parametricity condition of its type.

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash [\![x]\!] : x \in [\![A]\!]}$$

Note that because of this new construction $[\![\cdot]\!]$, the system $\mathscr{P}$ that we are defining in this section is not a Pure Type System. However it extends any PTS $\mathscr{O}$ such that $\mathscr{O} = [\![\mathscr{O}]\!]$.

At the same time, we must amend the parametric interpretation to keep track of which variables have been assigned an explicit witness, and which variables must wait for a concrete term. For instance in Example 1, only the bound variables of the identity f were assigned explicit witnesses. The parametric witnesses of a free variable x is given by our new syntactic construct $[\![x]\!]$, while that of a bound variable y is picked directly from the context as y_1. Hence we need to keep track of free variables when defining the interpretation; we write the list of assignments as an index to $[\![\cdot]\!]$, and extend $A \in [\![B]\!]$ to $A \in [\![B]\!]_{\xi}$ accordingly. (From here on, we let $[\![A]\!]$ mean $[\![A]\!]_{\emptyset}$.) For example, abstraction is translated as follows:

$$[\![\lambda x : A.\, B]\!]_{\xi} = \lambda x_0 : \{A\}_{\xi}.\lambda x_1 : x_0 \in [\![A]\!]_{\xi}.\, [\![B]\!]_{\xi, x \mapsto (x_0, x_1)}$$

and other cases are modified accordingly. In particular, the interpreta-

tion of variables becomes the following[1].

$$\begin{aligned} [\![x]\!]_\xi &= x_1 && \text{if } x \mapsto (x_0, x_1) \in \xi \\ [\![x]\!]_\xi &= \llbracket x \rrbracket && \text{if } x \notin \xi \end{aligned}$$

and $\{\cdot\}$ is generalized in a similar fashion: $\{A\}_\xi$ is A where each free variable in $x \in \xi$ is replaced with x_0, while variables that are not in ξ remain untouched.

The difference of treatment between free and bound variables is illustrated in the following example:

$$\begin{aligned} x : A \vdash b : B \implies x : A \vdash [\![b]\!] : b \in [\![B]\!] \\ \vdash \lambda x : A.\, b : \forall x : A.\, B \implies \vdash \lambda x_0 : A.\lambda x_1 : x_0 \in [\![A]\!].\, [\![b]\!]_{\{x \mapsto (x_0, x_1)\}} : \\ \forall x_0 : A.\, \forall x_1 : x_0 \in [\![A]\!].\, b \in [\![B]\!]_{\{x \mapsto (x_0, x_1)\}} \end{aligned}$$

The above construction solves the issue of context extension. That is, every term A of a PTS $\mathcal{Q}$ can be proved parametric by using $[\![A]\!]$ without extending the context where A is typeable. Another aspect of the result is that, assuming parametricity on variables, the parametricity for all terms can be derived. This means that, in a language featuring parametricity, the parametric construction can be used on any term, but in normal forms, $\llbracket\cdot\rrbracket$ only appears on variables, possibly in a nested way.

Unfortunately, internal parametricity does not quite hold at this stage, after the mere extension of the original calculus with the constructor $\llbracket\cdot\rrbracket$. Indeed, as we show in the next section, Subject Reduction does not hold.

2.3 Parametricity of parametricity

Assuming that $\mathcal{P}$ has Internalized Parametricity (Proposition 1), the fact that all values are parametric is also captured by the following theorem (internalized inside the calculus):

$$\begin{aligned} &\mathit{parametricity} : \forall a : \star.\, \forall x : a.\, x \in [\![a]\!] \\ &\mathit{parametricity} = \lambda a : \star.\lambda x : a.\, \llbracket x \rrbracket \end{aligned}$$

Since all terms are assumed parametric, it should be possible to apply $[\![\cdot]\!]$ to the above term. For a closed type $A : \star$, consider the term

$$[\![\mathit{parametricity}\, A]\!] = \lambda x_0 : A.\lambda x_1 : x_0 \in [\![A]\!].\, [\![\llbracket x \rrbracket]\!]_{\{x \mapsto (x_0, x_1)\}}$$

So far, we have not defined our meta-operation $[\![\cdot]\!]_\xi$ on the new constructor $\llbracket x \rrbracket$ of our system $\mathcal{P}$ (where x is a free variable). A perhaps natural

[1] Careful readers might worry that we discard the index in the second case. An informal justification is that if x has no explicit witness, then the free variables of its type do not either; thus types are preserved by this equation.

idea is to exchange the two occurrences of the parametric interpretation, by defining $[\![\lceil x \rceil]\!]_\xi = [\![[\![x]\!]_\xi]\!]$. In our case, that leads to

$$[\![\lceil x \rceil]\!]_{\{x \mapsto (x_0, x_1)\}} = [\![[\![x]\!]_{\{x \mapsto (x_0, x_1)\}}]\!] = \lceil x_1 \rceil$$

which is a proper normal form. Unfortunately, this definition *does not preserve types* (and therefore breaks subject reduction). Indeed, assuming $x : A$, the expression $[\![\lceil x \rceil]\!]_{\{x \mapsto (x_0, x_1)\}}$ has different types before and after reduction. Internal Parametricity gives $\lceil x \rceil : [\![A]\!]\, x$. By Abstraction (giving an explicit parametric witness for x), we get

$$\begin{aligned} [\![\lceil x \rceil]\!]_{\{x \mapsto (x_0, x_1)\}} &: [\![[\![A]\!]\, x]\!]_{\{x \mapsto (x_0, x_1)\}}\, \{[\![x]\!]\}_{\{x \mapsto (x_0, x_1)\}} \\ &: [\![[\![A]\!]]\!]_{\{x \mapsto (x_0, x_1)\}}\, x_0\, [\![x]\!]_{\{x \mapsto (x_0, x_1)\}}\, \lceil x_0 \rceil \\ &: [\![[\![A]\!]]\!] x_0\, [\![x]\!]_{\{x \mapsto (x_0, x_1)\}}\, \lceil x_0 \rceil \\ &: [\![[\![A]\!]]\!] x_0\, x_1\, \lceil x_0 \rceil \end{aligned} \tag{3}$$

On the other hand, by Abstraction we have $[\![x]\!]_{\{x \mapsto (x_0, x_1)\}} : [\![A]\!]\, x_0$, and by application of Internal Parametricity, we obtain

$$\begin{aligned} [\![[\![x]\!]_{\{x \mapsto (x_0, x_1)\}}]\!] = \lceil x_1 \rceil &: [\![[\![A]\!]\, x_0]\!]\, x_1 \\ &: [\![[\![A]\!]]\!]\, x_0\, \lceil x_0 \rceil\, x_1 \end{aligned} \tag{4}$$

That is, in the above example, the reduction rule suggested above has the effect to swap the second and third arguments to $[\![[\![A]\!]]\!]$ in the type, which means that Subject Reduction would not hold if we were to have the above, naive rule.

However, one observes that, for a *closed* type A, the relation $[\![[\![A]\!]]\!]\, x$ is symmetric, *i.e.*, $[\![[\![A]\!]]\!]\, x\, B\, C$ is *isomorphic* to $[\![[\![A]\!]]\!]\, x\, C\, B$. Thus the swapping observed above is harmless, and it is sufficient to deal with it in a technical fashion.

Example 2. *For instance, the relation $[\![[\![(a : \star) \to a \to a]\!]]\!]\, f$ is symmetric for any f. That is,*

$$[\![[\![(a : \star) \to a \to a]\!]]\!]\, f\, f_1 f_2 \quad \textit{and} \quad [\![[\![(a : \star) \to a \to a]\!]]\!]\, f\, f_2 f_1$$

are isomorphic for all f_1, f_2 of type $[\![(a : \star) \to a \to a]\!]\, f$.
Indeed, $[\![[\![(a : \star) \to a \to a]\!]]\!]\, f\, f_1 f_2$ expands to

$$\begin{array}{ll} \forall a : \star. & \forall P : a \to \star. \\ & \forall Q : a \to \star.\, \forall R : (x : a) \to P\, x \to Q\, x \to \star. \\ \forall x : a. & \forall p : Q\, x. \\ & \forall q : T\, x.\, \forall r : R\, x\, p\, q. \\ & R\, (f\, a\, x)\, (f_1\, a\, P\, x\, p)\, (f_2\, a\, Q\, x\, q) \end{array}$$

If $\varphi : [\![(a : \star) \to a \to a]\!]\, f\, f_1 f_2$*, an inhabitant of*

$$[\![(a : \star) \to a \to a]\!]\, f\, f_2 f_1$$

is given by swapping the abstractions of respectively P *and* Q*, and* p *and* q*:*

$$\lambda a.\, \lambda P.\, \lambda Q.\, \lambda R.\, \lambda x.\, \lambda p.\, \lambda q.\, \lambda r.\, \varphi\, a\, Q\, P\, (\lambda x.\, \lambda p.\, \lambda q.\, R\, x\, q\, p)\, x\, q\, p\, r$$

In the light of this observation, we introduce a special-purpose operator (pronounced exchange) $\cdot \ddagger^{\pi}$, which applies the given permutation π to the arguments of relations, and which permutes their types in the same way.

$$\frac{\Gamma \vdash A : B}{\Gamma \vdash A \ddagger^{\pi} : B \ddagger^{\pi}}$$

This rule generalizes the above example to open terms and types. Indeed, when instantiated to Example 2, the new operator merely swaps the abstractions:

$$\frac{\vdash \varphi : [\![(a : \star) \to a \to a]\!]\, f\, f_1 f_2}{\vdash \varphi \ddagger^{(1,2)} : [\![(a : \star) \to a \to a]\!]\, f\, f_2 f_1}$$

Thanks to this operation we can now properly define the parametric interpretation on the constructor $[\![\cdot]\!]$, in a way that preserves types. The above situation now becomes:

$$[\![[\![x]\!]\,]\!]_{\{x \mapsto (x_0, x_1)\}} = [\![[\![x]\!]_{\{x \mapsto (x_0, x_1)\}}]\!]\, \ddagger^{(1,2)}.$$

However, supporting exchange ($\cdot \ddagger$) requires deep changes in the syntax, exposed in the next section.

2.4 A syntax for hypercubes

In order to support the swapping operation, we need to indicate the role of each of the arguments to the relations explicitly, in the syntax. To this end, we amend the abstract syntax, and introduce a new version of application where arguments are tied together in a cubical structure. For instance, the type of $[\![[\![x]\!]\,]\!]_{\{x \mapsto (x_0, x_1)\}}$, which was written before as $[\![[\![A]\!]\,]\!]\, x_0\, x_1\, [\![x_0]\!]$, is now written

$$[\![[\![A]\!]\,]\!] \bullet \begin{pmatrix} x_0 & x_1 \\ [\![x_0]\!] & \cdot \end{pmatrix}.$$

that is, the 3 arguments of the relation $[\![[\![A]\!]\,]\!]$ are tied together into an (incomplete) 2×2 matrix. Its counterpart, corresponding to the former

$[\![[\![A]\!]\, x_0\, [\![x_0]\!]\, x_1$, can now be obtained by merely transposing the matrix:

$$\left([\![[\![A]\!]\bullet\begin{pmatrix} x_0 & x_1 \\ [\![x_0]\!] & \cdot \end{pmatrix}\right)\ddagger^{(1,2)} =_\beta [\![[\![A]\!]\bullet\begin{pmatrix} x_0 & [\![x_0]\!] \\ x_1 & \cdot \end{pmatrix}$$

One could understand hypercube application as a macro denoting a $(2^n - 1)$-place application. However, we need to make this explicit in the syntax to be able to perform exchanges without extra complication of the analysis of terms. Indeed, having grouped the arguments allows us to massage them all at once in the β-reduction and parametric interpretation; however they should really be read in their "linearized" form, such as the $[\![[\![A]\!]\, x_0\, x_1\, [\![x_0]\!]$ above.

In general, we need to remember the grouping of arguments when applying the relational interpretation. Essentially, one iteration of the relational interpretation transforms an application of an argument into application of two arguments. After a second iteration, there will be four arguments, and 2^n after n iterations. (We must change the abstract syntax of application to group these 2^n arguments together.) Abstraction and product follow the same pattern as application. Hence, we can arrange our bindings as oriented n-cubes in general. Using overbar to denote cube meta-variables, the syntax becomes the following:

$$\begin{array}{rcll} \mathsf{Term} & = & A\,\bar{B} & \text{application (of hypercubes)} \\ & | & \lambda\bar{x} : \bar{A}.\, B & \text{abstraction (of hypercubes)} \\ & | & \forall\bar{x} : \bar{A}.\, B & \text{function space} \\ & | & \ldots & \end{array}$$

In the above, a binding $\bar{x} : \bar{B}$ introduces 2^n variables x_i, where i is any bit-vector of size n, and n is the dimension of $\bar{B}$. Consider the binding $\bar{x} : \bar{B}$. If $\bar{B}$ has dimension zero, it stands for a single binding $x : B$. If it has dimension 1, it contains a type B_0, and a predicate B_1 over B_0. Abusing matrix notation, one could write

$$\bar{x} : \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} \quad \text{as a shorthand for the two bindings} \quad \begin{pmatrix} x_0 : B_0 \\ x_1 : B_1 x_0 \end{pmatrix}$$

At dimension two, the cube $\bar{B}$ contains a type B_{00}, two predicates B_{01} and B_{10} over B_{00}, and a relation B_{11}, between B_{00}, $B_{10}\, x_{00}$, and $B_{01}\, x_{00}$.

$$\bar{x} : \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \quad \text{means} \quad \begin{pmatrix} x_{00} : B_{00} & x_{01} : B_{01}\, x_{00} \\ x_{10} : B_{10}\, x_{00} & x_{11} : B_{11}\, x_{00}\, x_{01}\, x_{10} \end{pmatrix}$$

Since we refer to each vertex by its position in the hypercube, we define hypercubes of dimension n as mappings from bit-vectors of length n to terms. We write

$$\left[\ i \mapsto B_i\ \right]^n_{i \in \mathbf{2}^n} \qquad \text{and} \qquad \left[\ i \mapsto B_i\ \right]^n_{i \in \mathbf{2}^n - 1}$$

respectively for plain and incomplete cubes (those that lack an element at index 1...1, called top index in the following) of dimension n.

We furthermore need a special syntax for the introduction, elimination and formation of relations, which correspond to application, abstraction and quantification over incomplete cubes. Such a cube is found for example in the type of x_{11} above. Using a check $\check{\cdot}$ to denote incomplete cube (*i.e.*, one of those with $2^n - 1$ vertices) meta-variables:

$$\begin{array}{rcll} \mathsf{Term} & = & A \bullet \check{B} & \text{relation membership} \\ & | & \lambda^{\bullet} \check{x} : \check{A}.\, B & \text{relation formation} \\ & | & \check{A} \xrightarrow{\bullet} s^n & \text{relation space} \\ & | & \ldots & \end{array}$$

Using this syntax, we can finally write the type of x_{11}, previously linearized as $B_{11}\, x_{00}\, x_{01}\, x_{10}$, in the form we need: $B_{11} \bullet \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & \cdot \end{pmatrix}$. The type of B_{11} is $\begin{pmatrix} B_{00} & B_{01} \\ B_{10} & \cdot \end{pmatrix} \xrightarrow{\bullet} s$. For a plain cube $\bar{B}$ of arbitrary dimension, we have $x_{1\ldots1} : B_{1\ldots1} \bullet (\bar{x}/1\ldots1)$ and $B_i : (\bar{B}/i) \xrightarrow{\bullet} s$, where $\bar{B}/1\ldots1$ denotes the cube $\bar{B}$ with the top vertex removed. Further generalizing, x_i is a witness that the sub-cube found by removing all the dimensions d such that $i_d = 0$ satisfies the relation B_i:

$$x_i : B_i \bullet (\bar{x}/i)$$

where $\bar{B}/i$ is the cube obtained by discarding the elements of the cube $\bar{B}$ for each dimension d where $i_d = 0$, and then removing the top vertex.

$$\bar{B}/i = \left[\ j \mapsto B_{j \,\&\, i}\ \right]_{j \in \mathbf{2}^{|i|} - 1}^{\|i\|}$$

where $\|i\| = \sum_d i_d$ and $\&$ is the pointwise and between bitvectors:

$$\begin{array}{ll} (bj)\&(0i) & = 0(j\&i) \\ (bj)\&(1i) & = b(j\&i) \end{array}$$

B_i is then a relation over the corresponding sub-cube of $\bar{B}$, which is written formally:

$$B_i : (\bar{B}/i) \xrightarrow{\bullet} s$$

Remark. *In this presentation, free theorems, or more generally logical relations, can only take an incomplete cube (of $2^n - 1$ vertices) as argument, whereas their proofs involve applications of full cubes. In particular, partial applications of a parametric relation are not allowed. We should also stress that the syntax is an extension of the underlying PTS, which can be recovered by restricting to cubes of dimension zero.*

2.5 The interpretation of hypercubes

Having given the new syntax of terms, we can express the relational interpretation using this new syntax. The interpretation of a cube increases its dimension; to each element is associated its interpretation:

$$[\![\bar{A}]\!]_{\bar{\xi}} = \left[\begin{array}{l} 0i \mapsto \{A_i\}_{\bar{\xi}} \\ 1i \mapsto [\![A_i]\!]_{\bar{\xi}} \end{array}\right]_{i \in \mathbf{2}^{\dim \bar{A}}}^{\dim \bar{A}+1}$$

If a binding $\bar{x}$ has been extended by the interpretation, a variable x_i is then interpreted as x_{1i}.

$$[\![x_i]\!]_{\bar{\xi},x} = x_{1i}$$

The interpretation of terms mentioning full cubes (of size 2^n for some n) is the following:

$$\begin{aligned} [\![A\,\bar{B}]\!]_{\bar{\xi}} &= [\![A]\!]_{\bar{\xi}}\,[\![\bar{B}]\!]_{\bar{\xi}} \\ [\![\lambda\bar{x} : \bar{A}.\,B]\!]_{\bar{\xi}} &= \lambda\bar{x} : [\![\bar{A}]\!]_{\bar{\xi}}.\,[\![B]\!]_{\bar{\xi},x\mapsto(x_0,x_1)} \\ C \in [\![\forall\bar{x} : \bar{A}.\,B]\!]_{\bar{\xi}} &= \forall\bar{x} : [\![\bar{A}]\!]_{\bar{\xi}}.\,(C\,(\bar{x}/01\ldots1)) \in [\![B]\!]_{\bar{\xi},x\mapsto(x_0,x_1)} \end{aligned}$$

The interpretation of the cubes of size $2^n - 1$ used for relations requires some care. Because the index 1...1 is missing in such a cube, applying the same method as for full cubes leaves two elements missing, at indices 1...1 and 01...1. The former is supposed to be missing (because the resulting cube is also incomplete), but the latter is dependent on the context. Hence we introduce the following notation for interpretation of incomplete cubes where the "missing element" is explicitly specified to be B:

$$([\![\check{A}]\!]_{\bar{\xi}} \oplus B) = \left[\begin{array}{l} 0i \mapsto \{A_i\}_{\bar{\xi}} \\ 1i \mapsto [\![A_i]\!]_{\bar{\xi}} \\ 01\ldots1 \mapsto B \end{array}\right]_{i \in \mathbf{2}^{\dim \check{A}}-1}^{\dim \check{A}+1}$$

The parametric interpretation of the special forms for relation formation, membership and product are as follows[2].

$$\begin{aligned} C \in [\![\check{A} \overset{\bullet}{\to} s]\!]_{\bar{\xi}} &= ([\![\check{A}]\!]_{\bar{\xi}} \oplus C) \overset{\bullet}{\to} s \\ C \in [\![A \bullet \check{B}]\!]_{\bar{\xi}} &= [\![A]\!]_{\bar{\xi}} \bullet ([\![\check{B}]\!]_{\bar{\xi}} \oplus C) \\ [\![\lambda^\bullet \check{x} : \check{A}.\,B]\!]_{\bar{\xi}} &= \lambda^\bullet \check{x} : ([\![\check{A}]\!]_{\bar{\xi}} \oplus (\lambda^\bullet \check{x} : \check{A}.\,B)).\\ &\qquad x_{01\ldots1} \in [\![B]\!]_{\bar{\xi},x\mapsto(x_0,x_1)} \end{aligned}$$

[2]Note that the missing element (the right hand-side of the $\oplus$ operator) is always a subterm of the expression we start with.

They are a straightforward consequence of the usual parametric interpretation and our choice of grouping arguments in cubes. Readers familiar with realizability interpretations for the Calculus of Constructions (in the style for example of [Paulin-Mohring, 1989]) will notice a similarity here: the interpretation of a function space adds a quantification; and the other forms behave accordingly. Note that the form $A \bullet \check{B}$ is always a type, and therefore we interpret it as such.

We now revisit nested parametricity (presented above in section 2.3):

Example 3 (Nested application of $[\![\cdot]\!]$)**.**

$$[\![\mathit{parametricity}\, A]\!] = \lambda \bar{a} : [\![(A)]\!].\, \llbracket a_1 \rrbracket \ddagger^{(01)}$$

where $\bar{a} : [\![(A)]\!]$ *can be understood as* $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} : \begin{pmatrix} A \\ a_0 \in [\![A]\!] \end{pmatrix}$*. There we have on the one hand*

$$\begin{aligned} [\![\mathit{parametricity}\, A]\!] : (\mathit{parametricity}\, A) &\in [\![\forall a : A.\ \in [\![A]\!]]\!] \\ &= (\lambda a : A.\, \llbracket a \rrbracket) \in [\![\forall a : A.\ \in [\![A]\!]]\!] \\ &= \forall \bar{a} : [\![(A)]\!].\, [\![A]\!]^2 \bullet \begin{pmatrix} a_0 & a_1 \\ \llbracket a_0 \rrbracket & \cdot \end{pmatrix} \end{aligned}$$

while on the other hand

$$a_1 : a_0 \in [\![A]\!] = [\![A]\!] \bullet \begin{pmatrix} a_0 \\ \cdot \end{pmatrix}$$

hence

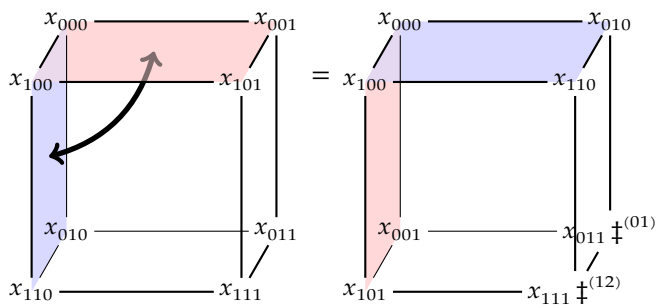
$$\begin{aligned} \llbracket a_1 \rrbracket \ddagger^{(01)} : (a_1 \in [\![a_0 \in [\![A]\!]]\!])\ddagger^{(01)} &= [\![A]\!]^2 \bullet \begin{pmatrix} a_0 & \llbracket a_0 \rrbracket \\ a_1 & \cdot \end{pmatrix} \ddagger^{(01)} \\ &= [\![A]\!]^2 \bullet \begin{pmatrix} a_0 & a_1 \\ \llbracket a_0 \rrbracket & \cdot \end{pmatrix} \end{aligned}$$

So Subject Reduction no longer fails as it did for the system without hypercubes presented in section 2.3.

2.6 Exchanging dimensions

Given the above definition of cubes, we can take advantage of the fact that vertices are tied to the structure and define an operation that applies an arbitrary permutation of its dimensions. For dimension $n = 0$ or

$n = 1$, there is no non-trivial permutation. In the case of a square ($n = 2$), there is a single non-trivial permutation, which is a simple swapping of the elements at indices 01 and 10. For higher dimensions ($n \geq 3$), the elements of the cube are multidimensional themselves (the element at index i has dimension $\|i\|$). Thus, one must take care to perform the exchange properly for each element. For instance, performing an exchange of dimensions 1 and 2 in a cube $\bar{x}$ for $n = 3$ involves exchanging dimensions 0 and 1 of the element x_{011}. Indeed, exchanging the dimensions 1 and 2 in the cube has the effect of exchanging dimensions in the square occupied by x_{011}; so an exchange has to be performed on x_{011} to restore the cube structure. Geometrically, exchanging the dimensions as above corresponds to twisting the cube: two faces are swapped, and the two other are twisted. The situation is shown graphically in the following picture.



In general, applying a permutation π on the dimensions of a cube $\bar{C}$ is done as follows:

Definition 1 (Cube exchange)**.**

$$\bar{C}\ddagger^{\pi} = \left[\ i \mapsto C_{\pi(i)}\ddagger^{\pi/i}\ \right]_{i \in \mathbf{2}^{\dim \bar{C}}}^{\dim \bar{C}}$$

Where π/i stands for the permutation π restricted to the dimensions d where $i_d = 1$.
Incomplete cubes are permuted in the same way (simply omitting the top vertex).

Definition 2. *If π is a permutation $\{d \mapsto x_d\}$,*
$\pi/i = \text{canon}\{d \mapsto x_d \mid i_d = 1\}$, where canon *maps the domain and codomain of the function $\{d \mapsto x_d \mid i_d = 1\}$ to the set $\{0..\|i\| - 1\}$, preserving the order. Renaming the dimensions in the permutation ensures that sub-cubes can be treated just like normal cubes.*

Example 4. *If $\pi = \{0 \mapsto 0, 1 \mapsto 2, 2 \mapsto 1\}$ swaps dimensions 1 and 2, we have*

i	$\{d \mapsto \pi(d) \mid i_d = 1\}$	π/i
001	$\{2 \mapsto 1\}$	$\{0 \mapsto 0\}$
010	$\{1 \mapsto 2\}$	$\{0 \mapsto 0\}$
100	$\{0 \mapsto 0\}$	$\{0 \mapsto 0\}$
011	$\{1 \mapsto 2, 2 \mapsto 1\}$	$\{0 \mapsto 1, 1 \mapsto 0\}$
101	$\{0 \mapsto 0, 2 \mapsto 1\}$	$\{0 \mapsto 0, 1 \mapsto 1\}$
110	$\{0 \mapsto 0, 1 \mapsto 2\}$	$\{0 \mapsto 0, 1 \mapsto 1\}$

Applying a permutation to terms is then a matter of permuting all the cubes encountered:

$$\begin{aligned}
(A\,\bar{B})\,\ddagger^{\pi}_{\xi} &= A\,\ddagger^{\pi}_{\xi}\,\bar{B}\,\ddagger^{\pi}_{\xi} \\
(\lambda\bar{x} : \bar{A}.\,B)\,\ddagger^{\pi}_{\xi} &= \lambda\bar{x} : \bar{A}\,\ddagger^{\pi}_{\xi}.B[\bar{x}\,\ddagger^{\pi}/\bar{x}]\,\ddagger^{\pi}_{\xi,x} \\
(\forall\bar{x} : \bar{A}.\,B)\,\ddagger^{\pi}_{\xi} &= \forall\bar{x} : \bar{A}\,\ddagger^{\pi}_{\xi}.B[\bar{x}\,\ddagger^{\pi}/\bar{x}]\,\ddagger^{\pi}_{\xi,x}
\end{aligned}$$

(and similarly for the incomplete cubes). It remains to explain the interaction with the special constructs, $[\![\cdot]\!]$ and $\cdot\ddagger$ itself. We do so by listing four laws which hold in our calculus.

The first law is not surprising: the composition of exchanges is the exchange of the composition.

$$A\,\ddagger^{\rho}\,\ddagger^{\pi} =_{\beta} A\,\ddagger^{\pi\circ\rho} \tag{5}$$

Regarding the interactions between $[\![\cdot]\!]$ and $\cdot\ddagger^{\pi}$, recall first that the relational interpretation adds one dimension to cubes. By convention, the dimension added by $[\![\cdot]\!]$ is at index 0, and all other dimensions are shifted by one. Therefore, the relational interpretation of an exchange merely lifts the exchange out, and shifts indices by one in its permutation, leaving dimension 0 intact.

$$[\![A\,\ddagger^{\pi}]\!] =_{\beta} [\![A]\!]\,\ddagger^{\pi+1} \tag{6}$$

where $\pi + 1$ denotes the permutation $\{d \mapsto \pi(d-1) \mid 0 < d \leq \dim\pi\}$.

The law that motivates the introduction of exchanges is the following:

$$[\![[\![A]\!]_{\xi}]\!] =_{\beta} [\![[\![A]\!]_{\xi}]\!]\,\ddagger^{(01)} \tag{7}$$

This law can also be explained by the convention that $[\![\cdot]\!]$ always increases each existing dimension and inserts a new dimension 0. By commuting the uses of parametricity, dimensions are swapped, and the exchange operator restores the order.

Last, one can also simplify exchanges in the presence of symmetric terms. We know that a term $[\![A]\!]^n$ is symmetric in its n first dimensions. Thus,

applying a permutation that touches only dimensions $0..n-1$ to such a term has no effect. Formally, we have:

$$[\![A]\!]^n \ddagger^{(x_1\ x_2 \ldots x_m)} =_\beta [\![A]\!]^n \qquad \text{if } \forall i \in 1..m, x_i < n \tag{8}$$

We have argued before that it suffices to provide parametricity only for variables, and that the construct $[\![\cdot]\!]$ acts as a "macro" on other constructs. The situation is the same in the presence of dimension exchanges: (6) explains how to compute the parametricity witness of an exchange. For the $\cdot \ddagger^\pi$ construct, the situation is analogous: it suffices to provide the construct for variables, possibly enclosed by $[\![\cdot]\!]$ themselves, while it is a macro on all other forms.

The reason is that the above laws give a way to compute the exchange for any term which is not a parametricity witness (the result is given in Definition 5). When we want to be explicit about exchange being the syntactic construct, we write simply $\mathbf{x} \dagger^\pi$. The syntax fragment for parametricity and exchange is as follows.

Var	$\in x, y, z$			
Param	$\in \mathbf{x}$	$\stackrel{\text{def}}{=}$	x	*variable*
		$\mid$	$[\![\mathbf{x}]\!]$	*parametric witness*
Term	$\in a, \ldots, u$	$\stackrel{\text{def}}{=}$	$\mathbf{x} \dagger^\pi$	*permutation of dimensions*
		$\mid$	$\ldots$	

2.7 Dimension checks

If a permutation acts on dimensions 0 to $n-1$, then every cube where it is applied to *must* exhibit at least n dimensions. So far we have not discussed this restriction, which is the final feature of the system to present. To implement it we choose to amend the syntax and annotate sorts with the dimension of the type which inhabits it. Since the sort s at dimension n is written s^n, we can capture the restriction in the following exchange rule.

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B : s^n}{\Gamma \vdash A \ddagger^\pi : B \ddagger^\pi} \dim(\pi) \leq n$$

(However, as with the Param rule, only the version where the term A is a *variable* is added to our typing rules; this is enough, since the general rule can be derived.)

If a type inhabits a sort of dimension n, then all the quantifications found inside the type must be over cubes of dimension at least n. This is enforced by modifying the product rule as follows:

$$\frac{\Gamma \vdash \bar{A} : s_1^n \qquad \Gamma, \bar{x} : \bar{A} \vdash B : s_2^m}{\Gamma \vdash (\forall \bar{x} : \bar{A}.\, B) : s_3^{m \sqcap n}}$$
$$\textsc{Product}\ (s_1, s_2, s_3) \in \mathscr{R}$$

Similarly, relations found in the type must be over cubes of dimension n.

3 A calculus with an internal parametricity theorem

Having concluded the informal presentation of our system $\mathscr{P}$, we now focus on a detailed description and will end with proofs of some fundamental meta-properties such as confluence (Theorem 3), strong normalization (Theorem 9), and consistency (Theorem 8).

3.1 Definitions

We start this section with the full definition of system $\mathscr{P}$, parameterized on a PTS specification $(\mathscr{S}, \mathscr{A}, \mathscr{R})$.

Definition 3 (Abstract syntax of $\mathscr{P}$)**.**

Sort	$\ni s, s_1, s_2, s_3$	$\stackrel{\text{def}}{=}$	$\mathscr{S}$	
Var	$\in x, y, z$			
Param	$\in \mathbf{x}$	$\stackrel{\text{def}}{=}$	x	*variable*
		$\mid$	$[\![\mathbf{x}]\!]$	*parametric witness*
Term	$\in a, \ldots, u$	$\stackrel{\text{def}}{=}$	$\mathbf{x}\dagger^{\pi}$	*permutation of dimensions*
	$A, \ldots, U$	$\mid$	s^n	*sort at dimension n*
		$\mid$	$A\,\bar{B}$	*application (of hypercubes)*
		$\mid$	$\lambda\bar{x} : \bar{A}.\, B$	*abstraction (of hypercubes)*
		$\mid$	$\forall\bar{x} : \bar{A}.\, B$	*function space*
		$\mid$	$A \bullet \check{B}$	*relation membership*
		$\mid$	$\lambda^{\bullet}\check{x} : \check{A}.\, B$	*relation formation*
		$\mid$	$\check{A} \stackrel{\bullet}{\to} s^n$	*relation space*
Cube	$\ni \bar{A}$	$\stackrel{\text{def}}{=}$	$\left[\; i \mapsto A_i \;\right]^n_{i \in \mathbf{2}^n}$	*cube of size* 2^n
Cube′	$\ni \check{A}$	$\stackrel{\text{def}}{=}$	$\left[\; i \mapsto A_i \;\right]^n_{i \in \mathbf{2}^n - 1}$	*cube of size* $2^n - 1$
Context	$\ni \Gamma, \Delta$	$\stackrel{\text{def}}{=}$	$\diamond$	*empty context*
		$\mid$	$\Gamma, x : A$	*context extension*

Where $\left[\; i \mapsto A_i \;\right]^n_{i \in \mathbf{2}^n}$ *(resp.* $\left[\; i \mapsto A_i \;\right]^n_{i \in \mathbf{2}^n - 1}$*) denotes a balanced binary tree (resp. a balanced binary tree without the lower-right leaf) of depth n, where for each bit-vector i of length n, the vertex* A_i *is the leaf reached from the root by following the left child on 1's and right one on 0's.*

The cube bindings can be defined formally once we introduce some convenient notations:

- $\mathbf{2}^n$ *stands for all bit-vectors of size n; and* $\mathbf{2}^n - 1$ *stands for all bit-vectors of size n, except* 1…1*.*

- $\mathrm{ind}(\bar{A})$ *stands for* $\mathbf{2}^{\dim \bar{A}}$*; and* $\mathrm{ind}(\check{A})$ *stands for* $\mathbf{2}^{\dim \check{A}} - 1$.

- $\bar{x} : \bar{A}$ *stands for the bindings* $x_i : A_i\bullet(\bar{x}/i)$ *where* $i \in \mathrm{ind}(\bar{A})$*; and* $\check{x} : \check{A}$ *stands for the bindings* $x_i : A_i\bullet(\check{x}/i)$ *where* $i \in \mathrm{ind}(\check{A})$.

- *Similarly,* $\bar{A} : s^n$ *stands for* $A_i : \bar{A}/i \xrightarrow{\bullet} s^{|i|}$ *and* $\check{A} : s^n$ *stands for* $A_i : \check{A}/i \xrightarrow{\bullet} s^{|i|}$.

Definition 4 (Relational interpretation of raw terms)**.**

$$
\begin{aligned}
[\![\lceil x \rceil^n]\!]_\xi &= \lceil x \rceil^{n+1} \qquad \textit{(in particular, } [\![x]\!]_\xi = \lceil x \rceil \textit{ for } n = 0) && \textit{if } x \notin \xi \\
[\![\lceil x_i \rceil^n]\!]_\xi &= \lceil x_{1i} \rceil^n \dagger^{(0..n)} \textit{ (in particular, } [\![x_i]\!]_\xi = x_{1i} \textit{ for } n = 0) && \textit{if } x \in \xi \\
[\![\mathbf{x}\dagger^\pi]\!]_\xi &= [\![\mathbf{x}]\!]_\xi \dagger^{\pi+1} \\
[\![\lambda\bar{x} : \bar{A}.\, B]\!]_\xi &= \lambda\bar{x} : [\![\bar{A}]\!]_\xi .\, [\![B]\!]_{\xi, x\mapsto(x_0,x_1)} \\
[\![\lambda^\bullet\check{x} : \check{A}.\, B]\!]_\xi &= \lambda^\bullet\check{x} : ([\![\check{A}]\!]_\xi \oplus (\lambda^\bullet\check{x} : \check{A}.\, B)).\\
&\qquad\qquad x_{01\ldots1} \in [\![B]\!]_{\xi, x\mapsto(x_0,x_1)} \\
[\![A\,\bar{B}]\!]_\xi &= [\![A]\!]_\xi\, [\![\bar{B}]\!]_\xi \\
[\![T]\!]_\xi &= \lambda\check{z} : \begin{pmatrix} T \\ \cdot \end{pmatrix}. z_0 \in [\![T]\!]_\xi && \textit{if } T \textit{ is } \forall, \bullet \textit{ or } s^n
\end{aligned}
$$

$$
\begin{aligned}
C \in [\![s^n]\!]_\xi &= \begin{pmatrix} C \\ \cdot \end{pmatrix} \xrightarrow{\bullet} s^{n+1} \\
C \in [\![\forall\bar{x} : \bar{A}.\, B]\!]_\xi &= \forall\bar{x} : [\![\bar{A}]\!]_\xi .\, (C\,(\bar{x}/01\ldots1)) \in [\![B]\!]_{\xi, x\mapsto(x_0,x_1)} \\
C \in [\![\check{A} \xrightarrow{\bullet} s^n]\!]_\xi &= ([\![\check{A}]\!]_\xi \oplus C) \xrightarrow{\bullet} s^{n+1} \\
C \in [\![A\bullet\check{B}]\!]_\xi &= [\![A]\!]_\xi\bullet([\![\check{B}]\!]_\xi \oplus C) \\
C \in [\![T]\!]_\xi &= [\![T]\!]_\xi\bullet\begin{pmatrix} C \\ \cdot \end{pmatrix} && \textit{if } T \textit{ is not } \forall, \bullet \textit{ nor } s^n
\end{aligned}
$$

$$
\begin{aligned}
[\![\diamond]\!]_\xi &= \diamond \\
[\![\Gamma, x : A]\!]_{\xi, x\mapsto(x_0,x_1)} &= [\![\Gamma]\!]_\xi, x_0 : A, x_1 : x_0 \in [\![A]\!]_\xi && \textit{if } x \in \xi \\
[\![\Gamma, x : A]\!]_\xi &= [\![\Gamma]\!]_\xi, x : A && \textit{if } x \notin \xi
\end{aligned}
$$

$$
\begin{aligned}
[\![\bar{A}]\!]_\xi &= \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto [\![A_i]\!]_\xi \end{array}\right]^{\dim\bar{A}+1}_{i\in\mathbf{2}^{\dim\bar{A}}} \\
([\![\check{A}]\!]_\xi \oplus B) &= \left[\begin{array}{l} 0i \mapsto \{A_i\}_\xi \\ 1i \mapsto [\![A_i]\!]_\xi \\ 01\ldots1 \mapsto B \end{array}\right]^{\dim\check{A}+1}_{i\in\mathbf{2}^{\dim\check{A}}-1}
\end{aligned}
$$

Definition 5 (Term exchange)**.**

$$
\begin{aligned}
[\![x]\!]^n \dagger^\rho \ddagger^\pi_\xi &= [\![x]\!]^n \dagger^\rho && \text{if } x \in \xi \\
[\![x]\!]^n \dagger^\rho \ddagger^\pi_\xi &= [\![x]\!]^n \dagger^{\mathrm{normal}_n(\pi \circ \rho)} && \text{if } x \notin \xi \\
(A\,\bar{B})\ddagger^\pi_\xi &= A\ddagger^\pi_\xi\ \bar{B}\ddagger^\pi_\xi \\
(\lambda\bar{x} : \bar{A}.\,B)\ddagger^\pi_\xi &= \lambda\bar{x} : \bar{A}\ddagger^\pi_\xi.B[\bar{x}\ddagger^\pi/\bar{x}]\ddagger^\pi_{\xi,x} \\
(\forall\bar{x} : \bar{A}.\,B)\ddagger^\pi_\xi &= \forall\bar{x} : \bar{A}\ddagger^\pi_\xi.B[\bar{x}\ddagger^\pi/\bar{x}]\ddagger^\pi_{\xi,x} \\
(A\bullet\check{B})\ddagger^\pi_\xi &= A\ddagger^\pi_\xi \bullet\bar{B}\ddagger^\pi_\xi \\
(\lambda^\bullet\check{x} : \check{A}.\,B)\ddagger^\pi_\xi &= \lambda^\bullet\check{x} : \check{A}\ddagger^\pi_\xi.B[\bar{x}\ddagger^\pi/\bar{x}]\ddagger^\pi_{\xi,x} \\
(\check{A} \overset{\bullet}{\to} s^n)\ddagger^\pi_\xi &= \check{A}\ddagger^\pi_\xi \overset{\bullet}{\to} s^n \\
s^n\ddagger^\pi_\xi &= s^n
\end{aligned}
$$

Where $\mathrm{normal}_n(\pi)$ *removes all cycles of* π *entirely contained in* $0..n-1$*.*

The β-reduction of the underlying PTS extends naturally to hypercube redexes.

Definition 6 (Reduction)**.**

$$
\begin{aligned}
(\lambda\bar{x} : \bar{A}.\,b)\,\bar{a} &\longrightarrow b[\bar{a}/\bar{x}] \\
(\lambda^\bullet\check{x} : \check{A}.\,b)\bullet\check{a} &\longrightarrow b[\check{a}/\check{x}]
\end{aligned}
$$

Where $b[\bar{a}/\bar{x}]$ *(resp.* $b[\check{a}/\check{x}]$*) denotes the* $2^{\dim \bar{a}}$ *(resp.* $2^{\dim \check{a}} - 1$*) substitutions* $b[x_i/a_i \mid i \in \mathbf{2}^{\dim \bar{a}}]$ *(resp.* $b[x_i/a_i \mid i \in \mathbf{2}^{\dim \check{a}} - 1]$*).*
We do not specify a reduction strategy, and the β*-reduction* $\longrightarrow$ *can be applied anywhere in a term, including under abstraction or application.*

We write $=_\beta$ *the reflexive, symmetric, transitive closure of the reduction* $\longrightarrow$*.*

Definition 7 (Substitution)**.** *In addition to the usual congruence rules, we extend the substitution meta-operation to our two new syntactic constructs.*

$$
\begin{aligned}
[\![x]\!][a/x] &= [\![a]\!]_\emptyset \\
x\dagger^\pi[a/x] &= a\ddagger^\pi_\emptyset
\end{aligned}
$$

Definition 8 (Typing rules of $\mathscr{P}$)**.**

$$\frac{}{\vdash s_1^n : s_2^n}\ (s_1, s_2) \in \mathscr{A} \qquad \textsc{Axiom}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s^n}{\Gamma, x : C \vdash A : B} \qquad \textsc{Weakening}$$

$$\frac{\Gamma \vdash F : (\check{A} \overset{\bullet}{\to} s^n) \qquad \Gamma \vdash \check{a} : \check{A}}{\Gamma \vdash F \bullet \check{a} : s^n} \qquad \textsc{Rel-elim}$$

$$\frac{\Gamma, \check{x} : \check{A} \vdash B : s^n \qquad \Gamma \vdash \check{A} : s^n}{\Gamma \vdash (\lambda^{\bullet} \check{x} : \check{A}.\, B) : (\check{A} \overset{\bullet}{\to} s^n)} \qquad \textsc{Rel-intro}$$

$$\frac{\Gamma \vdash \check{A} : s_1^n}{\Gamma \vdash (\check{A} \overset{\bullet}{\to} s_1^n) : s_2^n}\ (s_1, s_2) \in \mathscr{A} \qquad \textsc{Rel-form}$$

$$\frac{\Gamma \vdash F : (\forall \bar{x} : \bar{A}.\, B) \qquad \Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash F\, \bar{a} : B[\bar{a}/\bar{x}]} \qquad \textsc{Application}$$

$$\frac{\Gamma, \bar{x} : \bar{A} \vdash b : B \qquad \Gamma \vdash (\forall \bar{x} : \bar{A}.\, B) : s^n}{\Gamma \vdash (\lambda \bar{x} : \bar{A}.\, b) : (\forall \bar{x} : \bar{A}.\, B)} \qquad \textsc{Abstraction}$$

$$\frac{\Gamma \vdash \bar{A} : s_1^n \qquad \Gamma, \bar{x} : \bar{A} \vdash B : s_2^m}{\Gamma \vdash (\forall \bar{x} : \bar{A}.\, B) : s_3^{m \sqcap n}}\ (s_1, s_2, s_3) \in \mathscr{R} \qquad \textsc{Product}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s^n \qquad B =_\beta B'}{\Gamma \vdash A : B'} \qquad \textsc{Conversion}$$

$$\frac{\Gamma \vdash A : s^n}{\Gamma, x : A \vdash x : A} \qquad \textsc{Start}$$

$$\frac{\Gamma \vdash \mathbf{x} : A}{\Gamma \vdash [\![\mathbf{x}]\!] : \mathbf{x} \in [\![A]\!]_\emptyset} \qquad \textsc{Param}$$

$$\frac{\Gamma \vdash \mathbf{x} : A \qquad \Gamma \vdash A : s^n}{\Gamma \vdash \mathbf{x} \dagger^\pi : A \ddagger^\pi}\ \dim(\pi) \leq n \qquad \textsc{Exchange}$$

Where the typing judgment $\Gamma \vdash \bar{a} : \bar{A}$ *stands for the conjunction of the judgments* $\Gamma \vdash a_i : A_i \bullet (\bar{a}/i)$ *for* $i \in \mathrm{ind}(\bar{A})$*; and* $\Gamma \in \check{a} : \check{A}$ *stands for the conjunction of the judgments* $\Gamma \vdash a_i : A_i \bullet (\check{a}/i)$ *for* $i \in \mathrm{ind}(\check{A})$*.*

The syntactic changes made to the system require results to be adapted accordingly. In the case of Example 1, (proving that any function of type $\forall a : \star.\, a \to a$ is an identity), the definition of Equality must be amended to make it inhabit $\star^1$. This mostly involves augmenting the dimension of cubes by adding unit types as indices:

$$Eq : \forall a : \star . a \to \begin{pmatrix} a \\ \cdot \end{pmatrix} \overset{\bullet}{\to} \star^1$$

$$Eq = \lambda a : \star . \lambda x : A . \lambda^{\bullet} \begin{pmatrix} y \\ \cdot \end{pmatrix} : \begin{pmatrix} a \\ \cdot \end{pmatrix} . \forall \begin{pmatrix} - \\ P \end{pmatrix} : \left(\begin{pmatrix} a \\ \cdot \end{pmatrix} \overset{\top}{\overset{\bullet}{\to}} \star^1 \right) .$$
$$\left(P \bullet \begin{pmatrix} \top \\ \begin{pmatrix} x \\ \cdot \end{pmatrix} \end{pmatrix} \right) \to P \bullet \begin{pmatrix} y \\ \cdot \end{pmatrix}$$

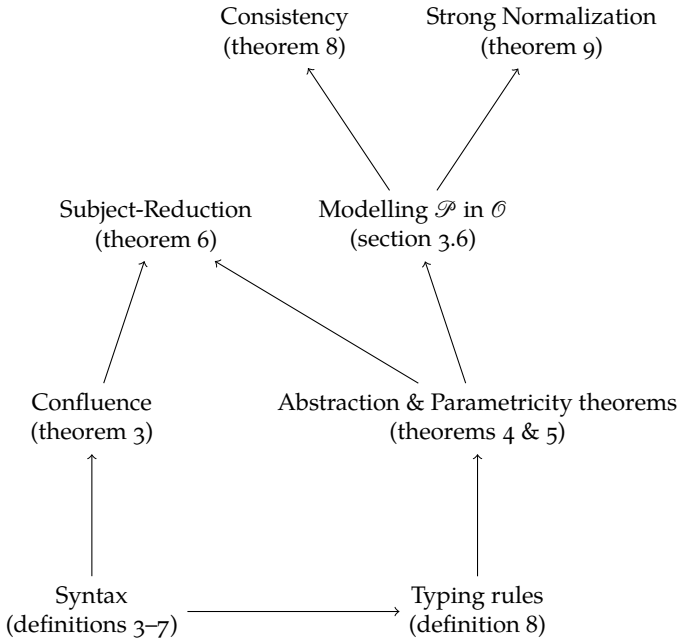
The proof term needs fewer amendments:

$$identities : \forall f : (\forall a : \star . a \to a) .$$
$$\forall a : \star . \forall x : a . Eq\, a\, (f\, a\, x) \bullet \begin{pmatrix} x \\ \cdot \end{pmatrix}$$
$$identities = \lambda f . \lambda a . \lambda x . . [\![f]\!] \begin{pmatrix} a \\ Eq\, a\, x \end{pmatrix} \begin{pmatrix} x \\ refl\, a\, x \end{pmatrix}$$

$$[\![f]\!] : \forall \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} : \begin{pmatrix} \star \\ \overset{\bullet}{\to} \\ a_0 \to \star^1 \end{pmatrix} . \forall \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} : \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} . a_1 \bullet (f a_0 x_0)$$

We have now defined our system $\mathscr{P}$. In the remainder of this section we prove the main meta-theoretic results about the system. More precisely, we prove confluence in section 3.3, the abstraction and parametricity theorems in section 3.4, and subject reduction in section 3.5. We then define in section 3.6 a reduction-preserving interpretation of $\mathscr{P}$ into the underlying PTS $\mathscr{O}$, hence model the former in the latter. This model is done by introducing explicit witnesses of parametricity for all variables. Provided that consistency and strong normalization hold for $\mathscr{O}$ (for instance when $\mathscr{O}$ is the Calculus of Constructions), we can then derive from the model that they also hold for our system $\mathscr{P}$.

Dependencies between these results can be summarized by the following directed graph:



3.2 Properties of the parametric interpretation

We start by proving weakening and commutation lemmas for our parametric interpretation. These lemmas are used to prove confluence (section 3.3), and abstraction and parametricity theorems (section 3.4).

Lemma 1. *For each term A and each variable z not free in A, we have:*

i) $\llbracket A \rrbracket_{\xi, z \mapsto (z_0, z_1)} = \llbracket A \rrbracket_{\xi}$; *and*

ii) $\{a\}_{\xi, z \mapsto (z_0, z_1)} \in \llbracket A \rrbracket_{\xi, z \mapsto (z_0, z_1)} = \{a\}_{\xi} \in \llbracket A \rrbracket_{\xi}$ *for all terms a.*

Proof. By simultaneous induction on the structure of the raw term A. Details can be found in appendix A. □

$\cdot \dagger^{\pi}$ commutes with $\llbracket \cdot \rrbracket$, but the permutation needs to be lifted.

Lemma 2. *For each term $A : s^m$ and each ρ of dimension at most m, we have:*

$$\llbracket A \ddagger_{\zeta}^{\rho} \rrbracket_{\xi} = \llbracket A \rrbracket_{\xi} \ddagger_{\zeta}^{1+\rho}$$

Proof. By induction on the structure of the raw term A. Details can be found in appendix A. □

When exchanging two occurrences of the parametric interpretation, one needs to permute the cube variables that are explicit in both interpretations:

Lemma 3. *For each term A, we have:*

$$[\![[\![A]\!]_{\vec{\xi}}]\!]_{\vec{\zeta}} = [\![[\![A]\!]_{\vec{\zeta}}]\!]_{\vec{\xi}}[\bar{x}\dagger^{(01)}/\bar{x} \mid x \in \vec{\xi} \cap \vec{\zeta}]\ddagger^{(01)}_{\vec{\xi}\cap\vec{\zeta}}$$

Proof. By structural induction on the raw term A. Details can be found in appendix A. □

In particular, when $\vec{\xi}$ is empty:

Corollary 1. *For each term A, we have:*

$$[\![[\![A]\!]^m]\!]_{\vec{\zeta}} = [\![[\![A]\!]_{\vec{\zeta}}]\!]^m \ddagger^{(0\ldots m)}$$

Note that (7) *is a special case of this result, taking $m = 0$.*

The parametric interpretation commutes with the substitution, but a special treatment is required when the variable to be substituted for is either free or known to the interpretation.

Lemma 4 ($[\![\cdot]\!]$ and substitution, part 1)**.** *For each term A, and each variable z not in $\vec{\xi}$, we have:*

i) $[\![A[u/z_i]]\!]_{\vec{\xi}} = [\![A]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]$; *and*

ii) $\{a[u/z_i]\}_{\vec{\xi}} \in [\![A[u/z_i]]\!]_{\vec{\xi}} = (\{a\}_{\vec{\xi},z} \in [\![A]\!]_{\vec{\xi},z})[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]$.

Lemma 5 ($[\![\cdot]\!]$ and substitution, part 2)**.** *For each term A, for variable z not free in A or contained in $\vec{\xi}$, we have:*

i) $[\![A[u/z_i]]\!]_{\vec{\xi}} = [\![A]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}]$; *and*

ii) $\{a[u/z_i]\}_{\vec{\xi}} \in [\![A[u/z_i]]\!]_{\vec{\xi}} = (\{a\}_{\vec{\xi}} \in [\![A]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]$.

Proof. By simultaneous induction on the structure of the raw term A. Details can be found in appendix A. □

The last lemma of this section states that our parametric interpretation *uniformly* expands cubes:

Lemma 6 (Symmetry)**.** *For each term A, $[\![A]\!]^n$ is symmetric in its n first dimensions. More specifically,*

i) $[\![A]\!]^n_{\vec{\xi}}\ddagger^{\pi}_{\vec{\xi}} = [\![A]\!]^n_{\vec{\xi}}\ddagger^{\mathrm{normal}_n(\pi)}_{\vec{\xi}}$; *and*

ii) $(a \in [\![A]\!]^n_{\vec{\xi}})\ddagger^{\pi}_{\vec{\xi}} = (a \in [\![A]\!]^n_{\vec{\xi}})\ddagger^{\mathrm{normal}_n(\pi)}_{\vec{\xi}}$.

Proof. By simultaneous induction on the structure of the raw term A. Details can be found in appendix A. □

Lemma 7 ($\cdot \ddagger^{\cdot}$ and substitution)**.** *If ξ does not contain either z or any of the free variables of E, then*

$$A[E/z] \ddagger^{\pi}_{\xi} = A \ddagger^{\pi}_{\xi}[E/z] \text{ for all } \pi.$$

Proof. By induction on A. The only interesting case is the one for variables, with $x = z$:

$$\begin{aligned}
[\![z]\!]^n \dagger^{\rho}[E/z] \ddagger^{\pi}_{\xi} &= [\![E]\!]^n \ddagger^{\rho} \ddagger^{\pi}_{\xi} \\
&= [\![E]\!]^n \ddagger^{\mathrm{normal}_n(\pi \circ \rho)} \quad \text{by Lem. 6} \\
&= [\![z]\!]^n \ddagger^{\mathrm{normal}_n(\pi \circ \rho)}[E/z] \\
&= [\![z]\!]^n \dagger^{\rho} \ddagger^{\pi}_{\xi}[E/z]
\end{aligned}$$

□

Lemma 8 (Substitution)**.**

$$A[E/z][E'/z'] = A[E'/z'][E[E'/z']/z]$$

Proof. By induction on A; the only non-trivial case is for the parametric witnesses $[\![z]\!]^n$:

$$\begin{aligned}
[\![z]\!]^n \dagger^{\pi}[E/z][E'/z'] &= [\![E]\!]^n_{\emptyset}[E'/z'] \ddagger^{\pi} \\
&= [\![E[E'/z']]\!]^n_{\emptyset} \ddagger^{\pi} = [\![z]\!]^n \dagger^{\pi}[E'/z'][E[E'/z']/z]
\end{aligned}$$

by Lemmas 5 and 7. □

3.3 Confluence

We now check that the Church-Rosser property holds, that is, we verify that the order in which the reductions are performed does not matter. To prove this property, we define a *parallel reduction* (following the Tait/Martin-Löf technique), and show that the diamond property holds for this reduction.

Definition 9 (Parallel nested reduction)**.**

$$\textsc{Refl}\ \frac{}{A \rhd A}$$

$$\beta\ \frac{b \rhd b' \qquad \bar{a} \rhd \bar{a}'}{(\lambda \bar{x} : \bar{A}.\, b)\ \bar{a} \rhd b'[\bar{a}'/\bar{x}]} \qquad \beta^\bullet\ \frac{b \rhd b' \qquad \check{a} \rhd \check{a}'}{(\lambda^\bullet \check{x} : \check{A}.\, b) \bullet \check{a} \rhd b'[\check{a}'/\check{x}]}$$

$$\textsc{App-Cong}\ \frac{F \rhd F' \qquad \bar{a} \rhd \bar{a}'}{F\,\bar{a} \rhd F'\,\bar{a}'} \qquad \textsc{App}^\bullet\textsc{-Cong}\ \frac{F \rhd F' \qquad \check{a} \rhd \check{a}'}{F \bullet \check{a} \rhd F' \bullet \check{a}'}$$

$$\textsc{Abs-Cong}\ \frac{\bar{A} \rhd \bar{A}' \qquad b \rhd b'}{\lambda \bar{x} : \bar{A}.\, b \rhd \lambda \bar{x} : \bar{A}'.\, b'} \qquad \textsc{Abs}^\bullet\textsc{-Cong}\ \frac{\check{A} \rhd \check{A}' \qquad b \rhd b'}{\lambda^\bullet \check{x} : \check{A}.\, b \rhd \lambda^\bullet \check{x} : \check{A}'.\, b'}$$

$$\textsc{All-Cong}\ \frac{\bar{A} \rhd \bar{A}' \qquad B \rhd B'}{\forall \bar{x} : \bar{A}.\, B \rhd \forall \bar{x} : \bar{A}'.\, B'} \qquad \textsc{All}^\bullet\textsc{-Cong}\ \frac{\check{A} \rhd \check{A}'}{\check{A} \overset{\bullet}{\to} s^n \rhd \check{A}' \overset{\bullet}{\to} s^n}$$

With $\bar{A} \rhd \bar{A}'$ iff. for all i, $A_i \rhd A_i'$ (and similarly for $\check{A} \rhd \check{A}'$).

We now need to prove congruence lemmas for the parallel reduction $\rhd$, for each of our 3 meta-operators: parametric interpretation $[\![\cdot]\!]$, term exchange $\cdot\ddagger$, and substitution.

Lemma 9 (Congruence of $[\![\cdot]\!]$)**.** *If $A \rhd A'$, then for all ξ,*

i) $[\![A]\!]_\xi \rhd [\![A']\!]_\xi$; *and*

ii) $a \in [\![A]\!]_\xi \rhd a' \in [\![A']\!]_\xi$ *for all $a \rhd a'$.*

Proof. By induction on $A \rhd A'$:

- The case of Refl is trivial.
- For β, one expects

  $$[\![(\lambda \bar{x} : \bar{A}.\, b)\ \bar{a}]\!]_\xi \rhd [\![b'[\bar{a}'/\bar{x}]]\!]_\xi,$$

  knowing $b \rhd b'$ and $\bar{a} \rhd \bar{a}'$.

$$\begin{array}{ll} & [\![(\lambda \bar{x} : \bar{A}.\, b)\ \bar{a}]\!]_\xi \\ = & \quad \{\text{by def. of } [\![\cdot]\!]_\xi\} \\ & (\lambda \bar{x} : [\![\bar{A}]\!]_\xi.\, [\![b]\!]_{\xi,x})\ [\![\bar{a}]\!]_\xi \\ \rhd & \quad \{\text{by } \beta, \textsc{Refl} \text{ and IH}\} \\ & [\![b']\!]_{\xi,x}[[\![\bar{a}']\!]_\xi/\bar{x}] \\ = & \quad \{\text{by Lemma 4}\} \\ & [\![b'[\bar{a}'/\bar{x}]]\!]_\xi \end{array}$$

- The case of $\beta^\bullet$ is similar.
- The cases of $\star$-Cong are straightforward using the definition of $[\![\cdot]\!]$.

□

Lemma 10 (Congruence of $\cdot \ddagger_\xi^\pi$)**.** *If $A \rhd A'$, then for all ξ and π, one has*

$$A \ddagger_\xi^\pi \rhd A' \ddagger_\xi^\pi$$

Proof. By induction on $A \rhd A'$. The only interesting cases are for the β- and $\beta^\bullet$-reductions. For β ($\beta^\bullet$ is similar), we have

$$\begin{array}{ll} & ((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a}) \ddagger_\xi^\pi \\ = & \quad \{\text{by def. of } \cdot \ddagger_\xi^\pi\} \\ & (\lambda \bar{x} : \bar{A} \ddagger_\xi^\pi .\, b[\bar{x} \ddagger_\xi^\pi / \bar{x}] \ddagger_{\xi,x}^\pi)\, \bar{a} \ddagger_\xi^\pi \\ \rhd & \quad \{\text{by } \beta, \textsc{Refl} \text{ and IH}\} \\ & b'[\bar{x} \ddagger_\xi^\pi / \bar{x}] \ddagger_{\xi,x}^\pi [\bar{a}' \ddagger_\xi^\pi / \bar{x}] \\ = & \quad b' \ddagger_{\xi,x}^\pi [\bar{a}' \ddagger_\xi^\pi / \bar{x} \ddagger_\xi^\pi] \\ = & \quad b'[\bar{a}'/\bar{x}] \ddagger_\xi^\pi \end{array}$$

□

Lemma 11 (Congruence of substitution)**.** *If $A \rhd A'$ and $E \rhd E'$, then*

$$A[E/z] \rhd A'[E'/z].$$

Proof. By induction on $A \rhd A'$:

- For Refl, the expected result follows from an induction on A (using n times Lemmas 9 and 10 for the case $[\![z]\!]^n \ddagger^\pi$).
- For β, one expects

  $$((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a})[E/z] \rhd b'[\bar{a}'/\bar{x}][E/z],$$

  knowing $b \rhd b'$ and $\bar{a} \rhd \bar{a}'$. We have

  $$\begin{array}{ll} & ((\lambda \bar{x} : \bar{A}.\, b)\, \bar{a})[E/z] \\ = & \quad \{\text{by def. of the substitution}\} \\ & (\lambda \bar{x} : A[E/z].\, b[E/z])\, \bar{a}[E/z] \\ \rhd & \quad \{\text{by } \beta \text{ and IH}\} \\ & b'[E'/z][\bar{a}'[E'/z]/\bar{x}] \\ = & \quad \{\text{by Lemma 8}\} \\ & b'[\bar{a}'/\bar{x}][E'/z] \end{array}$$

- The case of $\beta^\bullet$ is similar.

- The cases of $\star$-Cong stem from straightforward uses of induction hypotheses. □

Theorem 2 (Diamond property)**.** *The rewriting system ($\triangleright$) has the diamond property. In other words, for each A, B, B' such that $B \triangleleft A \triangleright B'$, there exists C such that $B \triangleright C \triangleleft B'$*

Proof. By induction on the derivations:

- If one of the derivations ends with Refl, one has either $A = B$, or $A = B'$. We pick $C = B'$ in the former case and $C = B$ in the latter.

- If one of the derivations ends with Abs-Cong, All-Cong, Abs•-Cong or All•-Cong, the other one has to end with the same rule, and the result is a straightforward use of the induction hypothesis.

- If one of the derivations ends with App-Cong, the other one has to end with App-Cong, or with β. The first case is straightforward; in the second one, one has

$$(\lambda \bar{x} : \bar{A}'.\, b')\, \bar{a}' \quad \triangleleft \quad (\lambda \bar{x} : \bar{A}.\, b)\, \bar{a} \quad \triangleright \quad b''[\bar{a}''/\bar{x}]$$

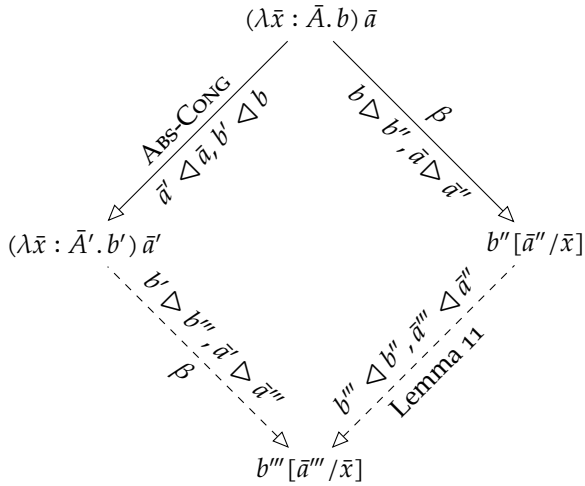
$$\text{with } \lambda \bar{x} : \bar{A}'.\, b' \quad \triangleleft \quad \lambda \bar{x} : \bar{A}.\, b, \quad b \triangleright b'' \quad \text{ and } \quad \bar{a}' \triangleleft \bar{a} \triangleright \bar{a}''$$

  The situation is summarized in the diagram below. In more details, the end of the derivation of $\lambda \bar{x} : \bar{A}'.\, b' \quad \triangleleft \quad \lambda \bar{x} : \bar{A}.\, b$ has to be either Abs-Cong, or Refl. In the first case (the last one is similar), one has $\bar{A}' \triangleleft \bar{A}$ and $b' \triangleleft b$.

  By induction hypothesis there exists b''', $\bar{a}'''$ such that $b' \triangleright b''' \triangleleft b''$ and $\bar{a}' \triangleright \bar{a}''' \triangleleft \bar{a}''$.

  The result follows by β and Lemma 11:

$$(\lambda \bar{x} : \bar{A}'.\, b')\, \bar{a}' \quad \triangleright \quad b'''[\bar{a}'''/\bar{x}] \quad \triangleleft \quad b''[\bar{a}''/\bar{x}]$$



- The case for App•-Cong is similar.
- If both derivations end with the same β or $\beta^\bullet$ rule, the result is a straightforward use of the induction hypothesis and Lemma 11. □

Theorem 3 (Church-Rosser property)**.** *Our calculus system has the confluence (Church-Rosser) property that is, for each A, B, B' such that $B \longleftarrow^\star A \longrightarrow^\star B'$, there exists C such that $B \longrightarrow^\star C \longleftarrow^\star B'$*

Proof. Direct consequence of Theorem 2, using the equality $\rhd^\star = \longrightarrow^\star$. □

3.4 Abstraction

In this section we check that our main goal, the integration of parametricity (see Proposition 1), is achieved by the design that we propose. (In particular, internalized parametricity holds for the Param rule itself.) At the same time, we check that the abstraction theorem also holds for our calculus. We do so by proving Lemma 12, which subsumes both theorems.

Lemma 12 (Generalized abstraction)**.** *Assuming that ξ conforms to Γ,*

i) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : \{A\}_\xi \in [\![B]\!]_\xi$

ii) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash \{A\}_\xi : \{B\}_\xi$

iii) $\Gamma \vdash B : s^n \;\Rightarrow\; [\![\Gamma]\!]_\xi, x : B \vdash x \in [\![B]\!]_\xi : s^{n+1}$

Proof. The proof is done by simultaneous induction on the derivation tree, and is similar to the proof of the Abstraction Theorem by Bernardy

and Lasson [2011]. The new parts occur in the special handling of the Start and Param rules. The proof of each sub-lemma can be sketched as follows (the full proof can be found in appendix A):

i) The cases of abstraction and application stem from the fact that their respective relational interpretations follow the same pattern as the relational interpretation of the product. The case of a variable x (Start) is more tricky: if $x \in \xi$, then the context contains an explicit witness of parametricity for x. This witness is used to justify the translated judgment. If $x \notin \xi$, then we can use the parametricity rule on x to translate the typing judgment. The Param rule is handled similarly, with the additional complexity that an exchange of dimensions must be added when $x \notin \xi$.

ii) This sub-lemma is used to justify weakening of contexts in the other sub-lemmas. It is a consequence of the thinning lemma and the fact that the interpretation of types in always well-typed (see the third item below).

iii) This sub-lemma expresses that if B is a well-sorted type, then so is $x \in [\![B]\!]$. It is easy to convince oneself of that result by checking that the translation of a type always yields a relation, and that the translation of a relation is itself a relation. □

Remark. *In summary, and roughly speaking, Lemma 12 replaces the occurrences of* Start *(resp.* Param*) for variables not in ξ by* Param *(resp. nested* Param + Exchange*). Occurrences on* Start *(resp.* Param*) for variables in ξ are preserved.*

Theorem 4 (Abstraction)**.**

i) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : (\{A\}_\xi \in [\![B]\!]_\xi)$*, where ξ contains all the variables in* Γ*.*

ii) *Furthermore, if the original judgment makes no use of* Param*, the resulting judgment does not either.*

Proof.

i) Direct consequence of Lemma 12i.

ii) In the proof of Lemma 12i, if ξ is full, then the target derivation trees contains Param iff. Param occurs in the derivation tree for $\Gamma \vdash A : B$. □

Theorem 5 (Parametricity)**.** *Each term, no matter if is closed or opened, satisfies the parametricity condition of its type:*

$$\Gamma \vdash A : B \;\Rightarrow\; \Gamma \vdash [\![A]\!] : (A \in [\![B]\!])$$

Proof. Take ξ empty in Lemma 12i. (We recall that $[\![\Gamma]\!]_\emptyset = \Gamma$.) □

Definition 10. *ξ conforms to Γ iff. ξ contains a suffix of Γ.*

Remark. *$[\![\cdot]\!]$ preserves conforming indices: if ξ conforms to Γ and A is well-typed in Γ, the definition of $[\![A]\!]_\xi$ makes only recursive calls with conforming substitutions.*

Proof. By induction on the typing derivation. In the definition of $[\![\cdot]\!]_\xi$, every bound variable in a term is added to the index ξ in recursive calls. □

3.5 Subject reduction

In this section we prove subject reduction (preservation of types). Since parametricity acts as a typing rule for $[\![\cdot]\!]$, subject reduction for our calculus stems directly from it. We start by discussing basic properties generally attributed to PTSs, which subject reduction (Theorem 6) depends on.

The weakening of contexts behaves in our calculus exactly in the same way as in all PTSs. Indeed, the usual thinning lemma holds.

Lemma 13 (Thinning)**.** *Let Γ and Δ be legal contexts such that $\Gamma \subseteq \Delta$. Then $\Gamma \vdash A : B \Longrightarrow \Delta \vdash A : B$.*

Proof. As in [Barendregt, 1992, Lemma 5.2.12]. □

The generation lemma for our calculus must account for the new parametricity construct.

Lemma 14 (Generation)**.** *The statement of the lemma is the same as that of the generation lemma for PTS [Barendregt, 1992, Lemma 5.2.13], but with the additional case for the* Param *rule:*

- *If $\Gamma \vdash [\![x]\!] : C$ then there exists B such that $\Gamma \vdash B : s^n$, $(x : B) \in \Gamma$, and $C =_\beta x \in [\![B]\!]$.*

Proof. As in [Barendregt, 1992]:

- We follow the derivation $\Gamma \vdash [\![x]\!] : C$ until $[\![x]\!]$ is introduced. It can only be done by the following rule

$$\frac{\Delta \vdash B : s_n}{\Delta, x : B \vdash [\![x]\!] : x \in [\![B]\!]} \textsc{Param}$$

with $C =_\beta x \in [\![B]\!]$, and $(\Delta, \bar{x} : B) \subseteq \Gamma$. The conclusion stems from Lemma 13. □

Theorem 6 (Subject Reduction)**.** *If $A \longrightarrow A'$ and $\Gamma \vdash A : T$, then*

$$\Gamma \vdash A' : T$$

Proof. Most of the technicalities of the proof by Barendregt [1992], concern β-reduction, and are not changed by our addition of parametricity.

Hence we discuss here only the handling of the parametricity construct: our task is to check that substituting a concrete term a for x in $[\![x]\!]$ preserves the type of the expression.

Facing a term such as $[\![x]\!]$ in context Γ, we know by generation that it must have type $x \in [\![B]\!]$ (for some type B valid in Γ, and $x : B$). We can then prove that substituting a term a of type B' (where B' is convertible to B) for x preserves the type of the expression. Indeed, the expression then reduces to $[\![a]\!]$, which has type $a \in [\![B']\!]$ by Theorem 5. In turn, $a \in [\![B']\!]$ is convertible to $x \in [\![B]\!]$ by Lemma 9. □

3.6 Reduction-preserving model into the underlying PTS

In this section we present a formalization of the intuitive model presented in section 2.2. We developed a "high-level" calculus $\mathscr{P}$ suitable to internalize parametricity results; we now model our system $\mathscr{P}$ into the underlying PTS $\mathscr{O}$, which can be seen as "low-level" in that context.

Each term is mapped to a term where parametricity witnesses are passed explicitly. Simultaneously, contexts are extended with explicit witnesses: in a first approximation, each binding $x : A$ is replaced by a multiple binding $x : A, \check{x} : x \in [\![A]\!]$. This means that $[\![x]\!]$ can be interpreted by the corresponding variable $\check{x}$ in the context. In fact, this is really what the term $[\![x]\!]$ means, as shown by the reduction rule $[\![x]\!][u/x] \longrightarrow [\![u]\!]$.

The following table shows how some example terms can be interpreted (for the sake of readability we omit type annotations in the abstractions, since they play no role in these examples):

original term A	its interpretation $\langle\!\langle A \rangle\!\rangle$
$\lambda x.\, [\![x]\!]$	$\lambda x.\, \lambda \check{x}.\, \check{x}$
$(\lambda x.\, [\![x]\!])\ (y\, z)$	$(\lambda x.\, \lambda \check{x}.\, \check{x})\ (y\, z)\ (\check{y}\, z\, \check{z})$
$(\lambda x.\, [\![x]\!])\ (\lambda y.\, [\![y]\!])$	$(\lambda x.\, \lambda \check{x}.\, \check{x})\ (\lambda y.\, \lambda \check{y}.\, \check{y})\ (\lambda y.\, \lambda \check{y}'.\, \lambda \check{\check{y}}.\, \check{\check{y}})$

Note that the third row in the above table shows how an instance of nested parametricity is modelled: the second argument (the parametricity witness of the first argument $\lambda y.\, \lambda \breve{y}.\, \breve{y}$), is itself expanded by adding an explicit witnesses $\breve{\breve{y}}$ of level two.

Given that the interpretation is sound with respect to $\mathcal{O}$ (Theorem 7) and that it preserves reductions (Lemma 16), we obtain strong normalization (Theorem 9). The rest of the section is devoted to defining the model formally, and proving its soundness.

In general, the transformation is not trivial, because of the interaction between functions and their arguments, occurring in the App rule. If a function uses parametricity on one of its argument, calls to the function must also compute explicit parametricity witnesses. (This may in turn trigger the need for more explicit witnesses at the call site). Furthermore, if the function is passed to another function, this will create further needs for explicit witnesses.

As we have seen above, each binding $x : A$ should be replaced by $x : A, \ldots, \breve{x}^n : x \in [\![A]\!]^n$ for some n. Our main task is to compute an n that would be big enough to make all the parametricity witnesses $[\![x]\!]^k$ explicit. To do so, we use an intermediate representation of the typing derivation, containing some *constraints* on the n's, by annotation of the derivation tree, as in figure 2. We assume without loss of generality that variable names are distinct, so the n's are given by a (partial) valuation $\epsilon : \mathsf{Var} \to \mathbb{N}$ defined on each *cube* variable. This annotation of the derivation with constraints is an instance of a technique known as type-based analysis [Svenningsson, 2007].

The Application and Rel-Elim rules require special care. Indeed, we need to "lift" the inequalities of the right sub-tree $\mathbf{t}$, since if F has to be extended to a term of type $\forall [\![x : A]\!]^n.\, B$, then it has to be fed with n extra parametricity witnesses $[\![a]\!] \cdots [\![a]\!]^n$, hence the context has to be extended enough to contain $\breve{y}^n$, for each y free in a. Note that the constraints $e + \epsilon(x) \leq \epsilon(y)$ we add in the Application and Rel-Elim rule are more restrictive than the corresponding $e \leq \epsilon(y)$ that are in $\mathbf{t}$, so one can simply ignore the latter.

We need to check that the system of constraints has a solution. In fact, the simplex it defines is unbounded: indeed, the only place where a variable appears on the left-hand side of a constraint is in Application and Rel-Elim when we "lift by x" the constraints in the sub-tree $\mathbf{t}$; it cannot create any cycle, since x does not appear in $\mathbf{t}$.

With our notion of cubes instead of usual bindings, extending the context with an explicit witness corresponds to adding one dimension to the cube. However, we *a priori* only need to access one of the new vertices, the one for which the new dimension is set to one. Hence in general, each of the $2^{\dim \bar{A}}$ vertices x_i of a binding $\bar{x} : \bar{A}$ will be extended with

$$\frac{\Gamma \vdash F : (\forall \bar{x} : \bar{A}.\, B) \qquad \mathfrak{t} :: \Gamma \vdash \bar{a} : \bar{A} \qquad \{e + \epsilon(x) \leq \epsilon(y) \mid e \leq \epsilon(y) \in \mathfrak{t}\}}{\Gamma \vdash F\,\bar{a} : B[\bar{a}/\bar{x}]}$$
Application

$$\frac{\Gamma \vdash F : (\forall^{\bullet} \check{x} : \check{A}.\, s^n) \qquad \mathfrak{t} :: \Gamma \vdash \check{a} : \check{A} \qquad \{e + \epsilon(x) \leq \epsilon(y) \mid e \leq \epsilon(y) \in \mathfrak{t}\}}{\Gamma \vdash F{\bullet}\check{a} : s^n}$$
Rel-Elim

$$\frac{\Gamma \vdash A : s^m \qquad n \leq \epsilon(x)}{\Gamma, x : A \vdash [\![x]\!]^n \ddagger^{\pi} : (x \in [\![A]\!]^n) \ddagger^{\pi}} \dim \pi \leq m + n$$
Param/n

Figure 2: Typing rules extended with constraints on the valuation ϵ. Rules omitted here remain unchanged (see Definition 8). The notation $e \leq \epsilon(y) \in \mathfrak{t}$ expresses that the constraint appears in the sub-derivation $\mathfrak{t}$. (For the sake of conciseness, we merged the rules Start, Param and Exchange into Param/n here.)

$x_{ji} : x_i \in [\![A_i{\bullet}(\bar{x}/i)]\!]^k$ for $0 \leq k \leq \epsilon(x)$ and $j = 0^{\epsilon(x)-k}1^k$.

Permutations on variables yield yet another difficulty, as one can see in the example $\lambda x : A.\lambda y_1 : x \in [\![A]\!].\, [\![y_1]\!] \ddagger^{(12)}$. (The cubes have been flattened for the sake of readability.) Here, $[\![y_1]\!] \ddagger^{(12)} : [\![A]\!]^2\, x\, y_1\, [\![x]\!]$ while $[\![y_1]\!] : [\![A]\!]^2\, x\, [\![x]\!]\, y_1$. Our solution is to not only extend the context with explicit parametricity witnesses, but also with explicit *permuted* parametricity witnesses. Hence a possible interpretation of the previous term in the naked system $\mathcal{O}$ is the following:

$$\begin{array}{l} \lambda x_0 : A.\lambda x_1 : ([\![A]\!]\, x_0). \\ \qquad \lambda y_{01} : ([\![A]\!]\, x_0).\lambda y_{11} : ([\![A]\!]^2\, x_0\, x_1\, y_{01}). \\ \qquad\qquad \lambda y_{11}^{(12)} : ([\![A]\!]^2\, x_0\, y_{01}\, x_1). \quad y_{11}^{(12)} \end{array}$$

We are not focusing on the minimal extension here, and we add witnesses for each possible permutation. It is however possible to refine this extension, since for instance the relations are symmetric in the new dimensions, hence we can ignore permutation cycles that are entirely contained in these new dimensions.

Definition 11 (Interpretation which inserts explicit witnesses)**.** *Writing*

$\mathfrak{S}_n$ *to be the group of permutations on* $\{0, \dots, n-1\}$,

$$
\begin{aligned}
\langle\!| s^n |\!\rangle &= s \\
\langle\!| [\![x_i]\!]^n \dagger^\pi |\!\rangle &= x^\pi_{ji} \quad \textit{where } j = \overbrace{0 \dots 0}^{\epsilon(x)} \underbrace{1 \dots 1}_{n} \\
\langle\!| \lambda \bar{x} : \bar{A}. B |\!\rangle &= \lambda \langle\!| \bar{x} : \bar{A} |\!\rangle . \langle\!| B |\!\rangle \\
\langle\!| \forall \bar{x} : \bar{A}. B |\!\rangle &= \forall \langle\!| \bar{x} : \bar{A} |\!\rangle . \langle\!| B |\!\rangle \\
\langle\!| F\ ^x \bar{a} |\!\rangle &= \langle\!| F |\!\rangle \ \{ \langle\!| [\![a_i]\!]^k \ddagger^\pi |\!\rangle \mid i \in \mathrm{ind}(\bar{a}), k \leq \epsilon(x), \\
&\qquad\qquad \pi \in \mathfrak{S}_{k+\dim s \bar{a}} \ \} \\
\langle\!| \lambda^\bullet \check{x} : \check{A}. B |\!\rangle &= \lambda \langle\!| \check{x} : \check{A} |\!\rangle . \langle\!| B |\!\rangle \\
\langle\!| \forall^\bullet \check{x} : \check{A}. s^n |\!\rangle &= \forall \langle\!| \check{x} : \check{A} |\!\rangle . s \\
\langle\!| F \bullet^x \check{A} |\!\rangle &= \langle\!| F |\!\rangle \ \{ \langle\!| [\![a_i]\!]^k \ddagger^\pi |\!\rangle \mid i \in \mathrm{ind}(\check{a}), k \leq \epsilon(x), \\
&\qquad\qquad \pi \in \mathfrak{S}_{k+\dim s \check{a}} \ \} \\
\hline
\langle\!| \diamond |\!\rangle &= \diamond \\
\langle\!| \Gamma, x_i : A |\!\rangle &= \langle\!| \Gamma |\!\rangle, \langle\!| x_i : A |\!\rangle
\end{aligned}
$$

We introduce a new macro $\langle\!| x_i : A |\!\rangle$*, which expands to the following multiple bindings:*

$$
\begin{aligned}
\langle\!| x_i : A : s^n |\!\rangle = \{ x^\pi_{ji} : \langle\!| (x_i \in [\![A]\!]^k) \ddagger^\pi |\!\rangle \mid\ & k \leq \epsilon(x), \\
& j = 0^{\epsilon(x)-k} 1^k, \\
& \pi \in \mathfrak{S}_{k+n} \ \}
\end{aligned}
$$

Bindings of cube variables are merely "flattened", using our previously defined macro:

$$
\begin{aligned}
\langle\!| \bar{x} : \bar{A} |\!\rangle &= \{ \langle\!| x_i : A_i \bullet (\bar{x} / i) |\!\rangle \mid i \in \mathrm{ind}(\bar{A}) \} \\
\langle\!| \check{x} : \check{A} |\!\rangle &= \{ \langle\!| x_i : A_i \bullet (\check{x} / i) |\!\rangle \mid i \in \mathrm{ind}(\check{A}) \}
\end{aligned}
$$

The essence of the model defined by $\langle\!| \cdot |\!\rangle$ is that a parametricity witness $[\![x_i]\!]^n \dagger^\pi$ is adequately modeled by the variable $x^\pi_{0\dots01\dots1i}$, that is, if x has type A, then $x_1 : x \in [\![A]\!]$, etc.

Lemma 15 ($\langle\!| \cdot |\!\rangle$ and substitution)**.**

$$
\langle\!| A[a_i/x_i] |\!\rangle = \langle\!| A |\!\rangle [\langle\!| [\![a_i]\!]^k \ddagger^\pi |\!\rangle / x^\pi_{ji}, k \leq \epsilon(x), j = 0^{\epsilon(x)-k} 1^k, \pi \in \dots]
$$

Proof. By induction on A; we illustrate how the proof proceeds by showing only the case for variables, since all the other cases stem from straight-

forward uses of the induction hypotheses.

$$\begin{aligned}
\langle\!\langle [\![x_i]\!]^n \dagger^\pi [a_i/x_i] \rangle\!\rangle &= \langle\!\langle [\![a_i]\!]^n \ddagger^\pi \rangle\!\rangle \\
&= x^\pi_{ji}[\langle\!\langle [\![a_i]\!]^n \ddagger^\pi \rangle\!\rangle / x^\pi_{ji}] \\
&= \langle\!\langle [\![x_i]\!]^n \dagger^\pi \rangle\!\rangle [\langle\!\langle [\![a_i]\!]^k \ddagger^\pi \rangle\!\rangle / x^\pi_{ji}, \dots] \qquad \square
\end{aligned}$$

Lemma 16 (Congruence of $\langle\!\langle \cdot \rangle\!\rangle$)**.** *If* $\Gamma \vdash A : B$ *with* $A \longrightarrow A'$*, then*

$$\langle\!\langle A \rangle\!\rangle \longrightarrow^+ \langle\!\langle A' \rangle\!\rangle.$$

Proof. By induction on $A \longrightarrow A'$. $\square$

Finally, we are now able to prove the soundness of our model, by proving that the transformation yields well-typed terms in $\mathcal{O}$.

Theorem 7 (Soundness)**.** *If* $\Gamma \vdash_{\mathscr{P}} A : B$*, then*

$$\langle\!\langle \Gamma \rangle\!\rangle \vdash_{\mathcal{O}} \langle\!\langle A \rangle\!\rangle : \langle\!\langle B \rangle\!\rangle.$$

Proof. We proceed by induction on the derivation; however the proof requires a stronger induction hypothesis when the derivation $\Gamma \vdash_{\mathscr{P}} A : B$ starts with the Application rule. See appendix A for details. $\square$

Theorem 8 (Consistency)**.** *If* $\mathcal{O}$ *is consistent, then so is* $\mathscr{P}$*, our system extended with* Param*.*

Proof. Since $\langle\!\langle \cdot \rangle\!\rangle$ transports the empty type from $\mathscr{P}$ to $\mathcal{O}$, by Theorem 7 any inhabitant of the empty type in $\mathscr{P}$ would give one in $\mathcal{O}$. $\square$

Theorem 9 (Strong Normalization)**.** *If* $\mathcal{O}$ *is strongly normalizing, then so is* $\mathscr{P}$*.*

Proof. Assume $\Gamma \vdash A : B$ and consider a chain of reductions $A \longrightarrow^n A'$. We have $\langle\!\langle A \rangle\!\rangle \longrightarrow^m \langle\!\langle A' \rangle\!\rangle$, and $m \geq n$ by Lemma 16. We also have that $\langle\!\langle A \rangle\!\rangle$ is typeable in $\mathcal{O}$, by Theorem 7. Therefore, only finite chains of reductions are possible. $\square$

Chapter 2

Type theory in color

1 Introduction

Intelligent use of color in a written argument can go a long way into conveying an idea. But how convincing can it really be? Consider the case of the computer scientist Philip Wadler, who is fond of using color in his papers. On multiple occasions, Wadler [2003, 2007, 2012] presents a programming language and its type-system, and shows that, by erasing the appropriate parts of the type-system, a logic appears. This is done by a straightforward but clever use of colors. Typically, in the presentation of a typing rule, the program parts are written in blue. The corresponding logic rule appears if one erases that color. As Wadler suggests, one can see the erasure simply by putting on blue glasses.

$$\frac{f : A \to B \qquad u : A}{f\ u : B} \qquad\qquad \frac{A \to B \qquad A}{B}$$

Typing rule for application — After erasure: *modus ponens*

The relationship between the programming language and the logic is deep: for every aspect of the language, there is a "blue part" that can be erased away to obtain the corresponding logical concept. For example, a computation step on the programming side yields a cut-elimination step on the logic side. That is, Wadler does not play mindlessly with colors, he is consistent; he follows in fact a precise (however unwritten) logic of colors[1]. In fact, his "proofs by putting on glasses" are extremely compelling.

In this paper, we set-out to formalize this kind of reasoning with colors as an extension of dependent type theory. The result is a more powerful

[1] A meta-level logic, not to be confused with the object logics studied by Wadler.

type theory: some definitions and proofs may be omitted as they become trivial; it becomes easier to program with precise types; and some propositions that were impossible to prove now become provable.

In section 2 we demonstrate how one can program and reason with colors via a number of examples, and introduce the fundamental concepts of type theory with color at the same time. In section 3, we describe CCCC (the main technical contribution of this paper), a core calculus of constructions with colors, and prove meta-theoretical properties (subject-reduction, normalization). In section 4, we discuss some possible extensions of CCCC. Related work is discussed in section 5, and we conclude in section 6.

2 Programming and reasoning with colors

In this section we explain how we envision a full-featured type theory with color (TTC) would be designed, in the form of a short tutorial. We assume familiarity with a proof assistant based on type theory such as Coq [The Coq development team, 2016] or Agda [Norell, 2007].

2.1 Colored lists

We start with an example similar in structure, but significantly simpler than those presented by Wadler. In the standard inductive definition of lists, the structure of the list does not depend on the elements it contains. Hence, it makes sense to color the elements: erasing that color yields a meaningful definition. In fact, the result is structurally equal to the usual unary representation of natural numbers:

$$\mathbb{N} = \lfloor \mathsf{List}\ a \rfloor_i$$

(In this section we assume a color i that we render in blue. Even though we strongly recommend reading the colored version, we index an i-tainted typing with i, so readers can make sense of what follows even if it is printed in black and white.)

$$\begin{array}{l}
\mathit{data}\ \mathsf{List}\ (a :_i \star) : \star \\
\quad \mathsf{stop} : \mathsf{List}\ a \\
\quad \mathsf{more} : (x :_i a) \to \mathsf{List}\ a \to \mathsf{List}\ a \\
\mathit{data}\ \mathsf{List} \quad\quad : \star \\
\quad \mathsf{stop} : \mathsf{List} \\
\quad \mathsf{more} : \quad\quad \mathsf{List} \to \mathsf{List}
\end{array}$$

As in Wadler's examples, the relationship between colored objects and their erasure carries over everywhere. For example, erasing color from a given list yields its length — and the typing relation is preserved.

$$\begin{aligned}&\mathsf{more}\ 'b'\ (\mathsf{more}\ 'l'\ (\mathsf{more}\ 'u'\ \mathsf{stop})) : \mathsf{List}\ \mathsf{Char}\\&\mathsf{more}\quad(\mathsf{more}\quad(\mathsf{more}\quad\mathsf{stop})) : \mathsf{List}\end{aligned}$$

Concatenation yields addition; assuming $a :_i \star$:

$$\begin{aligned}&(+) : \mathsf{List}\ a \to \mathsf{List}\ a \to \mathsf{List}\ a\\&\mathsf{stop} \quad + \quad x_s \quad = \quad x_s\\&\mathsf{more}\ x\ x_s \quad + \quad y_s \quad = \quad \mathsf{more}\ x\ (x_s + y_s)\\&(+) : \mathsf{List} \quad \to \mathsf{List} \quad \to \mathsf{List}\\&\mathsf{stop} \quad + \quad x_s \quad = \quad x_s\\&\mathsf{more}\quad x_s \quad + \quad y_s \quad = \quad \mathsf{more}\quad (x_s + y_s)\end{aligned}$$

This structural relation is a benefit of abiding to color discipline: colorless parts shall never refer to tainted ones, and in return one gets some equalities for free. For example, the length of the concatenation is the addition of the lengths. This proposition requires a proof in Agda or Coq, but thanks to colors, it holds by definition. Writing $\lfloor t \rfloor_i$ for the i-erasure of t, we have

$$\lfloor x_s + y_s \rfloor_i = \lfloor x_s \rfloor_i + \lfloor y_s \rfloor_i$$

In fact, TTC does not have a special purpose operator for erasure: the context determines whether variables refer to complete objects or to their erasures. For example in the following signature, the annotation $\dot{\imath}$ indicates that type of the first argument of $<$ is any type which yields $\mathbb{N}$ after erasing i. (We will say that x_s is oblivious to i in the definition of $<$.)

$$(<) : (x_s :_{\dot{\imath}} \mathbb{N}) \to (y : \mathbb{N}) \to \mathsf{Bool}$$

Hence, if $x_s : \mathsf{List}\ a$ then $x_s < 5$ is a valid expression[2]; and it tests whether x_s has less than 5 elements. The expression $3 < 5$ is also type-correct, because the erasure is idempotent. In general, it has no effect on terms which do not mention the erased color ($\lfloor \mathbb{N} \rfloor_i = \mathbb{N}$).

We note that substitution behaves specially on oblivious arguments. Consider again the expression $x_s < 5$. In it, one can substitute for x_s a concrete list containing information. The remarkable feature is that in the resulting term, the *erased* list will stand for x_s. For example:

$$\begin{aligned}&(x_s < 5)[\mathsf{more}\ 'b'\ (\mathsf{more}\ 'l'\ (\mathsf{more}\ 'u'\ \mathsf{stop}))/x_s]\\&\quad = \mathsf{more}\quad(\mathsf{more}\quad(\mathsf{more}\quad\mathsf{stop})) < 5\\&\quad = 3 < 5\end{aligned}$$

2.2 Types as predicates

By using colored types, one effectively specifies structural invariants. For example, the type of the above concatenation operation constrains the

[2] We take the liberty to use decimal notation for unary naturals.

length of its result. In our TTC it is possible to reveal these invariants explicitly by viewing types as predicates, and terms as proofs that the predicates are satisfied by the i-erasure. This is done by modularizing the typing judgment. For example, under the modality i, the type of lists is seen as a predicate over $\mathbb{N}$. To indicate that the judgment is modulated, the typing operator (colon) is indexed with i.

$$\mathsf{List}\ (a :_i \star)\ :_i\ (x_s :_\not{i}\ \mathbb{N}) \to \star$$

Any typing can be so modulated. For example, a list $x_s : \mathsf{List}\ a$ becomes a proof that $\lfloor x_s \rfloor_i$ satisfies the $\mathsf{List}\ a$ seen as a predicate.

$$x_s :_i \mathsf{List}\ a\ \bullet_\not{i}\ x_s$$

(To avoid confusion we write predicate test using the $\bullet_\not{i}$ operator. Formally it is just the application corresponding to i-oblivious abstraction.) Likewise, the concatenation returns a list whose length is the sum of the lengths of its inputs.

$$(+) :_i (x_s : \mathsf{List}\ a) \to (y_s : \mathsf{List}\ a) \to \mathsf{List}\ a\ \bullet_\not{i}\ (x_s + y_s)$$

(To make sense of the above typings, recall that the second argument to List becomes a natural number after erasing i; in general a variable $x_s :_i \mathsf{List}\ a\ \bullet_\not{i}\ n$ stands for a list of size n.)

Additionally, we remark that even though every type becomes a predicate, the computations (or data) that it represents do not essentially change. Taking our list example, the union of the types $\mathsf{List}\ a\ \bullet_\not{i}\ n$ for any $n : \mathbb{N}$ is isomorphic to $\mathsf{List}\ a$ seen as a type. Hence, assuming one has existential quantification, the type $A : \star$ remains available as a type under the modality i, as $\exists x.A\ x : \star$. Hence, one can continue to use types as types, even under a colored typing, referring implicitly to the above existential construction. For the sake of concision we will take advantage of this shortcut in section 2.4.

2.3 Colored pairs

Another (dual) way to introduce colors is via pairs. A *colored pair* type, whose general form is written $(x : A) \times_i B$, is similar to the usual type $\Sigma(x : A)\ B$ (in particular x may occur in B). The difference is that B is tainted with the color i; and A is oblivious to i. Given $a : A$ and $b :_i B[a/x]$ one can construct an inhabitant of the pair type. As usual, colors must match: $a ,_i b : (x : A) \times_i B$ is valid only if b is tainted and a is oblivious to i. Erasure extracts the first component of a pair, and interpreting a pair as a predicate yields its second component. The following example illustrates how colored pairs can be used to prove some parametricity

properties. Assume the following (i-oblivious) context.

$$f : (a : \star) \to a \to a$$
$$b : \star$$
$$y : b$$

Then we can define

$$t : (x : b) \times_i (x \equiv y)$$
$$t = f\ ((x : b) \times_i (x \equiv y))\ (y ,_i \mathsf{refl})$$

By definition of erasure:

$$\lfloor t \rfloor_i = f\ b\ y \tag{1}$$

The above equation can also be intuited by looking at t under i-glasses:

$$f\ ((\quad b) \qquad\qquad)\ (y \quad)$$

In an i-modulated typing, one implicitly refers to the colored component of pairs, and we therefore have: $t :_i \lfloor t \rfloor_i \equiv y$. By (1) we obtain

$$t :_i f\ b\ y \equiv y$$

This result is normally obtained by a logical relation argument *outside* the theory, while it is internalized here — albeit via a judgment with an extra color. (A fully formal version of this example is presented in section 4.) It may be worth stressing that, if one were to use a regular pair type, then, because f is abstract, the first component of t would *not* compute. In contrast, erasure is defined even on neutral terms.

2.4 Multiple colors

We propose to support arbitrarily many colors. This feature is important for compositionality: it ensures that one can always mark a binding as tainted without corrupting interactions with the rest of the program. Indeed the other parts of the programs will use other, orthogonal colors. For example, the $\mathsf{List}\ a$ type described above is sufficient, there is no need to define a version without color. If one needs to access the elements of the list in a function, one simply taints its typing with the color. For example, a summation function may be given the type $\mathsf{sum} :_i \mathsf{List}\ \mathbb{N} \to \mathbb{N}$. The taint will be transitively inherited by any function using sum (which can in turn use other colors at will).

In fact, it is advisable to use colors even more effectively, and instead define sum by erasure. Assume the following definition of concat, where the nested lists use a different color j for the type a of elements, which

we render in red. (In the type of concat, a occurs in an i-tainted context, so it is tainted both with i and j. We have $a :_{ij} \star$, and we render it with the combination of blue and red: magenta).

$$\begin{aligned}
&\mathsf{concat} :_i \mathsf{List}\ (\mathsf{List}\ a) \to \mathsf{List}\ a \\
&\mathsf{concat}\ \mathsf{stop} &&= \mathsf{stop} \\
&\mathsf{concat}\ (\mathsf{more}\ x\ x_s) &&= x + \mathsf{concat}\ x_s
\end{aligned}$$

Then one can obtain sum by erasing j from concat: $(\mathsf{sum} = \lfloor \mathsf{concat} \rfloor_j)$. This would be impossible with a single color: attempting to erase the elements of the inner lists would erase the whole function.

Another use for multiple colors is to nest pairs. One cannot nest pairs differing on a same given color, because this would break the rule that either side of an i-colored pair must respectively be oblivious to i or be tainted with it. However one can nest pairs which use different colors. The general form of j-colored pairs nested inside an i-colored one is the following:

$$(x : (w : A) \times_j B[w]) \times_i ((z : C[x]) \times_j D[x, z])$$

C can only refer to the j-oblivious part of x. C may not refer to the j-tainted part of w, since it does not carry that color itself. Looking at the above type successively with i and j glasses[3]:

$$\begin{array}{ll}
(& (w : A) \times_j B[w]) \\
(x : (& A) \qquad\qquad) \times_i ((\quad C[x]) \qquad\qquad)
\end{array}$$

Using nested pairs is syntactically inconvenient, hence in the rest of the section we use a record-like syntax. Using record syntax, the above pair would be written

$$\{w :_{\not{ij}} A;\ y :_{\not{i}} B[w];\ z :_{\not{j}} C[w];\ D((w,_j y)\ ,\ z)\}$$

The following example illustrates how multiple colors can be used to program with relations. Assume a definition of streams $\mathsf{Stream} : \star \to \star$. Stream is a functor as witnessed by $\mathsf{map} : (a : \star) \to (b : \star) \to (a \to b) \to \mathsf{Stream}\ a \to \mathsf{Stream}\ b$, with usual definitions. Assume furthermore the

[3]The analogy to perceptual colors still holds here: magenta both appears blue under blue glasses and red under red glasses. (*I.e.*, it fades into the background in both cases.) The analogy breaks only when one uses too many colors: most humans can only perceive three primary colors, while we allow an unbounded number of colors in TTC.

following abstract context:

$$\begin{aligned}
nth &: (a : \star) \to \mathsf{Stream}\, a \to \mathbb{N} \to a \\
a &: \star \\
b &: \star \\
f &: a \to b \\
x_s &: \mathsf{Stream}\, a \\
n &: \mathbb{N}
\end{aligned}$$

We define:

$$\begin{aligned}
r &: \star \\
r &= \{x :_{i} a;\, y :_{j} b;\, f\, x \equiv y\} \\
u &: a \to r \\
u &= \lambda z.\{x = z;\, y = f\, z;\, \mathsf{refl}\} \\
t &: r \\
t &= nth\, r\, (\mathsf{map}\, u\, x_s)\, n \\
&= nth\, \{x :_{i} a;\, y :_{j} b;\, f\, x \equiv y\} \\
&\qquad (\mathsf{map}\, (\lambda z.\{x = z;\, y = f\, z;\, \mathsf{refl}\})\, x_s) \\
&\qquad n
\end{aligned}$$

Erasing colors from t yields:

$$\begin{aligned}
\lfloor t \rfloor_i &= nth\, a\, (\mathsf{map}\, \mathsf{id}_a\, x_s)\, n = nth\, a\, x_s\, n \\
\lfloor t \rfloor_j &= nth\, b\, (\mathsf{map}\, f\, x_s)\, n
\end{aligned}$$

Indeed, looking at t respectively under i-colored and j-colored glasses:

$$\begin{array}{l}
nth\, \{\quad a \quad\quad\quad\quad \}\, (\mathsf{map}\, (\lambda z.\{\quad z \quad\quad\quad\quad \})\, x_s)\, n \\
nth\, \{\quad\quad\quad b \quad\quad \}\, (\mathsf{map}\, (\lambda z.\{\quad\quad\quad f\, z \quad \})\, x_s)\, n
\end{array}$$

Now, viewing t under the i then $\{i, j\}$-modalities:

$$\begin{aligned}
&t : r \\
&t :_{i} r\, \lfloor t \rfloor_i \\
&t :_{i,j} r\, \lfloor t \rfloor_i\, \lfloor t \rfloor_j \\
&t :_{i,j} f\, \lfloor t \rfloor_i \equiv \lfloor t \rfloor_j \\
&t :_{i,j} f\, (nth\, a\, x_s\, n) \equiv nth\, b\, (\mathsf{map}\, f\, x_s)\, n
\end{aligned}$$

Hence writing $nth\, a\, x_s\, n$ as $x_s\, !!\, n$, we obtain the expected commuting law

$$f\, (x_s\, !!\, n) \equiv \mathsf{map}\, f\, x_s\, !!\, n.$$

2.5 Conclusion

A motto of programming with dependent types is to use more and more types to express one's intentions more and more precisely. However, there is a drawback to precise types: hard work is often required to convince a type-checker that programs inhabit them. We observe that the use of colors is a way to specify invariants in types which does not complicate user code. For instance we have seen that it is just as easy to program with colored lists as with regular ones, and length invariants are captured. Furthermore, the system can automatically discover equations which would require a proof without the use of colors.

One cannot "go wrong" by using more colors in a library. In the worst case, colors can simply be ignored by the users of the library. In the best case, they serve to specify invariants concisely, facilitate reasoning, and provide a variant of the library to the user for each possible erasure combination.

We have implemented a prototype of TTC as an extension of the Agda proof assistant. The prototype, in its current version at the time of writing, features colored bindings and abstraction over colors, but is still lacking erasure, oblivious bindings and colored pairs. The prototype, together with a short tutorial for it, can be obtained online at `http://www.cse.chalmers.se/~mouling/Parametricity/TTC.html`.

3 CCCC: A Core Calculus of Colored Constructions

In this section we present CCCC (the Core Calculus of Colored Constructions) which is the formal core of the TTC we envision. Technically, CCCC is an extension of CC (the plain Calculus of Constructions [Coquand and Huet, 1988]) with the notion of color informally introduced in the previous section. Even though we use CC as a base, we do not rely on its specifics. Along the lines presented here, it is conceivable to construct a variant of any type-system with colors, including intuitionistic type theory [Martin-Löf, 1984].

The rest of the section describes the main features of CCCC in pedagogical order. A summary is shown in appendix B for reference. We emphasize that we describe only on the core features of TTC. Some features used in the previous section will not be included here, even though we believe they could be included with limited effort.

Even though we continue to render some expressions in color as a visual aid, the formal system does not rely on them in any way. This section can be read in black and white without any loss in precision.

3.1 CC as a PTS

We use CC as a base, so we recall briefly its definition, using a pure type system [Barendregt, 1992] presentation. The typing rules are as follows:

$$\textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad A =_\beta A'}{\Gamma \vdash a : A'} \qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star : \Box} \qquad \textsc{Var}\ \frac{\vdash \Gamma \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma, x : A \vdash B : s}{\Gamma \vdash (x : A) \to B : s} \qquad \textsc{Abs}\ \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A.b : (x : A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash F \bullet u : B[u/x]}$$

To limit clutter we omit the well-formedness conditions of types A and B in the rule Abs. The product $(x : A) \to B$ may be also written $A \to B$ when x does not occur free in B, and we generally omit the application operator $\bullet$. The metasyntactic variable s ranges over the sorts $\star$ and $\Box$. The context-lookup relation $(x : A \in \Gamma)$ is straightforward, and the context-formation rules are:

$$\textsc{Empty}\ \frac{}{\vdash \diamond} \qquad \textsc{Bind}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : s}{\vdash \Gamma, x : A}$$

Traditional presentations of PTSs, including that of Barendregt [1992], use another formulation, which integrates the context-lookup, context-formation and typing rules. Instead of the Var rule, one has the following Weakening and Start rules, and axioms can only be used in the empty context.

$$\textsc{Start}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad \textsc{Weakening}\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$$

While the presentation of Barendregt economizes a couple of derivation rules, it has the disadvantage to conflate separate concepts in a single definition. Consequently, is it harder to extend, and modern presentations tend to use separate context-lookup and context-formation relations.

3.2 Colors, taints and modalities

We assume an infinite supply of color names; the metasyntactic variables i and j stand for them in the remainder of this chapter. We call an

ordered set of such color names a *taint* and use θ or ι to range over taints. A color may be introduced in the context by its name. After such a mention, terms may contain i-tainted parts, but also non i-tainted parts. In contrast, before the mention of i, terms are i-oblivious: they cannot depend on i in any way.

Our typing judgment $\Gamma \vdash A :_\theta B$ is indexed by a taint θ, which must be a subset of the colors present in Γ. The presence of a given color i in θ indicates how i can be used in A and B.

- $i \notin \theta$ indicates that A is not tainted with i. A term A typed in such a taint may still mention the color i. For example $\lambda(x :_i T) \to a$ is allowed. However, the usage of i-tainted variables is forbidden in the target (a) of the term. For example $\lambda(x :_i T) \to x$ is forbidden.

- $i \in \theta$ indicates that the term A is tainted with i. In such a judgment, using i-tainted variables is allowed in the targets of terms and types. *Remark*: it does not make sense to erase i from a judgment using this taint; conceptually the whole typing is tainted, so erasure would entirely remove it.

For each taint θ we have two sorts $\star_\theta$ and $\Box_\theta$, with the axiom $\star_\theta : \Box_\theta$. The conversion rule merely preserves taints.

$$\textsc{Conv}\ \frac{\Gamma \vdash a :_\theta A \qquad A =_\beta A'}{\Gamma \vdash a :_\theta A'} \qquad\qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star_\theta :_\theta \Box_\theta}$$

A variable binding $x :_\psi A$ does not only carry a taint, but also a *modality*. (ψ and φ range over modalities.) A modality is composed of two disjoint sets of colors (say $\psi = (\theta, \iota)$ with $\theta \cap \iota = \emptyset$) that constrain what kind of term u can be substituted for x. The first set θ is the taint of u. The second set ι is an "anti-taint": a set of colors which u must be oblivious to. We often use the compact notation $i_1 ... i_n, \cancel{j_1} ... \cancel{j_m}$ for the modality $(\{i_1 ... i_n\}, \{j_1 ... j_n\})$. Similarly we will write $\cancel{j} \in \psi$ to mean that j is found in the second set. Using this notation, the following two contexts are equivalent (one can substitute one for another without changing the provability of a judgment):

$$\Gamma, x : A, i, \Delta$$
$$\Gamma, i, x :_{\cancel{i}} A, \Delta$$

That is, declaring a variable before i, or declaring it i-oblivious explicitly are equivalent. The product and abstraction rules can change the modal-

ity of the type quantified over; the application rule behaves accordingly.

$$\textsc{Prod}\ \frac{\Gamma, x :_\psi A \vdash B :_\theta s}{\Gamma \vdash (x :_\psi A) \to B :_\theta s} \qquad \textsc{Abs}\ \frac{\Gamma, x :_\psi A \vdash b :_\theta B}{\Gamma \vdash (\lambda x :_\psi A.b) :_\theta (x :_\psi A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F :_\theta (x :_\psi A) \to B \qquad \Gamma \vdash u :_\psi A}{\Gamma \vdash F \bullet_\psi u :_\theta B[u/x]}$$

(where the metasyntactic variable s now ranges over the sorts $\star_\iota$ or $\Box_\iota$, for any ι.) To be able to merely forward modalities, we need to extend the judgment to support arbitrary modalities φ, not just taints θ, as we do at the beginning of the next section.

We generally omit the modality annotation on applications, because they are easily inferred from the context. One can embed any derivation from CC into CCCC simply by using the empty taint everywhere.

3.3 Obliviousness and variable lookup

We extend the typing judgment to support any modality ψ (not just a taint) as follows:

Definition 1 (Oblivious judgment)**.**

$$\textit{If } \psi = (\theta, \iota) \textit{ then } \Gamma \vdash A :_\psi B \ \stackrel{\text{def}}{=}\ \lfloor \Gamma \rfloor_\iota \vdash A :_\theta B$$

This captures the intuition that A and B are oblivious to every color in ι. Indeed, erasure removes all mentions of i from the context Γ for every $i \in \iota$ (a complete definition and justification of erasure is given in the following section). In particular, if $x : A \in \Gamma$, then $\Gamma \vdash x :_\lightning \lfloor A \rfloor_i$; in words, referencing a variable from an i-oblivious judgment yields only a witness of the i-erased type.

Hence, variable lookup requires equality of taints, not merely inclusion:

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

There are two ways to access an i-oblivious variable x. First, it is accessible in an i-oblivious judgment, as can be seen by expanding Definition 1:

$$\Gamma, x :_\lightning A, \Delta \vdash x :_\lightning A \ \stackrel{\text{def}}{=}\ \lfloor \Gamma \rfloor_i, x : A, \lfloor \Delta \rfloor_i \vdash x : A$$

Second, it can be accessed from an i-aware judgment, as formalized in the context-lookup rules:

$$\textsc{Start}\ \frac{}{x :_\theta A \in \Gamma, x :_{(\theta,\iota)} A} \qquad \textsc{Col. Wk}\ \frac{x :_\theta A \in \Gamma \qquad i \notin \theta}{x :_\theta A \in \Gamma, i} \qquad \textsc{Wk}\ \frac{x :_\theta A \in \Gamma}{x :_\theta A \in \Gamma, y :_\psi B}$$

The Col. Wk rule ensures that x is accessible from an i-aware context (but not an i-tainted one), even if it is declared before the introduction the color i. The Start rule plays a similar role: x can be explicitly oblivious to any set of colors, it does not change its accessibility.

Consider as an example the following definition, a variant of Leibniz equality.

$$x \equiv^i_a y \quad \stackrel{\text{def}}{=} \quad (P :_i (z :_\lightning a) \to \star_i) \to P\, x \to P\, y$$

One can verify that it is a well-colored type as follows. Let $\Delta = a : \star, x : a, y : a$:

$$\textsc{Prod}\ \frac{\textsc{App}\ \dfrac{\vdots\ \text{Definition 1}\ \dfrac{\Delta \vdash y : a}{\Delta, i, \ldots \vdash y :_\lightning a}}{\Delta, i, P :_i (z :_\lightning a) \to \star_i, q :_i P\, x \vdash P\, y : \star_i}}{\Delta, i \vdash x \equiv^i_a y :_i \star_i}$$

With $\mathsf{refl}^i \stackrel{\text{def}}{=} \lambda(P :_i \star).\lambda(q :_i P).q$, one can also derive $a : \star, x : a, i \vdash \mathsf{refl}^i :_i x \equiv^i_a x$.

In order to use $\equiv$ as an equality, we need to access oblivious variables from non-oblivious contexts, as we explain in this paragraph. The key difference between $\equiv$ as defined above and the usual Leibniz equality here is that we use propositions P of type $(x :_\lightning a) \to \star_i$; that is, the parameter of P is i-oblivious.

Thanks to the context lookup rules, a variable $x :_\lightning \mathsf{Bool}$ may be used even in an i-aware context, so one can construct a proposition Q^i of the right type which returns truth if the Boolean is true and falsity otherwise. Then one can obtain falsity from $\mathsf{true} \equiv_{\mathsf{Bool}} \mathsf{false}$ by substituting Q^i for P. Assuming a definition $\mathsf{Test} :_i \mathsf{Bool} \to \star_i$ which does the adequate case analysis on booleans, one has:

$$\textsc{Def.}\ \frac{\textsc{Abs}\ \dfrac{\textsc{App}\ \dfrac{\vdots\ \textsc{Var}\ \dfrac{x : \mathsf{Bool} \in i, x :_\lightning \mathsf{Bool}}{i, x :_\lightning \mathsf{Bool} \vdash x : \mathsf{Bool}}}{i, x :_\lightning \mathsf{Bool} \vdash \mathsf{Test}\, x :_i \star_i}}{i \vdash \lambda(x :_\lightning \mathsf{Bool}).\mathsf{Test}\, x :_i (x :_\lightning \mathsf{Bool}) \to \star_i}}{i \vdash Q^i :_i (x :_\lightning \mathsf{Bool}) \to \star_i}$$

3.4 Erasure

Color erasure is defined by structural induction on terms. The effect on each modality is the following. Applying i-erasure on an i-tainted binding removes it. The i-erasure of a non i-tainted binding is the binding of the erasure. Erasing i from an i-oblivious binding has no effect besides removing the mention of $\dot{\imath}$.

Erasure of product, abstraction and application follows directly from the behavior on bindings. Erasure preserves all variable occurrences, as well as sorts. In the following table we sum up all cases. The erasure of terms which do not mention a modality are show in the first column. For terms which mention a modality, we show the various cases in various columns. The first column shows the case where the color does not occur anywhere in the modality. The second one shows the case where i occurs as a taint. The third one shows the case where the i occurs as an anti-taint.

$i \notin \psi$	$i \in \psi$	$\psi = \varphi, \dot{\imath}$
$\lfloor x \rfloor_i = x$		
$\lfloor s \rfloor_i = s$		
$\lfloor (x :_\psi A) \to B \rfloor_i = (x :_\psi \lfloor A \rfloor_i) \to \lfloor B \rfloor_i$	$\lfloor B \rfloor_i$	$(x :_\varphi A) \to \lfloor B \rfloor_i$
$\lfloor \lambda x :_\psi A.b \rfloor_i = \lambda x :_\psi \lfloor A \rfloor_i . \lfloor b \rfloor_i$	$\lfloor b \rfloor_i$	$\lambda x :_\varphi A . \lfloor b \rfloor_i$
$\lfloor F \bullet_\psi a \rfloor_i = (\lfloor F \rfloor_i) \bullet_\psi \lfloor a \rfloor_i$	$\lfloor F \rfloor_i$	$(\lfloor F \rfloor_i) \bullet_\varphi a$
$\lfloor \Gamma, x :_\psi A \rfloor_i = \lfloor \Gamma \rfloor_i, x :_\psi \lfloor A \rfloor_i$	$\lfloor \Gamma \rfloor_i$	$\lfloor \Gamma \rfloor_i, x :_\varphi A$
$\lfloor \Gamma, j \rfloor_i = \lfloor \Gamma \rfloor_i, j$		
$\lfloor \Gamma, i \rfloor_i = \Gamma$		

Lemma 1 (Erasure preserves typing)**.** *If* $\Gamma \vdash A :_\theta B$ *and* $i \notin \theta$ *then* $\lfloor \Gamma \rfloor_i \vdash \lfloor A \rfloor_i :_\theta \lfloor B \rfloor_i$.

Proof. By induction on the derivation. The proof relies on the color-discipline enforced by the typing rules. □

This lemma means that erasure makes sense as a meta-level definition. The precondition is important: erasing i makes no sense on an i-tainted term; conceptually the whole term would be erased in that case. This justifies for example the case for sorts in the definition: the precondition guarantees that erasure will not be applied to a sort $\star_i$. In the system, we use erasure only in situations where this precondition is satisfied.

Erasure is used in the definition of substitution (whose full definition is given in appendix B): when substituting in an oblivious argument, or in the type of an oblivious parameter, one needs to erase the substitutee. For example:

$$(f \bullet_{\dot{\imath}} u)[t/x] = f[t/x] \bullet_{\dot{\imath}} u[\lfloor t \rfloor_i / x]$$

Note that if x is i-tainted, the type-system prevents any occurrence of x in u, ensuring that the precondition of Lemma 1 is respected.

We have not yet defined how to reduce terms in CCCC, but it is worth mentioning already that erasure preserves computation. A proof is given later in Lemma 2 of appendix C.

$$\text{if } A \longrightarrow^\star B, \quad \text{then} \quad \lfloor A \rfloor_i \longrightarrow^\star \lfloor B \rfloor_i, \text{ for any color } i.$$

3.5 Types as predicates

By modulating a judgment with a color i, a type B becomes a predicate over $\lfloor B \rfloor_i$ [4]. The erasure of a term A of type B satisfies the type B seen as a predicate.

The parametric interpretation defined by Bernardy et al. [2010], can be extended to our theory with colors:

$$\begin{aligned}
[\![x]\!]^i &= x^i \\
[\![\lambda x :_\psi A.b]\!]^i &= \lambda x :_\psi A.\lambda x^i :_{\psi,i} x \in_\psi [\![A]\!]^i.[\![b]\!]^i && \text{if } i \notin \psi \\
&= [\![b]\!]^i && \text{if } i \in \psi \\
&= \lambda x :_\varphi A.[\![b]\!]^i && \text{if } \psi = \varphi, \dot{\iota} \\
[\![F \bullet_\psi a]\!]^i &= [\![F]\!]^i \bullet_\psi a \bullet_{\psi,i} [\![a]\!]^i && \text{if } i \notin \psi \\
&= [\![F]\!]^i && \text{if } i \in \psi \\
&= [\![F]\!]^i \bullet_\varphi a && \text{if } \psi = \varphi, \dot{\iota} \\
[\![T]\!]^i &= \lambda x :_i T.x \in [\![T]\!]^i \qquad \text{if } T \text{ is a type} \\
\hline
C \in_\psi [\![s]\!]^i &= (x :_{\psi,\dot{\iota}} C) \to s && \text{if } i \notin \psi \\
&= s && \text{if } i \in \psi \\
&= (x :_\varphi C) \to s && \text{if } \psi = \varphi, \dot{\iota} \\
C \in_\psi [\![(x :_i A) \to B]\!]^i &= \\
&(x :_\psi A) \to (x^i :_{\psi,i} x \in_\psi [\![A]\!]^i) \to (C \bullet_\psi x) \in_\psi [\![B]\!]^i && \text{if } i \notin \psi \\
&= C \in_\psi [\![B]\!]^i && \text{if } i \in \psi \\
&= (x :_\varphi A) \to (C \bullet_\varphi x) \in_\varphi [\![B]\!]^i && \text{if } \psi = \varphi, \dot{\iota} \\
C \in_\psi [\![T]\!]^i &= [\![T]\!]^i \bullet_{\psi,\dot{\iota}} \lfloor T \rfloor_i \qquad \text{if } T \text{ is not a type}
\end{aligned}$$

Theorem 1 (Parametricity of closed terms). *If* $\vdash A :_\theta B$ *and* $i \notin \theta$*, then*

$$\vdash [\![A]\!]^i :_{\theta,i} [\![B]\!]^i \bullet_{\theta,\dot{\iota}} \lfloor A \rfloor_i$$

[4] With the exception of terms of sort annotated with that color. For example, $\star_i :_i \Box_i$ and not $\star_i :_i \star_i \to \Box_i$. This feature prevents an "infinite descent" into deeper and deeper predicates.

Proof. The proof uses the standard techniques of logical relations, extended to dependent types by Bernardy et al. [2010]. □

We wish however not to be limited to closed terms, and want parametricity even on open terms. The presence of colors allows[5] to add the following rule, which internalizes the reinterpretation of terms as predicates and proofs.

$$\textsc{Param}\ \frac{\Gamma \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash A :_{\theta,i} B \bullet_{\theta,\dot{\imath}} \lfloor A \rfloor_i}$$

This rule is a generalization of Theorem 1 and it allows to deduce, within the calculus, theorems which could only be obtained meta-theoretically otherwise. However, in this paper, the treatment of logical relations differs from the usual one: a type is not *interpreted* as a predicate (via the above transformation of terms and types $[\![\cdot]\!]$), but is directly *used* as such in an i-modulated judgment.

In order to use T as a predicate we extend reduction rules as follows, where we recall that s ranges over sorts, while t ranges over terms.

$$s_\theta \bullet_\varphi t \longrightarrow (z :_\varphi t) \to s_{\theta \cup \iota} \qquad (1)$$
$$\text{where } \varphi = (\theta, \iota)$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) \qquad (2)$$
$$\text{if } \exists i \text{ such that } i \in \psi \text{ and } \dot{\imath} \in \varphi$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) \qquad (3)$$
$$\text{otherwise}$$
$$(\lambda x :_\psi A.b) \bullet_\psi t \longrightarrow b[t/x] \qquad (4)$$

- (1): Since a type becomes a predicate, a type of types (a sort) becomes a type of predicates. The target sort of the predicate type is adjusted, in order to obtain a type (and not again a predicate — see footnote 4).
- (2) and (3): A function t satisfies the predicate of a function type if an argument x (which implicitly satisfies the predicate of the domain A) is mapped by the function t to a value satisfying the codomain B. In the case of (2), the modality φ mandates erasure of the domain A, therefore x is not given as an argument to t.
- (4): The β-reduction is trivially amended to account for colors. The rules (1,2,3) do not interfere with β, because they concern other syntactic forms.

[5] The issues that one faces when attempting to internalize parametricity in a theory without colors are detailed by Bernardy and Moulin [2012].

The cases (1,2,3) agree with the standard interpretation of types as predicates. Bernardy and Moulin [2012], Bernardy et al. [2012] give a detailed account of logical relations in the presence of dependent types. As an example, one can check that reduction behaves as expected for the list concatenation type:

$$\begin{aligned} & ((x_s : \mathsf{List}\ a) \to (y_s : \mathsf{List}\ a) \to \mathsf{List}\ a) \bullet_\lambda c \\ \longrightarrow\ & (x_s : \mathsf{List}\ a) \to ((y_s : \mathsf{List}\ a) \to \mathsf{List}\ a) \bullet_\lambda (c\ x_s) \\ \longrightarrow\ & (x_s : \mathsf{List}\ a) \to (y_s : \mathsf{List}\ a) \to \mathsf{List}\ a \bullet_\lambda (c\ x_s\ y_s) \end{aligned}$$

3.6 Example

Assume the colors i and j as well as the context

$$\begin{aligned} \mathsf{List}_i &: (a :_i \star_i) \to \star \\ \mathsf{List}_j &: (b :_j \star_j) \to \star \\ \mathsf{fold} &: (a :_i \star_i) \to (b : \star) \to \\ &\quad (a \to b \to b) \to b \to \mathsf{List}_i\ a \to b \end{aligned}$$

One can define a variant of the ubiquitous *map* function as follows:

$$\begin{aligned} \mathit{map} &: (a :_i \star_i) \to (b :_j \star_j) \to \\ &\quad (f :_j (x :_i a) \to b) \to \mathsf{List}_i\ a \to \mathsf{List}_j\ b \\ \mathit{map} &= \lambda f.\mathsf{fold}\ a\ (\mathsf{List}_j\ b)\ (\lambda\ x\ x_s.\mathsf{more}\ (f\ x)\ x_s)\ \mathsf{stop} \end{aligned}$$

This version of *map* is versatile. After erasing j, the f argument disappears and *map* becomes a function returning the length of its input. After erasing i, f becomes a constant of type b, the input "list" becomes a natural number (n), and *map* returns a list containing n copies of f.

3.7 Analysis

In this section we state and prove a number of standard meta-theoretical results for our calculus. Details can be found in appendix C.

Corollary 1 (Confluence). *CCCC has the confluence property.*

Proof. A direct consequence of the diamond property and $\longrightarrow^\star = \rhd^\star$. □

Lemma 2 (Thinning). *Let* Γ *and* Δ *be legal contexts such that* $\Gamma \subseteq \Delta$. *Then* $\Gamma \vdash A :_\theta B \Longrightarrow \Delta \vdash A :_\theta B$.

Proof. As in [Barendregt, 1992, lem. 5.2.12]. □

The generation lemma for PTS can be extended to colored bindings. The only difficulty is the following. In [Barendregt, 1992], the generation lemma includes a case for applications. This case is difficult to extend to our calculus, since application can be done not only on lambda abstractions, but also on types, when they are used as predicates. Fortunately, that case is not used in the subject reduction lemma, and therefore we can omit it from our version of the generation lemma.

Lemma 3 (Generation)**.** *The statement is similar to that of [Barendregt, 1992, lem. 5.2.13]. Points 1. to 4. (constant, variable, product, abstraction) are adapted in a straightforward manner to colored binding. Point 5. (application) is removed. The following two points are added.*

- *If* $\Gamma \vdash (s_{\theta,i} \bullet_{\theta,\lightning} t) :_{\theta,i} C$*, then*

$$\Gamma \vdash t :_{\theta,\lightning} s_\theta \text{ and } C \equiv_\beta s'_\theta \text{ with } (s, s') \in \mathscr{A}$$

- *If* $\Gamma \vdash ((x :_\varphi A) \to B) \bullet_{\theta,\lightning} t :_{\theta,i} C$*, then*

$$\Gamma \vdash t :_{\theta,\lightning} \lfloor (x :_\varphi A) \to B \rfloor_i \text{ and } C \equiv_\beta s'_\theta \text{ with } \Gamma \vdash (x :_\varphi A) \to B :_\theta s_\theta$$

Proof. As in [Barendregt, 1992]: we follow the derivations until $s_{\theta,i}$ (resp. $(x :_\varphi A) \to B$) is introduced. It can only be done by the Param rule, and the conclusion follows from a use of the Thinning Lemma. (An example of such derivations can be found in figure 1.) □

Theorem 2 (Subject reduction)**.** *If* $A \longrightarrow A'$ *and* $\Gamma \vdash A : T$*, then*

$$\Gamma \vdash A' : T$$

Proof. Most of the technicalities of the proof of subject reduction for PTSs Barendregt [1992] concern β-reduction, and are not changed by the addition of colors.

Hence we discuss here only the handling of reduction rules (1) to (3). We treat first the case of sorts (1):

$$s_\theta \bullet_\varphi t \longrightarrow (z :_\varphi t) \to s_{\theta \cup \iota} \qquad (1)$$
$$\text{where } \varphi = (\theta, \iota)$$

In this case, the Generation Lemma indicates that the derivation tree must end with an App rule and contain a chain Param-Ax on the left-hand side of the derivation. A template for such a tree is shown in figure 1. One can then construct a typing derivation for the reduct, which ends with the Prod rule, and does not mention Param nor App. The template for typing the reduct is also shown in figure 1.

$$\textsc{App}\ \dfrac{\textsc{Conv}\ \dfrac{\textsc{Param}\ \dfrac{\textsc{Ax}\ \dfrac{\vdash \Gamma}{\Gamma \vdash \star : \square}}{\Gamma \vdash \star :_i \square \bullet_\lambda \star}}{\Gamma \vdash \star :_i (x :_\lambda \star) \to \square_i} \qquad \Gamma \vdash t :_\lambda \star}{\Gamma \vdash \star \bullet_\lambda t :_i \square_i}$$

$$\Downarrow$$

$$\textsc{Prod}\ \dfrac{\textsc{Ax}\ \dfrac{\textsc{Bind}\ \dfrac{\vdash \Gamma \qquad \Gamma \vdash t :_\lambda \star}{\vdash \Gamma, x :_\lambda t}}{\Gamma, x :_\lambda t \vdash \star_i :_i \square_i}}{\Gamma \vdash (x :_\lambda t) \to \star_i :_i \square_i}$$

$$\textsc{App}\ \dfrac{\textsc{Conv}\ \dfrac{\textsc{Param}\ \dfrac{\textsc{Prod}\ \dfrac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash (x : A) \to B : \star}}{\Gamma \vdash (x : A) \to B :_i \star \bullet_\lambda \lfloor (x : A) \to B \rfloor_i}}{\Gamma \vdash (x : A) \to B :_i \lfloor (x : A) \to B \rfloor_i \to \star_i} \qquad \Gamma \vdash t :_\lambda \lfloor (x : A) \to B \rfloor_i}{\Gamma \vdash ((x : A) \to B) \bullet_\lambda t :_i \star_i}$$

$$\Downarrow$$

$$\textsc{Prod}\ \dfrac{\textsc{App}\ \dfrac{\textsc{Param}\ \dfrac{\Gamma, x : A \vdash B : \star}{\Gamma, x : A \vdash B :_i \lfloor B \rfloor_i \to \star_i} \qquad \textsc{App}\ \dfrac{\Gamma \vdash t :_\lambda \lfloor (x : A) \to B \rfloor_i}{\Gamma, x : A \vdash t\, x :_\lambda B}}{\Gamma, x : A \vdash B \bullet_\lambda (t\, x) :_i \star_i}}{\Gamma \vdash (x : A) \to B \bullet_\lambda (t\, x) :_i \star_i}$$

Figure 1: Proof-reduction templates corresponding to the term-reductions (1) and (3). The sorts and taint annotations are specialized to the simplest case to reduce clutter; generalization to arbitrary taints and sorts is straightforward.

Second we treat the case of products (2) and (3).

$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) \qquad (2)$$
$$\text{if } \exists i \text{ such that } i \in \psi \text{ and } \dot{\imath} \in \varphi$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) \qquad (3)$$
$$\text{otherwise}$$

Again we can use the Generation Lemma to obtain the shape of a derivation tree of the reducible expression, and obtain a valid typing for the reduct (also shown in figure 1). In this case the typing of the source involves the chain of rules Param-Prod on the left-hand side of App. The typing of the reduct ends with Prod. □

We have taken the view in our presentation that the reduction rules are untyped, and therefore subject reduction must recover typings using the Generation Lemma. An alternative would be to have a typed reduction relation (this relation usually goes by the name of "judgmental equality"). In this case the reduction of proof trees shown in figure 1 would be part of the definition of reduction. The two approaches have been proved equivalent for arbitrary PTSs by Siles [2010].

Theorem 3 (Normalization)**.** *CCCC is strongly normalizing: every sequence of reductions eventually terminates.*

Proof. The new reduction rules, involving Param, are much easier to handle than β-reduction (rule 4). Indeed, the argument to the application is not duplicated by these reduction rules.

Hence, one can adapt the proofs of termination of CC to CCCC. At a high-level, the argument goes as follows. Consider reduction rules (1) to (3), that we collectively call Param-reductions below. Their effect is to make the left-hand-side of the $\bullet_\psi$ operator smaller. This can also be seen by examining the corresponding typing derivations: the reductions strictly decrease the size of the tree above the Param rule involved.

Consequently, between each β-reduction step, there can be only a finite number of Param-reductions. The question is now: do these Param-reductions create β-redexes? The reductions (1) and (2) do not, but (3) does. However, the new redex is harmless: it lies in an i-oblivious context, and therefore there is no risk of an (i-aware) Param-reduction to be created by that redex.

The situation is then that, by importing the proof techniques developed for CC, one can bound the chains of β-reductions, and use the above argument to finitely-bound the Param-reductions occurring between β-reduction steps. □

The above proof is an alternative to that done in earlier work Bernardy and Moulin [2012], which works by construction a model by translation into CC. Even though the earlier proof can be adapted to the present system, we find the above proof more modular, and thus easier to grasp.

3.8 Type-checking with colors

The rules Param and the definition of and oblivious judgment Definition 1 are not syntax-directed, therefore it may be non-obvious how they should be used in type-checking. In this subsection we sketch a possible type-checking algorithm and briefly argue for its soundness. A full description of the algorithm, together with a completeness proof, is left for future work.

We assume that the user supplies a term t, a type T, a modality φ and a context Γ. The task is to reconstruct a derivation of $\Gamma \vdash t :_\varphi T$.

Most of the implementation of the new rules is realized at the point of checking a variable ($t = x$). Let us assume that looking up x in the context Γ yields $x :_\psi A$.

For each color $i \in \Gamma$ we have to take into account its status in φ and ψ. For simplicity we assume that φ and ψ contain at most one color; a full implementation will do the same task for each color independently.

1. $\not{i} \in \varphi$. If i is not mentioned in ψ, then the derivation is the following.

$$\text{Definition 1}\ \dfrac{\text{Lemma 1}\ \dfrac{\text{Var}\ \dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}}{\lfloor \Gamma \rfloor_i \vdash x : \lfloor A \rfloor_i}}{\Gamma \vdash x :_{\not{i}} \lfloor A \rfloor_i}$$

   Hence we simply check that $T = \lfloor A \rfloor_i$. For $i \in \psi$, then x is not accessible and type-checking reports failure. For $\not{i} \in \psi$ one needs only to check $T = A$.

2. If the judgment is i-aware ($\varphi = \emptyset$), we have three cases. If $i \in \psi$, then x is not accessible and type-checking reports failure. Otherwise, then modalities match and we only need to check $T = A$. Notably, if $\not{i} \in \psi$, x is more oblivious than required, but this is accepted by the Start rule, which allows arbitrary extra anti-taints in the binding of the looked-up variable.

3. If $i \in \varphi$, we have two cases. If $i \in \psi$, then one simply checks $T = A$. Otherwise, one must take into account Param, that is, check $T = A \bullet_{\not{i}} x$ instead of $T = A$.

Lastly, when checking a type used as a predicate, that is a term t of the form $(\Delta \to B) \bullet_\psi u$ (for any telescope Δ) or $s \bullet_\psi u$, we reduce the term t

before checking it. The number of reduction steps is at most $|\Delta| + 1$ in the first case or 1 in the second case: the performance hit is minimal.

As usual in type-checking algorithms, all the equality-tests mentioned above have to be performed up-to the reduction relation, in order to take into account the Conv rule. This is done using standard means, for example normalizing terms before comparison, which is possible thanks to Theorem 3.

4 Extensions

4.1 Inductive definitions

The definitions presented in the previous section can be extended to work on inductive types in a straightforward manner, merely recursively applying the definitions on the types of each component of the inductive type, as we have done in the examples in section 2. The extension of a similar system to inductive definitions is described in full detail by Bernardy et al. [2012]. We conjecture that the addition of inductive definitions does not compromise any meta-theoretical property.

4.2 Colored pairs

In this section we formally introduce colored pairs, and formalize an example presented in section 2: inhabitants of the type $(a : \star) \to a \to a$ must be the identity function.

The formation and introduction rules (Sum, Pair) are similar to the usual rules for dependent pairs, with the difference that the modalities track that the first component is oblivious to the color while the second component is tainted.

$$\textsc{Sum}\ \frac{\Gamma, x :_{\not\iota} A \vdash B :_i s}{\Gamma \vdash (x : A) \times_i B : s} \qquad \textsc{Pair}\ \frac{\Gamma \vdash a :_{\not\iota} A \qquad \Gamma \vdash b :_i B[a/x]}{\Gamma \vdash a ,_i b : (x : A) \times_i B}$$

We do not provide special elimination rules for colored pairs. Instead, erasure and Param play this role. The erasure of a pair yields its first component if colors match, otherwise it acts structurally.

$$\begin{aligned} \lfloor (x : A) \times_i B \rfloor_i &= A \\ \lfloor (x : A) \times_j B \rfloor_i &= (x : \lfloor A \rfloor_i) \times_j \lfloor B \rfloor_i \\ \lfloor a ,_i b \rfloor_i &= a \\ \lfloor a ,_j b \rfloor_i &= \lfloor a \rfloor_{i ,_j} \lfloor b \rfloor_i \end{aligned}$$

We remark that the property that erasure preserves typing is conserved. In particular:

$$\text{if } \Gamma \vdash p : (x : A) \times_i B \text{ then } \lfloor \Gamma \rfloor_i \vdash \lfloor p \rfloor_i : A$$

Interpreting a pair as a predicate yields the second component if colors match, otherwise it acts structurally:

$$\begin{aligned} ((x : A) \times_i B) \bullet_{\psi, \not\downarrow} t &= B[t/x] \\ ((x : A) \times_j B) \bullet_\psi t &= (x : A \bullet_\psi \lfloor t \rfloor_j) \times_j (B \bullet_\psi t) \end{aligned}$$

We can now explain fully formally how one can derive that any function of type $(a : \star) \to a \to a$ is indeed an identity function. Let

$$\begin{aligned} \Gamma &\stackrel{\text{def}}{=} f : (a : \star) \to a \to a, \quad b : \star, \quad y : b \\ x \equiv_a y &\stackrel{\text{def}}{=} (P : a \to \star) \to P\, x \to P\, y \\ t &\stackrel{\text{def}}{=} f\, ((x : b) \times_i (x \equiv_b y))\, (y, _i\, \mathsf{refl}^i) \end{aligned}$$

Where $\equiv$ refers to the definition of the previous section. We first check that $(x : b) \times_i (x \equiv_b y)$ is well-colored (and well-sorted):

$$\dfrac{\dfrac{\Gamma \vdash b : \star}{\lfloor \Gamma, i \rfloor_i \vdash b : \star}\,\text{Erasure Def.}}{\Gamma, i \vdash b :_\not \star}\,\text{Definition 1} \qquad \dfrac{\Gamma, i, x :_\not b \vdash x :_\not b \qquad \Gamma, i, x :_\not b \vdash (x \equiv_b y) :_i \star_i}{\Gamma, i \vdash (x : b) \times_i (x \equiv_b y) : \star}\,\text{Sum}$$

And proceed with the main result:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash y : b \qquad \Gamma, i \vdash \mathsf{refl}^i :_i (y \equiv_b y)}{\Gamma, i \vdash (y, _i\, \mathsf{refl}^i) : (x : b) \times_i (x \equiv_b y)}\,\textsc{Pair} \quad \vdots}{\Gamma, i \vdash f\, ((x : b) \times_i (x \equiv_b y))\, (y, _i\, \mathsf{refl}^i) : ((x : b) \times_i (x \equiv_b y))}\,\textsc{App}}{\Gamma, i \vdash t : (x : b) \times_i (x \equiv_b y)}\,\textsc{Param}}{\Gamma, i \vdash t :_i (\lfloor t \rfloor_i \equiv_b y)}\,\text{Erasure Def.}}{\Gamma, i \vdash t :_i f\, b\, y \equiv_b y}\,\text{Def.}$$

An essential component of the trick is that f is i-oblivious. Hence it cannot distinguish in any way between i-tainted and non i-tainted terms, thus we can pass it a colored pair as its type argument.

The trick generalizes: one is able to derive useful properties about a polymorphic term q of type A by construction of an adequate term p of type $(x : A) \times_i B$ such that $\lfloor p \rfloor_i = q$. The construction of q involves specializing a type parameter to a colored pair type involving the property of interest.

In fact, by pairing a type A with a predicate $B[x]$, the colored pair type allows to override the default predicate interpretation of the type A with $B[x]$, for a given specific color.

4.3 Abstraction over colors

So far we have assumed that colors were available in the context. In a complete TTC, a mechanism to abstract over colors should be provided. At this stage we have thought of colors only as a first-order concept, that is, we propose only first-order quantification over colors.

As such, color abstraction is a relatively modest, straightforward addition. We need new syntax for abstraction, quantification, and application, as well as a constant color (written **0** in the following) used for erasure. A notable feature is that one cannot abstract over a color which is present in the modality; otherwise the color would escape the scope where it is introduced. Using k to range over color names or **0**, the typing rules are as follows:

$$\textsc{Col. Abs}\ \frac{\Gamma, i \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash \lambda i.A :_\theta \forall i.B} \qquad \textsc{Col. App}\ \frac{\Gamma \vdash A :_\theta \forall i.B}{\Gamma, k \vdash A\,k :_\theta B[k/i]}$$

and the reduction rules:

$$\begin{aligned} (\forall i.T)\ \bullet_\psi\, t &\longrightarrow \forall i.T \bullet_\psi (t\,i) \\ (\lambda i.t)\,\mathbf{0} &\longrightarrow \lfloor t \rfloor_i \\ (\lambda i.t)\,j &\longrightarrow t[j/i] \end{aligned}$$

Erasure must be extended as well to substitute occurrences of colors as arguments:

$$\begin{aligned} \lfloor t\,i \rfloor_i &= \lfloor t \rfloor_i\,\mathbf{0} \\ \lfloor t\,i \rfloor_j &= \lfloor t \rfloor_j\,i \\ \lfloor t\,\mathbf{0} \rfloor_i &= \lfloor t \rfloor_i\,\mathbf{0} \\ \lfloor \lambda i.\,t \rfloor_j &= \lambda i.\lfloor t \rfloor_j \end{aligned}$$

With this extension, the subject reduction property depends on Lemma 1.

5 Discussion and related work

Coq-style erasure. Thanks to Paulin-Mohring [1989], Coq features *program extraction*, which is an erasure of proofs to obtain programs. Such programs are external entities, that is, they cannot be referred to as an erasure from the Coq script they originate. This shortcoming is remedied in TTC. For the purpose of extraction, Coq separates types from propositions, using different sorts for each, and allows types to depend on the existence of proofs (but not their structure). In contrast, in the

system we present here, untainted terms cannot even depend on the existence of a tainted term. It is likely that the two notions of erasure can be combined in a single system, but we leave the study of that combination to future work.

Agda-style erasure. A number of systems with modalities for erasure have been proposed [Pfenning, 2001, Mishra-Linger and Sheard, 2008, Abel and Scherer, 2012], with the interpretation of *irrelevance*: types marked with a special modality (usually written $x \div A$) are understood as proofs, whose inhabitants are irrelevant for the execution of the programs.

The system presented here bears some similarity to such systems, but also presents important differences:

- Our binding form $x :_i A$ corresponds to the irrelevant binding $x \div A$.
- We have, in addition to irrelevant bindings, the complementary notion (written $x :_\not i A$). This allows us to mix erased terms with non-erased ones, and choose arbitrarily which version we mean. In contrast, systems with erasure usually fix a specific view on which parts of a term is accessible.
- We support an arbitrary number of colors instead of a single one, which is essential for compositionality and to support n-ary parametricity.
- We focus on erasure instead of irrelevance. Previous authors usually allow the use of the *ex falso quod libet* principle on irrelevant assumptions, while we forbid any use of a tainted variable in a non-tainted context.

Types for language-based security. Our notion of taint is reminiscent of that used in language-based security. More precisely, our tainted variables would correspond to variables at high security levels: tainted variables may be used only in tainted contexts. The present work can be seen as a generalization of [Abadi et al., 1999] to dependent types. A difference is that we use modalities instead of a different type former for security levels. As a consequence, we do not need a monad to relate security levels. To our knowledge, the combination of dependent types and security-levels in a type-system has not been realized before.

Ornaments. Relating variants of dependently-typed programs have been a concern for a long time. The idea of ornamenting inductive structures have been proposed to remedy this problem [Évariste Dagand and

McBride, 2012]. Here, we instead focus on erasure (let the user specify an ornamented type and recover the relation with its erasure) instead of specifying ornamentation of already existing types. This is not much of a difference in practice, because ornaments typically can only be applied to a single type. An advantage of colors over ornaments is that colors integrate natively with existing type-theories, while ornamentation relies on encodings; additionally we get free equalities when working with erasures, and colored pairs reveal parametricity properties. The chief advantage of ornaments is that any algebra can be used to ornament datatypes, while we are limited to structural relations.

Ko and Gibbons [2011] have shown how to compose ornaments. Composition is lacking from the system presented here, but we plan to support it in future extensions of TTC. Indeed, if two terms A and B share an erasure (for example $C = \lfloor A \rfloor_i = \lfloor B \rfloor_j$), it means that A and B are to versions of C ornamented differently. Under the same assumption, it is possible to automatically construct a term D such that $\lfloor D \rfloor_j = A$, $\lfloor D \rfloor_i = B$, and $\lfloor D \rfloor_{ij} = C$. That is, D contains both the ornamentations coming from A and B.

Parametricity. Bernardy and Moulin [2012] (expanded as chapter 1 of this thesis) have described a calculus which internalizes parametricity, and have shown that higher dimensions are necessary to nest parametricity. The present work has the same model as the previous one (colors are dimensions under another name). Besides re-framing dimensions as colors, which we find allows an easier grasp of intuitions, the present system features a number of technical simplifications:

1. In the system of Bernardy and Moulin [2012], one has to be explicit about the number of dimensions that a term has. In contrast, here, the dimensionality of a term is implicit. Indeed, contexts can be extended with an arbitrary number of dimensions. This means that a term can always be used in a context which has more (distinct) colors, whereas previously an explicit conversion had to take place. In other words, terms can be seen as infinitely-dimensional; but if they do not mention a dimension explicitly they behave uniformly over it. In particular, usual λ-terms are uniform, and parametricity is a consequence of this uniformity.

2. In this paper, we name dimensions, whereas they are numbered in Bernardy and Moulin [2012]. The situation is analogous to the issue of the representation of variables in lambda-calculi. One can either use explicit names or De Brujin indices, and using names is usually more convenient.

It is worth underlining that the notion of erasure we employ here is not

the same as that of Bernardy and Lasson [2011]. Here, we erase the colored component; whereas Bernardy and Lasson [2011] erase the oblivious components.

Parametricity has more "standard consequences" such as the deduction of induction principles. Unfortunately, many of these consequence also require extensionality (Wadler [2007] gives a complete development). Since extensionality is not well integrated to type theory (let alone TTC) at the moment, we cannot derive such constructions, at least not without postulating extensionality.

Higher-dimensional Equality. Recent work on the interpretation of the equality-type in intensional type theory suggests that it should be modeled using higher-dimensional structures [Licata and Harper, 2012]. In the present work we support the definition of higher-dimensional structures via dependencies on colors. In future work we wish to investigate whether the presence of colors can help encoding the higher-dimensional structure of equality.

A potential difficulty is that the structure of equality is simplectic, while we have here a cubic structure (colors are orthogonal). A simplex is however easily embedded in a cube, so there are grounds to believe that the two aspects can eventually be integrated.

6 Conclusion and future work

We have described an extension of type theory with colored terms, a notion of color erasure, and a way to interpret colored types as predicates. We have shown how that extension provides a new kind of genericity, and how the coloring discipline enforces invariants when writing programs. We also shown how to reveal these invariants (by typing the judgment in another modality) and internalize some parametricity results.

We detailed extensively a core version of that colored type theory, namely CCCC. In particular, we proved fundamental properties for that system, such as Church-Rosser's, subject-reduction, and strong normalization. We also implemented some features of CCCC in a prototype we aim to merge into the main stream Agda proof assistant.

Future work chiefly involves unifying colors as presented here with previous similar notions. On the implementation side, we aim to complete our prototype with all the features presented in this paper. Besides, we would like to investigate the feasibility of inference of color annotations before merging our implementation of TTC into the main branch of Agda. We will then be able to assess the practical power of TTC. A

first step in this direction is the retrofitting of the Agda standard library to use colors.

Acknowledgements

Many ideas underlying this paper have germinated and matured in discussions with Thierry Coquand, Simon Huber and Patrik Jansson. We thank Peter Dybjer, Cezar Ionescu, Nicolas Pouillard and Philip Wadler as well as anonymous reviewers for useful feedback.

This work has been partially funded by the Swedish Foundation for Strategic Research, under grant RAWFP.

Chapter 3

A new type theory in color and its presheaf model

1 Introduction

In chapter 1 we defined (by induction on raw expressions) a *meta-operator* $[\![\cdot]\!]$, called the parametric interpretation. If $a : A$ then the type $[\![A]\!]\, a$ represented the associated free theorem, and $[\![a]\!] : [\![A]\!]\, a$ a proof of it. On this basis, we defined a calculus where one can prove *internally* that each well-typed term a satisfies the parametric interpretation of its type. A property of the parametric interpretation is that it expands each binding to a double binding with a new parametricity witness. For instance we defined $[\![\lambda x : A.\, b]\!] \stackrel{\text{def}}{=} \lambda x : A.\, \lambda \dot{x} : [\![A]\!]\, x.\, [\![b]\!]$ (hence the parametricity witness $\dot{x}$ of x is in scope in $[\![b]\!]$). However on a free variable x, the parametricity witness $\dot{x}$ is not in scope and we defined $[\![x]\!] \stackrel{\text{def}}{=} \llbracket x \rrbracket$, where $\llbracket \cdot \rrbracket$ is a new constructor denoting the parametricity witness of a free variable.

Thanks to the internalization we were able to prove the parametricity theorem (if $\Gamma \vdash a : A$ then $\Gamma \vdash [\![a]\!] : [\![A]\!]\, a$) not only as a meta-theorem for the underlying calculus without the constructor $\llbracket \cdot \rrbracket$ (a Pure Type System such as the Calculus of Constructions [Coquand and Huet, 1988]), but also for the new system as an internal application (*i.e.*, we proved that the type $(X : \mathcal{U}) \to (x : X) \to [\![X]\!]\, x$ is inhabited). In other words, *parametricity itself is parametric*. As a consequence, we could nest applications of the parametric interpretation; in particular taking $t = \lambda x : A.\, \llbracket x \rrbracket$, of type $(x : A) \to [\![A]\!]\, x$, as hinted at in the general introduction (and developed in page 21 of chapter 1) we obtained the following non-convertible

types for the term $[\![t]\!]$:

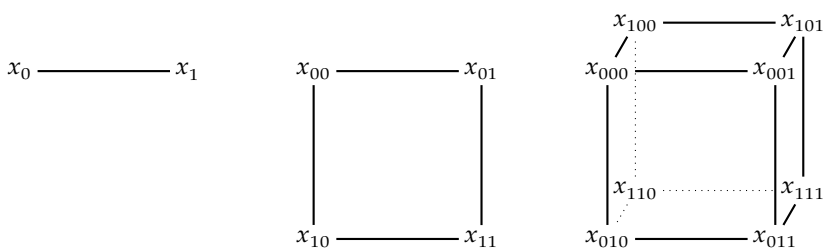
$$(x : A) \to (\dot{x} : [\![A]\!]\, x) \to [\![[\![A]\!]]\!]\, x\, \dot{x}\, [\![x]\!]; \text{ and} \tag{1}$$
$$(x : A) \to (\dot{x} : [\![A]\!]\, x) \to [\![[\![A]\!]]\!]\, x\, [\![x]\!]\, \dot{x}. \tag{2}$$

In order to avoid such lack of uniqueness of types, we grouped together each term with its parametricity witness. That is, instead of the two bindings $[\![\lambda x : A.\ ...]\!] = \lambda x : A.\ \lambda \dot{x} : [\![A]\!]\, x.\ ...$, we bound them together as a pair: $[\![\lambda x : A.\ ...]\!] = \lambda \bar{x} : (x : A) \times ([\![A]\!]\, x).\ ...$ (where x in the body denotes the first components of the pair $\bar{x}$, and $\dot{x}$ denotes its second component). When iterating the parametric interpretation, these pairs become 4-tuples, 8-tuples, etc. Therefore after n applications we abstracted over a 2^n-long tuple (which we represented as a n-dimensional hypercube), while the parametricity predicate was applied to a $(2^n - 1)$-long tuple (which we represented as a n-dimensional hypercube lacking its bottom-right vertex). Each vertex was annotated using a bit-vector of length n: for instance the previous variables x and $\dot{x}$ respectively correspond to x_0 and x_1. We can view these variables paired as the two endpoints of a line. Geometrically, the previous types can be seen as

$$(\ x_0 \text{ — } x_1\ ) : [\![A]\!] \to [\![[\![A]\!]]\!] \begin{pmatrix} x_0 \text{ — } x_1 \\ | \\ [\![x_0]\!] \end{pmatrix}; \text{ and} \tag{1}$$

$$(\ x_0 \text{ — } x_1\ ) : [\![A]\!] \to [\![[\![A]\!]]\!] \begin{pmatrix} x_0 - [\![x_0]\!] \\ | \\ x_1 \end{pmatrix}. \tag{2}$$

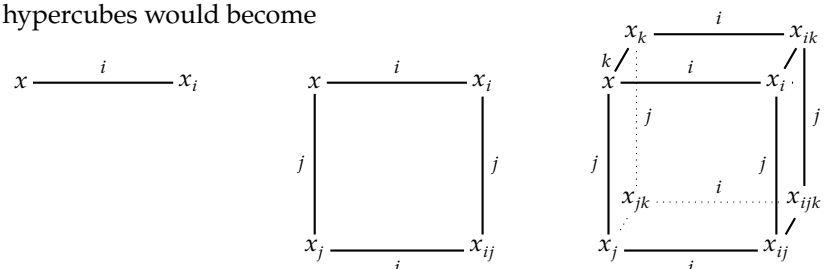
Applying parametricity to this pair, the *line* x_0—x_1, yields a square by duplicating the endpoints x_0 and x_1 respectively into the lines x_{00}—x_{10} and x_{01}—x_{11}, that we can view as the two sides of a square. Similarly, applying parametricity to a square yields a cube, etc.



In chapter 1 we showed how grouping a term together with its parametricity witnesses, and defining a cube rotation operator, allowed us

to change the definition of the parametric interpretation in a way that it preserves types.

An alternative approach, explored in chapter 2, is to *name* each direction: instead of having a single parametric interpretation operator $[\![\cdot]\!]$, we get one for each name (here writing $[\![\cdot]\!]_i$ for the parametric interpretation at dimension i to emphasize the connection with chapter 1). Nesting is still allowed, with the restriction that each occurrence of parametricity must be used with a *different name* (also called *color*). For instance the previous hypercubes would become



(Were we have labelled each edge with colors, and omitted some color labels k on the cube for the sake of readability.) A feature of this calculus is that each binding can be annotated with a finite set of colors. Hence hypercubes no longer need to be constructed explicitly. For instance instead of explicitly binding the above square, we would introduce 4 bindings each annotated with a subset of the color *set* $\{i, j\}$ (the order does not matter):

$$x : A, \quad x_i :_i [\![A]\!]_i\, x, \quad x_j :_j [\![A]\!]_j\, x, \quad \text{and } x_{ij} :_{ij} [\![[\![A]\!]_i]\!]_j\, x\, x_i\, x_j.$$

Moreover cube rotation can now be achieved by mere color renaming. The canonical parametricity witness $[\![a]\!]$ also needs to be parameterized with a color; it follows that applying parametricity is now a two-fold operation: 1. pick a *fresh* color i, and 2. apply $[\![\cdot]\!]_i$. Furthermore since in any given context there are only finitely many colors available (one for each level of parametricity), this limits how many nested levels of parametricity there can be in a typeable expression.

In the present chapter we will present a new colored type theory (section 2) extending Martin-Löf's Logical Framework, which internalizes parametricity (as we show in section 4) and can be seen as a simplification and generalization of the systems of chapters 1 and 2. In particular we introduce a new constructor $(\!|\cdot,\cdot|\!)$, which we call a *ray*, pairing a term (then called the *origin* of the ray) with its parametricity proof. The use of the name "ray" is motivated by the correspondence with cubical type theory [Cohen et al., 2015], as we detail later in this section. Unlike before, the parametricity proof does not depend on a direction. Instead, a ray $(\!|a, p|\!)$ can be applied to a fresh color i (we write $(\!|a, p|\!)@i$ for the application) to "specify its direction", which in the calculus from chapter 2

corresponds to forming an i-labelled edge $a \xrightarrow{i} p$ with the ray's components as endpoints. Moreover we have a special symbol '$\mathbf{0}$', which when applied to a ray yields its origin: $(\!|a, p|\!)@\mathbf{0} \equiv a$. The second component of a ray is the parametricity proof of its origin, and is obtained by applying another operation $\cdot!$: $(\!|a, p|\!)! \equiv p$. Furthermore a color i can be abstracted over using the notation $\langle i \rangle$, and if u is a ray then the color η-expansion yields $u \equiv (\langle i \rangle u)@i$. The canonical parametricity witness of a given term a is $(\langle i \rangle a)!$, which as in the previous paragraph is obtained by first abstracting over a new color (which yields a ray) and then applying the parametricity operator $\cdot!$.

As we will show in Example 1, these new constructions enable us to prove the proposition (†) *internally*. (This is not possible with usual pairs and projections since the first projection does not commute with application.)

$$(f : (X : \mathcal{U}) \to X \to X) \to \\ (A : \mathcal{U}) \to (P : A \to \mathcal{U}) \to (a : A) \to P\,a \to P\,(f\,A\,a) \quad (\dagger)$$

However, unlike previous type theories with internalized parametricity [Bernardy and Moulin, 2012, 2013], the system presented here *does not compute* parametricity types: for instance, parametricity conditions are *isomorphic to* function types, rather than function types themselves. More precisely, the binary *meta-operator* $[\![A]\!] \ni a$ of chapter 1 (computing parametricity predicates) corresponds to the *syntactic construction* $(\forall i.A) \ni a$. Bernardy et al. [2010] showed that $\vdash [\![a]\!] : [\![A]\!] \ni a$ follows from $\vdash a : A$; in our new calculus the conclusion becomes $\vdash (\langle i \rangle a)! : (\forall i.A) \ni a$, but unlike $[\![A]\!] \ni a$, the type $(\forall i.A) \ni a$ does not *compute* to the free theorem associated with $a : A$. For instance while one has

$$[\![(x : A) \to B]\!] \ni f = (x : A) \to (\dot{x} : [\![A]\!] \ni x) \to [\![B]\!] \ni (f\,x)$$

in the system of chapter 1, in our new system the corresponding type $(\forall i.((x : A) \to B)) \ni f$ does not reduce to an iterated dependent product (in fact it does not reduce at all). As we show in section 4, this does not appear to be a problem in practice.

Furthermore unlike for the previous chapters, we provide (in section 5) a *denotational* semantics, in the form of a presheaf model, for this type theory. This model is a refinement of the presheaf semantics used to interpret nominal sets with restrictions [Bezem et al., 2013, Pitts, 2014].

Cubical type theory [Cohen et al., 2015] is closely related to our type theory, but there are some differences. The first difference is that our theory is about unary predicates (although as we argue in section 4, our system generalizes to binary parametricity), whereas cubical type theory is about binary relations. However, we could consider a unary version of cubical type theory, and use the notation Ray for the unary version of

the Path constructor of cubical type theory. Ray would then come with the following inference rules:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{Ray}\, A\, a} \qquad \frac{\Gamma \vdash A \qquad \Gamma, i \vdash a : A}{\Gamma \vdash \langle i \rangle a : \mathsf{Ray}\, A\, a\, (i\, \mathbf{0})}$$

$$\frac{\Gamma \vdash t : \mathsf{Ray}\, A\, a \qquad \Gamma \vdash i}{\Gamma \vdash t@i : A} \qquad \frac{\Gamma \vdash t : \mathsf{Ray}\, A\, a}{\Gamma \vdash t@\mathbf{0} \equiv a : A}$$

The type $\mathsf{Ray}\, A\, a$ is analogous to our parametricity type $(\forall i.A) \ni a$. For instance in both cases applying the special symbol **'0'** to a ray yields its origin. However in our system the formation rules differ in that a color is bound in the type A as well. Indeed the corresponding rules would be

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash a : A@\mathbf{0}}{\Gamma \vdash A \ni a} \qquad \frac{\Gamma, i \vdash A \qquad \Gamma, i \vdash a : A}{\Gamma \vdash (\langle i \rangle a)! : (\forall i.A) \ni a\, (i\, \mathbf{0})}$$

$$\frac{\Gamma \vdash t : \forall i.A \qquad i \notin \Gamma}{\Gamma, i \vdash t@i : A} \qquad \frac{\Gamma \vdash t : \forall i.A}{\Gamma \vdash t@\mathbf{0} : A\, (i\, \mathbf{0})}$$

It is not clear how to prove Example 1 with the version from cubical type theory, because our method heavily relies on our Pair-Pred rule which requires a bound color in the type.

2 Syntax and typing rules

In this section we define the syntax and typing rules of our parametric type theory. We proceed step by step, starting with the underlying type theory (section 2.1), on top of which we later add the nominal (section 2.2) then parametric (section 2.3) fragments. We formally present our system in section 2.4; the raw syntax and typing rules can respectively be found in Definitions 1 and 2.

2.1 Underlying type theory

We consider Martin-Löf's Logical Framework [Nordström et al., 1990] (with an untyped conversion instead of equality judgments), a minimal type theory with one universe of small types and dependent function types. We formulate our presentation à la Russel, in that we use the same notation for the codes for small types and the types themselves.

We consider three kinds of judgments, respectively expressing the correctness of contexts, correctness of types and well-typeness of terms.

$$\Gamma \vdash \qquad \Gamma \vdash A \qquad \Gamma \vdash t : A$$

The raw syntax of expressions and context are given by

$$
\begin{array}{rll}
\mathsf{Expr} \ni A, B, t, u & \stackrel{\text{def}}{=} \mathcal{U} & \textit{universe of small types} \\
& \mid (x : A) \to B & \textit{dependent function space} \\
& \mid t\ u & \textit{application} \\
& \mid \lambda(x : A).t & \textit{abstraction} \\
& \mid x & \textit{variable}
\end{array}
$$

$$
\begin{array}{rll}
\mathsf{Ctx} \ni \Gamma & \stackrel{\text{def}}{=} \Diamond & \textit{empty context} \\
& \mid \Gamma, x : A & \textit{context extension}
\end{array}
$$

The typing judgments are

$\boxed{\Gamma \vdash}$

$$
\textsc{Empty}\ \frac{}{\Diamond \vdash} \qquad \textsc{Ext}\ \frac{\Gamma \vdash \qquad \Gamma \vdash A}{\Gamma, x : A \vdash}
$$

$\boxed{\Gamma \vdash A}$

$$
\textsc{Universe}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathcal{U}} \qquad \textsc{Prod}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash (x : A) \to B} \qquad \textsc{Small}\ \frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash A}
$$

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

$$
\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B \qquad A \equiv B}{\Gamma \vdash t : B} \qquad \textsc{Var}\ \frac{\Gamma \vdash A}{\Gamma, x : A, \Delta \vdash x : A}
$$

$$
\textsc{Abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : (x : A) \to B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x \mapsto u]}
$$

$$
\textsc{Prod-}\mathcal{U}\ \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \to B : \mathcal{U}}
$$

The equality used in Conv is defined as the smallest congruence containing the following β- and η-equalities:

$$
\begin{array}{ll}
\beta & \eta \\
(\lambda(x : A).t)\ u \equiv t[x \mapsto u] & t \equiv \lambda(x : A).(t\ x)
\end{array}
$$

Where $t[x \mapsto u]$ denotes the term t where the variable x is substituted by the term u. Substitution is a meta-operation defined on the raw syntax in Definition 4.

2.2 Nominal extension

We assume a countably infinite set $\mathbb{I}$ of symbols, which we also sometimes call *colors* following [Bernardy and Moulin, 2013], and extend the type theory presented in the previous section with new syntactic constructions for color binders and color application.

The main innovation of the type theory presented here is that terms may depend on (a finite number of) colors.

We further assume a special symbol $\mathbf{0} \notin \mathbb{I}$. The metasyntactic variables $i, j, \ldots$ range over colors, while φ range over $\mathbb{I} \cup \{\mathbf{0}\}$. The postfix meta-operator $(i\,\varphi)$ denotes substitution of the color i by φ (if $\varphi \in \mathbb{I}$) or erasure of the color i (if $\varphi = \mathbf{0}$). It is defined by induction on the raw syntax in Definition 5.

$$
\begin{array}{rll}
\mathsf{Expr} \ni A, t \stackrel{\text{def}}{=} \ldots & & \\
\quad \mid \forall i.A & & \textit{color product} \\
\quad \mid t@\varphi & & \textit{color application} \\
\quad \mid \langle i \rangle t & & \textit{color abstraction} \\
\mathsf{Ctx} \ni \Gamma \stackrel{\text{def}}{=} \ldots & & \\
\quad \mid \Gamma, i & & \textit{color context extension}
\end{array}
$$

$$
\textsc{ColExt}\ \frac{\Gamma \vdash}{\Gamma, i \vdash} \qquad \textsc{ColPi}\ \frac{\Gamma, i \vdash A}{\Gamma \vdash \forall i.A}
$$

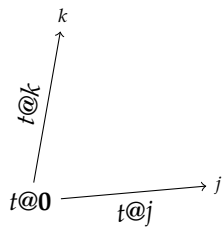
$$
\textsc{ColAbs}\ \frac{\Gamma, i \vdash t : A}{\Gamma \vdash \langle i \rangle t : \forall i.A} \qquad \textsc{ColPi-}\mathcal{U}\ \frac{\Gamma, i \vdash A : \mathcal{U}}{\Gamma \vdash \forall i.A : \mathcal{U}}
$$

$$
\textsc{ColApp}\ \frac{\Gamma, \vec{j} \vdash t : \forall i.A \qquad i \notin \Gamma, \vec{j}}{\Gamma, i, \vec{j} \vdash t@i : A} \qquad \textsc{ColApp-Orig}\ \frac{\Gamma \vdash t : \forall i.A}{\Gamma \vdash t@\mathbf{0} : A(i\,\mathbf{0})}
$$

Remark 1. *The special symbol* $\mathbf{0}$ *can only be used as argument in a color application; color names can only be used in color bindings or as argument in a color application: in particular* $\langle i \rangle i$ *violates the syntax. Furthermore, color application is linear, in the sense that applying the same color twice (such as in* $t@i@i$*)*

yields an ill-typed term. Linearity is enforced by the ColApp *rule: the color being applied must be fresh in the conclusion.* $\vec{j}$ *denotes a list of colors* $j_1, ..., j_n$*. Allowing the fresh color* i *to be introduced before* $\vec{j}$ *enables color swapping; for instance if* $\Gamma \vdash t : \forall i.\forall j.A$ *then one can derive both* $\Gamma \vdash \langle j\rangle\langle i\rangle t@i@j : \forall j.\forall i.A$ *and* $\Gamma \vdash \langle i\rangle\langle j\rangle t@i@j : \forall j.\forall i.A$*.*

We call any term of type $\forall i.A$, such as $\langle i\rangle t$, a *ray*. If t is a ray we then say that the term $t@\mathbf{0}$ is its *origin*; applying a ray to a color can be seen as "specifying its direction". For instance if $t : \forall i.A$ then its origin $t@\mathbf{0}$ and the terms $t@j$ and $t@k$ (respectively of type $A(i\,\mathbf{0})$, $A(i\,j)$ and $A(i\,k)$) can be seen geometrically as



The conversion definition is amended by the addition of the following β- and η-equalities for colors.

Col-β	Col-η
$(\langle i\rangle t)@\varphi \equiv t(i\,\varphi)$	$t \equiv \langle i\rangle(t@i)$

We remark that since color substitution does not add any redex, unlike the usual β-reduction the colored version does not introduce any termination difficulty.

2.3 Parametric extension

As explained in the introduction, we introduce a new syntactic construction (in fact a new kind of ray) $(\!|a, p|\!)$ to pair any term a with its parametricity witness p. We also introduce a new postfix constructor $\cdot!$ corresponding to the second projection for such pairs: $(\!|a, p|\!)! \equiv p$. The first projection is obtained by applying the special symbol $\mathbf{0}$: $(\!|a, p|\!)@\mathbf{0} \equiv a$.

Reynolds associates each type with a predicate. Here, each type is associated not with a single predicate, but with a predicate for every color. By making the new pair constructor a term former for our color product, we are able to distinguish between multiple applications of parametricity using color abstraction and application. Furthermore the linear property of ColApp forces nested applications of parametricity to be used with different colors. As shown in the introduction, this property is crucial for a system where parametricity can be nested.

$$
\begin{aligned}
\mathsf{Expr} \ni A, P, a, p, u \stackrel{\text{def}}{=} \ & \dots \\
& \mid (\!| a, p |\!) && \textit{colored pair} \\
& \mid P \ni a && \textit{parametricity type} \\
& \mid u! && \textit{parametricity proof} \\
& \mid \Psi_A P && \textit{parametricity predicate} \\
& \mid \Phi_t u && \textit{parametricity function}
\end{aligned}
$$

The origin of the ray $(\!| a, p |\!) : \forall i.A$ is its first component a. If $a : A(i\,\mathbf{0})$, then a *ray from* a is any $p : (\forall i.A) \ni a$; they can be paired to form the ray $(\!| a, p |\!)$ of origin a.

While we have a single notion of rays, it really hides three kinds of objects ("term-rays", "type-rays" and "function-rays") used respectively for parametricity proofs, types and functions.

- $\cdot!$ takes a ray $u : \forall i.A$ and forms a ray from its origin $u@\mathbf{0} : A(i\,\mathbf{0})$;
- Ψ takes the parametric interpretation of a type A (*i.e.*, a predicate $P : A \to \mathcal{U}$) and forms a ray from A; and
- Φ takes the parametric interpretation of a dependent function

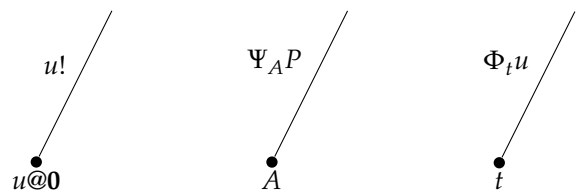
  $$t : (x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}),$$

  i.e., a function of rays

  $$u : (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni t\,(x@\mathbf{0}),$$

  and forms a ray from t.

One may remark that the constructors Ψ and Φ are respectively reminiscent of the parametricity clauses for sorts and functions types. We represent the rays $(\!| u@\mathbf{0}, u! |\!)$, $(\!| A, \Psi_A P |\!)$, and $(\!| t, \Phi_t u |\!)$ as follows. The dot represents a term (resp. type), while the line represents its parametricity proof (resp. predicate) for an arbitrary dimension.



These rays are typed using In-Abs followed by Param, In-Pred and In-Fun, respectively.

Remark 2. *Unlike the systems of Bernardy et al. [2010] or Bernardy and Moulin [2012], we have constructors for parametricity types, predicates or functions, rather than a meta-operator. For instance with the earlier calculi one would obtain*

$$\vdash [\![f]\!] : [\![(x : A) \to B]\!] \ni f = (x : A) \to (\dot{x} : [\![A]\!] \ni x) \to [\![B]\!] \ni (f\ x)$$

from $\vdash f : (x : A) \to B$*, and*

$$\vdash [\![P]\!] : [\![A \to \mathcal{U}]\!] \ni P = (x : A) \to (\dot{x} : [\![A]\!] \ni x) \to P\ x \to \mathcal{U}$$

from $\vdash P : A \to \mathcal{U}$*. In other words, the types* $[\![(x : A) \to B]\!] \ni f$ *and* $[\![A \to \mathcal{U}]\!] \ni P$ *compute to the free theorem associated with* $f : (x : A) \to B$ *and* $P : A \to \mathcal{U}$*, respectively.*

However, in our new system the corresponding types, respectively $(\forall i.((x : A) \to B)) \ni f$ *and* $\forall i.(A \to \mathcal{U}) \ni P$*, do not compute. But as we will show in section 4.2 these types are isomorphic (Definition 6) to the aforementioned free theorems.*

$$\textsc{Out}\ \frac{\Gamma \vdash \forall i.A \qquad \Gamma \vdash a : A(i\,\mathbf{0})}{\Gamma \vdash (\forall i.A) \ni a} \qquad \textsc{Out-}\mathcal{U}\ \frac{\Gamma \vdash \forall i.A : \mathcal{U} \qquad \Gamma \vdash a : A(i\,\mathbf{0})}{\Gamma \vdash (\forall i.A) \ni a : \mathcal{U}}$$

$$\textsc{In-Pred}\ \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash P : A \to \mathcal{U}}{\Gamma \vdash \Psi_A P : (\forall i.\mathcal{U}) \ni A}$$

$$\textsc{In-Fun}\ \frac{\Gamma \vdash t : (x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}) \qquad \Gamma \vdash u : (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni t\ (x@\mathbf{0})}{\Gamma \vdash \Phi_t u : (\forall i.((x : A) \to B)) \ni t}$$

$$\textsc{In-Abs}\ \frac{\Gamma \vdash a : A(i\,\mathbf{0}) \qquad \Gamma \vdash p : (\forall i.A) \ni a}{\Gamma \vdash (\!|a, p|\!) : \forall i.A} \qquad \textsc{Param}\ \frac{\Gamma \vdash u : \forall i.A}{\Gamma \vdash u! : (\forall i.A) \ni u@\mathbf{0}}$$

One can pair $\Psi_A P$ (resp. $\Phi_t u$) with A (resp. t) to obtain a ray. For convenience we define the two following macros:

$$A \bowtie P \stackrel{\text{def}}{=} (\!|A, \Psi_A P|\!) \qquad\qquad \langle\!| t, u |\!\rangle \stackrel{\text{def}}{=} (\!|t, \Phi_t u|\!)$$

Remark 3. *We could have defined* $\cdot \bowtie \cdot$ *and* $\langle\!| \cdot, \cdot |\!\rangle$ *as syntactic constructors instead of* Ψ *and* Φ*, at the expense of additional term formers for rays and equalities between rays. Instead, we derive the two following introduction rules for*

our new macros:

$$\textsc{In-Abs-Pred}\ \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash P : A \to \mathcal{U}}{\Gamma \vdash A \bowtie P : \forall i.\mathcal{U}}$$

$$\textsc{In-Abs-Fun}\ \frac{\Gamma \vdash t : (x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}) \qquad \Gamma \vdash u : (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni t\,(x@\mathbf{0})}{\Gamma \vdash \langle\!|t, u|\!\rangle : \forall i.((x : A) \to B)}$$

*(*In-Abs-Pred *and* In-Pred *are interdefinable, and so are* In-Abs-Fun *and* In-Fun.*)*

Proof.

In-Abs-Pred**.** By In-Pred we get $\Gamma \vdash \Psi_A P : (\forall i.\mathcal{U}) \ni A$, and by In-Abs we deduce $\Gamma \vdash \langle\!|A, \Psi_A P|\!\rangle : \forall i.\mathcal{U}$.

In-Abs-Fun**.** We get $\Gamma \vdash \Phi_t u : (\forall i.(A \to B)) \ni t$ by applying In-Fun on the premises, and deduce $\Gamma \vdash \langle\!|t, \Phi_t u|\!\rangle : \forall i.(A \to B)$ from In-Abs. □

We also define macros for when a ray is applied to a color:

$$\langle\!|a\,,_i\,p|\!\rangle \stackrel{\text{def}}{=} \langle\!|a, p|\!\rangle@i \qquad A \bowtie_i P \stackrel{\text{def}}{=} (A \bowtie P)@i \qquad \langle\!|t\,,_i\,u|\!\rangle \stackrel{\text{def}}{=} \langle\!|t, u|\!\rangle@i$$

Finally, we extend the definition of the conversion $\equiv$. Since the origin of the ray $\langle\!|a, p|\!\rangle$ is a, $\cdot@\mathbf{0}$ is the first projection. Similarly, $\cdot!$ is the second projection, since the parametricity witness for the first component is the second component. We also have η-expansion on rays.

$$\textsc{Pair-Orig}\ \langle\!|a, p|\!\rangle@\mathbf{0} \equiv a \qquad \textsc{Pair-Param}\ \langle\!|a, p|\!\rangle! \equiv p$$

$$\textsc{Surj-Param}\ \langle\!|a@\mathbf{0}, a!|\!\rangle \equiv a \qquad \textsc{Surj-Type}\ \Psi_{T@\mathbf{0}}(\lambda(x : T@\mathbf{0}).\ (\forall i.T@i) \ni x) \equiv T!$$

$$\textsc{Surj-Fun}\ \Phi_{t@\mathbf{0}}(\lambda(x : \forall i.A).\ (\langle i\rangle (t@i)\ (x@i))!\ ) \equiv t!$$

$A \bowtie P$ forms a ray obtained by pairing the type A with its parametricity theorem P, while $\cdot \ni a$ destroys a ray and applies its parametricity theorem to a. Therefore combining the two merely yields $P\,a$. Similarly, $\langle\!|t, u|\!\rangle$ forms a ray obtained by pairing a function with its parametricity theorem (represented as a function of rays). Therefore applying the ray to an argument distributes the application over the pair of functions.

$$\textsc{Pair-Pred}\ (\forall i.(A \bowtie_i P)) \ni a \equiv P\,a \qquad \textsc{Pair-App}\ \langle\!|t\,,_i\,u|\!\rangle\,(a@i) \equiv \langle\!|t\,(a@\mathbf{0})\,,_i\,u\,a|\!\rangle$$

Remark 4. *One might think that we could define $\Phi_t u$ as the term*

$$\langle i\rangle\lambda(x : A).\ (\!|t\, x\, ,_i\, u\, (\langle i\rangle x)|\!).$$

However this term is ill-typed due the double color binding $\langle i\rangle$: indeed, one cannot derive $i, x : A \vdash \langle i\rangle x : \forall i.A$.

Remark 5. *We derive the following rules:*

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash P : A \to \mathcal{U}}{\Gamma \vdash \forall i.A \bowtie_i P : \mathcal{U}}\ \textsc{In-Abs-Pred}'$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash p : P\,a}{\Gamma \vdash (\!|a, p|\!) : \forall i.A \bowtie_i P}\ \textsc{In-Abs}' \qquad \frac{i \notin \Gamma \qquad \Gamma \vdash a : A \qquad \Gamma \vdash p : P\,a}{\Gamma, i \vdash (\!|a\, ,_i\, p|\!) : A \bowtie_i P}\ \textsc{In-Abs}_i$$

$$\frac{i \notin \Gamma \qquad \Gamma \vdash t : A(i\,\mathbf{0}) \to B(i\,\mathbf{0}) \qquad \Gamma \vdash u : (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni t\,(x@\mathbf{0})}{\Gamma, i \vdash \langle\!|t\, ,_i\, u|\!\rangle : A \to B}\ \textsc{In-Abs-Fun}_i$$

Proof.

In-Abs-Pred′. In-Abs-Pred gives $\Gamma \vdash A \bowtie P : \forall i.\mathcal{U}$, hence $\Gamma, i \vdash A \bowtie_i P : \mathcal{U}$ by ColApp, and $\Gamma \vdash \forall i.A \bowtie_i P : \mathcal{U}$ follows by ColPi-$\mathcal{U}$.

In-Abs′. We remark by Pair-Orig that $A \equiv (\!|A, \Psi_A P|\!)@\mathbf{0} = (A \bowtie P)@\mathbf{0}$, and Pair-Pred gives that $P\,a \equiv (\forall i.(A \bowtie_i P)) \ni a$. Hence by Conv we respectively get $\Gamma \vdash a : (\!|A, \Psi_A P|\!)@\mathbf{0}$ and $\Gamma \vdash p : (\forall i.(A \bowtie_i P)) \ni a$. We finally deduce $\Gamma \vdash (\!|a, p|\!) : \forall i.(A \bowtie_i P)$ by In-Abs.

In-Abs$_i$. Direct consequence of In-Abs′ and ColApp.

In-Abs-Fun$_i$. Direct consequence of In-Abs-Fun and ColApp. □

Before making the syntax more formal, we start with an example and note that we can already prove parametricity for the polymorphic identity. (Which does not come as a surprise since as shown in the introduction, introducing pairing is not necessary for this example.)

Example 1. *The type*

$$(f : (X : \mathcal{U}) \to X \to X) \to \\ (A : \mathcal{U}) \to (P : A \to \mathcal{U}) \to (a : A) \to P\,a \to P\,(f\,A\,a)$$

is inhabited.

Let Γ be a context with

- $f : (X : \mathcal{U}) \to X \to X$;
- $A : \mathcal{U}$;
- $P : A \to \mathcal{U}$;
- $a : A$; *and*
- $p : P\,a$.

We now show how to construct a term of type $P\,(f\,A\,a)$ *in context* Γ.

We deduce from In-Abs-Pred *and* ColApp *that* $\Gamma, i \vdash A \bowtie_i P : \mathcal{U}$. *Furthermore* In-Abs$_i$ *gives that* $\Gamma, i \vdash (\!|a\,,_i\,p|\!) : A \bowtie_i P$. *Thus by* App *and* ColAbs *we get* $\Gamma \vdash \langle i\rangle(f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!)) : \forall i.(A \bowtie_i P)$, *then by* Param *and* Col-β

$$\Gamma \vdash (\langle i\rangle(f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!)))\,!\,!\,:\,(\forall i.(A \bowtie_i P)) \ni (f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!))\,(i\,\mathbf{0})$$

Now by definition and Pair-Orig

$$(f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!))\,(i\,\mathbf{0}) = f\,(i\,\mathbf{0})\,(A \bowtie_i P)\,(i\,\mathbf{0})\,(\!|a\,,_i\,p|\!)\,(i\,\mathbf{0}) = f\,A\,a,$$

thus by Pair-Pred $((\forall i.(A \bowtie_i P)) \ni (f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!))\,(i\,\mathbf{0})) \equiv P\,(f\,A\,a)$. *We therefore conclude by* Conv, *that*

$$\Gamma \vdash (\langle i\rangle(f\,(A \bowtie_i P)\,(\!|a\,,_i\,p|\!)))\,!\,!\,:\,P\,(f\,A\,a)$$

(We reuse this method, namely introducing a new color with $\langle i\rangle$ *and instancing the polymorphic function at type* $\cdot \bowtie_i \cdot$, *in the other examples of this chapter.)*

2.4 Full system

We now formally define our system by giving its raw syntax (Definition 1) and typing judgments (Definition 2). We also define the untyped conversion (Definition 3), as well as variable and color substitutions (Definitions 4 and 5).

Definition 1 (Raw syntax)**.**

$$\begin{array}{lll}
\mathsf{Expr} \ni A, B, P, t, u, a, p \stackrel{\text{def}}{=} \mathcal{U} & & \textit{universe of small types} \\
\quad | \ (x : A) \to B & & \textit{dependent function space} \\
\quad | \ t\ u & & \textit{application} \\
\quad | \ \lambda(x : A).t & & \textit{abstraction} \\
\quad | \ x & & \textit{variable} \\
\quad | \ \forall i.A & & \textit{color product} \\
\quad | \ t@\varphi & & \textit{color application} \\
\quad | \ \langle i\rangle t & & \textit{color abstraction}
\end{array}$$

$$
\begin{array}{rr}
\mid (\!|a, p|\!) & \textit{colored pair} \\
\mid P \ni a & \textit{parametricity type} \\
\mid u! & \textit{parametricity proof} \\
\mid \Psi_A P & \textit{parametricity predicate} \\
\mid \Phi_t u & \textit{parametricity function}
\end{array}
$$

$$
\begin{array}{rr}
\mathsf{Ctx} \ni \Gamma \stackrel{\text{def}}{=} \diamond & \textit{empty context} \\
\mid \Gamma, x : A & \textit{context extension} \\
\mid \Gamma, i & \textit{color context extension}
\end{array}
$$

Definition 2 (Typing judgments)**.**

$\boxed{\Gamma \vdash}$

$$
\textsc{Empty}\ \frac{}{\diamond \vdash} \qquad \textsc{Ext}\ \frac{\Gamma \vdash \qquad \Gamma \vdash A}{\Gamma, x : A \vdash} \qquad \textsc{ColExt}\ \frac{\Gamma \vdash}{\Gamma, i \vdash}
$$

$\boxed{\Gamma \vdash A}$

$$
\textsc{Universe}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathcal{U}} \qquad \textsc{Prod}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash (x : A) \to B} \qquad \textsc{Small}\ \frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash A}
$$

$$
\textsc{ColPi}\ \frac{\Gamma, i \vdash A}{\Gamma \vdash \forall i.A} \qquad \textsc{Out}\ \frac{\Gamma \vdash \forall i.A \qquad \Gamma \vdash a : A(i\,\mathbf{0})}{\Gamma \vdash (\forall i.A) \ni a}
$$

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

$$
\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B \qquad A \equiv B}{\Gamma \vdash t : B} \qquad \textsc{Var}\ \frac{\Gamma \vdash A}{\Gamma, x : A, \Delta \vdash x : A}
$$

$$
\textsc{Abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : (x : A) \to B} \qquad \textsc{App}\ \frac{\Gamma \vdash t : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x \mapsto u]}
$$

$$
\textsc{Prod-}\mathcal{U}\ \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \to B : \mathcal{U}} \qquad \textsc{Out-}\mathcal{U}\ \frac{\Gamma \vdash \forall i.A : \mathcal{U} \qquad \Gamma \vdash a : A(i\,\mathbf{0})}{\Gamma \vdash (\forall i.A) \ni a : \mathcal{U}}
$$

$$
\textsc{ColAbs}\ \frac{\Gamma, i \vdash t : A}{\Gamma \vdash \langle i \rangle t : \forall i.A} \qquad \textsc{In-Abs}\ \frac{\Gamma \vdash a : A(i\,\mathbf{0}) \qquad \Gamma \vdash p : (\forall i.A) \ni a}{\Gamma \vdash (\!|a, p|\!) : \forall i.A}
$$

$$\textsc{ColPi-}\mathcal{U}\ \frac{\Gamma, i \vdash A : \mathcal{U}}{\Gamma \vdash \forall i.A : \mathcal{U}} \qquad \textsc{In-Pred}\ \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash P : A \to \mathcal{U}}{\Gamma \vdash \Psi_A P : (\forall i.\mathcal{U}) \ni A}$$

$$\textsc{In-Fun}\ \frac{\Gamma \vdash t : (x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}) \qquad \Gamma \vdash u : (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni t\,(x@\mathbf{0})}{\Gamma \vdash \Phi_t u : (\forall i.((x : A) \to B)) \ni t}$$

$$\textsc{ColApp}\ \frac{\Gamma, \vec{j} \vdash t : \forall i.A \qquad i \notin \Gamma, \vec{j}}{\Gamma, i, \vec{j} \vdash t@i : A} \qquad \textsc{ColApp-Orig}\ \frac{\Gamma \vdash t : \forall i.A}{\Gamma \vdash t@\mathbf{0} : A(i\,\mathbf{0})} \qquad \textsc{Param}\ \frac{\Gamma \vdash u : \forall i.A}{\Gamma \vdash u! : (\forall i.A) \ni u@\mathbf{0}}$$

Definition 3 (Conversion). *The equality used in* Conv *is defined as the smallest congruence containing the following equalities:*

$$\beta\ (\lambda(x : A).t)\ u \equiv t[x \mapsto u] \qquad \eta\ t \equiv \lambda(x : A).(t\,x) \qquad \textsc{Col-}\beta\ (\langle i \rangle t)@\varphi \equiv t(i\,\varphi)$$

$$\textsc{Col-}\eta\ t \equiv \langle i \rangle (t@i) \qquad \textsc{Pair-Orig}\ (\!|a, p|\!)@\mathbf{0} \equiv a \qquad \textsc{Pair-Param}\ (\!|a, p|\!)! \equiv p$$

$$\textsc{Surj-Param}\ (\!|a@\mathbf{0}, a!|\!) \equiv a \qquad \textsc{Surj-Type}\ \Psi_{T@\mathbf{0}}(\lambda(x : T@\mathbf{0}).\ (\forall i.T@i) \ni x) \equiv T!$$

$$\textsc{Surj-Fun}\ \Phi_{t@\mathbf{0}}(\lambda(x : \forall i.A).\ (\langle i \rangle (t@i)\ (x@i))!\ ) \equiv t!$$

$$\textsc{Pair-Pred}\ (\forall i.(A \bowtie_i P)) \ni a \equiv P\,a \qquad \textsc{Pair-App}\ (\!|t\ ,_i u|\!)\ (a@i) \equiv (\!|t\ (a@\mathbf{0})\ ,_i u\,a|\!)$$

Definition 4 (Substitution). *For any $t, v \in \mathsf{Expr}$, we define the term $t[z \mapsto v]$ (denoting t where the variable z was replaced by v) by induction on the raw syntax of t. (We assume that the variable z is not bound in t.)*

$$\begin{aligned}
\mathcal{U}[z \mapsto v] &= \mathcal{U} \\
((x : A) \to B)[z \mapsto v] &= (x : A[z \mapsto v]) \to B[z \mapsto v] \\
(t\,u)[z \mapsto v] &= (t[z \mapsto v])\ (u[z \mapsto v]) \\
(\lambda(x : A).t)[z \mapsto v] &= \lambda(x : A[z \mapsto v]).(t[z \mapsto v]) \\
x[z \mapsto v] &= v \text{ if } x = z, \text{ and } x \text{ otherwise} \\
(\forall i.A)[z \mapsto v] &= \forall i.(A[z \mapsto v]) \\
(t@\varphi)[z \mapsto v] &= (t[z \mapsto v])@\varphi \\
(\langle i \rangle t)[z \mapsto v] &= \langle i \rangle (t[z \mapsto v])
\end{aligned}$$

$$
\begin{aligned}
(\!|t, u|\!)[z \mapsto v] &= (\!|t[z \mapsto v], u[z \mapsto v]|\!) \\
(P \ni a)[z \mapsto v] &= (P[z \mapsto v]) \ni (a[z \mapsto v]) \\
(a!)[z \mapsto v] &= (a[z \mapsto v])! \\
(\Psi_A P)[z \mapsto v] &= \Psi_{A[z \mapsto v]}(P[z \mapsto v]) \\
(\Phi_t u)[z \mapsto v] &= \Phi_{t[z \mapsto v]}(u[z \mapsto v])
\end{aligned}
$$

Definition 5 (Color erasure and substitution)**.** *For any $t \in \mathsf{Expr}$ and $\varphi \in \mathbb{I} \cup \{\mathbf{0}\}$, we define the term $t(j\,\varphi)$ (denoting the color substitution of j by k if $\varphi = k \in \mathbb{I}$, or the color erasure of j in $\varphi = \mathbf{0}$) by induction on the raw syntax of t. (We assume that the color j is not bound in t.)*

$$
\begin{aligned}
\mathcal{U}(j\,\varphi) &= \mathcal{U} \\
((x : A) \to B)(j\,\varphi) &= (x : A(j\,\varphi)) \to B(j\,\varphi) \\
(t\,u)(j\,\varphi) &= (t(j\,\varphi))\,(u(j\,\varphi)) \\
(\lambda(x : A).t)(j\,\varphi) &= \lambda(x : A(j\,\varphi)).(t(j\,\varphi)) \\
x(j\,\varphi) &= x \\
(\forall i.A)(j\,\varphi) &= \forall i.(A(j\,\varphi)) \\
(t@\varphi')(j\,\varphi) &= t@\varphi \text{ if } \varphi' = j \in \mathbb{I}, \text{ and } (t(j\,\varphi))@\varphi' \text{ otherwise} \\
(\langle i \rangle t)(j\,\varphi) &= \langle i \rangle (t(j\,\varphi)) \\
(\!|t, u|\!)(j\,\varphi) &= (\!|t(j\,\varphi), u(j\,\varphi)|\!) \\
(P \ni a)(j\,\varphi) &= (P(j\,\varphi)) \ni (a(j\,\varphi)) \\
(a!)(j\,\varphi) &= (a(j\,\varphi))! \\
(\Psi_A P)(j\,\varphi) &= \Psi_{A(j\,\varphi)}(P(j\,\varphi)) \\
(\Phi_t u)(j\,\varphi) &= \Phi_{t(j\,\varphi)}(u(j\,\varphi))
\end{aligned}
$$

Remark 6. *We deduce by definition that*

$$
(\!|a\,,_i p|\!)(i\,j) = (\!|a\,,_j p|\!) \qquad (A \bowtie_i P)(i\,j) = A \bowtie_j P \qquad \langle\!|u\,,_i t|\!\rangle(i\,j) = \langle\!|u\,,_j t|\!\rangle
$$

Furthermore using Pair-Orig*, we get*

$$
(\!|a\,,_i p|\!)(i\,\mathbf{0}) \equiv a \qquad (A \bowtie_i P)(i\,\mathbf{0}) \equiv A \qquad \langle\!|u\,,_i t|\!\rangle(i\,\mathbf{0}) \equiv u
$$

3 Meta-properties of the type theory

Lemma 1 (Substitution)**.** *If $t, u, u' \in \mathsf{Expr}$ and $z \neq z'$ are two variables such that z is not free in u', then $t[z \mapsto u][z' \mapsto u'] = t[z' \mapsto u'][z \mapsto u[z' \mapsto u']]$.*

Proof. By structural induction on the raw syntax of t. Substitution is defined (Definition 4) by uniformly applying it recursively on each subterm, except for the variable case. We are therefore only treating the case where t is a variable here; other cases stem from straightforward uses of the induction hypothesis.

- If $x = z$ we get
  $z[z \mapsto u][z' \mapsto u'] = u[z' \mapsto u'] = z[z' \mapsto u'][z \mapsto u[z' \mapsto u']]$
- If $x = z'$ we get
  $z'[z \mapsto u][z' \mapsto u'] = u' = u'[z \mapsto u[z' \mapsto u']]$
  $= z'[z' \mapsto u'][z \mapsto u[z' \mapsto u']]$
- Otherwise we have
  $x[z \mapsto u][z' \mapsto u'] = x = x[z' \mapsto u'][z \mapsto u[z' \mapsto u']]$ □

Theorem 1 (Substitution preserves typing)**.**
If $\Gamma, x : B \vdash t : A$ *and* $\Gamma \vdash u : B$*, then* $\Gamma \vdash t[x \mapsto u] : A[x \mapsto u]$*.*

Proof. By structural induction on the typing judgment. □

Theorem 2 (Color erasure and substitution preserve typing)**.**
If $\Gamma, i \vdash t : A$ *then*

- $\Gamma \vdash t(i\,\mathbf{0}) : A(i\,\mathbf{0})$*; and*
- $\Gamma, j \vdash t(i\,j) : A(i\,j)$*.*

Proof. By structural induction on the typing judgment. □

4 Parametricity

In this section we prove that our system properly internalizes unary parametricity, using the special symbol $\mathbf{0}$ for erasure of the argument of parametricity predicates. (Our system can be naturally extended to the n-ary case by using further special symbols $\mathbf{1}, \ldots, \mathbf{n-1}$ for erasure of the other arguments.)

With Example 1 we have already seen an example of application of the Param rule to internally prove the parametricity theorem of a concrete type. We further illustrate the system by giving a few other simple proofs relying on parametricity, including iterated parametricity.

With the system of Bernardy and Moulin [2012], one could use parametricity generically as follows

$$p : (X : \mathcal{U}) \to (x : X) \to [\![X]\!] \ni x$$
$$p = \lambda(X : \mathcal{U}).\, \lambda(x : X).\, [\![x]\!]$$

Here, each application of parametricity needs to be annotated with a color. We thus bind a new name using $\langle i \rangle$ (resp. $\forall i.$), then use $\cdot!$ (resp. $\cdot \ni x$) as our generic parametricity operation:

$$p : (X : \mathcal{U}) \to (x : X) \to (\forall i.X) \ni x$$
$$p = \lambda(X : \mathcal{U}).\, \lambda(x : X).\, (\langle i \rangle x)!$$

We have already seen that $(\forall i.A) \ni$ corresponds to a parametricity predicate for the type A. As we hinted at in the introduction, the color binding i allows us to distinguish each application of parametricity. (As a side remark, since the Param 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.)

However, unlike with [Bernardy et al., 2010] or [Bernardy and Moulin, 2012], there is no meta-operator expanding to any free theorem instance in our new calculus. Indeed, while we can deduce $\vdash (\langle i \rangle a)! : (\forall i.A) \ni a$ from $\vdash a : A$, unlike $[\![A]\!] \ni a$ the parametricity type $(\forall i.A) \ni a$ does not reduce to the free theorem associated with $a : A$. (We will expand on that in section 4.2.) As shown in the following examples, we can also prove unary parametricity for polymorphic functions (using for instance the mechanical operator of Bernardy et al. [2010] to derive the associated free theorem) by introducing a new color with $\langle i \rangle$ and instancing the polymorphic function at type $\cdot \bowtie_i \cdot$.

4.1 Examples

Example 1 gives an internal proof of the free theorem associated with the polymorphic identity using our syntactic construction for parametricity predicates (Ψ) and the Pair-Pred equality rule. As shown in Example 2, the constructor Φ (or the macro $(\!|\cdot,\cdot|\!)$) is required to give internal proofs of free theorems associated with higher-order types, since there is no other way to form a ray of functions from a function of rays.

Example 2. *Consider the type of predicative Church numerals* $\mathbf{N} \stackrel{\text{def}}{=} (X : \mathcal{U}) \to X \to (X \to X) \to X$*. Proving (unary) parametricity for* $\mathbf{N}$ *means that, assuming a context* Γ *with*

- $f : \mathbf{N}$*;*
- $A : \mathcal{U}$*;*
- $P : A \to \mathcal{U}$*;*
- $z : A$*;*
- $\dot{z} : P\, z$*;*
- $s : A \to A$*; and*
- $\dot{s} : (x : A) \to P\, x \to P\, (s\, x)$*,*

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

As for Example 1, we deduce from In-Abs-Pred *and* ColApp *that* $\Gamma, i \vdash A \bowtie_i P : \mathcal{U}$, *and* In-Abs$_i$ *gives* $\Gamma, i \vdash (\!| z \,,_i \dot{z} |\!) : A \bowtie_i P$. *Furthermore* $\Gamma, x : (\forall i.A \bowtie_i P) \vdash x@\mathbf{0} : A$ *by* ColApp-Orig *and* $\Gamma, x : (\forall i.A \bowtie_i P) \vdash x! : P\,(x@\mathbf{0})$ *by* Param, Pair-Pred *and* Conv, *hence defining* $\dot{s}' \stackrel{\text{def}}{=} \lambda(x : \forall i.A \bowtie_i P).\ \dot{s}\,(x@\mathbf{0})\,(x!)$ *we obtain* $\Gamma \vdash \dot{s}' : (x : \forall i.A \bowtie_i P) \to P\,(s\,(x@\mathbf{0}))$ *by* App *and* Abs, *and* $\Gamma \vdash \dot{s}' : (x : \forall i.A \bowtie_i P) \to (\forall i.A \bowtie_i P) \ni (s\,(x@\mathbf{0}))$ *by* Pair-Pred *and* Conv.

Thus $\Gamma, i \vdash \langle\!| s \,,_i \dot{s}' |\!\rangle : A \bowtie_i P \to A \bowtie_i P$ *by* In-Abs-Fun$_i$, *and*

$$\Gamma \vdash \langle i \rangle (f\ (A \bowtie_i P)\ (\!| z \,,_i \dot{z} |\!)\ \langle\!| s \,,_i \dot{s}' |\!\rangle) : \forall i.A \bowtie_i P$$

follows by App *and* ColAbs. *Now applying* Param *gives*

$$\Gamma \vdash (\langle i \rangle (f\ (A \bowtie_i P)\ (\!| z \,,_i \dot{z} |\!)\ \langle\!| s \,,_i \dot{s}' |\!\rangle))! : (\forall i.A \bowtie_i P) \ni (f\ A\ z\ s)$$

and we conclude by Conv *since* $(\forall i.A \bowtie_i P) \ni f\,A\,z \equiv P\,(f\,A\,z\,s)$ *by* Pair-Pred.

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. It turns out that $\langle\!| t, u |\!\rangle$ can be combined with $(\!| a, p |\!)$ to pair higher order functions with the parametricity proof of their type. The following example illustrates this technique.

Example 3. *Let* $F = (X : \mathcal{U}) \to ((X \to X) \to X) \to X$. *Proving (unary) parametricity for* F *means that, assuming a context* Γ *with*

- $f : F$;
- $A : \mathcal{U}$;
- $P : A \to \mathcal{U}$;
- $g : (A \to A) \to A$; *and*
- $\dot{g} : (h : A \to A) \to ((x : A) \to P\,x \to P\,(h\,x)) \to P\,(g\,h)$,

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

As before we get such a proof by using $\Gamma \vdash (\langle i \rangle (f\ (A \bowtie_i P)\ \langle\!| g, \dot{g}' |\!\rangle))! : P\,(f\,A\,g)$, *where we need to find a function* $\dot{g}'$ *such that*

$$\Gamma \vdash \dot{g}' : (h : \forall i.(A \bowtie_i P \to A \bowtie_i P)) \to P\,(g\,(h@\mathbf{0}))$$

Let $T \stackrel{\text{def}}{=} A \bowtie_i P \to A \bowtie_i P$. *We have* $\Gamma, h : \forall i.T \vdash h@\mathbf{0} : A \to A$ *by* ColApp-Orig. *Furthermore*

$\Gamma, h : \forall i.T, x : A, \dot{x} : P\,x, i \vdash h@i : A \bowtie_i P \to A \bowtie_i P$ *by* ColApp, *and*
$\Gamma, h : \forall i.T, x : A, \dot{x} : P\,x, i \vdash (\!| x \,,_i \dot{x} |\!) : A \bowtie_i P$ *by* In-Abs$_i$, *hence*
$\Gamma, h : \forall i.T, x : A, \dot{x} : P\,x, i \vdash (h@i)\ (\!| x \,,_i \dot{x} |\!) : A \bowtie_i P$ *by* App, *and finally*
$\Gamma, h : \forall i.T, x : A, \dot{x} : P\,x \vdash (\langle i \rangle ((h@i)\ (\!| x \,,_i \dot{x} |\!)))! : (\forall i.A \bowtie_i P) \ni ((h@\mathbf{0})\ x)$
by ColAbs *and* Param.

Since Pair-Pred *gives* $(\forall i.A \bowtie_i P) \ni ((h@\mathbf{0})\ x) \equiv P\,((h@\mathbf{0})\ x)$, *we obtain*

$$\Gamma, h : \forall i.T \vdash \dot{g}\ (h@\mathbf{0})\ (\lambda x : A.\ \lambda \dot{x} : P\,x.\ (\langle i \rangle ((h@i)\ (\!| x \,,_i \dot{x} |\!)))!) : P\,(g\,(h@\mathbf{0})),$$

We can therefore conclude by Abs *and define*

$$\dot{g}' \stackrel{\text{def}}{=} \lambda h : (\forall i.(A \bowtie_i P \to A \bowtie_i P)).\\ \dot{g}\ (h@\mathbf{0})\ (\lambda x : A.\ \lambda \dot{x} : P\,x.\ (\langle i\rangle((h@i)\ (\!|x\ ,_i\ \dot{x}|\!)))!\ )$$

Example 4. *Although our calculus does not have any base type, we could extend it and include base types. For instance we could add three constants* $\mathbb{B} : \mathcal{U}$ *and* $\mathsf{tt}, \mathsf{ff} : \mathbb{B}$ *for booleans. We would then obtain*

- $[\![\mathbb{B}]\!] \stackrel{\text{def}}{=} \lambda(x : \mathbb{B}).\ (\forall i.\mathbb{B}) \ni x\ :\ \mathbb{B} \to \mathcal{U}$*;*
- $[\![\mathsf{tt}]\!] \stackrel{\text{def}}{=} (\langle i\rangle\mathsf{tt})!\ :\ (\forall i.\mathbb{B}) \ni \mathsf{tt}$*; and*
- $[\![\mathsf{ff}]\!] \stackrel{\text{def}}{=} (\langle i\rangle\mathsf{ff})!\ :\ (\forall i.\mathbb{B}) \ni \mathsf{ff}$*.*

4.2 General results

Unlike previous type theories with internalized parametricity [Bernardy and Moulin, 2012, 2013], the system presented in section 2.4 lacks equalities which allow us to compute parametricity types. Expressed in our new syntax, those equalities would become the following conversion rules:

$$(\forall i.\mathcal{U}) \ni A \ \equiv\ A \to \mathcal{U}\text{; and}$$
$$(\forall i.(x : A) \to B) \ni f \ \equiv\ (x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni (f\ (x@\mathbf{0})).$$

The absence of the above equalities did not prevent us from giving proofs of free theorems in the examples in the previous section. This also allows for a simpler system, but how can we ensure that all parametricity theorems hold? The answer is that while the types $(\forall i.\mathcal{U}) \ni A$ and $A \to \mathcal{U}$ are not convertible, they are *isomorphic* in the following sense.

Definition 6 (Isomorphism)**.** *We say that two types A and B are isomorphic when the following 4 conditions hold:*

1. *there exists* $\pi : A \to B$*;*
2. *there exists* $\sigma : B \to A$*;*
3. $\pi\ (\sigma\ x) \equiv x$ *for any x; and*
4. $\sigma\ (\pi\ x) \equiv x$ *for any x.*

Where $\equiv$ *is the untyped conversion relation (Definition 3).*

Theorem 3. *The types* $(\forall i.\mathcal{U}) \ni A$ *and* $A \to \mathcal{U}$ *are isomorphic.*

Proof.

1. We take $\pi \stackrel{\text{def}}{=} \lambda(Q : (\forall i.\mathcal{U}) \ni A).\ \lambda(x : A).\ (\forall i.(\!|A\ ,_i\ Q|\!)) \ni x$. Indeed $\pi : (\forall i.\mathcal{U}) \ni A \to A \to \mathcal{U}$ by Out-$\mathcal{U}$ and In-Abs.
2. We take $\sigma \stackrel{\text{def}}{=} \lambda(P : A \to \mathcal{U}).\ \Psi_A P$. Indeed $\sigma : (A \to \mathcal{U}) \to (\forall i.\mathcal{U}) \ni A$ by In-Pred.

3. We obtain $\pi\,(\sigma\,P) \equiv \lambda x : A.\,(\forall i.A \bowtie_i P) \ni x$ by β-reduction. Hence $\pi\,(\sigma\,P) \equiv \lambda x : A.\,P\,x$ by Pair-Pred, and we conclude that $\pi\,(\sigma\,P) \equiv P$ by η-contraction.
4. We obtain $\sigma\,(\pi\,Q) \equiv \Psi_A(\lambda(x : A).\,(\forall i.(\!|A\,,_i\,Q|\!) \ni x)$ by β-reduction. Hence $\sigma\,(\pi\,Q) \equiv (\!|A, Q|\!)!$ by Pair-Orig and Surj-Type, and we conclude that $\sigma\,(\pi\,Q) \equiv Q$ by Pair-Param. □

Theorem 4. *The types* $(\forall i.(x : A) \to B) \ni f$ *and*

$$(x : \forall i.A) \to (\forall i.B[x \mapsto x@i]) \ni (f\,(x@\mathbf{0}))$$

are isomorphic.

Proof.

1. We take

   $$\pi \stackrel{\text{def}}{=} \lambda(q : (\forall i.(x : A) \to B[x]) \ni f).\,\lambda(x : \forall i.A).\,(\langle i\rangle(\!|f\,,_i\,q|\!)\,(x@i))!\,.$$

   Indeed $(\!|f\,,_i\,q|\!) : (x : A) \to B[x]$ by In-Abs and ColApp. Thus $\langle i\rangle(\!|f\,,_i\,q|\!)\,(x@i) : \forall i.B[x@i]$ by App and ColAbs and we conclude by Param and Abs that

   $$\pi : (q : (\forall i.(x : A) \to B[x]) \ni f) \to (x : \forall i.A) \to \forall i.B[x@i] \ni f\,(x@\mathbf{0}).$$

2. We take $\sigma \stackrel{\text{def}}{=} \lambda(p : (x : \forall i.A) \to (\forall i.B[x@i]) \ni f\,(x@\mathbf{0})).\,\Phi_f p$. Indeed $\sigma : ((x : \forall i.A) \to (\forall i.B[x@i]) \ni f\,(x@\mathbf{0})) \to (\forall i.(x : A) \to B[x]) \ni f$ by In-Fun.
3. We obtain $\pi\,(\sigma\,p) \equiv \lambda(x : \forall i.A).\,(\langle i\rangle(\!|f\,,_i\,p|\!)\,(x@i))!$ by β-reduction. Hence by Pair-App and Col-η we have

   $$\begin{aligned}\pi\,(\sigma\,P) &\equiv \lambda(x : \forall i.A).\,(\langle i\rangle(\!|f\,(x@\mathbf{0})\,,_i\,p\,x|\!))!\\ &\equiv \lambda(x : \forall i.A).\,(\!|f\,(x@\mathbf{0}), p\,x|\!)!\,,\end{aligned}$$

   and we conclude that $\pi\,(\sigma\,P) \equiv \lambda(x : \forall i.A).\,p\,x \equiv p$ by Param and η-contraction.
4. We obtain that $\sigma\,(\pi\,q) \equiv \Phi_f(\lambda(x : \forall i.A).\,(\langle i\rangle(\!|f\,,_i\,q|\!)\,(x@i)))!$ by β-reduction. $\sigma\,(\pi\,q) \equiv (\!|f, q|\!)!$ follows by Pair-Orig and Surj-Fun, and we conclude that $\sigma\,(\pi\,q) \equiv q$ by Pair-Param. □

4.3 Iterating parametricity

The complex technicalities of [Bernardy and Moulin, 2012], in particular the hypercube structures and the swapping operator, is due to the interpretation of nested parametricity. For instance consider $p \stackrel{\text{def}}{=} \lambda(X :$

$\mathcal{U}).\, \lambda(x : X).\, \llbracket x \rrbracket$ (of type $(X : \mathcal{U}) \to (x : X) \to \llbracket X \rrbracket \ni x$). Recall that using the "naive" parametric interpretation (where each binding x is expanded into two bindings, x and its parametricity witness $\dot{x}$, without hypercube structure), applying the Param rule yields:

$$\begin{aligned}
&\llbracket p \rrbracket : \llbracket (X : \mathcal{U}) \to (x : X) \to \llbracket X \rrbracket \ni x \rrbracket \ni p \\
&\quad = (X : \mathcal{U}) \to (\dot{X} : \llbracket \mathcal{U} \rrbracket \ni X) \to (x : X) \to \\
&\qquad\quad (\dot{x} : \llbracket X \rrbracket \ni x) \to \llbracket \llbracket X \rrbracket \ni x \rrbracket \ni (p\, X\, x) \\
&\quad = (X : \mathcal{U}) \to (\dot{X} : X \to \mathcal{U}) \to (x : X) \to (\dot{x} : \llbracket X \rrbracket \ni x) \to \llbracket \llbracket X \rrbracket \rrbracket\, x\, \dot{x}\, \llbracket x \rrbracket
\end{aligned}$$

On the other hand, $\llbracket p \rrbracket$ intuitively β-reduces to

$$\lambda(X : \mathcal{U}).\, \lambda(\dot{X} : X \to \mathcal{U}).\, \lambda(x : X).\, \lambda(\dot{x} : \llbracket X \rrbracket \ni x).\, \llbracket \dot{x} \rrbracket$$

which is of type

$$(X : \mathcal{U}) \to (\dot{X} : X \to \mathcal{U}) \to (x : X) \to (\dot{x} : \llbracket X \rrbracket \ni x) \to \llbracket \llbracket X \rrbracket \rrbracket\, x\, \llbracket x \rrbracket\, \dot{x}$$

Hence, as already mentioned in the introduction, $\llbracket p \rrbracket$ has two non-convertible types $\llbracket \llbracket X \rrbracket \rrbracket\, x\, \dot{x}\, \llbracket x \rrbracket$ and $\llbracket \llbracket X \rrbracket \rrbracket\, x\, \llbracket x \rrbracket\, \dot{x}$. As a way to deal with this problem, Bernardy and Moulin [2012] introduced hypercubes in order to "glue together" the arguments of the relation $\llbracket \llbracket X \rrbracket \rrbracket$, to obtain the equality $\llbracket \llbracket X \rrbracket \rrbracket\, (x, \dot{x}, \llbracket x \rrbracket) = \llbracket \llbracket X \rrbracket \rrbracket\, (x, \llbracket x \rrbracket, \dot{x})$.

This issue is not present in our system because the parametricity operator $\cdot!$ does not reduce under λ. However we can iterate the operator $A \ni$ to construct relations between parametricity witnesses. That is, given a context with

$$x : A,\ \ y : (\forall i.A) \ni x,\ \ z : (\forall i.A) \ni x,$$

the type $(\forall i.(\forall j.A) \ni (\!| x\, ,_i\, y |\!)) \ni z$ is well formed, 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 5. *If the type A does not depend on either colors i or j, then the relation*

$$\lambda(y : (\forall i.A) \ni x).\, \lambda(z : (\forall i.A) \ni x).\, (\forall i.(\forall j.A) \ni (\!| x\, ,_i\, y |\!)) \ni z$$

is symmetric for each $x : A$.

Proof. We define

$$\begin{aligned}
\sigma_1 \stackrel{\text{def}}{=}\ & \lambda(y : (\forall i.A) \ni x).\, \lambda(z : (\forall i.A) \ni x). \\
& \lambda(w : (\forall i.(\forall j.A) \ni (\!| x\, ,_i\, y |\!)) \ni z).\, (\langle i \rangle (\langle j \rangle (\!| (\!| x\, ,_j\, y |\!)\, ,_i (\!| z\, ,_j\, w |\!) |\!))!)!
\end{aligned}$$

Let $y : (\forall i.A) \ni x$, $z : (\forall i.A) \ni x$, and $w : (\forall i.(\forall j.A) \ni (\!| x\, ,_i\, y |\!)) \ni z$. Since A does not depend or i nor j we have $j \vdash (\!| x\, ,_j\, y |\!) : A$ by In-Abs and

ColApp. Furthermore $w : (\forall j.(\forall i.A) \ni (\!| x ,_j y |\!)) \ni z$. by α-equivalence on colors and since $z : (\forall i.A) \ni (\!| x, y |\!)@\mathbf{0}$ by Pair-Orig and Conv, we get $j \vdash (\!| z ,_j w |\!) : (\forall i.A) \ni (\!| x, y |\!)$ by In-Abs and ColApp. Following the same line of reasoning we obtain $i, j \vdash (\!| (\!| x ,_j y |\!) ,_i (\!| z ,_j w |\!) |\!) : A$.

Thus by ColAbs, Param and Pair-Orig

$$i \vdash (\langle j \rangle (\!| (\!| x ,_j y |\!) ,_i (\!| z ,_j w |\!) |\!))! : (\forall j.A) \ni (\!| x ,_i z |\!),$$

then

$$\vdash (\langle i \rangle (\langle j \rangle (\!| (\!| x ,_j y |\!) ,_i (\!| z ,_j w |\!) |\!))!)! : (\forall i.(\forall j.A) \ni (\!| x ,_i z |\!)) \ni x.$$

We therefore conclude that

$$\sigma_1 : (y : (\forall i.A) \ni x) \to (z : (\forall i.A) \ni x) \to (\forall i.(\forall j.A) \ni (\!| x ,_i y |\!)) \ni z \to (\forall i.(\forall j.A) \ni (\!| x ,_i z |\!)) \ni y \quad \square$$

Theorem 6. *If the type A does not depend on either colors i or j, then the types $(\forall i.(\forall j.A) \ni (\!| x ,_i y |\!)) \ni z$ and $(\forall i.(\forall j.A) \ni (\!| x ,_i z |\!)) \ni y$ are isomorphic for each $x : A$.*

Proof. Let $y : (\forall i.A) \ni x$, $z : (\forall i.A) \ni x$, and $w : (\forall i.(\forall j.A) \ni (\!| x ,_i y |\!)) \ni z$. We show that the function σ_1 (defined in the proof of Theorem 5) is involutive in its last argument, *i.e.*, that $\sigma_1\, y\, z\, (\sigma_1\, z\, y\, w) \equiv w$.

Let $t \stackrel{\text{def}}{=} \langle i \rangle (\langle j \rangle (\!| (\!| x ,_j z |\!) ,_i (\!| y ,_j w |\!) |\!))!$ and $w' \stackrel{\text{def}}{=} t! = \sigma_1\, z\, y\, w$. We have $t@\mathbf{0} \equiv (\langle j \rangle (\!| x ,_j z |\!))! \equiv (\!| x, z |\!)! = z$ by Col-β, Pair-Orig, Col-η and Pair-Param. Hence $(\!| z ,_j w' |\!) \equiv t@j \equiv (\langle i \rangle (\!| (\!| x ,_i z |\!) ,_j (\!| y ,_i w |\!) |\!))!$ by Surj-Param and Col-β.

Let $t' \stackrel{\text{def}}{=} \langle i \rangle (\!| (\!| x ,_i z |\!) ,_j (\!| y ,_i w |\!) |\!)$. We have $t'@\mathbf{0} \equiv (\!| x ,_j y |\!)$ by Col-β and Pair-Orig. Hence $(\!| (\!| x ,_j y |\!) ,_i (\!| z ,_j w' |\!) |\!) \equiv t'@i \equiv (\!| (\!| x ,_i z |\!) ,_j (\!| y ,_i w |\!) |\!)$ by Surj-Param and Col-β.

Therefore $\sigma_1\, y\, z\, (\sigma_1\, z\, y\, w) \equiv (\langle i \rangle (\langle j \rangle (\!| (\!| x ,_i z |\!) ,_j (\!| y ,_i w |\!) |\!))!)!$ and we conclude by using twice Col-η and Pair-Param, that $\sigma_1\, y\, z\, (\sigma_1\, z\, y\, w) \equiv w$. $\square$

Remark 7. *One may wonder if our system could have been set up to have the types $(\forall i.(\forall j.A) \ni (\!| x ,_i y |\!)) \ni z$ and $(\forall i.(\forall j.A) \ni (\!| x ,_i z |\!)) \ni y$ convertible rather than isomorphic. In fact, the equality*

$$(\forall i.(\forall j.A) \ni (\!| x ,_i y |\!)) \ni z \equiv (\forall i.(\forall j.A) \ni (\!| x ,_i z |\!)) \ni y$$

is inconsistent. In particular for the universe one gets

$$(\forall i.(\forall j.\mathcal{U}) \ni (\!| X ,_i P |\!)) \ni Q \equiv (\forall i.(\forall j.\mathcal{U}) \ni (\!| X ,_i Q |\!)) \ni P$$

for arbitrary P and Q of type $(\forall i.\mathcal{U}) \ni X$*. By Theorem 3 this equality in turn implies*

$$(x : X) \to P\,x \to Q\,x \to \mathcal{U} \equiv (x : X) \to Q\,x \to P\,x \to \mathcal{U}$$

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

Theorem 7. *If the type A and the term* $a : A$ *does not depend on either colors i or j, and* $\dot{a} : (\forall i.A) \ni a$ *(not depending on i or j either), then* $\dot{a}$ *is related to the canonical proof* $(\langle i \rangle a)!$*, in the sense that the type*

$$(\forall i.(\forall j.A) \ni (\!|a\,,_i \dot{a}|\!)) \ni (\langle i \rangle a)!$$

is inhabited.

Proof. Since $(\langle i \rangle a)@\mathbf{0} \equiv a$ we get $(\!|a\,,_i (\langle i \rangle a)!|\!) \equiv a$ by Surj-Param. Hence $\dot{a} : (\forall j.A) \ni (\!|a\,,_i (\langle i \rangle a)!|\!)$ by Conv, and finally ColAbs and Param give $(\langle i \rangle \dot{a})! : (\forall i.(\forall j.A) \ni (\!|a\,,_i \dot{a}|\!)) \ni (\langle i \rangle a)!$. □

Remark 8. *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* $i < n$ *one can construct a function* σ_i*, which exchanges the arguments i and* $i+1$ *of a relation. Furthermore, these functions satisfy the laws of the generators of the symmetric group on n symbols* $\mathfrak{S}_n$*:*

- *any permutation of* $\{0, \ldots, n-1\}$ *can be written as a* $\sigma_{i_1}^{\alpha_1} \ldots \sigma_{i_k}^{\alpha_k}$ *where* $\alpha_j = \pm 1$ *and* $1 \leq i_j < n$ *for each* $1 \leq j \leq k$*;*
- $\sigma_i^2 = 1$ *for each* $1 \leq i < n$*;*
- $\sigma_i \sigma_j = \sigma_j \sigma_i$ *for each* $1 \leq i, j < n$ *such that* $|j - i| \neq 1$*; and*
- $(\sigma_i \sigma_{i+1})^3 = 1$ *for each* $1 \leq i < n - 1$*.*

5 Presheaf model

In this chapter we show how to equip our type theory with presheaf semantics. We start with general background. In particular, we recall how to interpret terms and types of the Logical Framework presented in section 2.1, following Hofmann [1997] and Bezem et al. [2013]. We will then modify Hofmann's presheaf model in order to capture the invariants of our type theory, and show how to adapt the interpretation functions accordingly in section 5.3.

5.1 Background

If $\mathscr{C}$ is a category, we write $X \in \mathscr{C}$ if X is an object of $\mathscr{C}$, and for two objects $X, Y \in \mathscr{C}$, we write $X \longrightarrow Y$ for the Hom-set $\mathbf{Hom}_{\mathscr{C}}(X, Y)$. Fur-

thermore $1 : X \longrightarrow X$ denotes the identity morphism, and we write morphism composition in diagrammatical order: if $X, Y, Z \in \mathscr{C}$, then the composition of $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$ is written $fg : X \longrightarrow Z$.

Definition 7 (Presheaf)**.** *Let $\mathscr{C}$ be a small category. A presheaf over $\mathscr{C}$ is a functor $F : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathbf{Set}$, where $\mathbf{Set}$ is the category of small sets and functions. (Where the set of small sets is a Grothendieck universe $\mathscr{U}_{type}$ containing another Grothendieck universe $\mathscr{U}$, used in the interpretation of the type-theoretic universe $\mathcal{U}$.)*

In other words, a presheaf F over $\mathscr{C}$ is given by a family of sets $F(X)$ indexed by the objects of $\mathscr{C}$, together with restrictions maps $F(f) : F(X) \longrightarrow F(Y)$, $u \longmapsto uf$ for $f : Y \longrightarrow X$ satisfying $u1 = u$ and $(uf)g = u(gf)$ for $g : Z \longrightarrow Y$.

$$\begin{array}{rcccl} & Z & \xrightarrow{g} & Y & \xrightarrow{f} X \\ F(X) & \longrightarrow & F(Y) & \longrightarrow & F(Z) \\ u & \longmapsto & uf & \longmapsto & (uf)g = u(gf) \end{array}$$

Hofmann [1997, sec. 4] shows how to equip the Logical Framework with presheaf semantics. There are three interpretation functions, each partially defined by induction on the raw syntax:

- one to interpret contexts;
- one to interpret types, relative to a context; and
- one to interpret terms, relative to a context and a type.

(The first is defined on the set of raw contexts, while the last two are both defined on the set of raw expressions.) They are not necessarily defined on an arbitrary raw context, raw type or raw term, but Hofmann shows later that they are defined on all correct contexts, correct types with a context, and correct terms with a context and a type, and that the following properties hold:

- A well-formed context $\Gamma \vdash$ is interpreted by a presheaf Γ, *i.e.*, by a family of sets $\Gamma(X)$ indexed by $X \in \mathscr{C}$, together with restrictions maps $\Gamma(f) : \Gamma(X) \longrightarrow \Gamma(Y)$, $\rho \longmapsto \rho f$ for $f : Y \longrightarrow X$, satisfying $\rho 1 = \rho$ and $(\rho f)g = \rho(gf)$ for $g : Z \longrightarrow Y$.

- A type $\Gamma \vdash A$ is interpreted by a family of sets $A\rho$ indexed by $\rho \in \Gamma(X)$ for each $X \in \mathscr{C}$, together with restriction maps $A\rho \longrightarrow A(\rho f)$, $u \longmapsto uf$ for $f : Y \longrightarrow X$, satisfying $u1 = u$ and $(uf)g = u(gf)$ for $g : Z \longrightarrow Y$.

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

- If $\Gamma \vdash A \equiv A'$ then $A\rho = A'\rho$ for any $\rho \in \Gamma(X)$.

- If $\Gamma \vdash a \equiv a' : A$ then $a\rho = a'\rho$ for any $\rho \in \Gamma(X)$.

We now show some interesting cases of the definition of Hofmann's interpretation functions. The presheaf laws (the first three items in the above list of properties) hold by definition, while the remaining two properties (namely that the interpretation functions are total on valid judgments, and that convertible terms and types yield equal semantic values) are proven by induction on the derivation of judgments. (Unlike Hofmann we considered an untyped conversion rather than equality judgments in our system; as a consequence our validity results from section 5.4 are not expressed in the same way.)

Empty Context. The presheaf $\diamond$ is defined as follows.

Object part. $\diamond(X)$ is a singleton: $\diamond(X) = \{\star\}$ for each $X \in \mathscr{C}$.

Arrow part. For $f : Y \longrightarrow X$, we take $\diamond(f) : \diamond(X) \longrightarrow \diamond(Y)$ the trivial (constant) map, mapping the element $\star \in \diamond(X)$ to $\star \in \diamond(Y)$.

Preservation of identities. Direct.

Preservation of composition. Direct.

Context extension. The presheaf $(\Gamma, x : A)$ is defined as follows.

Object part. We let $(\Gamma, x : A)(X) = (\rho \in \Gamma(X)) \times A\rho$ be the dependent product of the interpretations of Γ and A. We also write $\langle \rho, x = u \rangle \in (\Gamma, x : A)(X)$ to mean $\rho \in \Gamma(X)$ and $u \in A\rho$. (The notation is tagged with the variable x to allow named-based lookup.)

Arrow part. For $f : Y \longrightarrow X$, we take $(\Gamma, x : A)(f) : (\Gamma, x : A)(X) \longrightarrow (\Gamma, x : A)(Y)$, $\langle \rho, x = u \rangle \longmapsto \langle \rho f, x = uf \rangle$.

Preservation of identities. For $\langle \rho, x = u \rangle \in (\Gamma, x : A)(X)$ we have $\rho \in \Gamma(X)$ and $u \in A\rho$, hence by induction hypothesis $\rho 1 = \rho$ and $u1 = u$. Therefore $\langle \rho, x = u \rangle 1 = \langle \rho 1, x = u1 \rangle = \langle \rho, x = u \rangle$.

Preservation of composition. For $\langle \rho, x = u \rangle \in (\Gamma, x : A)(X), f : Y \longrightarrow X$ and $g : Z \longrightarrow Y$ we have $\rho \in \Gamma(X)$ and $u \in A\rho$, hence by induction hypothesis $(\rho f)g = \rho(gf)$ and $(uf)g = u(gf)$. Therefore $(\langle \rho, x = u \rangle f)g = \langle (\rho f)g, x = (uf)g \rangle = \langle \rho(gf), x = u(gf) \rangle = \langle \rho, x = u \rangle (gf)$.

Variable. The element $x\rho$ is defined only for ρ of the form $\langle \rho', z = u \rangle$. In that case we take $x\langle \rho', x = u \rangle = u$, and $x\langle \rho', z = u \rangle = x\rho'$ if $x \neq z$.

We now show that $(x\rho)f = x(\rho f)$ for each $f : Y \longrightarrow X$. We have

- $(x\langle \rho', x = u \rangle)f = uf = x\langle \rho' f, x = uf \rangle = x(\langle \rho', x = u \rangle f)$; and
- $(x\langle \rho', z = u \rangle)f = (x\rho')f = x(\rho' f) = x(\langle \rho', z = u \rangle f)$ if $x \neq z$.

Dependent function space. The raw type $((x : A) \to B)$ is interpreted (relative to Γ) as follows.

For $\rho \in \Gamma(X)$, $((x : A) \to B)\rho$ is the set of families $\lambda = (\lambda_f)_{f:Y \longrightarrow X}$ where each λ_f is a dependent function such that $\lambda_f(u) \in B\langle \rho f, x = u\rangle$ for each $u \in A\rho f$, and $(\lambda_f(u))g = \lambda_{gf}(ug)$ for each $u \in A\rho f$ and $g : Z \longrightarrow Y$.

Restriction maps. For $f : Y \longrightarrow X$, we take $\lambda f_g = \lambda_{gf}$ for each $g : Z \longrightarrow Y$. In other words, we take $((x : A) \to B)\rho \longrightarrow ((x : A) \to B)\rho f$ to be defined as $(\lambda_h)_{h:Z \longrightarrow X} \longmapsto (\lambda_{gf})_{g:Z \longrightarrow Y}$.

Preservation of identities. We have $\lambda 1_g = \lambda_{g1} = \lambda_g$ for each $g : Z \longrightarrow X$, hence $\lambda 1 = \lambda$.

Preservation of composition. For $f : Y \longrightarrow X$ and $g : Z \longrightarrow Y$ we have $(\lambda f)g_h = \lambda f_{hg} = \lambda_{(hg)f} = \lambda_{h(gf)} = \lambda(gf)_h$ for each $h : T \longrightarrow Z$, hence $(\lambda f)g = \lambda(gf)$.

Universe. The type-theoretic universe $\mathcal{U}$ is interpreted (relative to Γ) as follows.

For $\rho \in \Gamma(X)$, $\mathcal{U}\rho$ is the set of a families $\mathcal{A} = (\mathcal{A}_f)_{f:Y \longrightarrow X}$ where each $\mathcal{A}_f$ is a $\mathscr{U}$-small set, together with restriction maps $\mathcal{A}_f \longrightarrow \mathcal{A}_{gf}$, $u \longmapsto ug$ for any $f : Y \longrightarrow X$ and $g : Z \longrightarrow Y$, satisfying the equalities $u1 = u$ and $(ug)h = u(hg)$ for each $h : T \longrightarrow Z$.

Restriction maps. For $f : Y \longrightarrow X$, we take $\mathcal{U}\rho \longrightarrow \mathcal{U}\rho f$ to be defined as $(\mathcal{A}_h)_{h:Z \longrightarrow X} \longmapsto (\mathcal{A}_{gf})_{g:Z \longrightarrow Y}$. In other words, we take $\mathcal{A}f_g = \mathcal{A}_{gf}$ for each $g : Z \longrightarrow Y$ together with restriction maps $\mathcal{A}f_g \longrightarrow \mathcal{A}f_{hg}$ defined as the given maps $\mathcal{A}_{gf} \longrightarrow \mathcal{A}_{hgf}$.

Preservation of identities. We have $\mathcal{A}1_g = \mathcal{A}_{g1} = \mathcal{A}_g$ for each $g : Z \longrightarrow X$, hence $\mathcal{A}_1 = \mathcal{A}$.

Preservation of composition. For $f : Y \longrightarrow X$ and $g : Z \longrightarrow Y$ we have $(\mathcal{A}f)g_h = \mathcal{A}f_{hg} = \mathcal{A}_{(hg)f} = \mathcal{A}_{h(gf)} = \mathcal{A}(gf)_h$ for each $h : T \longrightarrow Z$, hence $(\mathcal{A}f)g = \mathcal{A}(gf)$.

Small dependent function space. The small type $((x : A) \to B)$ is interpreted (relative to Γ and $\mathcal{U}$) as an element of the set which interprets $\mathcal{U}$, namely as a family of $\mathscr{U}$-small sets indexed by $f : Y \longrightarrow X$, by taking $((x : A) \to B)\rho_f$ to be the $\mathscr{U}$-small set of dependent functions λ_f mapping each $u \in (A\rho f)_1$ to an element of $(B\langle \rho f, x = u\rangle)_1$. (Interpreting the small types A and B relative to the universe $\mathcal{U}$ yields two families of $\mathscr{U}$-small sets; we take the elements at index $1 : Y \longrightarrow Y$.) For $f : Y \longrightarrow X$ and $g : Z \longrightarrow Y$, the restriction map $((x : A) \to B)\rho_f \longrightarrow ((x : A) \to B)\rho_{gf}$ is defined to be $\lambda_f \longmapsto \lambda_f g = \lambda_{gf}$.

Let $f : Y \longrightarrow X$. We now show that $(((x : A) \to B)\rho)f = ((x : A) \to B)(\rho f)$. For each $g : Z \longrightarrow Y$, we have $(A(\rho f)g)_1 = (A\rho(gf))_1$ and $(B\langle(\rho f)g, x = u\rangle)_1 = (B\langle\rho(gf), x = u\rangle)_1$ by induction hypothesis, hence by definition the two sets $((x : A) \to B)\rho f_g = ((x : A) \to B)\rho_{gf}$ and $((x : A) \to B)(\rho f)_g$ coincide.

5.2 The category pI and the notion of I-sets

We will modify Hofmann's model to deal with colors and parametricity rules. We interpret a context Γ as a *refined* presheaf over $\mathbf{pI}^{\mathrm{op}}$ (Definition 15 below), where $\mathbf{pI}$ is the category of colors and partial injections (Definition 9). A type $\Gamma \vdash A$ is interpreted as a family of I-sets $A\rho$ indexed by $I \in \mathbf{pI}$ and $\rho \in \Gamma(I)$, where an I-set is a set of tuples indexed by the subsets of I (Definition 14). (The reason for considering I-sets instead of usual sets is due to our conversion rules such as Pair-Pred and Surj-Param, and is motivated in section 5.2.3.) A term $\Gamma \vdash a : A$ is interpreted as an I-element $a\rho \in A\rho$ for each $I \in \mathbf{pI}$ and $\rho \in \Gamma(I)$. Furthermore, we prove in section 5.4 (Theorem 10) that two valid convertible types (resp. terms) are interpreted as equal I-sets (resp. I-elements). Note that unlike Hofmann we considered an untyped conversion rather than equality judgments; thus our validity result (section 5.4) consists of 4 mutually proven theorems (Theorems 8 to 11).

5.2.1 The category pI of colors and partial injections

Recall that $\mathbb{I}$ is a fixed infinite set of colors, and that $\mathbf{0} \notin \mathbb{I}$ is a special symbol. We further assume a fixed function $\mathsf{fresh}(\cdot)$ such that $\mathsf{fresh}(I) \in \mathbb{I} \backslash I$ for any finite $I \subset \mathbb{I}$.

Definition 8 (Color map)**.** *If I and J are two finite subsets of $\mathbb{I}$, we say that a color map, or a partial injection, is any set-theoretic function $f : I \to J \cup \{\mathbf{0}\}$ such that $i_1 = i_2$ for any $i_1, i_2 \in I$ with $f(i_1) = f(i_2) \in J$.*

Definition 9 (Category $\mathbf{pI}$)**.** *We consider the category $\mathbf{pI}$ of finite color sets and partial injections. We use the Kleisli composition: if $f : I \longrightarrow J$ and $g : J \longrightarrow K$, then $fg : I \longrightarrow K$ is defined as $fg(i) = \mathbf{0}$ if $f(i) = \mathbf{0}$ and $fg(i) = g(f(i))$ if $f(i) \in J$.*

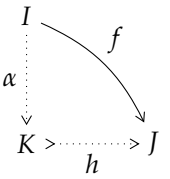
Remark 9. *Pitts [2013, ex. 9.7 p. 176] gives another description of the category $\mathbf{pI}$. Furthermore, the category of presheaves on $\mathbf{pI}^{\mathrm{op}}$ is equivalent to the category $\mathbf{Res}$ of nominal restriction sets [Pitts, 2013, rem. 9.9 p. 161].*

We also observe that any morphism has a unique decomposition into a projection map (Definition 10) and a total map (Definition 11).

Definition 10 (Projection map)**.** *We say that a color map $\alpha : I \uplus J \longrightarrow I$ is a projection if $\alpha(i) = i$ for each $i \in I$, and $\alpha(j) = \mathbf{0}$ for each $j \in J$.*

Definition 11 (Total map)**.** *We say that a color map $h : I \longrightarrow J$ is total, and note $h : I \rightarrowtail J$, if it is injective, i.e., if $h(i) \neq \mathbf{0}$ for each $i \in I$.*

Remark 10 (Morphism decomposition). *Any morphism $f : I \longrightarrow J$ has a unique decomposition into a projection map $\alpha : I \longrightarrow K$ and a total map $h : K \rightarrowtail J$.*



5.2.2 Interpretation of the color context extension

After introducing a new notation for color map restriction, we show how to define the presheaf (Γ, i) for color context extension.

Definition 12 (Color map restriction). *If $f : I, i \longrightarrow J$ (resp. $f : I, i \longrightarrow J, j$) is a color map such that $f(i) = \mathbf{0}$ (resp. $f(i) = j$), we let $f - i : I \longrightarrow J$ be the color map defined by $(f - i)(k) = f(k)$ for every $k \in I$.*
If $I = J$, we write $(i\,\mathbf{0})$ as a shorthand for $1 - i$ (where we recall that $1 : I \longrightarrow I$ is the identity morphism).

We now define the presheaf (Γ, i) as follows (we also note that this definition reflects the linear behavior or ColApp mentioned in Remark 1):

Object part. We take $(\Gamma, i)(I) = \Gamma(I) \uplus \cup_{j \in I} \Gamma(I \backslash \{j\})\}$ that is, the *separated product* [Pitts, 2013, sec. 3.4 p. 54] of the interpretation of Γ and $\mathbb{I} \cup \{\mathbf{0}\}$. We also write $[\rho, i = \varphi] \in (\Gamma, i)(I)$ to mean either

- $\varphi = \mathbf{0}$ and $\rho \in \Gamma(I)$; or
- $\varphi = j \in I$ and $\rho \in \Gamma(I \backslash \{j\})$.

Arrow part. For $f : I \longrightarrow J$, we take $(\Gamma, i)(f) : (\Gamma, i)(I) \longrightarrow (\Gamma, i)(J)$ to be defined as

$$[\rho, i = \varphi] \longmapsto \begin{cases} [\rho f, i = \mathbf{0}] & \text{if } \varphi = \mathbf{0} \\ [\rho(f - j), i = f(j)] & \text{if } \varphi = j \in I \end{cases}$$

Preservation of identities. Let $[\rho, i = \varphi] \in (\Gamma, i)(I)$. By induction hypothesis $\rho 1 = \rho$, and:

- for $\varphi = \mathbf{0}$ we get $[\rho, i = \mathbf{0}]1 = [\rho 1, i = \mathbf{0}] = [\rho, i = \mathbf{0}]$; and
- for $\varphi = j \in I$ we get $[\rho, i = j]1 = [\rho(1 - j), i = 1(j)] = [\rho 1, i = j] = [\rho, i = j]$.

Preservation of composition. Let $[\rho, i = \varphi] \in (\Gamma, i)(I), f : I \longrightarrow J$ and $g : J \longrightarrow K$. By induction hypothesis $(\rho f)g = \rho(fg)$, and:

- for $\varphi = \mathbf{0}$ or $\varphi = j \in I$ such that $f(j) = \mathbf{0}$ we get $([\rho, i = \varphi]f)g = [(\rho f)g, i = \mathbf{0}] = [\rho(fg), i = \mathbf{0}] = [\rho, i = \varphi](fg)$; and
- for $\varphi = j \in I$ such that $f(j) \neq \mathbf{0}$ we get $([\rho, i = j]f)g = [(\rho f)g, i = g(f(j))] = [\rho(fg), i = fg(j)] = [\rho, i = j](fg)$.

5.2.3 I-sets and I-elements

Definition 13 (Color map extension)**.** *Let $f : I \longrightarrow J$ be a color map. If $i \notin I$ and $j \notin J$, we let $(f, i = j) : I, i \longrightarrow J, j$ (resp. $(f, i = \mathbf{0}) : I, i \longrightarrow J$) be the color map (where I, i is a shorthand for $I \cup \{i\}$) defined by*

- $(f, i = \varphi)(i) = \varphi$; *and*
- $(f, i = \varphi)(k) = f(k)$ *for every $k \in I$.*

If $I = J$, we write $(i\,j)$ as a shorthand for $(1, i = j)$.

We now attempt to define the interpretation of the color product $\forall i.A$ (relative to a context Γ) by defining the set $(\forall i.A)\rho$ for $\rho \in \Gamma$. Since the color i is free in A a first attempt would be to define $(\forall i.A)\rho = A[\rho, i = j]$ for $j = \mathsf{fresh}(I)$. However since the color i is also bound in the ray, we need to consider all fresh colors and quotient by color α-equivalence. We thus define

$$(\forall i.A)\rho = \{\bar{u}^j \mid j = \mathsf{fresh}(I), u \in A[\rho, i = j]\},$$

where

$$\bar{u}^j = \{v \mid k \in \mathbb{I} \backslash I, v \in A[\rho, i = k], v(k\,j) = u\}$$

is the equivalence class of the $v \in A[\rho, i = k]$ (quotiented by color α-equivalence) for fresh ks.

However with such a definition it is not clear how to validate our new conversion rules, in particular Pair-Pred ($(\forall i.(A \bowtie_i P)) \ni a \equiv P\,a$) and Surj-Param ($(\!|a@\mathbf{0}, a!|\!) \equiv a$). This seems to be due to some lack of structure in the target of the presheaf interpretation (the category of small sets). However by replacing usual sets by something more structured (I-sets, Definition 14), and by refining the notion of presheaf (Definition 15), we are able to capture in the model the invariants of our type theory and validate all our equalities in the model (Theorem 10).

Definition 14 (I-set)**.** *Let an I-element be any tuple indexed by the subsets of I: $(u_J)_{J \subseteq I}$. Alternatively, such an I-element can be seen as a tuple (u_α) indexed by the projections $\alpha : I \longrightarrow J$. An I-set is a set of I-elements. For instance, the elements of an $\{i, j\}$-set are of the form $u = (u_\emptyset, u_i, u_j, u_{i,j})$.*

Definition 15 (Refined presheaf on $\mathbf{pI}^{\mathrm{op}}$)**.** *We refine Definition 7 of presheaves on $\mathbf{pI}^{\mathrm{op}}$ by requiring two extra conditions:*

1. *for any object I, $F(I)$ is an I-set; and*
2. *for any projection map $\alpha : I \longrightarrow J$, the restriction map $F(\alpha) : F(I) \longrightarrow F(J)$, $u \longmapsto u\alpha$ is the projection operation, i.e., $u\alpha_K = u_K$ for any $K \subseteq J$ (alternatively, seeing J-elements as tuples indexed by projection maps, $(u\alpha)_\beta = u_{\alpha\beta}$).*

We therefore use a "dependent" notion of presheaves $(I \in \mathbf{pI}^{\mathrm{op}}) \longrightarrow I$-$\mathbf{Set}$.

We then refine the interpretation of raw types and terms accordingly:

- A well-formed context $\Gamma \vdash$ is interpreted by a refined presheaf Γ, *i.e.*, by a family of I-sets $\Gamma(I)$ indexed by $I \in \mathbf{pI}$, together with restrictions maps $\Gamma(f) : \Gamma(I) \longrightarrow \Gamma(J)$, $\rho \longmapsto \rho f$ for $f : I \longrightarrow J$, satisfying $\rho 1 = \rho$ and $(\rho f)g = \rho(fg)$ for $g : J \longrightarrow K$, and such that the map $\Gamma(I) \longrightarrow \Gamma(J)$, $\rho \longmapsto \rho\alpha$ is the projection operation for each projection map $\alpha : I \longrightarrow J$.

- 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 $A\rho \longrightarrow A(\rho f)$, $u \longmapsto uf$ for $f : I \longrightarrow J$, satisfying $u1 = u$ and $(uf)g = u(fg)$ for $g : J \longrightarrow K$, and such that the map $A\rho \longrightarrow A(\rho\alpha)$, $u \longmapsto u\alpha$ is the projection operation for each projection map $\alpha : I \longrightarrow J$.

- 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 $f : I \longrightarrow J$.

The following definition will be used in the interpretation of rays:

Definition 16. *If u is an (I, i)-element, we let $\bar{u}^i$ be the I-element defined by $(\bar{u}^i)_J = (u_J, u_{J,i})$ for each $J \subseteq I$. We also let*

- *$\bar{u}^i@\mathbf{0}$ be the I-element defined by $(\bar{u}^i@\mathbf{0})_J = u_J$ for each $J \subseteq I$;*
- *$\bar{u}^i@j$ be the (I, j)-element defined by $(\bar{u}^i@j)_J = u_J$ and $(\bar{u}^i@j)_{J,j} = u_{J,i}$ for each $J \subseteq I$; and*
- *$\bar{u}^i!$ be the I-element defined by $(\bar{u}^i!)_J = u_{J,i}$ for each $J \subseteq I$.*

5.3 Presheaf model of the parametric type theory

We now equip our parametric type theory with a modified presheaf model over $\mathbf{pI}^{\mathrm{op}}$. Moreover we refine the interpretation functions by requiring that $F(I)$ is an I-set for each $I \in \mathbf{pI}$, where I-sets are sets of tuples indexed by the subsets of I (Definition 14 above).

As before, there are three interpretation functions, each partially defined by induction on the raw syntax:

- one to interpret contexts;
- one to interpret types, relative to a context; and
- one to interpret terms, relative to a context and a type.

They are not defined on an arbitrary raw contexts, raw types or raw terms, but as we will show in section 5.4, convertible terms and types yield equal semantic values (Theorem 10), and the interpretation functions are total on valid judgments (Theorem 11). Like for section 5.1, we prove the presheaf laws in the definition of the interpretation functions.

5.3.1 Underlying type theory

Due to the restriction described in section 5.2.3, we cannot directly take the interpretation from Hofmann [1997, sec. 4] since for a type A and $\rho \in \Gamma(I)$, $A\rho$ is not required to be an I-set hence violates our presheaf refinement (Definition 15). We show how to adapt the interpretation of the underlying type theory (section 2.1) to satisfy the refinement.

Interpretation of Contexts

Empty Context.

Object part. We cannot directly take the interpretation from section 5.1, since the object part is not required to be an I-set. Instead, we define it as the I-singleton $\{(\diamond_J)_{J \subseteq I}\}$ such that $\rho_\emptyset$ is a singleton $\{\star\}$ and $\rho_J = \emptyset$ for any non-empty $J \subseteq I$.

Arrow part. For $f : I \longrightarrow J$, we take $\diamond(f) : \diamond(I) \longrightarrow \diamond(J)$ the trivial (constant) map.

Preservation laws. Direct.

Context extension. In section 5.1 we defined

$$(\Gamma, x : A)(I) = \{\langle \rho, x = u\rangle \mid \rho \in \Gamma(I), u \in A\rho\}.$$

For $\rho \in \Gamma(I)$ and an I-element $u \in A\rho$, we now redefine $\langle \rho, x = u\rangle$ as an I-element by taking $\langle \rho, x = u\rangle_J = (\rho_J, u_J)$ for each $J \subseteq I$.

We already proved preservation of identities and composition (independently of the definition of $\langle \rho, x = u\rangle$) in section 5.1. As for preservation of projections, if $\alpha : I \longrightarrow J$ is a projection and $K \subseteq J$, we have $\rho\alpha_K = \rho_K$ and $u\alpha_K = u_K$ by induction hypothesis and therefore $\langle \rho, x = u\rangle\alpha_K = \langle \rho\alpha, x = u\alpha\rangle_K = (\rho\alpha_K, u\alpha_K) = (\rho_K, u_K) = \langle \rho, x = u\rangle_K$.

Interpretation of Types

Universe. The type-theoretic universe $\mathcal{U}$ is interpreted (relative to Γ) as follows.

We cannot directly take the interpretation of $\mathcal{U}$ from section 5.1, since the object part is not required to be an I-set. However, using Remark 10, we can define an isomorphic I-set by reindexing each element as follows.

For $\rho \in \Gamma(I)$, $\mathcal{U}\rho$ is the I-set of families $(\mathcal{A}_\alpha)_{\alpha : I \longrightarrow J}$ indexed by projection maps $\alpha : I \longrightarrow J$ with $J \subseteq I$, where each $\mathcal{A}_\alpha$ is a tuple $(\mathcal{A}_{\alpha h})_{h : J \rightarrowtail K}$ indexed by *total* maps $h : J \rightarrowtail K$, such that each $\mathcal{A}_{\alpha h}$ is a $\mathscr{U}$-small K-set, together with restriction maps $\mathcal{A}_{\alpha h} \longrightarrow \mathcal{A}_{\alpha hg}, u \longmapsto ug$ for any $g : K \longrightarrow L$, satisfying the equalities $u1 = u$ and $(ug)h = u(gh)$ for each $h : L \longrightarrow M$.

Restriction maps. Since by Remark 10 any map $f : I \longrightarrow J$ has a unique decomposition $f = \alpha h$ as a projection map $\alpha : I \longrightarrow K$ and a total map $h : K \rightarrowtail J$, we can consider the set $\mathcal{A}_f$ and the restriction maps $\mathcal{A}_f \longrightarrow \mathcal{A}_{fg}$ for two arbitrary maps $f : I \longrightarrow K$ and $g : K \longrightarrow L$.

For an arbitrary map $f : I \longrightarrow J$, we then take $\mathcal{U}\rho \longrightarrow \mathcal{U}\rho f$ to be defined as $(\mathcal{A}_g)_{g:I\longrightarrow K} \longmapsto (\mathcal{A}_{fg'})_{g':J\longrightarrow K}$. In other words, we take $\mathcal{A}f_{\alpha h} = \mathcal{A}_{f\alpha h}$, together with restriction maps $\mathcal{A}f_{\alpha h} \longrightarrow \mathcal{A}f_{\alpha hg}$ defined as the given maps $\mathcal{A}_{f\alpha h} \longrightarrow \mathcal{A}_{f\alpha hg}$.

Preservation of identities. We have $\mathcal{A}1_{\alpha h} = \mathcal{A}_{1\alpha h} = \mathcal{A}_{\alpha h}$ for each projection map $\alpha : I \longrightarrow K$ and total map $h : K \rightarrowtail J$, hence $\mathcal{A}1 = \mathcal{A}$.

Preservation of composition. Let $f : I \longrightarrow J$ and $g : J \longrightarrow K$. Since $(\mathcal{A}f)g_{\alpha h} = \mathcal{A}f_{g\alpha h} = \mathcal{A}_{f(g\alpha h)} = \mathcal{A}_{(fg)(\alpha h)} = \mathcal{A}(fg)_{\alpha h}$ for each projection map $\alpha : K \longrightarrow K'$ and total map $h : K' \rightarrowtail L$, we have $(\mathcal{A}f)g = \mathcal{A}(fg)$.

Preservation of projections. Let $\alpha : I \longrightarrow J$ be a projection map. For each projection $\beta : J \longrightarrow K$ and total map $h : K \rightarrowtail L$, we have $\mathcal{A}\alpha_{\beta h} = \mathcal{A}_{\alpha(\beta h)} = \mathcal{A}_{(\alpha\beta)h}$. Therefore by definition the two families of $\mathcal{U}$-small L-sets indexed by total maps $h : K \rightarrowtail L$ coincide: $\mathcal{A}\alpha_\beta = \mathcal{A}_{\alpha\beta}$.

Dependent function space. The raw type $((x : A) \to B)$ is interpreted (relative to Γ) as follows.

Like for the case of the universe above we cannot directly take the interpretation from section 5.1 since it not an I-set. Instead, we define $((x : A) \to B)\rho$ to be an isomorphic I-set by reindexing each element as follows.

For $\rho \in \Gamma(I)$, $((x : A) \to B)\rho$ is the I-set of families $\lambda = (\lambda_\alpha)_{\alpha:I\longrightarrow J}$ indexed by projection maps $\alpha : I \longrightarrow J$ with $J \subseteq I$, where each λ_α is a tuple $(\lambda_{\alpha h})_{h:J\rightarrowtail K}$ indexed by *total* maps $h : J \rightarrowtail K$, such that each $\lambda_{\alpha h}$ is a dependent function satisfying $\lambda_{\alpha h}(u) \in B\langle \rho\alpha h, x = u\rangle$ for each $u \in A\rho\alpha h$, and $(\lambda_{\alpha h}(u))f = \lambda_{\alpha hf}(uf)$ for each $u \in A\rho\alpha h$ and $f : K \longrightarrow L$.

Restriction maps. As before (thanks to Remark 10), we can consider the dependent function λ_f for an arbitrary map $f : I \longrightarrow J$.

For an arbitrary map $f : I \longrightarrow J$, we then take $((x : A) \to B)\rho \longrightarrow ((x : A) \to B)\rho f$ to be defined as $(\lambda_g)_{g:I\longrightarrow K} \longmapsto (\lambda_{fg'})_{g':J\longrightarrow K}$. In other words, we take $\lambda f_{\alpha h} = \lambda_{f\alpha h}$.

Preservation of identities. We have $\lambda 1_{\alpha h} = \lambda_{1\alpha h} = \lambda_{\alpha h}$ for each projection map $\alpha : I \longrightarrow K$ and total map $h : K \rightarrowtail J$, hence $\lambda 1 = \lambda$.

Preservation of composition. Let $f : I \longrightarrow J$ and $g : J \longrightarrow K$. Since $(\lambda f)g_{\alpha h} = \lambda f_{g\alpha h} = A_{f(g\alpha h)} = \lambda_{(fg)(\alpha h)} = \lambda(fg)_{\alpha h}$ for each projection map $\alpha : K \longrightarrow K'$ and total map $h : K' \rightarrowtail L$, we have $(\lambda f)g = \lambda(fg)$.

Preservation of projections. Let $\alpha : I \longrightarrow J$ be a projection map. For each projection $\beta : J \longrightarrow K$ and total map $h : K \rightarrowtail L$, we have $\lambda\alpha_{\beta h} = \lambda_{\alpha(\beta h)} =$

$\lambda_{(\alpha\beta)h}$. Therefore by definition the two families of dependent functions indexed by total maps $h : K \rightarrowtail L$ coincide: $\lambda\alpha_\beta = \lambda_{\alpha\beta}$.

Interpretation of Terms
Note that since we have a universe à la Russel, small types need to be interpreted as terms as well (relative to a context and $\mathcal{U}$).

Dependent function space. We interpret the (small) dependent function space $(x : A) \to B$ (relative to Γ and $\mathcal{U}$) as an I-element of the I-set which interprets $\mathcal{U}$, namely as a nested family of $\mathscr{U}$-small K-sets indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, by taking $((x : A) \to B)\rho_{\alpha h}$ to be the $\mathscr{U}$-small K-set of families $\lambda = (\lambda_{\alpha'})_{\alpha' : K \longrightarrow L}$ indexed by projection maps $\alpha' : K \longrightarrow L$ with $L \subseteq K$, where each $\lambda_{\alpha'}$ is a tuple $(\lambda_{\alpha' h'})_{h' : L \rightarrowtail M}$ indexed by *total* maps $h' : L \rightarrowtail M$, such that each $\lambda_{\alpha' h'}$ is a dependent function satisfying $\lambda_{\alpha' h'}(u) \in (B\langle \rho\alpha h, x = u\rangle)_{\alpha' h'}$ for each $u \in (A\rho\alpha h)_{\alpha' h'}$, and $(\lambda_{\alpha' h'}(u))f = \lambda_{\alpha' h' g}(uf)$ for each $u \in (A\rho\alpha h)_{\alpha' h'}$ and $f : M \longrightarrow N$.

Let $f : I \longrightarrow J$. We now show that $(((x : A) \to B)\rho)f = ((x : A) \to B)(\rho f)$. Let $\alpha : J \longrightarrow K$ and $h : K \rightarrowtail L$. We have $A(\rho f)\alpha h = A\rho(f\alpha h)$ and $B\langle(\rho f)\alpha h, x = u\rangle = B\langle\rho(f\alpha h), x = u\rangle$ by induction hypotheses, hence by definition the two L-sets $((x : A) \to B)\rho f_{\alpha h} = ((x : A) \to B)\rho_{f\alpha h}$ and $((x : A) \to B)(\rho f)_{\alpha h}$ coincide.

Application. For $\rho \in \Gamma(I)$, we interpret $a\, b$ as follows. Seeing the interpretation of a relative to a function type as a family of dependent functions indexed by arbitrary maps, the element at index $1 : I \longrightarrow I$ is a dependent function $(a\rho)_1$. We then define the I-element $(a\, b)\rho$ as $(a\rho)_1(b\rho)$.

Let $f : I \longrightarrow J$. We now show that $((a\, b)\rho)f = (a\, b)(\rho f)$. By induction hypothesis and properties of the family of dependent functions $a\rho$ we have $(a\rho)_1 f = (a\rho)_f = (a(\rho f))_1$ and $(b\rho)f = b(\rho f)$. Therefore $((a\, b)\rho)f = ((a\rho)_1 f)\,((b\rho)f) = (a(\rho f))_1\,(b(\rho f)) = (a\, b)(\rho f)$.

Abstraction. We interpret $\lambda(x : A).t$ (relative to Γ and a function type $(x : A) \to B$) as follows. Let $\rho \in \Gamma(I)$. The I-element $(\lambda(x : A).t)\rho$, belonging to the I-set $((x : A) \to B)\rho$, is defined as the nested family λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where each $\lambda_{\alpha h}$ is the dependent function mapping any $u \in A\rho\alpha h$ to $t\langle\rho\alpha h, x = u\rangle$.

Let $f : I \longrightarrow J$. We now show that $((\lambda(x : A).t)\rho)f = (\lambda(x : A).t)(\rho f)$. Let $\alpha : J \longrightarrow K$ and $h : K \rightarrowtail L$. By induction hypothesis $A\rho(f\alpha h) = A(\rho f)\alpha h$ and $t\langle\rho(f\alpha h), x = u\rangle = t\langle(\rho f)\alpha h, x = u\rangle$. Furthermore by property of the family of dependent functions $(\lambda(x : A).t)\rho$ we have $((\lambda(x : A).t)\rho)f_{\alpha h} = (\lambda(x : A).t)\rho_{f\alpha h}$. Therefore by definition the two families of dependent functions indexed by projections $\alpha : J \longrightarrow K$ and total maps $h : K \rightarrowtail L$ coincide: $((\lambda(x : A).t)\rho)f = (\lambda(x : A).t)(\rho f)$.

Variable. We can reuse the definition from section 5.1 since an I-set is a set.

Remark 11. *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 by the I-set of tuples $(n_J)_{J\subseteq I}$ where $n_\emptyset \in \mathbb{N}$ and $n_J = \emptyset$ for any non-empty $J \subseteq I$.*

5.3.2 Nominal extension

We now show how to interpret the nominal extension (section 2.2) of our calculus.

Interpretation of Contexts

Color context extension. In section 5.2.2 we defined

$$(\Gamma, i)(I) = \{[\rho, i = \mathbf{0}] \mid \rho \in \Gamma(I)\} \uplus \{[\rho, i = j] \mid j \in I, \rho \in \Gamma(I\backslash\{j\})\}.$$

For $\rho \in \Gamma(I)$ (resp. $j \in I$, $\rho \in \Gamma(I\backslash\{j\})$), we now redefine $[\rho, i = \mathbf{0}]$ (resp. $[\rho, i = j]$) as an I-element by taking

- $[\rho, i = \mathbf{0}]_K = (\rho_K, \mathbf{0})$ for each $K \subseteq I$; and
- $[\rho, i = j]_K = (\rho_K, \mathbf{0})$ and $[\rho, i = j]_{K,j} = (\rho_K, j)$ for each $K \subseteq I\backslash\{j\}$.

We already proved preservation of identities and composition (independently of the definition of $[\rho, i = \varphi]$) in section 5.2.2. As for preservation of projections, if $\alpha : I \longrightarrow J$ is a projection, we have by induction hypothesis $\rho\alpha_K = \rho_K$ for each $K \subseteq J$ (if $\varphi = \mathbf{0}$); and $\rho(\alpha - j)_K = \rho_K$ for each $K \subseteq J\backslash\{j\}$ (if $\varphi = j \in I$). Now, by case analysis on φ:

- $[\rho, i = \mathbf{0}]\alpha_K = [\rho\alpha, i = \mathbf{0}]_K = (\rho_K, \mathbf{0}) = [\rho, i = \mathbf{0}]_K$ for $\rho \in \Gamma(I)$ and $K \subseteq J \subseteq I$;
- $[\rho, i = j]\alpha_K = [\rho(\alpha - j), i = \mathbf{0}]_K = (\rho_K, \mathbf{0}) = [\rho, i = j]_K$ for $\rho \in \Gamma(I\backslash\{j\})$ and $K \subseteq J \subseteq I\backslash\{j\}$ (then $\alpha(j) = \mathbf{0}$); and
- $[\rho, i = j]\alpha_K = [\rho(\alpha - j), i = j]_K = (\rho_K, \mathbf{0}) = [\rho, i = j]_K$ and $[\rho, i = j]\alpha_{K,j} = [\rho(\alpha - j), i = j]_{K,j} = (\rho_K, j) = [\rho, i = j]_{K,j}$ for $\rho \in \Gamma(I\backslash\{j\})$ and $K \subseteq J \subseteq I$ such that $j \notin K$ (then $\alpha(j) = j$).

Interpretation of Types

Color product. The raw type $\forall i.A$ is interpreted (relative to Γ) as follows.

For $\rho \in \Gamma(I)$, we take $(\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$, where $j = \mathsf{fresh}(I)$ and $\bar{u}^j$ turns the (I, j)-element $u \in A[\rho, i = j]$ into an I-element (defined in Definition 16 by $\bar{u}^j_J = (u_J, u_{J,i})$ for each $J \subseteq I$).

Restriction maps. For $f : I \longrightarrow J$, we take $(\forall i.A)\rho \longrightarrow (\forall i.A)\rho f$ to be defined as $\bar{u}^j \longmapsto \bar{v}^k$, where $v = u(f, j = k) \in A[\rho, i = j](f, j = k)$ for $u \in A[\rho, i = j]$ with $j = \mathsf{fresh}(I)$ and $k = \mathsf{fresh}(J)$.

Preservation of identities. Direct, since $v = u(1, j = j) = u1 = u$.

Preservation of composition. Let $f : I \longrightarrow J$ and $g : J \longrightarrow K$. By definition we have $\bar{u}^j f = \bar{v}^k$ where $k = \mathsf{fresh}(J)$ and $v = u(f, j = k)$, and $\bar{v}^k g = \bar{w}^l$ where $l = \mathsf{fresh}(K)$ and $w = v(g, k = l)$. Now since $w = u(f, j = k)(g, k = l) = u(fg, j = l)$, we deduce that $(\bar{u}^j f)g = \bar{w}^l = \bar{u}^j(fg)$.

Preservation of projections. Let $\alpha : I \longrightarrow J$ a projection and $\bar{u}^j \in (\forall i.A)\rho$, where $j = \mathsf{fresh}(I)$. By definition we have $\bar{u}^j \alpha = \bar{v}^k$, where $k = \mathsf{fresh}(J)$ and $v = u(\alpha, j = k)$. Let $K \subseteq J$. We get by induction hypothesis that $v_K = (u\alpha(jk))_K = u\alpha_K = u_K$ and $v_{K,k} = (u\alpha(jk))_{K,k} = u\alpha_{K,j} = u_{K,j}$. Therefore $(\bar{u}^j \alpha)_K = \bar{v}^k_K = (v_K, v_{K,k}) = (u_K, u_{K,j}) = \bar{u}^j_K$.

Interpretation of Terms

Color product. We interpret the (small) color product $\forall i.A$ (relative to Γ and $\mathcal{U}$) as an I-element of the I-set which interprets $\mathcal{U}$, namely as a nested family of $\mathscr{U}$-small sets indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, by taking $(\forall i.A)\rho_{\alpha h}$ to be the $\mathscr{U}$-small K-set of K-elements $\bar{u}^k$ such that $u \in A[\rho\alpha h, i = k]_1$, where $k = \mathsf{fresh}(K)$. (Seeing the interpretation of the small type A relative to the universe $\mathcal{U}$ as a family of $\mathscr{U}$-small sets indexed by arbitrary maps, we take the element at index $1 : K, k \longrightarrow K, k$.) For $\alpha : I \longrightarrow J$, $h : J \rightarrowtail K$ and $g : K \longrightarrow L$, the restriction map $(\forall i.A)\rho_{\alpha h} \longrightarrow (\forall i.A)\rho_{\alpha hg}$ is defined to be $\bar{u}^k \longmapsto \bar{u}^k g = \bar{v}^l$ where $l = \mathsf{fresh}(L)$ and $v = u(g, k = l)$.

Let $f : I \longrightarrow J$. We now show that $((\forall i.A)\rho)f = ((\forall i.A)(\rho f)$. Let $\alpha : J \longrightarrow K$ and $h : K \rightarrowtail L$. We have $[A\rho f\alpha h, i = l]_1 = [A(\rho f)\alpha h, i = l]_1$ by induction hypothesis, hence by definition the two L-sets $(\forall i.A)\rho f_{\alpha h} = (\forall i.A)\rho_{f\alpha h}$ and $(\forall i.A)(\rho f)_{\alpha h}$ coincide.

Color application. We interpret $a@\varphi$ by case analysis on φ. For $\rho \in \Gamma(I)$,

- we take $(a@\mathbf{0})\rho = (a\rho)@\mathbf{0}$; and
- $(a@i)\rho$ is only defined for ρ of the form $[\rho', i = \varphi']$, and we take $(a@i)[\rho', i = \varphi'] = a\rho'@\varphi'$

$(a\rho)@\mathbf{0}$ (resp. $a\rho'@\mathbf{0}$) is an I-element since interpreting a relative to a color product type yields an I-element $a\rho$ (resp. $a\rho'$) of the form $\bar{u}^j$ (where $j = \mathsf{fresh}(I)$). Similarly $(a\rho')@i'$ is an I-element since interpreting a relative to a color product type yields an $(I\backslash\{i'\})$-element $a\rho'$ of the form $\bar{u}^j$ (where $j = \mathsf{fresh}(I\backslash\{i'\})$).

Let $f : I \longrightarrow J$. We now show that $((a@\varphi)\rho)f = (a@\varphi)(\rho f)$. By induction hypothesis $a\rho f = a(\rho f)$ (resp. $a\rho' f = a(\rho' f)$), and we distinguish three cases.

- for $\varphi = \mathbf{0}$, we have $((a@\mathbf{0})\rho)f = (a\rho@\mathbf{0})f = (a\rho)f@\mathbf{0} = a(\rho f)@\mathbf{0} = (a@\mathbf{0})(\rho f)$;
- for $\varphi = i$ and $\varphi' = \mathbf{0}$, we have $((a@i)[\rho', i = \mathbf{0}])f = (a\rho'@\mathbf{0})f = (a\rho')f@\mathbf{0} = a(\rho' f)@\mathbf{0} = (a@i)[\rho f, i = \mathbf{0}]$; and
- for $\varphi = i$ and $\varphi' = j \in I$, we have $((a@i)[\rho', i = j])f = (a\rho'@j)f = (a\rho')(f - j)@f(j) = a(\rho'(f - j))@f(j) = (a@i)([\rho', i = j]f)$.

Color abstraction. We interpret $\langle i\rangle a$ (relative to Γ and a color product type $\forall i.A$) as follows. Let $\rho \in \Gamma(I)$. The I-element $(\langle i\rangle a)\rho$, belonging to the I-set $(\forall i.A)\rho$, is defined as $\bar{u}^j$, where $j = \mathsf{fresh}(I)$ and $u = a[\rho, i = j]$. (Indeed $\bar{u}^j$ is an I-element since u is (I, j)-element.) In other words (see Definition 16), $(\langle i\rangle a)\rho_J = (a[\rho, i = j]_J, a[\rho, i = j]_{J,j})$ for each $J \subseteq I$.

Let $f : I \longrightarrow J$. We now show that $((\langle i\rangle a)\rho)f = (\langle i\rangle a)(\rho f)$. Let $u = a[\rho, i = j]$ with $j = \mathsf{fresh}(I)$, and $v = u(f, j = k) = a[\rho f, i = k]$ with $k = \mathsf{fresh}(J)$. By induction hypothesis we have $a[\rho, i = j](f, j = k) = a([\rho, i = j](f, j = k))$, hence $((\langle i\rangle a)\rho)f = \bar{u}^j f = \bar{v}^k = (\langle i\rangle a)(\rho f)$.

5.3.3 Parametric extension

We now show how to interpret the parametric extension (section 2.3) of our calculus.

Interpretation of Types

Parametricity type. The raw type $P \ni a$ is interpreted (relative to Γ) as follows.

For $\rho \in \Gamma(I)$, $(P \ni a)\rho$ is the I-set of I-elements $u!$ (see Definition 16) for $u \in P\rho$ such that $u@\mathbf{0} = a\rho$.

In particular, if P is a color product $\forall i.A$ we have $(\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$, where $j = \mathsf{fresh}(I)$, hence by Definition 16 $((\forall .A) \ni a)\rho$ is the I-sets of I-elements $(u_{J,j})_{J \subseteq I}$ such that $u \in A[\rho, i = j]$ and $u(j\,\mathbf{0}) = a\rho$.

Restriction maps. For $f : I \longrightarrow J$, we take $(P \ni a)\rho \longrightarrow (P \ni a)\rho f$ to be defined as $u! \longmapsto uf!$. (Indeed for each $u \in P\rho$ such that $u@\mathbf{0} = a\rho$ we have $uf \in P\rho f$ and $(u@\mathbf{0})f = uf@\mathbf{0} = a\rho f$.)

Preservation of identities. Direct, since $u!\,1 = u1! = u!$.

Preservation of composition. Direct, since for each maps $f : I \longrightarrow J$ and $g : J \longrightarrow K$ we have $(u!f)g = (uf)!\,g = u(fg)! = u!\,(fg)$.

Preservation of projections. Let $\alpha : I \longrightarrow J$ a projection and $u! \in (P \ni a)\rho$. Let $j = \mathsf{fresh}(J)$ and $K \subseteq J$. By Definition 16 we have $(u\alpha!)_K = u\alpha_{K,j}$ and $u!_K = u_{K,j}$. Now by induction hypothesis $u\alpha_{K,j} = u_{K,j}$, hence $(u!\,\alpha)_K = (u\alpha!)_K = u\alpha_{K,j} = u_{K,j} = u!_K$.

Interpretation of Terms

Colored pair. If $\rho \in \Gamma(I)$, the I-element $(\!|a,p|\!)\rho = (u_J)_{J\subseteq I}$ is defined by $u_J = ((a\rho)_J, (p\rho)_J)$ for each $J \subseteq I$.

Let $f : I \longrightarrow J$. Since by induction hypothesis $a\rho f = (a\rho)f$ and $p\rho f = (p\rho)f$, for each $K \subseteq J$ we get that $((\!|a,p|\!)\rho f)_K = ((a(\rho f))_K, (p(\rho f))_K) = ((a\rho)f)_K, ((p\rho)f)_K) = (((\!|a,p|\!)\rho)f)_K$. Hence $(\!|a,p|\!)\rho f = ((\!|a,p|\!)\rho)f$.

Parametricity type. We interpret the (small) parametricity type $P \ni a$ (relative to Γ and $\mathcal{U}l$) as an I-element of the I-set which interprets $\mathcal{U}l$, namely as a nested family of $\mathcal{U}$-small K-sets indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, by taking $(P \ni a)\rho_{\alpha h}$ to be the $\mathcal{U}$-small K-set of K-elements $u!$ for $u \in P\rho_{\alpha h}$ such that $u@\mathbf{0} = a\rho\alpha h$. (Taking the interpretation of the small type P relative to the universe $\mathcal{U}l$ yields a nested family of $\mathcal{U}$-small sets indexed by projection maps and total maps, and we take the element at indices α and h to obtain a K-set $P\rho_{\alpha h}$.) For $\alpha : I \longrightarrow J$, $h : J \rightarrowtail K$ and $g : K \longrightarrow L$, the restriction map $(P \ni a)\rho_{\alpha h} \longrightarrow (P \ni a)\rho_{\alpha hg}$ is defined to be $u! \longmapsto ug!$.

Let $f : I \longrightarrow J$. For $\alpha : J \longrightarrow K$ and $h : K \rightarrowtail L$ the two L-sets $(P \ni a)\rho f_{\alpha h} = (P \ni a)\rho_{f\alpha h}$ and $(P \ni a)(\rho f)_{\alpha h}$ coincide, since by induction hypothesis $P\rho f = P(\rho f)$ and $a\rho f\alpha h = a(\rho f)\alpha h$.

Parametricity proof. If $\rho \in \Gamma(I)$, we take $(a!)\rho = (a\rho)!$. (Indeed if $a\rho$ is an I-element then so is $(a\rho)!$.)

Let $f : I \longrightarrow J$. Since by induction hypothesis $a\rho f = (a\rho)f$, we directly obtain $((a!)\rho)f = a\rho!f = (a\rho)f! = a(\rho f)! = (a!)(\rho f)$.

Parametricity predicate. We interpret the parametricity predicate $\Psi_A P$ (relative to Γ and $(\forall i.\mathcal{U}l) \ni A$) as an I-element of the I-set

$$((\forall i.\mathcal{U}l) \ni A)\rho = \{(u_{J,j})_{J\subseteq I} \mid u \in \mathcal{U}l[\rho, i = j], u(j\,\mathbf{0}) = A\rho\},$$

where $j = \mathsf{fresh}(I)$. For $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, we define $(\Psi_A P)\rho_{\alpha h}$ as the $\mathcal{U}$-small K-set of K-elements $v! \in ((P\rho)_{\alpha h}(u))_1$ for $u \in A\rho_{\alpha h}$ such that $u = v@\mathbf{0}$. (Taking the interpretation of P relative to the simple function type $A \to \mathcal{U}l$ yields a nested family of functions indexed by projection maps and total maps, and we take the element at indices α and h to obtain a set-theoretic function $(P\rho)_{\alpha h} : A\rho_{\alpha h} \to \mathcal{U}l\rho\alpha h$. Hence $(P\rho)_{\alpha h}(u)$ is a nested family of $\mathcal{U}$-small sets, which we turn into a K-set by taking the element at index $1 : K \longrightarrow K$.)

Let $f : I \longrightarrow J$. For $\alpha : J \longrightarrow K$ and $h : K \rightarrowtail L$ the two L-sets $(\Psi_A P)\rho f_{\alpha h} = (\Psi_A P)\rho_{f\alpha h}$ and $(P \ni a)(\rho f)_{\alpha h}$ coincide, since by induction hypothesis $(P\rho f)_{\alpha h} = (P\rho)_{f\alpha h} = ((P\rho)f)_{\alpha h}$ and $A\rho f = A(\rho f)$.

Parametricity function. If $\rho \in \Gamma(I)$, we take $(\Phi_t u)\rho = \bar{\lambda}^j!$, where $j = \mathsf{fresh}(I)$, $\lambda = ((\lambda_{\alpha h})_{h:J\rightarrowtail K})_{\alpha:I,j\longrightarrow J}$ and for $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$, the dependent function $\lambda_{\alpha h}$ is defined as follows by case analysis on $\alpha(j)$:

- if $\alpha(j) = \mathbf{0}$, we take $\lambda_{\alpha h} = (t\rho)_{\alpha h - j}$; and
- if $h(\alpha(j)) = k \in K$, we take for $\lambda_{\alpha h}$ the dependent function mapping each $v \in A[\rho(\alpha h - j), i = k]$ to $w(l\,k)$, where $l = \mathsf{fresh}(K \backslash \{k\})$, $(t\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0}) = \bar{w}^l @ \mathbf{0}$ and $(u\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l !$.

5.4 Validity results

Our interpretation functions are defined by induction on raw contexts, types and term. Totality on valid judgments (Theorem 11) is proven by mutual induction with equality on convertible terms (Theorem 10) and substitutions laws (Theorems 8 and 9).

Theorem 8 (Substitution law)**.**

- *If* $\Gamma, z : A \vdash B$ *and* $\Gamma \vdash a : A$ *then for any* $\rho \in \Gamma(I)$ *we have* $B[z \mapsto a]\rho = B\langle \rho, z = a\rho \rangle$*; and*
- *If* $\Gamma, z : A \vdash b : B$ *and* $\Gamma \vdash a : A$ *then for any* $\rho \in \Gamma(I)$ *we have* $b[z \mapsto a]\rho = b\langle \rho, z = a\rho \rangle$.

Proof. By simultaneous induction on the typing judgments $\Gamma, z : A \vdash B$ and $\Gamma, z : A \vdash b : B$ (Definition 2).

Universe**.** We directly have $\mathcal{U}\rho = \mathcal{U}\langle \rho, z = a\rho \rangle$ since the interpretation of the universe does not depend on the environment.

Prod**.** We need to prove that $((x : A[z \mapsto a]) \to B[z \mapsto a])\rho = ((x : A) \to B)\langle \rho, z = a\rho \rangle$. By definition $((x : A[z \mapsto a]) \to B[z \mapsto a])\rho$ is the I-set of nested families λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, such that each $\lambda_{\alpha h}$ is a dependent function satisfying $\lambda_{\alpha h}(u) \in B[z \mapsto a]\langle \rho\alpha h, x = u \rangle$ for each $u \in A[z \mapsto a]\rho\alpha h$, and $(\lambda_{\alpha h}(u))g = \lambda_{\alpha hg}(ug)$ for each $u \in A[z \mapsto a]\rho\alpha h$ and $g : K \longrightarrow L$.

Now by induction hypothesis $A[z \mapsto a]\rho\alpha h = A\langle \rho, z = a\rho \rangle \alpha h$ and $B[z \mapsto a]\langle \rho\alpha h, x = u \rangle = B\langle \langle \rho\alpha h, z = a\rho\alpha h \rangle, x = u \rangle$ for each $u \in A[z \mapsto a]\rho\alpha h = A\langle \rho, z = a\rho \rangle \alpha h$. Therefore the two I-sets $((x : A[z \mapsto a]) \to B[z \mapsto a])\rho$ and $((x : A) \to B)\langle \rho, z = a\rho \rangle$ coincide.

Small**.** Direct use of the induction hypothesis.

ColPi**.** We need to prove that $(\forall i.A[z \mapsto a])\rho = (\forall i.A)\langle \rho, z = a\rho \rangle$. Let $j = \mathsf{fresh}(I)$. By induction hypothesis we get $A[z \mapsto a][\rho, i = j] = A\langle [\rho, i = j], z = a[\rho, i = j] \rangle = A[\langle \rho, z = a\rho \rangle, i = j]$. We therefore obtain

$$\begin{aligned}(\forall i.A[z \mapsto a])\rho &= \{\bar{u}^j \mid u \in A[z \mapsto a][\rho, i = j]\} \\ &= \{\bar{u}^j \mid u \in A[\langle \rho, z = a\rho \rangle, i = j]\} \\ &= (\forall i.A)\langle \rho, z = a\rho \rangle\end{aligned}$$

Out. We need to show that $((\forall i.A[z \mapsto a]) \ni t[z \mapsto a])\rho = (\forall i.A \ni t)\langle \rho, z = a\rho\rangle$.

By definition $((\forall i.A[z \mapsto a]) \ni t[z \mapsto a])\rho$ is the I-set of I-elements u! such that $u \in (\forall i.A[z \mapsto a])\rho$ and $u\mathbf{@0} = a[z \mapsto a]\rho$. Similarly $(\forall i.A \ni t)\langle \rho, z = a\rho\rangle$ is the I-set of I-elements v! such that $v \in (\forall i.A)\langle \rho, z = a\rho\rangle$ and $v\mathbf{@0} = a\langle \rho, z = a\rho\rangle$.

We conclude that the two I-sets coincide since the induction hypothesis gives that $(\forall i.A[z \mapsto a])\rho = (\forall i.A)\langle \rho, z = a\rho\rangle$ and $a[z \mapsto a]\rho = a\langle \rho, z = a\rho\rangle$.

Conv. Direct use of the induction hypothesis and Theorem 10.

Var. Trivial.

Abs. We need to show that $(\lambda(x : A[z \mapsto a]).t[z \mapsto a])\rho = (\lambda(x : A).t)\langle \rho, z = a\rho\rangle$.

By definition $(\lambda(x : A[z \mapsto a]).t[z \mapsto a])\rho$ is a nested family λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\lambda_{\alpha h}$ is the dependent function mapping any $u \in A[z \mapsto a]\rho\alpha h$ to $t[z \mapsto a]\langle \rho\alpha h, x = u\rangle$ Similarly $(\lambda(x : A).t)\langle \rho, z = a\rho\rangle$ is a nested family λ' indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\lambda'_{\alpha h}$ is the dependent function mapping any $u \in A\langle \rho, z = a\rho\rangle\alpha h$ to $t\langle\langle \rho, z = a\rho\rangle\alpha h, x = u\rangle$. Now by induction hypothesis we have

$$\begin{aligned} A[z \mapsto a]\rho\alpha h &= A\langle \rho\alpha h, z = a\rho\alpha h\rangle = A\langle \rho, z = a\rho\rangle\alpha h; \text{ and} \\ t[z \mapsto a]\langle \rho\alpha h, x = u\rangle &= t\langle\langle \rho\alpha h, x = u\rangle, z = a\langle \rho\alpha h, x = u\rangle\rangle \\ &= t\langle\langle \rho\alpha h, x = u\rangle, z = a\rho\alpha h\rangle \\ &= t\langle\langle \rho\alpha h, z = a\rho\alpha h\rangle, x = u\rangle \\ &= t\langle\langle \rho, z = a\rho\rangle\alpha h, x = u\rangle, \end{aligned}$$

hence we deduce $\lambda_{\alpha h} = \lambda'_{\alpha h}$.

App. We need to prove that $(t[z \mapsto a]\, u[z \mapsto a])\rho = (t\, u)\langle \rho, z = a\rho\rangle$.

By induction hypothesis we get that $t[z \mapsto a]\rho = t\langle \rho, z = a\rho\rangle$ and $u[z \mapsto a]\rho = u\langle \rho, z = a\rho\rangle$, hence by definition

$$\begin{aligned} (t[z \mapsto a]\, u[z \mapsto a])\rho &= (t[z \mapsto a]\rho)_1(u[z \mapsto a]\rho) \\ &= (t\langle \rho, z = a\rho\rangle)_1(u\langle \rho, z = a\rho\rangle) \\ &= (t\, u)\langle \rho, z = a\rho\rangle. \end{aligned}$$

Prod-$\mathcal{U}$. Like for Prod.

Out-$\mathcal{U}$. Like for Out.

ColAbs. We need to prove that $(\langle i\rangle t[z \mapsto a])\rho = (\langle i\rangle t)\langle \rho, z = a\rho\rangle$. Let $j = \mathsf{fresh}(I)$. By definition $(\langle i\rangle t[z \mapsto a])\rho = \bar{u}^j$, where $u = (t[z \mapsto a])[\rho, i = j]$. Similarly $(\langle i\rangle t)\langle \rho, z = a\rho\rangle = \bar{v}^j$, where $v = t[\langle \rho, z = a\rho\rangle, i = j]$.

Now since by induction hypothesis we have

$$\begin{aligned}(t[z \mapsto a])[\rho, i = j] &= t\langle [\rho, i = j], z = a[\rho, i = j]\rangle \\ &= t\langle [\rho, i = j], z = a\rho\rangle \\ &= t[\langle \rho, z = a\rho\rangle, i = j],\end{aligned}$$

we deduce $u = v$ and therefore $\bar{u}^j = \bar{v}^j$.

In-Abs. We need to prove that $(\!|t[z \mapsto a], u[z \mapsto a]|\!)\rho = (\!|t, u|\!)\langle \rho, z = a\rho\rangle$. By induction hypothesis we get $t[z \mapsto a]\rho = t\langle \rho, z = a\rho\rangle$ and $u[z \mapsto a]\rho = u\langle \rho, z = a\rho\rangle$. Since for each $J \subseteq I$ we have

$$\begin{aligned}(\!|t[z \mapsto a], u[z \mapsto a]|\!)\rho_J &= (t[z \mapsto a]\rho_J, u[z \mapsto a]\rho_J) \\ &= (t\langle \rho, z = a\rho\rangle_J, u\langle \rho, z = a\rho\rangle_J) \\ &= (\!|t, u|\!)\langle \rho, z = a\rho\rangle_J,\end{aligned}$$

we deduce that the I-elements $(\!|t[z \mapsto a], u[z \mapsto a]|\!)\rho$ and $(\!|t, u|\!)\langle \rho, z = a\rho\rangle$ are equal.

ColPi-$\mathcal{U}$. Like for ColPi.

In-Pred. We need to show that $(\Psi_{A[z\mapsto a]}P[z \mapsto a])\rho = (\Psi_A P)\langle \rho, z = a\rho\rangle$.

By definition $(\Psi_{A[z\mapsto a]}P[z \mapsto a])\rho$ is a nested family $\mathcal{P}$ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\mathcal{P}_{\alpha h}$ is the $\mathscr{U}$-small K-set of K-elements $v! \in ((P[z \mapsto a]\rho)_{\alpha h}(u))_1$ for $u \in A[z \mapsto a]\rho_{\alpha h}$ such that $u = v\mathbf{@0}$. Similarly, $(\Psi_A P)\langle \rho, z = a\rho\rangle$ is a nested family $\mathcal{P}'$ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\mathcal{P}'_{\alpha h}$ is the $\mathscr{U}$-small K-set of K-elements $v'! \in ((P\langle \rho, z = a\rho\rangle)_{\alpha h}(u'))_1$ for $u' \in A\langle \rho, z = a\rho\rangle_{\alpha h}$ such that $u' = v'\mathbf{@0}$.

Since by induction hypothesis we have $P[z \mapsto a]\rho = P\langle \rho, z = a\rho\rangle$ and $A[z \mapsto a]\rho = A\langle \rho, z = a\rho\rangle$, we deduce that $\mathcal{P}_{\alpha h} = \mathcal{P}'_{\alpha h}$ for each $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, and therefore that $\mathcal{P} = \mathcal{P}'$.

In-Fun. We need to prove that $(\Phi_{t[z\mapsto a]}u[z \mapsto a])\rho = (\Phi_t u)\langle \rho, z = a\rho\rangle$.

Let $j = \mathsf{fresh}(I)$. By definition $(\Phi_{t[z\mapsto a]}u[z \mapsto a])\rho = \bar{\lambda}^j!$, where λ is a nested family indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$. Similarly $(\Phi_t u)\langle \rho, z = a\rho\rangle = \bar{\lambda'}^j!$, where λ' is a nested family indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$.

We now show by case analysis on $\alpha(j)$ that $\lambda_{\alpha h} = \lambda'_{\alpha h}$ for each $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$.

- If $\alpha(j) = \mathbf{0}$ we have $\lambda_{\alpha h} = (t[z \mapsto a]\rho)_{\alpha h - j} = (t\langle \rho, z = a\rho\rangle)_{\alpha h - j} = \lambda'_{\alpha h}$ since by induction hypothesis $t[z \mapsto a]\rho = t\langle \rho, z = a\rho\rangle$.

- If $h(\alpha(j)) = k \in K$, by definition $\lambda_{\alpha h}$ is the dependent function mapping each $v \in A[z \mapsto a][\rho(\alpha h - j), i = k]$ to $w(l\,k)$ where $(t[z \mapsto a]\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0}) = \bar{w}^l @ \mathbf{0}$, $(u[z \mapsto a]\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l !$ and $l = \mathsf{fresh}(K \backslash \{k\})$; similarly $\lambda'_{\alpha h}$ is the dependent function mapping each $v' \in A[\langle \rho, z = a\rho\rangle(\alpha h - j), i = k]$ to $w'(l\,k)$ where $(t\langle \rho, z = a\rho\rangle)_{\alpha h - j}(\bar{v'}^k @ \mathbf{0}) = \bar{w'}^l @ \mathbf{0}$ and $(u\langle \rho, z = a\rho\rangle)_{\alpha h - j}(\bar{v'}^k) = \bar{w'}^l !$. Now since by induction hypothesis

$$\begin{aligned} t[z \mapsto a]\rho &= t\langle \rho, z = a\rho\rangle; \\ u[z \mapsto a]\rho &= u\langle \rho, z = a\rho\rangle; \text{ and} \\ A[z \mapsto a][\rho(\alpha h - j), i = k] &= A\langle [\rho(\alpha h - j), i = k], z = a\rho(\alpha h - j)\rangle \\ &= A[\langle \rho(\alpha h - j), z = a\rho(\alpha h - j)\rangle, i = k] \\ &= A[\langle \rho, z = a\rho\rangle(\alpha h - j), i = k], \end{aligned}$$

  we deduce that $\lambda_{\alpha h} = \lambda'_{\alpha h}$.

ColApp. Absurd since the context $\Gamma, i, \vec{j}$ cannot end with a variable $z : A$.

ColApp-Orig. We need to show that $(t[z \mapsto a]@\mathbf{0})\rho = (t@\mathbf{0})\langle \rho, z = a\rho\rangle$. By induction hypothesis $t[z \mapsto a]\rho = t\langle \rho, z = a\rho\rangle$, and therefore $(t[z \mapsto a]@\mathbf{0})\rho = t[z \mapsto a]\rho@\mathbf{0} = t\langle \rho, z = a\rho\rangle@\mathbf{0} = (t@\mathbf{0})\langle \rho, z = a\rho\rangle$.

Param. We need to show that $(t[z \mapsto a]!)\rho = (t!)\langle \rho, z = a\rho\rangle$. By induction hypothesis $t[z \mapsto a]\rho = t\langle \rho, z = a\rho\rangle$, and therefore $(t[z \mapsto a]!)\rho = t[z \mapsto a]\rho! = t\langle \rho, z = a\rho\rangle! = (t!)\langle \rho, z = a\rho\rangle$. □

It also satisfies the substitution law on colors:

Theorem 9 (Substitution law on colors)**.**

- *If $\Gamma, i \vdash A$ then for any $\rho \in \Gamma(I)$ and $j \notin I$ we have $A(i\,\mathbf{0})\rho = A[\rho, i = \mathbf{0}] = A[\rho, i = j](j\,\mathbf{0})$. (Since $[\rho, i = \mathbf{0}] \in (\Gamma * \mathbb{I})(I)$ and $[\rho, i = j] \in (\Gamma * \mathbb{I})(I, j)$, $A(i\,\mathbf{0})\rho$ and $A[\rho, i = \mathbf{0}]$ are both I-sets while $A[\rho, i = j]$ is a (I, j)-set.)*
- *Similarly if $\Gamma, i \vdash a : A$ then for any $\rho \in \Gamma(I)$ and $j \notin I$ we have $a(i\,\mathbf{0})\rho = a[\rho, i = \mathbf{0}] = a[\rho, i = j](j\,\mathbf{0})$.*

Proof. By simultaneous induction on the typing judgments $\Gamma, i \vdash A$ and $\Gamma, i \vdash a : A$ (Definition 2).

Universe. We directly have $\mathcal{U}(i\,\mathbf{0})\rho = \mathcal{U}[\rho, i = \mathbf{0}]$ since the interpretation of the universe does not depend on the environment.

Prod. We need to prove that $((x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}))\rho = ((x : A) \to B)[\rho, i = \mathbf{0}]$. By definition $((x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}))\rho$ is the I-set of nested families λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, such that each $\lambda_{\alpha h}$ is a dependent function satisfying $\lambda_{\alpha h}(u) \in B(i\,\mathbf{0})\langle \rho\alpha h, x = u\rangle$ for each

$u \in A(i\,\mathbf{0})\rho\alpha h$, and $(\lambda_{\alpha h}(u))g = \lambda_{\alpha hg}(ug)$ for each $u \in A(i\,\mathbf{0})\rho\alpha h$ and $g : K \longrightarrow L$.

Now by induction hypothesis $A(i\,\mathbf{0})\rho\alpha h = A[\rho, i = \mathbf{0}]\alpha h$ and for each $u \in A(i\,\mathbf{0})\rho\alpha h = A[\rho, i = \mathbf{0}]\alpha h$, one has $B(i\,\mathbf{0})\langle\rho\alpha h, x = u\rangle = B[\langle\rho\alpha h, x = u\rangle, i = \mathbf{0}] = B\langle[\rho\alpha h, i = \mathbf{0}], x = u\rangle$. Therefore the two I-sets $((x : A(i\,\mathbf{0})) \to B(i\,\mathbf{0}))\rho$ and $((x : A) \to B)[\rho, i = \mathbf{0}]$ coincide.

Small. Direct use of the induction hypothesis.

ColPi. We need to prove that $(\forall j.A(i\,\mathbf{0}))\rho = (\forall j.A)[\rho, i = \mathbf{0}]$. Let $k = \mathsf{fresh}(I)$. By induction hypothesis we get $A(i\,\mathbf{0})[\rho, j = k] = A[[\rho, j = k], i = \mathbf{0}]\rangle = A[[\rho, i = \mathbf{0}], j = k]$. We therefore obtain

$$\begin{aligned}(\forall j.A(i\,\mathbf{0}))\rho &= \{\vec{u}^k \mid u \in A(i\,\mathbf{0})[\rho, j = k]\}\\ &= \{\vec{u}^k \mid u \in A[[\rho, i = \mathbf{0}], j = k]\}\\ &= (\forall j.A)[\rho, i = \mathbf{0}]\end{aligned}$$

Out. We need to show that $((\forall j.A(i\,\mathbf{0})) \ni t(i\,\mathbf{0}))\rho = (\forall j.A \ni t)[\rho, i = \mathbf{0}]$.

By definition $((\forall j.A(i\,\mathbf{0})) \ni t(i\,\mathbf{0}))\rho$ is the I-set of I-elements u! such that $u \in (\forall j.A(i\,\mathbf{0}))\rho$ and $u@\mathbf{0} = a(i\,\mathbf{0})\rho$. Similarly $(\forall j.A \ni t)[\rho, i = \mathbf{0}]$ is the I-set of I-elements v! such that $v \in (\forall j.A)[\rho, i = \mathbf{0}]$ and $v@\mathbf{0} = a[\rho, i = \mathbf{0}]$.

We conclude that the two I-sets coincide since the induction hypothesis gives that $(\forall j.A(i\,\mathbf{0}))\rho = (\forall j.A)[\rho, i = \mathbf{0}]$ and $a(i\,\mathbf{0})\rho = a[\rho, i = \mathbf{0}]$.

Conv. Direct use of the induction hypothesis and Theorem 10.

Var. Trivial.

Abs. We need to show that $(\lambda(x : A(i\,\mathbf{0})).t(i\,\mathbf{0}))\rho = (\lambda(x : A).t)[\rho, i = \mathbf{0}]$.

By definition $(\lambda(x : A(i\,\mathbf{0})).t(i\,\mathbf{0}))\rho$ is a nested family λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\lambda_{\alpha h}$ is the dependent function mapping any $u \in A(i\,\mathbf{0})\rho\alpha h$ to $t(i\,\mathbf{0})\langle\rho\alpha h, x = u\rangle$ Similarly $(\lambda(x : A).t)[\rho, i = \mathbf{0}]$ is a nested family λ' indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\lambda'_{\alpha h}$ is the dependent function mapping any $u \in A[\rho, i = \mathbf{0}]\alpha h$ to $t\langle[\rho, i = \mathbf{0}]\alpha h, x = u\rangle$. Now by induction hypothesis we have

$$\begin{aligned}A(i\,\mathbf{0})\rho\alpha h = A[\rho\alpha h, i = \mathbf{0}] &= A[\rho, i = \mathbf{0}]\alpha h; \text{ and}\\ t(i\,\mathbf{0})\langle\rho\alpha h, x = u\rangle &= t[\langle\rho\alpha h, x = u\rangle, i = \mathbf{0}]\\ &= t\langle[\rho\alpha h, i = \mathbf{0}], x = u\rangle\\ &= t\langle[\rho, i = \mathbf{0}]\alpha h, x = u\rangle,\end{aligned}$$

hence we deduce $\lambda_{\alpha h} = \lambda'_{\alpha h}$.

App. We need to prove that $(t(i\,\mathbf{0})\,u(i\,\mathbf{0}))\rho = (t\,u)[\rho, i = \mathbf{0}]$.

By induction hypothesis we get that $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$ and $u(i\,\mathbf{0})\rho = u[\rho, i = \mathbf{0}]$, hence by definition

$$\begin{aligned}(t(i\,\mathbf{0})\,u(i\,\mathbf{0}))\rho &= (t(i\,\mathbf{0})\rho)_1(u(i\,\mathbf{0})\rho)\\ &= (t[\rho, i = \mathbf{0}])_1(u[\rho, i = \mathbf{0}])\\ &= (t\,u)[\rho, i = \mathbf{0}].\end{aligned}$$

Prod-$\mathcal{U}$. Like for Prod.

Out-$\mathcal{U}$. Like for Out.

ColAbs. We need to prove that $(\langle j\rangle t(i\,\mathbf{0}))\rho = (\langle j\rangle t)[\rho, i = \mathbf{0}]$. Let $k = \mathsf{fresh}(I)$. By definition $(\langle j\rangle t(i\,\mathbf{0}))\rho = \bar{u}^k$, where $u = (t(i\,\mathbf{0}))[\rho, j = k]$. Similarly $(\langle j\rangle t)[\rho, i = \mathbf{0}] = \bar{v}^k$, where $v = t[[\rho, i = \mathbf{0}], j = k]$.

Now since by induction hypothesis we have $(t(i\,\mathbf{0}))[\rho, j = k] = t[[\rho, j = k], i = \mathbf{0}]] = t[[\rho, i = \mathbf{0}], j = k]$, we deduce $u = v$ and therefore $\bar{u}^k = \bar{v}^k$.

In-Abs. We need to prove that $(\!|t(i\,\mathbf{0}), u(i\,\mathbf{0})|\!)\rho = (\!|t, u|\!)[\rho, i = \mathbf{0}]$. By induction hypothesis we get $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$ and $u(i\,\mathbf{0})\rho = u[\rho, i = \mathbf{0}]$. Since for each $J \subseteq I$ we have

$$\begin{aligned}(\!|t(i\,\mathbf{0}), u(i\,\mathbf{0})|\!)\rho_J &= (t(i\,\mathbf{0})\rho_J, u(i\,\mathbf{0})\rho_J)\\ &= (t[\rho, i = \mathbf{0}]_J, u[\rho, i = \mathbf{0}]_J)\\ &= (\!|t, u|\!)[\rho, i = \mathbf{0}]_J,\end{aligned}$$

we deduce that the I-elements $(\!|t(i\,\mathbf{0}), u(i\,\mathbf{0})|\!)\rho$ and $(\!|t, u|\!)[\rho, i = \mathbf{0}]$ are equal.

ColPi-$\mathcal{U}$. Like for ColPi.

In-Pred. We need to show that $(\Psi_{A(i\,\mathbf{0})}P(i\,\mathbf{0}))\rho = (\Psi_A P)[\rho, i = \mathbf{0}]$.

By definition $(\Psi_{A(i\,\mathbf{0})}P(i\,\mathbf{0}))\rho$ is a nested family $\mathcal{P}$ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\mathcal{P}_{\alpha h}$ is the $\mathcal{U}$-small K-set of K-elements $v! \in ((P(i\,\mathbf{0})\rho)_{\alpha h}(u))_1$ for $u \in A(i\,\mathbf{0})\rho_{\alpha h}$ such that $u = v@\mathbf{0}$; and $(\Psi_A P)[\rho, i = \mathbf{0}]$ is a nested family $\mathcal{P}'$ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where $\mathcal{P}'_{\alpha h}$ is the $\mathcal{U}$-small K-set of K-elements $v'! \in ((P[\rho, i = \mathbf{0}])_{\alpha h}(u'))_1$ for $u' \in A[\rho, i = \mathbf{0}]_{\alpha h}$ such that $u' = v'@\mathbf{0}$.

Since by induction hypothesis we have $P(i\,\mathbf{0})\rho = P[\rho, i = \mathbf{0}]$ and $A(i\,\mathbf{0})\rho = A[\rho, i = \mathbf{0}]$, we deduce that $\mathcal{P}_{\alpha h} = \mathcal{P}'_{\alpha h}$ for each $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, and therefore that $\mathcal{P} = \mathcal{P}'$.

In-Fun. We need to prove that $(\Phi_{t(i\,\mathbf{0})}u(i\,\mathbf{0}))\rho = (\Phi_t u)[\rho, i = \mathbf{0}]$.

Let $j = \mathsf{fresh}(I)$. By definition $(\Phi_{t(i\,\mathbf{0})} u(i\,\mathbf{0}))\rho = \bar{\lambda}^j!$, where λ is a nested family indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$. Similarly $(\Phi_t u)[\rho, i = \mathbf{0}] = \bar{\lambda'}^j!$, where λ' is a nested family indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$.

We now show by case analysis on $\alpha(j)$ that $\lambda_{\alpha h} = \lambda'_{\alpha h}$ for each $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$.

- If $\alpha(j) = \mathbf{0}$ we have $\lambda_{\alpha h} = (t(i\,\mathbf{0})\rho)_{\alpha h - j} = (t[\rho, i = \mathbf{0}])_{\alpha h - j} = \lambda'_{\alpha h}$ since by induction hypothesis $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$.
- If $h(\alpha(j)) = k \in K$, by definition $\lambda_{\alpha h}$ is the dependent function mapping each $v \in A(i\,\mathbf{0})[\rho(\alpha h - j), i' = k]$ to $w(l\,k)$ where $l = \mathsf{fresh}(K \backslash \{k\})$, $(t(i\,\mathbf{0})\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0}) = \bar{w}^l @ \mathbf{0}$ and $(u(i\,\mathbf{0})\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l!$; similarly $\lambda'_{\alpha h}$ is the dependent function mapping each $v' \in A[[\rho, i = \mathbf{0}](\alpha h - j), i' = k]$ to $w'(l\,k)$ where $(t[\rho, i = \mathbf{0}])_{\alpha h - j}(\bar{v'}^k @ \mathbf{0}) = \bar{w'}^l @ \mathbf{0}$ and $(u[\rho, i = \mathbf{0}])_{\alpha h - j}(\bar{v'}^k) = \bar{w'}^l!$. Now since by induction hypothesis

$$\begin{aligned} t(i\,\mathbf{0})\rho &= t[\rho, i = \mathbf{0}]; \\ u(i\,\mathbf{0})\rho &= u[\rho, i = \mathbf{0}]; \text{ and} \\ A(i\,\mathbf{0})[\rho(\alpha h - j), i' = k] &= A[[\rho(\alpha h - j), i' = k], i = \mathbf{0}] \\ &= A[[\rho(\alpha h - j), i = \mathbf{0}], i' = k] \\ &= A[[\rho, i = \mathbf{0}](\alpha h - j), i' = k], \end{aligned}$$

  we deduce that $\lambda_{\alpha h} = \lambda'_{\alpha h}$.

ColApp. For simplicity we only consider the case where $\vec{j}$ is empty. We need to show that $(t(i\,\mathbf{0})@j)[\rho, j = \varphi] = (t@j)[[\rho, j = \varphi], i = \mathbf{0}]$. By induction hypothesis $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$, and therefore $(t(i\,\mathbf{0})@j)[\rho, j = \varphi] = t(i\,\mathbf{0})\rho@\varphi = t[\rho, i = \mathbf{0}]@\varphi = (t@j)[[\rho, i = \mathbf{0}], j = \varphi]$.

ColApp-Orig. We need to show that $(t(i\,\mathbf{0})@\mathbf{0})\rho = (t@\mathbf{0})[\rho, i = \mathbf{0}]$. By induction hypothesis $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$, and therefore $(t(i\,\mathbf{0})@\mathbf{0})\rho = t(i\,\mathbf{0})\rho@\mathbf{0} = t[\rho, i = \mathbf{0}]@\mathbf{0} = (t@\mathbf{0})[\rho, i = \mathbf{0}]$.

Param. We need to show that $(t(i\,\mathbf{0})!)\rho = (t!)[\rho, i = \mathbf{0}]$. By induction hypothesis $t(i\,\mathbf{0})\rho = t[\rho, i = \mathbf{0}]$, and therefore $(t(i\,\mathbf{0})!)\rho = t(i\,\mathbf{0})\rho! = t[\rho, i = \mathbf{0}]! = (t!)[\rho, i = \mathbf{0}]$. □

Theorem 10 (Convertible terms are semantically equal)**.**

- *If* $\Gamma \vdash A$ *and* $\Gamma \vdash A'$ *with* $A \equiv A'$*, then* $A\rho = A'\rho$ *for any* $\rho \in \Gamma(I)$*.*
- *If* $\Gamma \vdash a : A$ *and* $\Gamma \vdash a' : A$ *with* $a \equiv a'$*, then* $a\rho = a'\rho$ *for any* $\rho \in \Gamma(I)$*.*

Proof. By simultaneous induction on the conversion rules $A \equiv A'$ and $a \equiv a'$ (Definition 3).

β. We need to show that $((\lambda(x : A).t)\, u)\rho = t[x \mapsto u]\rho$. Seeing the interpretation of $\lambda(x : A).t$ relative to a function type as a family of dependent functions indexed by arbitrary maps, the element at index $1 : I \longrightarrow I$ gives a dependent function $((\lambda(x : A).t)\rho)_1$ mapping any $w \in A\rho$ to $t\langle\rho, x = w\rangle$. In particular, since $u\rho \in A\rho$ by Theorem 11, we deduce $((\lambda(x : A).t)\, u)\rho = ((\lambda(x : A).t)\rho)_1(u\rho) = t\langle\rho, x = u\rho\rangle$. We conclude that $((\lambda(x : A).t)\, u)\rho = t[x \mapsto u]\rho$ since we have $t\langle\rho, x = u\rho\rangle = t[x \mapsto u]\rho$ by Theorem 8.

η. We need to show that $t\rho = (\lambda(x : A).(t\, x))\rho$. Interpreting the term t relative to a function type yields a nested family indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$.

We show that $(\lambda(x : A).(t\, x))\rho_{\alpha h} = t\rho_{\alpha h}$ for each $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$. By definition $(\lambda(x : A).(t\, x))\rho_{\alpha h}$ is the dependent function $\lambda_{\alpha h}$ mapping any $u \in A\rho\alpha h$ to $(t\, x)\langle\rho\alpha h, x = u\rangle = (t\langle\rho\alpha h, x = u\rangle)_1(x\langle\rho\alpha h, x = u\rangle) = (t\rho)_{\alpha h}(u)$.

Col-β. There are two cases, depending on whether the color that is being applied is $\mathbf{0}$ or not. We need to show that $((\langle i\rangle t)@\mathbf{0})\rho = t(i\,\mathbf{0})\rho$, and that $((\langle i\rangle t)@j)[\rho, j = \varphi] = t(i\, j)[\rho, j = \varphi]$.

- By definition $((\langle i\rangle t)@\mathbf{0})\rho = (\langle i\rangle t)\rho@\mathbf{0} = \bar{u}^k@\mathbf{0}$, where $k = \mathsf{fresh}(I)$ and $u = t[\rho, i = k]$. Furthermore $\bar{u}^k@\mathbf{0} = t[\rho, i = k](k\,\mathbf{0}) = t[\rho, i = \mathbf{0}]$ and Theorem 9 gives $t[\rho, i = \mathbf{0}] = t(i\,\mathbf{0})\rho$. We therefore deduce $((\langle i\rangle t)@\mathbf{0})\rho = t(i\,\mathbf{0})\rho$.
- By definition $((\langle i\rangle t)@j)[\rho, j = \varphi] = (\langle i\rangle t)\rho@\varphi = \bar{u}^k@\varphi$, where $k = \mathsf{fresh}(I)$ and $u = t[\rho, i = k]$. Furthermore $\bar{u}^k@\varphi = t[\rho, i = k](k\,\varphi) = t[\rho, i = \varphi]$ and Theorem 9 gives $t[\rho, i = \varphi] = t(i\, j)[\rho, j = \varphi]$. We therefore deduce $((\langle i\rangle t)@j)[\rho, j = \varphi] = t(i\, j)[\rho, j = \varphi]$.

Col-η. We need to show that $t\rho = (\langle i\rangle(t@i))\rho$. By definition $(\langle i\rangle(t@i))\rho$ is the I-element $\bar{u}^j$ where $j = \mathsf{fresh}(I)$ and $u = (t@i)[\rho, i = j] = t\rho@j$, hence $(\langle i\rangle(t@i))\rho = \bar{u}^j = t\rho$.

Pair-Orig. We need to show that $(\!|a, p|\!)@\mathbf{0})\rho = a\rho$. By definition the I-element $u \stackrel{\text{def}}{=} (\!|a, p|\!)\rho$ is defined by $u_J = ((a\rho)_J, (p\rho)_J)$ for each $J \subseteq I$, and the I-element $v \stackrel{\text{def}}{=} ((\!|a, p|\!)@\mathbf{0})\rho = (\!|a, p|\!)\rho@\mathbf{0} = u@\mathbf{0}$ is defined by $v_J = (a\rho)_J$ for each $J \subseteq I$. Therefore $(\!|a, p|\!)@\mathbf{0})\rho = a\rho$.

Pair-Param. We need to show that $(\!|a, p|\!)!\,)\rho = p\rho$. By definition the I-element $u \stackrel{\text{def}}{=} (\!|a, p|\!)\rho$ is defined by $u_J = ((a\rho)_J, (p\rho)_J)$ for each $J \subseteq I$, and the I-element $v \stackrel{\text{def}}{=} (\!|a, p|\!)!\,)\rho = (\!|a, p|\!)\rho! = u!$ is defined by $v_J = (p\rho)_J$ for each $J \subseteq I$. Therefore $(\!|a, p|\!)!\,)\rho = p\rho$.

Surj-Param. We need to show that $(\!|a@\mathbf{0}, a!|\!)\rho = a\rho$. The I-element $u \stackrel{\text{def}}{=} (\!|a@\mathbf{0}, a!|\!)\rho$ is defined by $u_J = (((a@\mathbf{0})\rho)_J, ((a!\,)\rho)_J)$ for each $J \subseteq I$. Moreover, interpreting the term a relative to a color product $\forall i.A$ yields the I-element $a\rho = \bar{v}^j$, where $j = \mathsf{fresh}(I)$ and $v = a[\rho, i = j]$.

The I-elements $(a@\mathbf{0})\rho$ and $(a!\,)\rho$ are respectively defined by $((a@\mathbf{0})\rho)_J = ((a\rho@\mathbf{0}))_J = v_J$ and $(a!\,)\rho_J = (a\rho!\,)_J = v_{J,j}$. Since $u_J = (v_J, v_{J,j})$ for each $J \subseteq I$ we deduce that $(\!|a@\mathbf{0}, a!|\!)\rho = u = \bar{v}^j = a\rho$.

Surj-Type. We show $((\Psi_{T@\mathbf{0}}(\lambda(x : T@\mathbf{0}).\,(\forall i.T@i) \ni x))\rho)_{\alpha h} = (T!\,\rho)_{\alpha h}$. for each $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$. Let $A \stackrel{\text{def}}{=} T@\mathbf{0}$ and $P \stackrel{\text{def}}{=} \lambda(x : A).\,(\forall i.T@i) \ni x$.

For each $u \in A\rho_{\alpha h}$, $(((\forall i.T@i) \ni x)\langle\rho\alpha h, x = u\rangle)_1$ is by definition the K-set of K-elements $v!$ for $v \in ((\forall i.T@i)\langle\rho\alpha h, x = u\rangle)_1$ such that $v@\mathbf{0} = x\langle\rho\alpha h, x = u\rangle = u$; *i.e.*, the K-set of K-elements $\bar{w}^k!$ for $w \in (T@i)[\rho\alpha h, i = k]_1 = (T\rho\alpha h@k)_1$ such that $w(k\,\mathbf{0}) = u$, where $k = \mathsf{fresh}(K)$.

Moreover $((\Psi_A P)\rho)_{\alpha h}$ is the K-set of K-elements $v! \in ((P\rho)_{\alpha h}(u))_1$ for $u \in A\rho_{\alpha h} = (T\rho@\mathbf{0})_{\alpha h}$ such that $u = v@\mathbf{0}$; *i.e.*, the K-set of K-elements $\bar{w}^k!$ for $w \in (T\rho\alpha h@k)_1$ such that $w(k\,\mathbf{0}) \in (T\rho\alpha h@\mathbf{0})_1$. Now since $w(k\,\mathbf{0}) \in (T\rho\alpha h@\mathbf{0})_1$ for each $w \in (T\rho\alpha h@k)_1$, we deduce that $(\Psi_A P)\rho$ is merely the K-set of K-elements $\bar{w}^k!$ for $w \in (T\rho\alpha h@k)_1 = T[\rho\alpha h, i = k]_1$.

Finally, interpreting the term T relative to the color product $\forall i.\mathcal{U}$ yields the I-element $T\rho = \bar{\mathcal{A}}^j$, where $j = \mathsf{fresh}(I)$ and $\mathcal{A} = T[\rho, i = j] \in \mathcal{U}[\rho, i = j]$ is a nested family of $\mathscr{U}$-small sets indexed by projections $\alpha' : I, j \longrightarrow J$ and total maps $h' : J \rightarrowtail K$. It follows that $(T!\,\rho)_{\alpha h} = (T\rho!\,)_{\alpha h}$ is the $\mathscr{U}$-small set of K-elements $\bar{w}^k!$ for $w \in \mathscr{A}_{(\alpha h, j=k)} = T[\rho\alpha h, i = k]_1$. We therefore deduce that $(\Psi_A P)\rho_{\alpha h} = (T!\,)\rho_{\alpha h}$.

Surj-Fun. We need to show $(\Phi_{t@\mathbf{0}}(\lambda(x : \forall i.A).\,(\langle i\rangle(t@i)\,(x@i))!\,))\rho = t!\,\rho$. Let $t' \stackrel{\text{def}}{=} t@\mathbf{0}$ and $u' \stackrel{\text{def}}{=} \lambda(x : \forall i.A).\,(\langle i\rangle(t@i)\,(x@i))!$. Let $j = \mathsf{fresh}(I)$. By definition $(\Phi_{t'}u')\rho = \bar{\lambda}^j!$, where λ is a nested family of dependent functions indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$. Moreover interpreting the term t relative to a type of the form $\forall i.((x : A) \to B)$ yields an I-element $\bar{\lambda'}^j$, where λ' is a nested family of dependent functions indexed by $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$. We now show by case analysis on $\alpha(j)$ that $\lambda_{\alpha h} = \lambda'_{\alpha h}$ for each $\alpha : I, j \longrightarrow J$ and $h : J \rightarrowtail K$.

- If $\alpha(j) = \mathbf{0}$, we have $\lambda_{\alpha h} = (t'\rho)_{\alpha h - j} = t\rho_{\alpha h - j}@\mathbf{0} = \lambda'_{\alpha h}$.
- If $h(\alpha(j)) = k \in K$, by definition $\lambda_{\alpha h}$ is the dependent function mapping each $v \in A[\rho(\alpha h - j), i = k]$ to $w(l\,k)$, where $l = \mathsf{fresh}(K\backslash\{k\})$, $(t'\rho)_{\alpha h - j}(\bar{v}^k@\mathbf{0}) = \bar{w}^l@\mathbf{0}$ and $(u'\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l!$. Moreover

$$\begin{aligned}
(t'\rho)_{\alpha h - j}(\bar{v}^k@\mathbf{0}) &= ((t\rho)_{\alpha h - j}@\mathbf{0})(\bar{v}^k@\mathbf{0}) \\
&= ((t\rho_{\alpha h - j}@l)(\bar{v}^k@l))@\mathbf{0}\text{; and} \\
(u'\rho)_{\alpha h - j}(\bar{v}^k) &= (\langle i\rangle(t@i)\,(x@i))!\,\langle\rho(\alpha h - j), x = \bar{v}^k\rangle \\
&= ((t@i)\,(x@i))[\langle\rho(\alpha h - j), x = \bar{v}^k\rangle, i = l]! \\
&= ((t\rho_{\alpha h - j}@l)(\bar{v}^k@l))!\,.
\end{aligned}$$

  Therefore $\bar{w}^k = (t\rho_{\alpha h - j}@k)(v)$ and we deduce that $\lambda_{\alpha h}(v) = \lambda'_{\alpha h}(v)$.

Pair-Pred. We need to prove that $((\forall i.A \bowtie_i P) \ni a)\rho = (P\,a)\rho$. We show that $((\forall i.A \bowtie_i P) \ni a)\rho_{\alpha h} = (P\,a)\rho_{\alpha h}$ for each $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$.

Let $k = \mathsf{fresh}(K)$. By definition $(\forall i.A \bowtie_i P)\rho_{\alpha h} = (\forall i.(\!|A, \Psi_A P|\!)@i)\rho_{\alpha h}$ is the $\mathscr{U}$-small set of K-elements $\bar{u}^k$ for $u \in (\!|A, \Psi_A P|\!)@i)[\rho\alpha h, i = k] = (\!|A, \Psi_A P|\!)\rho\alpha h@k$, *i.e.*, such that $(u_L)_{L\subseteq K} = \bar{u}^k@\mathbf{0} \in A\rho_{\alpha h}$ and $(u_{L,k})_{L\subseteq K} = \bar{u}^k! \in (\Psi_A P)\rho_{\alpha h}$.

Furthermore $((\forall i.A \bowtie_i P) \ni a)\rho_{\alpha h}$ is the $\mathscr{U}$-small set of K-elements $v!$ for $v \in (\forall i.A \bowtie_i P)\rho_{\alpha h}$ such that $v@\mathbf{0} = a\rho\alpha h$; *i.e.*, the $\mathscr{U}$-small set of K-elements $\bar{u}^k! \in (\Psi_A P)\rho_{\alpha h}$ such that $\bar{u}^k@\mathbf{0} = a\rho\alpha h \in A\rho_{\alpha h}$; *i.e.*, the $\mathscr{U}$-small K-set $(P\rho)_{\alpha h}(a\rho\alpha h) = (P\,a)\rho_{\alpha h}$.

Pair-App. We show $(\langle\!| t \,,_i u |\!\rangle\, (a@i))[\rho, i = \varphi] = (\!|t\,(a@\mathbf{0})\,,_i u\,a|\!)[\rho, i = \varphi]$. By definition we get $(\langle\!| t \,,_i u |\!\rangle\, (a@i))[\rho, i = \varphi] = (\langle\!| t, u |\!\rangle\rho@\varphi)_1(a\rho@\varphi)$ and $(\!|t\,(a@\mathbf{0})\,,_i u\,a|\!)[\rho, i = \varphi] = (\!|t\,(a@\mathbf{0}), u\,a|\!)\rho@\varphi$. We now proceed by case analysis on φ.

- For $\varphi = \mathbf{0}$, we have on the one hand

  $$(\langle\!| t \,,_i u |\!\rangle\, (a@i))[\rho, i = \mathbf{0}] = (\langle\!| t, u |\!\rangle\rho@\mathbf{0})_1(a\rho@\mathbf{0}) = (t\rho)_1(a\rho@\mathbf{0}),$$

  and on the other hand

  $$(\!|t\,(a@\mathbf{0})\,,_i u\,a|\!)[\rho, i = \mathbf{0}] = (t\,(a@\mathbf{0}))\rho = (t\rho)_1(a\rho@\mathbf{0}).$$

- For $\varphi = j \in \mathbb{I}$, the (I, j)-element $(\langle\!| t, u |\!\rangle\rho@j)_1$ is the dependent function mapping $v \in A[\rho, i = j]$ to $w(k\,j)$, where $k = \mathsf{fresh}(I\backslash\{j\})$, $(t\rho)_1(\bar{v}^j@\mathbf{0}) = \bar{w}^k@\mathbf{0}$ and $(u\rho)_1(\bar{v}^j) = \bar{w}^k!$. In particular, for $v = a\rho@j \in A[\rho, i = j]$ we get $\bar{v}^j = a\rho$, $\bar{w}^k@\mathbf{0} = (t\rho)_1(a\rho@\mathbf{0}) = (t\,(a@\mathbf{0}))\rho$ and $\bar{w}^k! = (u\rho)_1(a\rho) = (u\,a)\rho$. Therefore $\bar{w}^k = (\!|t\,(a@\mathbf{0}), u\,a|\!)\rho$ and $(\langle\!| t, u |\!\rangle\rho@j)_1(a\rho@j) = (\!|t\,(a@\mathbf{0}), u\,a|\!)\rho@j$. □

We proved the presheaf laws in the definition of the interpretation functions. We now prove the remaining property, namely totality on valid judgments.

Theorem 11 (Validity)**.**

- *If* $\Gamma \vdash$ *then* Γ *is a refined presheaf (Definition 15);*
- *if* $\Gamma \vdash A$ *then* $A\rho$ *is an* I*-set for each* $\rho \in \Gamma(I)$*;*
- *if* $\Gamma \vdash a : A$ *then* $a\rho \in A\rho$ *for each* $\rho \in \Gamma(I)$*.*

Proof. By simultaneous induction on the typing judgments $\Gamma \vdash$, $\Gamma \vdash A$ and $\Gamma \vdash a : A$ (Definition 2). For most introduction rules, the conclusion directly follows from the definition of the relevant interpretation function.

Empty. Direct by definition.

Ext. By induction hypotheses Γ is a refined presheaf, and $A\rho$ is an I-set for each $\rho \in \Gamma$. Hence by definition $(\Gamma, x : A)$ is a refined presheaf.

ColExt. By induction hypothesis Γ is a refined presheaf, hence so is (Γ, i) by definition.

Universe. Direct by definition.

Prod. Let $\alpha : I \longrightarrow J$, $h : J \rightarrowtail K$. Since $\Gamma \vdash A$, we deduce by induction hypothesis that $A\rho$ is an I-set. Therefore $(\Gamma, x : A)$ is a refined presheaf and $\langle \rho\alpha h, x = u\rangle \in (\Gamma, x : A)(I)$ for each K-element $u \in A\rho\alpha h$. Now since $\Gamma, x : A \vdash B$, we deduce by induction hypothesis that $B\langle \rho\alpha h, x = u\rangle$ is a K-set for each K-element $u \in A\rho\alpha h$. Hence $\lambda_{\alpha h}$ from the definition is a well-typed function mapping each K-element $u \in A\rho\alpha h$ to a K-element of $B\langle \rho\alpha h, x = u\rangle$. Moreover the family λ is trivially an I-set.

Small. Since $\Gamma \vdash A : \mathcal{U}$, by induction hypothesis $\mathcal{U}\rho$ is an I-set and $A\rho \in \mathcal{U}\rho$; hence by definition $A\rho$ is a nested family $\mathcal{A}$ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, such that each $\mathcal{A}_{\alpha h}$ is an $\mathscr{U}$-small K-set. Hence taking the interpretation of the small type A turned into a proper type yields the I-set $A\rho_{11}$.

ColPi. Since $\Gamma, i \vdash A$ we get by induction hypothesis that Γ is a refined presheaf. Therefore $[\rho, i = j] \in \Gamma(I, j)$ and $A[\rho, i = j]$ is an (I, j)-set, where $j = \mathsf{fresh}(I)$. Hence $(\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$ is an I-set.

Out. By induction hypothesis we get that $(\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$ is an I-set, where $j = \mathsf{fresh}(I)$. Furthermore Theorem 9 gives that $A(i\,\mathbf{0})\rho = A[\rho, i = \mathbf{0}]$ is an I-set, and we deduce that $a\rho \in A[\rho, i = \mathbf{0}]$ is an I-element. Hence $((\forall i.A) \ni a)\rho = \{\bar{u}^j! \mid u \in A[\rho, i = j], u(j\,\mathbf{0}) = a\rho\}$ is a well-formed I-set.

Conv. Direct consequence of the induction hypothesis and Theorem 10.

Var. Direct by definition.

Abs. We need to show that $(\lambda(x : A).t)\rho \in ((x : A) \to B)\rho$. By definition $(\lambda(x : A).t)\rho$ is the nested family λ indexed by $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$, where each $\lambda_{\alpha h}$ is the dependent function mapping any $u \in A\rho\alpha h$ to $t\langle \rho\alpha h, x = u\rangle$. Moreover since $\Gamma, x : A \vdash t : B$, we deduce by induction hypothesis that $\Gamma, x : A$ is a refined presheaf, $\langle \rho\alpha h, x = u\rangle \in (\Gamma, x : A)(I)$, and $\lambda_{\alpha h}(u) = t\langle \rho\alpha h, x = u\rangle \in B\langle \rho\alpha h, x = u\rangle$. Therefore $(\lambda(x : A).t)\rho \in ((x : A) \to B)\rho$.

App. We need to show that $(t\,u)\rho \in B[x \mapsto u]\rho$. By definition $(t\,u)\rho = (t\rho)_1(u\rho)$, and by Theorem 8 $B[x \mapsto u]\rho = B\langle \rho, x = u\rho\rangle$. Moreover since $\Gamma \vdash u : A$, we deduce by induction hypothesis that $u\rho \in A\rho$.

Furthermore since $\Gamma \vdash t : (x : A) \to B$, we deduce by induction hypothesis that $(t\rho)_1$ is the dependent function λ_1 mapping any $v \in A\rho$ to an I-element in $B\langle\rho, x = v\rangle$; in particular, for $v = u\rho$ we obtain that $(t\rho)_1(u\rho) \in B\langle\rho, x = u\rho\rangle$.

Prod-$\mathcal{U}$. Direct by definition of the interpretation of the small dependent function space.

Out-$\mathcal{U}$. We need to show that $((\forall i.A) \ni a)\rho \in \mathcal{U}\rho$. Since $\Gamma \vdash \forall i.A : \mathcal{U}$, we deduce by induction hypothesis that $(\forall i.A)\rho$ is an I-element of the I-set $\mathcal{U}\rho$. Let $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$. By definition $(\forall i.A)\rho_{\alpha h}$ is the K-set of K-elements $\bar{u}^k$ such that $u \in A[\rho\alpha h, i = k]_1$, where $k = \mathsf{fresh}(K)$. Now since $\Gamma \vdash a : A(i\,\mathbf{0})$, we deduce by induction hypothesis and Theorem 9 that $a\rho \in A(i\,\mathbf{0})\rho_{\alpha h} = A[\rho\alpha h, i = \mathbf{0}]_1$.

By definition $((\forall i.A) \ni a)\rho_{\alpha h}$ is the $\mathscr{U}$-small K-set of K-elements $u!$ for $u \in (\forall i.A)\rho_{\alpha h}$ such that $u@\mathbf{0} = a\rho\alpha h$; *i.e.*, the $\mathscr{U}$-small K-set of K-elements $\bar{v}^k!$ for $v \in A[\rho\alpha h, i = k]_1$ such that $v(k\,\mathbf{0}) = a\rho\alpha h$.

ColAbs. Since $\Gamma, i \vdash t : A$ we get by induction hypothesis that Γ is a refined presheaf; therefore $[\rho, i = j] \in \Gamma(I, j)$ and we obtain an (I, j)-element $u \stackrel{\text{def}}{=} t[\rho, i = j] \in A[\rho, i = j]$, where $j = \mathsf{fresh}(I)$. Hence $\bar{u}^j \in (\forall i.A)\rho = \{\bar{v}^j \mid v \in A[\rho, i = j]\}$.

In-Abs. Since $\Gamma \vdash a : A(i\,\mathbf{0})$, we deduce by induction hypothesis and Theorem 9 that $a\rho \in A(i\,\mathbf{0})\rho = A[\rho, i = \mathbf{0}]$. Furthermore since $\Gamma \vdash p : (\forall i.A) \ni a$, we deduce that there exists $u \in A[\rho, i = j]$ (where $j = \mathsf{fresh}(I)$) such that $p\rho = \bar{u}^j!$ and $u(j\,\mathbf{0}) = a\rho$.

Now, by definition the tuple $(\!|a, p|\!)\rho$ is defined by $(\!|a, p|\!)\rho_J = (a\rho_J, p\rho_J) = (u_J, u_{J,j})$ for each $J \subseteq I$. Thus $(\!|a, p|\!)\rho = \bar{u}^j \in (\forall i.A)\rho = \{\bar{v}^j \mid v \in A[\rho, i = j]\}$.

ColPi-$\mathcal{U}$. Like for ColPi, direct by definition of the interpretation of the small color product.

In-Pred. We need to show that $(\Psi_A P)\rho \in ((\forall i.\mathcal{U}) \ni A)\rho$ Let $\alpha : I \longrightarrow J$ and $h : J \rightarrowtail K$. Since $\Gamma \vdash A : \mathcal{U}$ we deduce by induction hypothesis that $A\rho \in \mathcal{U}\rho$; hence $A\rho_{\alpha h}$ is a K-set. Furthermore since $\Gamma \vdash P : A \to \mathcal{U}$ we deduce by induction hypothesis that $P\rho \in (A \to \mathcal{U})\rho$; hence we obtain a set-theoretic function $P\rho_{\alpha h} : A\rho_{\alpha h} \to \mathcal{U}\rho\alpha h$.

Hence for each K-element $u \in A\rho_{\alpha h}$, we obtain a K-set $((P\rho)_{\alpha h}(u))_1$ and we can define the $\mathscr{U}$-small K-set of K-elements $v! \in ((P\rho)_{\alpha h}(u))_1$ for $u \in A\rho_{\alpha h}$ such that $u = v@\mathbf{0}$.

In-Fun. We need to show that $(\Phi_t u)\rho \in ((\forall i.(A \to B)) \ni t)\rho$. Let $j = \mathsf{fresh}(I)$. We have $(\Phi_t u)\rho = \bar{\lambda}^j!$, so we have to show that $\lambda \in (\forall i.((x : A) \to B))\rho = ((x : A) \to B)[\rho, i = j]$, and that $\bar{\lambda}^j@\mathbf{0} = t\rho$.

Let $\alpha : I, j \longrightarrow J$, $h : J \rightarrowtail K$. We need to show that $\lambda_{\alpha h}(v) \in B\langle[\rho, i = j]\alpha h, x = v\rangle$ for each $v \in A[\rho, i = j]\alpha h$. We proceed by case analysis on $\alpha(j)$:

- For $\alpha(j) = \mathbf{0}$, we have $\lambda_{\alpha h} = (t\rho)_{\alpha h - j}$. Since we get by Theorem 9 that $A[\rho(\alpha h - j), i = \mathbf{0}] = A(i\,\mathbf{0})\rho(\alpha h - j)$, we obtain by induction hypothesis that

$$\lambda_{\alpha h}(v) = (t\rho)_{\alpha h - j}(v) \in B(i\,\mathbf{0})\langle\rho(\alpha h - j), x = v\rangle = B\langle[\rho(\alpha h - j), i = \mathbf{0}], x = v\rangle$$

  for each $v \in A[\rho(\alpha h - j), i = \mathbf{0}]$.
- For $h(\alpha(j)) = k \in K$, $\lambda_{\alpha h}$ is the dependent function mapping each $v \in A[\rho(\alpha h - j), i = k]$ to $w(l\,k)$, where $(t\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0}) = \bar{w}^l @ \mathbf{0}$, $(u\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l !$, and $l = \mathsf{fresh}(K \backslash \{k\})$.
  Let $v \in A[\rho(\alpha h - j), i = k]$. As before by Theorem 9 we have $\bar{v}^k @ \mathbf{0} \in A[\rho(\alpha h - j), i = \mathbf{0}]$, and $(t\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0}) \in B(i\,\mathbf{0})\langle\rho(\alpha h - j), x = v\rangle = B\langle[\rho(\alpha h - j), i = \mathbf{0}], x = v\rangle$. Furthermore $\bar{v}^k \in (\forall i.A)\rho(\alpha h - j)$, hence by induction hypothesis

$$(u\rho)_{\alpha h - j}(\bar{v}^k) \in ((\forall i.B[x \mapsto x@i]) \ni t\,(x@\mathbf{0}))\langle\rho(\alpha h - j), x = \bar{v}^k\rangle$$

  Thus by definition there exists a $(K\backslash\{k\}, l)$-element $w \in B[x \mapsto x@i][\langle\rho(\alpha h - j), x = \bar{v}^k\rangle, i = l]$ such that $(u\rho)_{\alpha h - j}(\bar{v}^k) = \bar{w}^l !$ and $\bar{w}^l @ \mathbf{0} = (t\,(x@\mathbf{0}))\langle\rho(\alpha h - j), x = \bar{v}^k\rangle = (t\rho)_{\alpha h - j}(\bar{v}^k @ \mathbf{0})$
  Furthermore by Theorem 8 $B[x \mapsto x@i][\langle\rho(\alpha h - j), x = \bar{v}^k\rangle, i = l] = B\langle[\rho(\alpha h - j), i = l], x = \bar{v}^k @ l\rangle$, hence we conclude that $\lambda_{\alpha h}(v) = w(l\,k) \in B\langle[\rho(\alpha h - j), i = k], x = \bar{v}^k @ k\rangle = B\langle[\rho, i = j]\alpha h, x = v\rangle$.

ColApp. For simplicity we only consider the case where $\vec{j}$ is empty. (The reasoning easily generalizes to non-empty $\vec{j}$ since the presheaves $\Gamma, i, \vec{j}$ and $\Gamma, \vec{j}, i$ are equal.)

We need to prove that $(t@i)[\rho, i = \varphi] = t\rho@\varphi \in A[\rho, i = \varphi]$. Since $\Gamma \vdash t : \forall i.A$, we deduce by induction hypothesis that $t\rho \in (\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$, where $j = \mathsf{fresh}(I)$. Hence there exists an (I, j)-element $u \in A[\rho, i = j]$ such that $t\rho = \bar{u}^j$. Therefore $t\rho@\varphi \in A[\rho, i = j](j\,\varphi) = A[\rho, i = \varphi]$ by Theorem 9.

ColApp-Orig. We need to prove that $(t@\mathbf{0})\rho = t\rho@\mathbf{0} \in A\rho$. Since $\Gamma \vdash t : \forall i.A$, we deduce by induction hypothesis that $t\rho \in (\forall i.A)\rho = \{\bar{u}^j \mid u \in A[\rho, i = j]\}$, where $j = \mathsf{fresh}(I)$. Hence there exists an (I, j)-element $u \in A[\rho, i = j]$ such that $t\rho = \bar{u}^j$. Therefore $t\rho@\mathbf{0} \in A[\rho, i = j](j\,\mathbf{0}) = A[\rho, i = \mathbf{0}]$ by Theorem 9.

Param. We need to prove that $t!\,\rho = t\rho! \in ((\forall i.A) \ni t@\mathbf{0})\rho$. Since $\Gamma \vdash t : \forall i.A$, we deduce by induction hypothesis that $t\rho \in (\forall i.A)\rho = \{\bar{u}^j \mid$

$u \in A[\rho, i = j]\}$, where $j = \mathsf{fresh}(I)$. Hence there exists an (I, j)-element $u \in A[\rho, i = j]$ such that $t\rho = \bar{u}^j$. We conclude that $t\rho! \in ((\forall i.A) \ni t@\mathbf{0})\rho$ since $\bar{u}^j@\mathbf{0} = (t@\mathbf{0})\rho$. □

6 Related Work

Our own line of work. This work continues a line of work aiming at a smooth integration of parametricity with dependent types [Bernardy et al., 2010, Bernardy and Lasson, 2011, Bernardy et al., 2012, Bernardy and Moulin, 2012, 2013]. (The last two papers are respectively expanded as chapters 1 and 2 of this thesis.) The present work offers two improvements over previous publications: 1. denotational semantics; and 2. a much simplified syntax, suitable as the basis of a proof assistant.

The simplification of syntax is made possible by not requiring the preservation of functions by parametricity, where by "preservation of functions by parametricity" we mean the property that if f is a function, then the canonical proof that f is parametric (denoted $(\langle i \rangle f)!$ here) is also a function. To our knowledge, 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 β-reduction, as observed by Bernardy and Moulin [2012]. In the present system, the parametric interpretation of functions is instead merely *isomorphic* to a function, thanks to the In-Fun rule (Theorem 4). 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 dependent type theory [Krishnaswami and Dreyer, 2013, Atkey et al., 2014], 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 characteristic feature of proposals for parametricity is the kind of model underlying the semantics. Krishnaswami and Dreyer [2013] propose a model based on quasi-PERs.

Atkey et al. [2014] propose a model based on reflexive graphs. The model that we use is based on cubes (functions from finite subsets of colors). In Bernardy and Moulin [2012] 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. [2013], Pitts [2014], and Cohen et al. [2015] to model univalence in type theory. Not only do these models use a presheaf, but they also use a category closely related to the underlying category $\mathbf{pI}$. 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 [2014] give a syntax for Bezem et al.'s cubical set model, effectively modelling univalence by internalization of their model. The present work further refines the model by interpreting terms as I-elements, which we believe is essential to interpret our special-purpose pairing constructions.

7 Future work and conclusion

We have defined a new type theory, closely related to the cubical type theory of Cohen et al. [2015], with internalized parametricity. This work attempts to better integrate parametricity by simplifying the syntax of our previous type theories with internalized parametricity [Bernardy and Moulin, 2012, 2013].

Unlike our previous work, the system presented here does not compute parametricity types, hence does not not have preservation of functions by parametricity. As shown in section 4, this does not appear to be an issue in practice.

Furthermore, thanks to our model construction, we have proved the consistency of the system. Future work would involve proving decidability of conversion and type-checking. The type-checker could then be used as a minimal proof assistant for a type theory with parametricity.

Bibliography

M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, pages 157–170. ACM, 1993. ISBN 0-89791-560-7.

M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 147–160. ACM, 1999. ISBN 1-58113-095-3.

A. Abel and G. Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1):1–36, 2012. TYPES'10 special issue.

T. Altenkirch and A. Kaposi. A syntax for cubical type theory. aug 2014. Draft.

R. Atkey, S. Lindley, and J. Yallop. Unembedding domain-specific languages. In *Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell*, pages 37–48. ACM, 2009.

R. Atkey, N. Ghani, and P. Johann. A relationally parametric model of dependent type theory. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 503–515. ACM, 2014. ISBN 978-1-4503-2544-8.

H. P. Barendregt. Handbook of logic in computer science. volume 2, chapter Lambda Calculi with Types, pages 117–309. Oxford University Press, Inc., 1992. ISBN 0-19-853761-1.

J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *FoSSaCS*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.

J.-P. Bernardy and G. Moulin. Towards a computational interpretation of parametricity. Submitted to PoPL, 2011.

J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science*, pages 135–144, 2012.

J.-P. Bernardy and G. Moulin. Type theory in color. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 61–72. ACM, 2013.

J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 345–356. ACM, 2010. ISBN 978-1-60558-794-3.

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

J.-P. Bernardy, T. Coquand, and G. Moulin. A presheaf model of parametric type theory. volume 319, pages 67–82, 2015. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

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*, pages 107–128, 2013.

A. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ICFP '08, pages 143–156. ACM, 2008.

C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom, 2015. Preprint.

T. Coquand and G. Huet. The calculus of constructions. *Information and Computation – Semantics of Data Types*, 76(2-3):95–120, 1988. ISSN 0890-5401.

A. Gill, J. Launchbury, and S. L. Peyton Jones. A short cut to deforestation. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, FPCA '93, pages 223–232. ACM, 1993. ISBN 0-89791-595-X.

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

P. Johann. A generalization of short-cut fusion and its correctness proof. *Higher-Order and Symbolic Computation*, 15(4):273–300, 2002.

C. Keller and M. Lasson. Parametricity in an impredicative sort. In P. Cégielski and A. Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 381–395. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012. ISBN 978-3-939897-42-2.

H.-S. Ko and J. Gibbons. Modularising inductive families. In *Proceedings of the Seventh ACM SIGPLAN Workshop on Generic Programming*, WGP '11, pages 13–24. ACM, 2011. ISBN 978-1-4503-0861-8.

N. R. Krishnaswami and D. Dreyer. Internalizing relational parametricity in the extensional calculus of constructions. In S. R. D. Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 432–451. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013. ISBN 978-3-939897-60-6.

D. R. Licata and R. Harper. Canonicity for 2-dimensional type theory. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 337–348. ACM, 2012. ISBN 978-1-4503-1083-3.

H. G. Mairson. Outline of a proof theory of parametricity. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 313–327. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-54396-1.

P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

N. Mishra-Linger and T. Sheard. Erasure and polymorphism in pure type systems. In *Proceedings of the Theory and Practice of Software, 11th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'08/ETAPS'08, pages 350–364. Springer-Verlag, 2008. ISBN 3-540-78497-7, 978-3-540-78497-5.

G. Moulin. *Pure Type Systems with an Internalized Parametricity Theorem*. Licentiate thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, 2013.

B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's type theory*, volume 200. Oxford University Press, 1990.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, 2007.

C. Paulin-Mohring. Extracting F_ω's programs from proofs in the calculus of constructions. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 89–104. ACM, 1989. ISBN 0-89791-294-2.

F. Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, LICS '01, pages 221–230. IEEE Computer Society, 2001.

F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In *MFPS*, volume 442 of *LNCS*, pages 209–228. Springer, 1990.

A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013. ISBN 1107017785, 9781107017788.

A. M. Pitts. An equivalent presentation of the Bezem-Coquand-Huber category of cubical sets. *CoRR*, abs/1401.7807, 2014.

G. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-56517-8.

N. Pouillard. Nameless, painless. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 320–332. ACM, 2011.

J. C. Reynolds. Types, abstraction and parametric polymorphism. pages 513–523, 1983.

V. Siles. *Investigation on the typing of equality in type systems*. PhD thesis, École Polytechnique, 2010.

J. Svenningsson. *Scalable Program Analysis*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, 2007.

The Coq development team. The Coq proof assistant, 2016.

P. Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional Programming languages and Computer Architecture*, pages 347–359, 1989.

P. Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, ICFP '03, pages 189–201. ACM, 2003. ISBN 1-58113-756-7.

P. Wadler. The Girard–Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1–3):201–226, 2007. ISSN 0304-3975.

P. Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 273–286. ACM, 2012. ISBN 978-1-4503-1054-3.

P. Évariste Dagand and C. McBride. Transporting functions across ornaments. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 103–114. ACM, 2012. ISBN 978-1-4503-1054-3.

Appendix A

Additional proofs for chapter 1

Lemma 1. *For each term A and each variable z not free in A, we have:*

i) $\llbracket A \rrbracket_{\xi, z \mapsto (z_0, z_1)} = \llbracket A \rrbracket_{\xi}$; *and*

ii) $\{a\}_{\xi, z \mapsto (z_0, z_1)} \in \llbracket A \rrbracket_{\xi, z \mapsto (z_0, z_1)} = \{a\}_{\xi} \in \llbracket A \rrbracket_{\xi}$ *for all terms a.*

Proof. By simultaneous induction on the structure of the raw term A. Following the definition of our relational interpretation, we prove only i) for the case of variable, lambda, relation introduction and application; we prove ii) in the other cases, namely product, sort, and relation elimination.

Variable $\ulcorner x_i \urcorner^n \dagger^\pi$

$$\begin{aligned} \llbracket \ulcorner x_i \urcorner^n \dagger^\pi \rrbracket_{\xi, z} &= \llbracket \ulcorner x_i \urcorner^n \rrbracket_{\xi, z} \ddagger^{\pi+1} \\ &= \ulcorner x_i \urcorner^{1+n} \ddagger^{\pi+1} \\ &= \llbracket \ulcorner x_i \urcorner^n \rrbracket_{\xi} \ddagger^{\pi+1} \\ &= \llbracket \ulcorner x_i \urcorner^n \dagger^\pi \rrbracket_{\xi} \end{aligned}$$

Lambda $\lambda \bar{x} : \bar{A}. B$

$$\begin{aligned} \llbracket \lambda \bar{x} : \bar{A}. B \rrbracket_{\xi, z} &= \lambda \bar{x} : \llbracket \bar{A} \rrbracket_{\xi, z}. \llbracket B \rrbracket_{\xi, z, x} \\ &= \lambda \bar{x} : \llbracket \bar{A} \rrbracket_{\xi}. \llbracket B \rrbracket_{\xi, x} && \text{by IH} \\ &= \llbracket \lambda \bar{x} : \bar{A}. B \rrbracket_{\xi} \end{aligned}$$

Lambda• $\lambda^\bullet \check{x} : \check{A}.\, B$

$$\begin{aligned}
[\![\lambda^\bullet \check{x} : \check{A}.\, B]\!]_{\xi,z} &= \lambda^\bullet \check{x} : [\![\check{A}]\!]_{\xi,z} . [\![B]\!]_{\xi,z,x} \\
&= \lambda^\bullet \check{x} : [\![\check{A}]\!]_{\xi} . [\![B]\!]_{\xi,x} && \text{by IH} \\
&= [\![\lambda^\bullet \check{x} : \check{A}.\, B]\!]_{\xi}
\end{aligned}$$

Application $F\,\bar{a}$

$$\begin{aligned}
[\![F\,\bar{a}]\!]_{\xi,z} &= [\![F]\!]_{\xi,z}\, [\![\bar{a}]\!]_{\xi,z} \\
&= [\![F]\!]_{\xi}\, [\![\bar{a}]\!]_{\xi} && \text{by IH} \\
&= [\![F\,\bar{a}]\!]_{\xi}
\end{aligned}$$

Sort s^n

$$\begin{aligned}
\{a\}_{\xi,z} \in [\![s^n]\!]_{\xi,z} &= \begin{pmatrix} \{a\}_{\xi,z} \\ \cdot \end{pmatrix} \stackrel{\bullet}{\to} s^{1+n} \\
&= \{a\}_{\xi} \in [\![s^n]\!]_{\xi}
\end{aligned}$$

Product $\forall \bar{x} : \bar{A}.\, B$

$$\begin{aligned}
\{a\}_{\xi,z} \in [\![\forall \bar{x} : \bar{A}.\, B]\!]_{\xi,z} &= \forall \bar{x} : [\![\bar{A}]\!]_{\xi,z} . (\{a\}_{\xi,z}\,(\bar{x}/01...1)) \in [\![B]\!]_{\xi,z,x} \\
&= \forall \bar{x} : [\![\bar{A}]\!]_{\xi} . (\{a\}_{\xi}\,(\bar{x}/01...1)) \in [\![B]\!]_{\xi,x} && \text{by IH} \\
&= \{a\}_{\xi} \in [\![\forall \bar{x} : \bar{A}.\, B]\!]_{\xi}
\end{aligned}$$

Arrow• $\check{A} \stackrel{\bullet}{\to} s^n$

$$\begin{aligned}
\{a\}_{\xi,z} \in [\![\check{A} \stackrel{\bullet}{\to} s^n]\!]_{\xi,z} &= ([\![\check{A}]\!]_{\xi,z} \oplus \{a\}_{\xi,z}) \stackrel{\bullet}{\to} s^{1+n} \\
&= ([\![\check{A}]\!]_{\xi} \oplus \{a\}_{\xi}) \stackrel{\bullet}{\to} s^{1+n} && \text{by IH} \\
&= \{a\}_{\xi} \in [\![\check{A} \stackrel{\bullet}{\to} s^n]\!]_{\xi}
\end{aligned}$$

Application• $F \bullet \check{b}$

$$\begin{aligned}
\{a\}_{\xi,z} \in [\![F \bullet \check{b}]\!]_{\xi,z} &= [\![F]\!]_{\xi,z} \bullet ([\![\check{b}]\!]_{\xi} \oplus \{a\}_{\xi,z}) \\
&= [\![F]\!]_{\xi} \bullet ([\![\check{b}]\!]_{\xi} \oplus \{a\}_{\xi}) && \text{by IH} \\
&= \{a\}_{\xi} \in [\![F \bullet \check{b}]\!]_{\xi}
\end{aligned}$$

□

Lemma 2. *For each term $A : s^m$ and each ρ of dimension at most m, we have:*

$$[\![A \ddagger_\zeta^\rho]\!]_{\xi} = [\![A]\!]_{\xi} \ddagger_\zeta^{1+\rho}$$

Proof. By induction on the structure of raw term A.

Variable (1) $\llbracket x_i \rrbracket^n \dagger^\pi,\ x \in \xi \cap \zeta$

$$\begin{aligned}
\llbracket \llbracket x_i \rrbracket^n \dagger^\pi \ddagger^\rho_\zeta \rrbracket_\xi &= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \\
&= \llbracket \llbracket x_i \rrbracket^n \rrbracket_\xi \ddagger^{\pi+1} \\
&= \llbracket x_{1i} \rrbracket^n \dagger^{\mathrm{normal}_n((0\ldots n)\circ\pi+1)} \\
&= \llbracket x_{1i} \rrbracket^n \dagger^{\mathrm{normal}_n((0\ldots n)\circ\pi+1)} \ddagger^{\rho+1}_\zeta \\
&= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \ddagger^{\rho+1}_\zeta
\end{aligned}$$

Variable (2) $\llbracket x_i \rrbracket^n \dagger^\pi,\ x \notin \xi,\ x \in \zeta$

$$\begin{aligned}
\llbracket \llbracket x_i \rrbracket^n \dagger^\pi \ddagger^\rho_\zeta \rrbracket_\xi &= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \\
&= \llbracket \llbracket x_i \rrbracket^n \rrbracket_\xi \ddagger^{\pi+1} \\
&= \llbracket x_{0i} \rrbracket^n \dagger^{\mathrm{normal}_n((0\ldots n)\circ\pi+1)} \\
&= \llbracket x_{0i} \rrbracket^n \dagger^{\mathrm{normal}_n((0\ldots n)\circ\pi+1)} \ddagger^{\rho+1}_\zeta \\
&= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \ddagger^{\rho+1}_\zeta
\end{aligned}$$

Variable (3) $\llbracket x_i \rrbracket^n \dagger^\pi,\ x \in \xi,\ x \notin \zeta$

$$\begin{aligned}
\llbracket \llbracket x_i \rrbracket^n \dagger^\pi \ddagger^\rho_\zeta \rrbracket_\xi &= \llbracket \llbracket x_i \rrbracket^n \dagger^{\mathrm{normal}_n(\rho\circ\pi)} \rrbracket_\xi \\
&= \llbracket x_{1i} \rrbracket^n \dagger^{(0\ldots n)} \dagger^{1+\mathrm{normal}_n(\rho\circ\pi)} \\
&= \llbracket x_{1i} \rrbracket^n \dagger^{(0\ldots n)} \dagger^{1+\pi} \ddagger^{1+\rho}_\zeta \\
&= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \ddagger^{1+\rho}_\zeta
\end{aligned}$$

Variable (4) $\llbracket x_i \rrbracket^n \dagger^\pi,\ x \notin \xi,\ x \notin \zeta$

$$\begin{aligned}
\llbracket \llbracket x_i \rrbracket^n \dagger^\pi \ddagger^\rho_\zeta \rrbracket_\xi &= \llbracket \llbracket x_i \rrbracket^n \dagger^{\mathrm{normal}_n(\rho\circ\pi)} \rrbracket_\xi \\
&= \llbracket x_{0i} \rrbracket^n \dagger^{(0\ldots n)} \dagger^{1+\mathrm{normal}_n(\rho\circ\pi)} \\
&= \llbracket x_{0i} \rrbracket^n \dagger^{(0\ldots n)} \dagger^{1+\pi} \ddagger^{1+\rho}_\zeta \\
&= \llbracket \llbracket x_i \rrbracket^n \dagger^\pi \rrbracket_\xi \ddagger^{1+\rho}_\zeta
\end{aligned}$$

Lambda $\lambda\bar{x} : \bar{A}.\, B$

$$\begin{aligned}
\llbracket (\lambda\bar{x} : \bar{A}.\, B) \ddagger^\rho_\zeta \rrbracket_\xi &= \llbracket \lambda\bar{x} : \bar{A} \ddagger^\rho_\zeta .\, B[\bar{x} \dagger^\rho / \bar{x}] \ddagger^\rho_{\zeta,x} \rrbracket_\xi \\
&= \lambda\bar{x} : \llbracket \bar{A} \ddagger^\rho_\zeta \rrbracket_\xi .\, \llbracket B[\bar{x} \dagger^\rho / \bar{x}] \ddagger^\rho_{\zeta,x} \rrbracket_{\xi,x} \\
&= \lambda\bar{x} : \llbracket \bar{A} \rrbracket_\xi \ddagger^{1+\rho}_\zeta .\, \llbracket B[\bar{x} \dagger^\rho / \bar{x}] \rrbracket_{\xi,x} \ddagger^{1+\rho}_{\zeta,x} && \text{by IH} \\
&= \lambda\bar{x} : \llbracket \bar{A} \rrbracket_\xi \ddagger^{1+\rho}_\zeta .\, \llbracket B \rrbracket_{\xi,x} [\bar{x} \dagger^{1+\rho} / \bar{x}] \ddagger^{1+\rho}_{\zeta,x} \\
&= (\lambda\bar{x} : \llbracket \bar{A} \rrbracket_\xi .\, \llbracket B \rrbracket_{\xi,x}) \ddagger^{1+\rho}_\zeta \\
&= \llbracket \lambda\bar{x} : \bar{A}.\, B \rrbracket_\xi \ddagger^{1+\rho}_\zeta
\end{aligned}$$

Lambda• $\lambda^\bullet \check{x} : \check{A}.\, B$

$$
\begin{aligned}
[\![(\lambda^\bullet \check{x} : \check{A}.\, B)\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} &= [\![\lambda^\bullet \check{x} : \check{A}\, \ddagger^\rho_\zeta .\, B[\check{x} \dagger^\rho / \check{x}]\, \ddagger^\rho_{\zeta,x}]\!]_{\tilde{\xi}} \\
&= \lambda^\bullet \check{x} : ([\![\check{A}\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \oplus \{\lambda^\bullet \check{x} : \check{A}\, \ddagger^\rho_\zeta .\, B[\check{x} \dagger^\rho / \check{x}]\, \ddagger^\rho_{\zeta,x}\}_{\tilde{\xi}}). \\
&\qquad\qquad x_{01\ldots1} \in [\![B[\check{x} \dagger^\rho / \check{x}]\, \ddagger^\rho_{\zeta,x}]\!]_{\tilde{\xi},x} \\
&= \lambda^\bullet \check{x} : ([\![\check{A}]\!]_{\tilde{\xi}} \oplus \{\lambda^\bullet \check{x} : \check{A}.\, B\}_{\tilde{\xi}})\, \ddagger^{1+\rho}_\zeta . \qquad \text{by IH} \\
&\qquad (x_{01\ldots1} \in [\![B]\!]_{\tilde{\xi},x})[\check{x} \dagger^{1+\rho} / \check{x}]\, \ddagger^{1+\rho}_{\zeta,x} \\
&= (\lambda^\bullet \check{x} : ([\![\check{A}]\!]_{\tilde{\xi}} \oplus \{\lambda^\bullet \check{x} : \check{A}.\, B\}_{\tilde{\xi}}).\, x_{01\ldots1} \in [\![B]\!]_{\tilde{\xi},x})\, \ddagger^{1+\rho}_\zeta \\
&= [\![\lambda^\bullet \check{x} : \check{A}.\, B]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta
\end{aligned}
$$

Application $F\, \bar{a}$

$$
\begin{aligned}
[\![(F\, \bar{a})\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} &= [\![F\, \ddagger^\rho_\zeta\ \bar{a}\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \\
&= [\![F\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}}\, [\![\bar{a}\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \\
&= [\![F]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta\ [\![\bar{a}]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \qquad \text{by IH} \\
&= [\![F\, \bar{a}]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta
\end{aligned}
$$

Sort s^n

$$
\begin{aligned}
\{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![s^n\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} &= \{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![s^n]\!]_{\tilde{\xi}} \\
&= \begin{pmatrix} \{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \\ \cdot \end{pmatrix} \dot{\to}\ s^{1+n} \\
&= (\{a\}_{\tilde{\xi}} \in [\![s^n]\!]_{\tilde{\xi}})\, \ddagger^{1+\rho}_\zeta
\end{aligned}
$$

Product $\forall \bar{x} : \bar{A}.\, B$

$$
\begin{aligned}
&\{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![(\forall \bar{x} : \bar{A}.\, B)\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \\
&= \{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![\forall \bar{x} : \bar{A}\, \ddagger^\rho .\, B[\bar{x} \dagger^\rho / \bar{x}]\, \ddagger^\rho_{\zeta,x}]\!]_{\tilde{\xi}} \\
&= \forall \bar{x} : [\![\bar{A}\, \ddagger^\rho]\!]_{\tilde{\xi}}.\, (\{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta\ (\bar{x}/01\ldots1)) \in [\![B[\bar{x} \dagger^\rho / \bar{x}]\, \ddagger^\rho_{\zeta,x}]\!]_{\tilde{\xi},x} \\
&= \forall \bar{x} : [\![\bar{A}]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho} .\, (\{a\}_{\tilde{\xi}}\, (\bar{x}/01\ldots1)) \in [\![B]\!]_{\tilde{\xi},x})[\bar{x} \dagger^{1+\rho} / \bar{x}]\, \ddagger^{1+\rho}_\zeta \qquad \text{by IH} \\
&= [\![\forall \bar{x} : \bar{A}.\, B]\!]_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta
\end{aligned}
$$

Arrow• $\check{A} \dot{\to} s^n$

$$
\begin{aligned}
&\{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![(\check{A} \dot{\to} s^n)\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \\
&= \{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta \in [\![\check{A}\, \ddagger^\rho_\zeta \dot{\to} s^n]\!]_{\tilde{\xi}} \\
&= ([\![\check{A}\, \ddagger^\rho_\zeta]\!]_{\tilde{\xi}} \oplus \{a\}_{\tilde{\xi}}\, \ddagger^{1+\rho}_\zeta) \dot{\to} s^{1+n} \\
&= ([\![\check{A}]\!]_{\tilde{\xi}} \oplus \{a\}_{\tilde{\xi}})\, \ddagger^{1+\rho} \dot{\to} s^{1+n} \qquad \text{by IH} \\
&= (\{a\}_{\tilde{\xi}} \in [\![\check{A} \dot{\to} s^n]\!]_{\tilde{\xi}})\, \ddagger^{1+\rho}_\zeta
\end{aligned}
$$

Application• $F \bullet \check{b}$

$$\begin{aligned}
\{a\}_{\xi} \ddagger_{\zeta}^{1+\rho} \in [\![(F \bullet \check{b}) \ddagger_{\zeta}^{\rho}]\!]_{\xi} &= \{a\}_{\xi} \ddagger_{\zeta}^{1+\rho} \in [\![F \ddagger_{\zeta}^{\rho} \bullet \check{b} \ddagger^{\rho}]\!]_{\xi} \\
&= [\![F \ddagger_{\zeta}^{\rho}]\!]_{\xi} \bullet ([\![\check{b} \ddagger_{\zeta}^{\rho}]\!]_{\xi} \oplus \{a\}_{\xi} \ddagger_{\zeta}^{1+\rho}) \\
&= [\![F]\!]_{\xi} \ddagger_{\zeta}^{1+\rho} \bullet ([\![\check{b}]\!]_{\xi} \oplus \{a\}_{\xi}) \ddagger_{\zeta}^{1+\rho} \qquad \text{by IH} \\
&= (\{a\}_{\xi} \in [\![F \bullet \check{b}]\!]_{\xi}) \ddagger_{\zeta}^{1+\rho}
\end{aligned}$$

□

Lemma 3. *For each term A, we have:*

$$[\![[\![A]\!]_{\xi}]\!]_{\zeta} = [\![[\![A]\!]_{\zeta}]\!]_{\xi} [\bar{x} \dagger^{(01)} / \bar{x} \mid x \in \xi \cap \zeta] \ddagger_{\xi \cap \zeta}^{(01)}$$

Proof. By structural induction on A.

Variable (1) $[\![z_i]\!]^n \dagger^{\pi}, z \in \xi, z \in \zeta$

$$\begin{aligned}
[\![[\![[\![z_i]\!]^n \dagger^{\pi}]\!]_{\xi}]\!]_{\zeta} &= [\![[\![[\![z_i]\!]_{\xi}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_{\zeta} \\
&= [\![[\![z_{1i}]\!]^n \dagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_{\zeta} \\
&= [\![[\![z_{1i}]\!]^n]\!]_{\zeta} \ddagger^{(1\ldots 1+n)} \ddagger^{2+\pi} \qquad \text{by Lemma 2} \\
&= [\![[\![z_{1i}]\!]_{\zeta}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{(1\ldots 1+n)} \ddagger^{2+\pi} \\
&= [\![z_{11i}]\!]^n \dagger^{(0\ldots n)} \ddagger^{(1\ldots 1+n)} \ddagger^{2+\pi} \\
&= [\![z_{11i}]\!]^n \dagger^{\cdots} \ddagger^{2+\pi} \\
&= [\![z_{11i}]\!]^n \dagger^{\cdots} \ddagger_{\xi \cap \zeta}^{(01)} \ddagger^{2+\pi} \\
&= [\![z_{11i}]\!]^n \dagger^{\cdots} \ddagger^{2+\pi} \ddagger_{\xi \cap \zeta}^{(01)} \\
&\text{since } 2+\pi \text{ and } (01) \text{ are disjoints} \\
&= [\![[\![z_{1i}]\!]_{\xi}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{(1\ldots 1+n)} \ddagger^{2+\pi} \ddagger_{\xi \cap \zeta}^{(01)} \\
&= [\![[\![z_{1i}]\!]^n]\!]_{\xi} \ddagger^{(1\ldots 1+n)} \ddagger^{2+\pi} \ddagger_{\xi \cap \zeta}^{(01)} \\
&= [\![[\![z_{1i}]\!]^n \dagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_{\xi} \ddagger_{\xi \cap \zeta}^{(01)} \qquad \text{by Lemma 2} \\
&= [\![[\![[\![z_i]\!]_{\zeta}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_{\xi} \ddagger_{\xi \cap \zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^{\pi}]\!]_{\zeta}]\!]_{\xi} \ddagger_{\xi \cap \zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^{\pi}]\!]_{\zeta}]\!]_{\xi} [\bar{x} \dagger^{(01)} / \bar{x} \mid x \in \xi \cap \zeta] \ddagger_{\xi \cap \zeta}^{(01)}
\end{aligned}$$

Variable (2) $[\![z_i]\!]^n \dagger^\pi, z \notin \xi, z \in \zeta$

$$
\begin{aligned}
[\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\xi]\!]_\zeta &= [\![[\![z_i]\!]^{1+n} \dagger^{1+\pi}]\!]_\zeta \\
&= [\![[\![z_i]\!]_\zeta]\!]^{1+n} \ddagger^{(0\ldots1+n)} \ddagger^{2+\pi} \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(0\ldots1+n)} \ddagger^{2+\pi} \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(1\ldots1+n)} \ddagger_z^{(01)} \ddagger^{2+\pi} \\
&\quad \text{since } 2+\pi \text{ and } (01) \text{ are disjoints} \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(1\ldots1+n)} \ddagger^{2+\pi} \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![z_{1i}]\!]^n \dagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]_\zeta]\!]^n \ddagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi [\bar{x} \dagger^{(01)} / \bar{x} \mid x \in \xi \cap \zeta] \ddagger_{\xi\cap\zeta}^{(01)}
\end{aligned}
$$

Variable (3) $[\![z_i]\!]^n \dagger^\pi, z \in \xi, z \notin \zeta$

$$
\begin{aligned}
[\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\xi]\!]_\zeta &= [\![[\![z_{1i}]\!]^n \dagger^{(0\ldots n)} \ddagger^{1+\pi}]\!]_\zeta \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(1\ldots1+n)} \ddagger^{2+\pi} \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(0\ldots1+n)} \ddagger_z^{(01)} \ddagger^{2+\pi} \\
&\quad \text{since } 2+\pi \text{ and } (01) \text{ are disjoints} \\
&= [\![z_{1i}]\!]^{1+n} \dagger^{(0\ldots1+n)} \ddagger^{2+\pi} \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![z_i]\!]^{1+n} \dagger^{1+\pi}]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi [\bar{x} \dagger^{(01)} / \bar{x} \mid x \in \xi \cap \zeta] \ddagger_{\xi\cap\zeta}^{(01)}
\end{aligned}
$$

Variable (4) $[\![z_i]\!]^n \dagger^\pi, z \notin \xi, z \notin \zeta$

$$
\begin{aligned}
[\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\xi]\!]_\zeta &= [\![z_i]\!]^{2+n} \dagger^{2+\pi} \\
&\quad \text{since } \dim(01) < 2 \\
&= [\![z_i]\!]^{2+n} \ddagger_{\xi\cap\zeta}^{(01)} \ddagger^{2+\pi} \\
&\quad \text{since } 2+\pi \text{ and } (01) \text{ are disjoints} \\
&= [\![z_i]\!]^{2+n} \dagger^{2+\pi} \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi \ddagger_{\xi\cap\zeta}^{(01)} \\
&= [\![[\![[\![z_i]\!]^n \dagger^\pi]\!]_\zeta]\!]_\xi [\bar{x} \dagger^{(01)} / \bar{x} \mid x \in \xi \cap \zeta] \ddagger_{\xi\cap\zeta}^{(01)}
\end{aligned}
$$

Lambda $\lambda\bar{x} : \bar{A}.\, B$

$$
\begin{aligned}
[\![[\![\lambda\bar{x} : \bar{A}.\, B]\!]_{\bar{\xi}}]\!]_{\zeta} &= \lambda\bar{x} : [\![[\![\bar{A}]\!]_{\bar{\xi}}]\!]_{\zeta}.\, [\![[\![B]\!]_{\zeta,x}]\!]_{\zeta,x} \\
&= \lambda\bar{x} : [\![[\![\bar{A}]\!]_{\zeta}]\!]_{\bar{\xi}}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta}. \qquad \text{by IH} \\
&\qquad [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in (\xi \cap \zeta) \cup \{x\}]\, \ddagger^{(01)}_{(\xi\cap\zeta)\cup\{x\}} \\
&= \lambda\bar{x} : [\![[\![\bar{A}]\!]_{\zeta}]\!]_{\bar{\xi}}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta}. \\
&\qquad [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta} \\
&= (\lambda\bar{x} : [\![[\![\bar{A}]\!]_{\zeta}]\!]_{\bar{\xi}}.\, [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x})[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta} \\
&= [\![[\![\lambda\bar{x} : \bar{A}.\, B]\!]_{\zeta}]\!]_{\bar{\xi}}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}
$$

Lambda• $\lambda^{\bullet}\check{x} : \check{A}.\, B$

$$
\begin{aligned}
&[\![[\![\lambda^{\bullet}\check{x} : \check{A}.\, B]\!]_{\bar{\xi}}]\!]_{\zeta} \\
&= \lambda^{\bullet}\check{x} : \check{C}.\, [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x} \bullet \begin{pmatrix} x_{001\ldots1} & x_{011\ldots1} \\ x_{101\ldots1} & \cdot \end{pmatrix} \\
&= \lambda^{\bullet}\check{x} : \check{C}.\, [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in (\xi \cap \zeta) \cup \{x\}]\, \ddagger^{(01)}_{(\xi\cap\zeta)\cup\{x\}} \bullet \qquad \text{by IH} \\
&\qquad\qquad \begin{pmatrix} x_{001\ldots1} & x_{011\ldots1} \\ x_{101\ldots1} & \cdot \end{pmatrix} \\
&= \lambda^{\bullet}\check{x} : \check{C}'[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta}. \\
&\qquad \left([\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x} \bullet \begin{pmatrix} x_{001\ldots1} & x_{011\ldots1} \\ x_{101\ldots1} & \cdot \end{pmatrix} \right) [\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta} \\
&= \left(\lambda^{\bullet}\check{x} : \check{C}'.\, [\![[\![B]\!]_{\zeta,x}]\!]_{\bar{\xi},x} \bullet \begin{pmatrix} x_{001\ldots1} & x_{011\ldots1} \\ x_{101\ldots1} & \cdot \end{pmatrix} \right) \\
&\qquad\qquad [\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta} \\
&= [\![[\![\lambda^{\bullet}\check{x} : \check{A}.\, B]\!]_{\zeta}]\!]_{\bar{\xi}}[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}
$$

Where

$$
\check{C} = \left[\begin{array}{l|l}
00i \mapsto \{\{A_i\}_{\bar{\xi}}\}_{\zeta} & 10i \mapsto [\![\{A_i\}_{\bar{\xi}}]\!]_{\zeta} \\
01i \mapsto \{[\![A_i]\!]_{\bar{\xi}}\}_{\zeta} & 11i \mapsto [\![[\![A_i]\!]_{\bar{\xi}}]\!]_{\zeta} \\
001\ldots1 \mapsto \{\{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\bar{\xi}}\}_{\zeta} & 101\ldots1 \mapsto [\![\{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\bar{\xi}}]\!]_{\zeta} \\
011\ldots1 \mapsto \{\lambda^{\bullet}\check{x} : ([\![\check{A}]\!]_{\bar{\xi}} \oplus \{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\bar{\xi}}).\, x_{01\ldots1} \in [\![B]\!]_{\bar{\xi},x}\}_{\zeta} &
\end{array} \right]
$$

$$
= \check{C}'[\bar{z} \dagger^{(01)} / \bar{z} \mid z \in \xi \cap \zeta]\, \ddagger^{(01)}_{\xi\cap\zeta} \quad \text{by IH}
$$

and

$$
\check{C}' = \left[\begin{array}{l|l}
00i \mapsto \{\{A_i\}_{\zeta}\}_{\bar{\xi}} & 10i \mapsto [\![\{A_i\}_{\zeta}]\!]_{\bar{\xi}} \\
01i \mapsto \{[\![A_i]\!]_{\zeta}\}_{\bar{\xi}} & 11i \mapsto [\![[\![A_i]\!]_{\zeta}]\!]_{\bar{\xi}} \\
001\ldots1 \mapsto \{\{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\zeta}\}_{\bar{\xi}} & 101\ldots1 \mapsto [\![\{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\zeta}]\!]_{\bar{\xi}} \\
011\ldots1 \mapsto \{\lambda^{\bullet}\check{x} : ([\![\check{A}]\!]_{\zeta} \oplus \{\lambda^{\bullet}\check{x} : \check{A}.\, B\}_{\zeta}).\, x_{01\ldots1} \in [\![B]\!]_{\zeta,x}\}_{\bar{\xi}} &
\end{array} \right]
$$

Application $F\,\bar{a}$

$$\begin{aligned}
[\![[\![F\,\bar{a}]\!]_{\xi}]\!]_{\zeta} &= [\![[\![F]\!]_{\xi}]\!]_{\zeta}\,[\![[\![\bar{a}]\!]_{\xi}]\!]_{\zeta} \\
&= [\![[\![F]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta} && \text{by IH} \\
&\quad\ [\![[\![\bar{a}]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta} \\
&= [\![[\![F\,\bar{a}]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}$$

Sort s^n

$$\begin{aligned}
\{\{a\}_{\xi}\}_{\zeta} \in [\![[\![s^n]\!]_{\xi}]\!]_{\zeta} &= \begin{pmatrix} \{\{a\}_{\xi}\}_{\zeta} & \{[\![a]\!]_{\xi}\}_{\zeta} \\ [\![\{a\}_{\xi}]\!]_{\zeta} & \cdot \end{pmatrix} \dot{\to} s^{n+2} \\
&= \left(\begin{pmatrix} \{\{a\}_{\zeta}\}_{\xi} & \{[\![a]\!]_{\zeta}\}_{\xi} \\ [\![\{a\}_{\zeta}]\!]_{\xi} & \cdot \end{pmatrix} \dot{\to} s^{n+2} \right) \\
&\qquad [\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta} \\
&= (\{\{a\}_{\zeta}\}_{\xi} \in [\![[\![s^n]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta])\,\ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}$$

Product $\forall \bar{x} : \bar{A}.\, B$

$$\begin{aligned}
&\{\{a\}_{\xi}\}_{\zeta} \in [\![[\![\forall \bar{x} : \bar{A}.\, B]\!]_{\xi}]\!]_{\zeta} \\
&= \forall \bar{x} : [\![[\![\bar{A}]\!]_{\xi}]\!]_{\zeta}.\, [\![[\![\bar{B}]\!]_{\xi,x}]\!]_{\zeta,x} \bullet \begin{pmatrix} \{\{a\}_{\xi}\}_{\zeta}(\bar{x}/001\ldots1) & \{[\![a]\!]_{\xi}\}_{\zeta}(\bar{x}/011\ldots1) \\ [\![\{a\}_{\xi}]\!]_{\zeta}(\bar{x}/101\ldots1) & \cdot \end{pmatrix} \\
&= \forall \bar{x} : [\![[\![\bar{A}]\!]_{\xi}]\!]_{\zeta}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta}. && \text{by IH} \\
&\quad \left([\![[\![\bar{B}]\!]_{\xi,x}]\!]_{\zeta,x} \bullet \begin{pmatrix} \{\{a\}_{\xi}\}_{\zeta}(\bar{x}/001\ldots1) & \{[\![a]\!]_{\xi}\}_{\zeta}(\bar{x}/011\ldots1) \\ [\![\{a\}_{\xi}]\!]_{\zeta}(\bar{x}/101\ldots1) & \cdot \end{pmatrix} \right) \\
&\qquad\qquad [\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta \cup \{x\}]\,\ddagger^{(01)}_{\xi\cap\zeta\cup\{x\}} \\
&= (\{\{a\}_{\zeta}\}_{\xi} \in [\![[\![\forall \bar{x} : \bar{A}.\, B]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta])\,\ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}$$

Arrow• $\check{A} \dot{\to} s^n$

$$\begin{aligned}
&\{\{a\}_{\xi}\}_{\zeta} \in [\![[\![\check{A} \dot{\to} s^n]\!]_{\xi}]\!]_{\zeta} \\
&= \check{C} \dot{\to} s^{2+n} \\
&= (\check{C}' \dot{\to} s^{2+n})[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta])\,\ddagger^{(01)}_{\xi\cap\zeta} \\
&= (\{\{a\}_{\zeta}\}_{\xi} \in [\![[\![\check{A} \dot{\to} s^n]\!]_{\zeta}]\!]_{\xi}[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta])\,\ddagger^{(01)}_{\xi\cap\zeta}
\end{aligned}$$

Where

$$\begin{aligned}
\check{C} &= \left[\begin{array}{l|l} 00i \mapsto \{\{A_i\}_{\xi}\}_{\zeta} & 10i \mapsto [\![\{A_i\}_{\xi}]\!]_{\zeta} \\ 01i \mapsto \{[\![A_i]\!]_{\xi}\}_{\zeta} & 11i \mapsto [\![[\![A_i]\!]_{\xi}]\!]_{\zeta} \\ 001\ldots1 \mapsto \{\{a\}_{\xi}\}_{\zeta} & 101\ldots1 \mapsto [\![\{a\}_{\xi}]\!]_{\zeta} \\ 011\ldots1 \mapsto \{[\![a]\!]_{\xi}\}_{\zeta} & \end{array} \right] \\
&= \check{C}'[\bar{z}\dagger^{(01)}/\bar{z} \mid z \in \xi \cap \zeta]\,\ddagger^{(01)}_{\xi\cap\zeta} \quad \text{by IH}
\end{aligned}$$

and

$$\check{C}' = \left[\begin{array}{l|l} 00i \mapsto \{\{A_i\}_\zeta\}_\xi & 10i \mapsto [\![\{A_i\}_\zeta]\!]_\xi \\ 01i \mapsto \{[\![A_i]\!]_\zeta\}_\xi & 11i \mapsto [\![[\![A_i]\!]_\zeta]\!]_\xi \\ 001\ldots1 \mapsto \{\{a\}_\zeta\}_\xi & 101\ldots1 \mapsto [\![\{a\}_\zeta]\!]_\xi \\ 011\ldots1 \mapsto \{[\![a]\!]_\zeta\}_\xi & \end{array}\right]$$

Application• $F{\bullet}\check{A}$

$$\begin{aligned}\{\{a\}_{\bar\xi}\}_\zeta \in [\![[\![F{\bullet}\check{A}]\!]_{\bar\xi}]\!]_\zeta &= [\![[\![F]\!]_{\bar\xi}]\!]_\zeta{\bullet}\check{C} \\ &= (\{\{a\}_\zeta\}_{\bar\xi} \in [\![F{\bullet}\check{A}]\!]_{\bar\xi}[\bar z \dagger^{(01)}/\bar z \mid z \in \xi \cap \zeta]) \ddagger^{(01)}_{\xi\cap\zeta}\end{aligned}$$

Where

$$\begin{aligned}\check{C} &= \left[\begin{array}{l|l} 00i \mapsto \{\{A_i\}_{\bar\xi}\}_\zeta & 10i \mapsto [\![\{A_i\}_{\bar\xi}]\!]_\zeta \\ 01i \mapsto \{[\![A_i]\!]_{\bar\xi}\}_\zeta & 11i \mapsto [\![[\![A_i]\!]_{\bar\xi}]\!]_\zeta \\ 001\ldots1 \mapsto \{\{a\}_{\bar\xi}\}_\zeta & 101\ldots1 \mapsto [\![\{a\}_{\bar\xi}]\!]_\zeta \\ 011\ldots1 \mapsto \{[\![a]\!]_{\bar\xi}\}_\zeta & \end{array}\right] \\ &= \check{C}'[\bar z \dagger^{(01)}/\bar z \mid z \in \xi \cap \zeta] \ddagger^{(01)}_{\xi\cap\zeta} \quad \text{by IH}\end{aligned}$$

and

$$\check{C}' = \left[\begin{array}{l|l} 00i \mapsto \{\{A_i\}_\zeta\}_\xi & 10i \mapsto [\![\{A_i\}_\zeta]\!]_\xi \\ 01i \mapsto \{[\![A_i]\!]_\zeta\}_\xi & 11i \mapsto [\![[\![A_i]\!]_\zeta]\!]_\xi \\ 001\ldots1 \mapsto \{\{a\}_\zeta\}_\xi & 101\ldots1 \mapsto [\![\{a\}_\zeta]\!]_\xi \\ 011\ldots1 \mapsto \{[\![a]\!]_\zeta\}_\xi & \end{array}\right]$$

□

Lemma 4 ($[\![\cdot]\!]$ and substitution, part 1)**.** *For each term A, and each variable z not in ξ, we have:*

i) $[\![A[u/z_i]]\!]_\xi = [\![A]\!]_{\xi,z}[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}]$; *and*

ii) $\{a[u/z_i]\}_\xi \in [\![A[u/z_i]]\!]_\xi = (\{a\}_{\xi,z} \in [\![A]\!]_{\xi,z})[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}]$.

Proof. By simultaneous induction on the structure of A.

Variable (1) $[\![z_i]\!]^n \dagger^\pi$

$$\begin{aligned}[\![[\![z_i]\!]^n \dagger^\pi[u/z_i]]\!]_\xi &= [\![[\![u]\!]^n \ddagger^\pi]\!]_\xi \\ &= [\![[\![u]\!]^n]\!]_\xi \ddagger^{\pi+1} && \text{by Lemma 2} \\ &= [\![[\![u]\!]_\xi]\!]^n \ddagger^{(0\ldots n)} \ddagger^{\pi+1} && \text{by Corollary 1} \\ &= [\![z_{1i}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{\pi+1}[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}] \\ &= [\![[\![z_i]\!]_{\xi,z}]\!]^n \ddagger^{(0\ldots n)} \ddagger^{\pi+1}[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}] \\ &= [\![[\![z_i]\!]^n]\!]_{\xi,z} \ddagger^{\pi+1}[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}] \\ &= [\![[\![z_i]\!]^n \dagger^\pi]\!]_{\xi,z\mapsto(z_0,z_1)}[\{u\}_\xi/z_{0i}][[\![u]\!]_\xi/z_{1i}] && \text{by Lemma 2}\end{aligned}$$

Variable (2) $[\![x_j]\!]^n \dagger^\pi$, with $x \neq z$

$$\begin{aligned}
[\![[\![x_j]\!]^n \dagger^\pi[u/z_i]]\!]_{\vec{\xi}} &= [\![[\![x_j]\!]^n \dagger^\pi]\!]_{\vec{\xi}} \\
&= [\![[\![x_j]\!]^n \dagger^\pi]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \\
&= [\![[\![x_j]\!]^n \dagger^\pi]\!]_{\vec{\xi},z\mapsto(z_0,z_1)}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \quad \text{by Lemma 1}
\end{aligned}$$

Lambda $\lambda\bar{x} : \bar{A}.\, B$

$$\begin{aligned}
[\![(\lambda\bar{x} : \bar{A}.\, B)[u/z_i]]\!]_{\vec{\xi}} &= [\![\lambda\bar{x} : \bar{A}[u/z_i].\, B[u/z_i]]\!]_{\vec{\xi}} \\
&= \lambda\bar{x} : [\![\bar{A}[u/z_i]]\!]_{\vec{\xi}}.\, [\![B[u/z_i]]\!]_{\vec{\xi},x} \\
&= \lambda\bar{x} : [\![\bar{A}]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]. \quad \text{by IH} \\
&\qquad\qquad [\![B]\!]_{\vec{\xi},x,z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \\
&= [\![\lambda\bar{x} : \bar{A}.\, B]\!]_{\vec{\xi},z\mapsto(z_0,z_1)}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]
\end{aligned}$$

Lambda• $\lambda^\bullet\check{x} : \check{A}.\, B$

$$\begin{aligned}
&[\![(\lambda^\bullet\check{x} : \check{A}.\, B)[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![\lambda^\bullet\check{x} : \check{A}[u/z_i].\, B[u/z_i]]\!]_{\vec{\xi}} \\
&= \lambda^\bullet\check{x} : ([\![\check{A}[u/z_i]]\!]_{\vec{\xi}} \oplus \{\lambda^\bullet\check{x} : \check{A}[u/z_i].\, B[u/z_i]\}_{\vec{\xi}}).\, x_{01\ldots1} \in [\![B[u/z_i]]\!]_{\vec{\xi},x} \\
&= \lambda^\bullet\check{x} : ([\![\check{A}]\!]_{\vec{\xi}} \oplus \{\lambda^\bullet\check{x} : \check{A}[u/z_i].\, B[u/z_i]\}_{\vec{\xi}}). \\
&\qquad\qquad x_{01\ldots1} \in [\![B[u/z_i]]\!]_{\vec{\xi},x} \\
&= \lambda^\bullet\check{x} : ([\![\check{A}]\!]_{\vec{\xi},z} \oplus \lambda^\bullet\check{x} : \check{A}.\, B)[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]. \quad \text{by IH} \\
&\qquad\qquad (x_{01\ldots1} \in [\![B]\!]_{\vec{\xi},x,z})[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \\
&= (\lambda^\bullet\check{x} : ([\![\check{A}]\!]_{\vec{\xi},z} \oplus \{\lambda^\bullet\check{x} : \check{A}.\, B\}_{\vec{\xi},z}).\, x_{01\ldots1} \in [\![B]\!]_{\vec{\xi},x,z}) \\
&\qquad\qquad [\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \\
&= [\![\lambda^\bullet\check{x} : \check{A}.\, B]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]
\end{aligned}$$

Application $F\,\bar{a}$

$$\begin{aligned}
[\![(F\,\bar{a})[u/z_i]]\!]_{\vec{\xi}} &= [\![F[u/z_i]\,\bar{a}[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![F[u/z_i]]\!]_{\vec{\xi}}\,[\![\bar{a}[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![F]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]\,[\![\bar{a}]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \quad \text{by IH} \\
&= ([\![F]\!]_{\vec{\xi},z}\,[\![\bar{a}]\!]_{\vec{\xi},z})[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \\
&= [\![F\,\bar{a}]\!]_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]
\end{aligned}$$

Sort s^n

$$\begin{aligned}
\{a[u/z_i]\}_{\vec{\xi}} \in [\![s^n[u/z_i]]\!]_{\vec{\xi}} &= \begin{pmatrix} \{a\}_{\vec{\xi},z}[\{u\}_{\vec{\xi}}/z_{0i}] \\ \cdot \end{pmatrix} \overset{\bullet}{\to} s^{1+n} \\
&= (\{a\}_{\vec{\xi},z} \in [\![s^n]\!]_{\vec{\xi},z})[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}]
\end{aligned}$$

Product $\forall \bar{x} : \bar{A}.\, B$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\bar{\xi}} \in [\![(\forall \bar{x} : \bar{A}.\, B)[u/z_i]]\!]_{\bar{\xi}} \\
&= \forall \bar{x} : [\![\bar{A}[u/z_i]]\!]_{\bar{\xi}}.(\{a[u/z_i]\}_{\bar{\xi}}\,(\bar{x}/01...1)) \in [\![B[u/z_i]]\!]_{\bar{\xi},x,z} \\
&= (\forall \bar{x} : [\![\bar{A}]\!]_{\bar{\xi},z}.(\{a\}_{\bar{\xi},z}\,(\bar{x}/01...1)) \in [\![B]\!]_{\bar{\xi},x,z})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}] \quad \text{by IH} \\
&= (\{a\}_{\bar{\xi},z} \in [\![\forall \bar{x} : \bar{A}.\, B]\!]_{\bar{\xi},z})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}]
\end{aligned}
$$

Arrow• $\check{A} \overset{\bullet}{\to} s^n$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\bar{\xi}} \in [\![(\check{A} \overset{\bullet}{\to} s^n)[u/z_i]]\!]_{\bar{\xi}} \\
&= ([\![\check{A}[u/z_i]]\!]_{\bar{\xi}} \oplus \{a[u/z_i]\}_{\bar{\xi}}) \overset{\bullet}{\to} s^{1+n} \\
&= (([\![\check{A}]\!]_{\bar{\xi},z} \oplus \{a\}_{\bar{\xi},z}) \overset{\bullet}{\to} s^{1+n})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}] \qquad \text{by IH} \\
&= (\{a\}_{\bar{\xi},z} \in [\![\check{A} \overset{\bullet}{\to} s^n]\!]_{\bar{\xi},z})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}]
\end{aligned}
$$

Application• $F \bullet \check{b}$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\bar{\xi}} \in [\![(F \bullet \check{b})[u/z_i]]\!]_{\bar{\xi}} \\
&= [\![F[u/z_i]]\!]_{\bar{\xi}} \bullet ([\![\check{b}[u/z_i]]\!]_{\bar{\xi}} \oplus \{a[u/z_i]\}_{\bar{\xi}}) \\
&= [\![F]\!]_{\bar{\xi},z}[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}] \bullet ([\![\check{b}]\!]_{\bar{\xi},z} \oplus \{a\}_{\bar{\xi},z})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}] \quad \text{by IH} \\
&= (\{a\}_{\bar{\xi},z} \in [\![F \bullet \check{b}]\!]_{\bar{\xi},z})[\{u\}_{\bar{\xi}}/z_{0i}][[\![u]\!]_{\bar{\xi}}/z_{1i}]
\end{aligned}
$$

□

Lemma 5 ($[\![\cdot]\!]$ and substitution, part 2). *For each term A, for variable z not free in A or contained in ξ, we have:*

i) $[\![A[u/z_i]]\!]_{\xi} = [\![A]\!]_{\xi}[\{u\}_{\xi}/z_{0i}]$; *and*

ii) $\{a[u/z_i]\}_{\xi} \in [\![A[u/z_i]]\!]_{\xi} = (\{a\}_{\xi} \in [\![A]\!]_{\xi})[\{u\}_{\xi}/z_{0i}]$.

Proof. By simultaneous induction on the structure of A.

Variable (1) $[\![z_i]\!]^n \dagger^\pi$ Impossible.

Variable (2) $[\![x_j]\!]^n \dagger^\pi$, with $x \neq z$

$$
\begin{aligned}
[\![[\![x_j]\!]^n \dagger^\pi[u/z_i]]\!]_{\xi} &= [\![[\![x_j]\!]^n \dagger^\pi]\!]_{\xi} \\
&= [\![[\![x_j]\!]^n \dagger^\pi]\!]_{\xi}[\{u\}_{\xi}/z_{0i}]
\end{aligned}
$$

Lambda $\lambda \bar{x} : \bar{A}.\, B$

$$
\begin{aligned}
[\![(\lambda \bar{x} : \bar{A}.\, B)[u/z_i]]\!]_{\bar{\xi}} &= [\![\lambda \bar{x} : \bar{A}[u/z_i].\, B[u/z_i]]\!]_{\bar{\xi}} \\
&= \lambda \bar{x} : [\![\bar{A}[u/z_i]]\!]_{\bar{\xi}}.[\![B[u/z_i]]\!]_{\bar{\xi},x} \\
&= \lambda \bar{x} : [\![\bar{A}]\!]_{\bar{\xi}}[\{u\}_{\bar{\xi}}/z_{0i}].[\![B]\!]_{\bar{\xi},x}[\{u\}_{\bar{\xi}}/z_{0i}] \qquad \text{by IH} \\
&= [\![\lambda \bar{x} : \bar{A}.\, B]\!]_{\bar{\xi}}[\{u\}_{\bar{\xi}}/z_{0i}]
\end{aligned}
$$

Lambda• $\lambda^\bullet \check{x} : \check{A}.\, B$

$$
\begin{aligned}
&[\![(\lambda^\bullet \check{x} : \check{A}.\, B)[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![\lambda^\bullet \check{x} : \check{A}[u/z_i].\, B[u/z_i]]\!]_{\vec{\xi}} \\
&= \lambda^\bullet \check{x} : ([\![\check{A}[u/z_i]]\!]_{\vec{\xi}} \oplus \{\lambda^\bullet \check{x} : \check{A}[u/z_i].\, B[u/z_i]\}_{\vec{\xi}}).\, x_{01\ldots 1} \in [\![B[u/z_i]]\!]_{\vec{\xi},x} \\
&= \lambda^\bullet \check{x} : ([\![\check{A}]\!]_{\vec{\xi}} \oplus \{\lambda^\bullet \check{x} : \check{A}.\, B\}_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]. \qquad \text{by IH} \\
&\qquad\qquad (x_{01\ldots 1} \in [\![B]\!]_{\vec{\xi},x})[\{u\}_{\vec{\xi}}/z_{0i}] \\
&= [\![\lambda^\bullet \check{x} : \check{A}.\, B]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

Application $F\,\bar{a}$

$$
\begin{aligned}
[\![(F\,\bar{a})[u/z_i]]\!]_{\vec{\xi}} &= [\![F[u/z_i]\,\bar{a}[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![F[u/z_i]]\!]_{\vec{\xi}}\,[\![\bar{a}[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![F]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}]\,[\![\bar{a}]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}] \qquad \text{by IH} \\
&= ([\![F]\!]_{\vec{\xi}}\,[\![\bar{a}]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}] \\
&= [\![F\,\bar{a}]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

Sort s^n

$$
\begin{aligned}
\{a[u/z_i]\}_{\vec{\xi}} \in [\![s^n[u/z_i]]\!]_{\vec{\xi}} &= \begin{pmatrix} \{a\}_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}] \\ \cdot \end{pmatrix} \stackrel{\bullet}{\to} s^{1+n} \\
&= (\{a\}_{\vec{\xi}} \in [\![s^n]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

Product $\forall \bar{x} : \bar{A}.\, B$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\vec{\xi}} \in [\![(\forall \bar{x} : \bar{A}.\, B)[u/z_i]]\!]_{\vec{\xi}} \\
&= \forall \bar{x} : [\![\bar{A}[u/z_i]]\!]_{\vec{\xi}}.\, (\{a[u/z_i]\}_{\vec{\xi}}\,(\bar{x}/01\ldots 1)) \in [\![B[u/z_i]]\!]_{\vec{\xi},x} \\
&= (\forall \bar{x} : [\![\bar{A}]\!]_{\vec{\xi}}.\, (\{a\}_{\vec{\xi}}\,(\bar{x}/01\ldots 1)) \in [\![B]\!]_{\vec{\xi},x})[\{u\}_{\vec{\xi}}/z_{0i}] \qquad \text{by IH} \\
&= (\{a\}_{\vec{\xi}} \in [\![\forall \bar{x} : \bar{A}.\, B]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

Arrow• $\check{A} \stackrel{\bullet}{\to} s^n$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\vec{\xi}} \in [\![(\check{A} \stackrel{\bullet}{\to} s^n)[u/z_i]]\!]_{\vec{\xi}} \\
&= ([\![\check{A}[u/z_i]]\!]_{\vec{\xi}} \oplus \{a[u/z_i]\}_{\vec{\xi}}) \stackrel{\bullet}{\to} s^{1+n} \\
&= (([\![\check{A}]\!]_{\vec{\xi}} \oplus \{a\}_{\vec{\xi}}) \stackrel{\bullet}{\to} s^{1+n})[\{u\}_{\vec{\xi}}/z_{0i}] \qquad \text{by IH} \\
&= (\{a\}_{\vec{\xi}} \in [\![\check{A} \stackrel{\bullet}{\to} s^n]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

Application• $F \bullet \check{b}$

$$
\begin{aligned}
&\{a[u/z_i]\}_{\vec{\xi}} \in [\![(F \bullet \check{b})[u/z_i]]\!]_{\vec{\xi}} \\
&= [\![F[u/z_i]]\!]_{\vec{\xi}} \bullet ([\![\check{b}[u/z_i]]\!]_{\vec{\xi}} \oplus \{a[u/z_i]\}_{\vec{\xi}}) \\
&= [\![F]\!]_{\vec{\xi}}[\{u\}_{\vec{\xi}}/z_{0i}][[\![u]\!]_{\vec{\xi}}/z_{1i}] \bullet ([\![\check{b}]\!]_{\vec{\xi}} \oplus \{a\}_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}] \qquad \text{by IH} \\
&= (\{a\}_{\vec{\xi}} \in [\![F \bullet \check{b}]\!]_{\vec{\xi}})[\{u\}_{\vec{\xi}}/z_{0i}]
\end{aligned}
$$

□

Lemma 6 (Symmetry)**.** *For each term A, $[\![A]\!]^n$ is symmetric in its n first dimensions. More specifically,*

i) $[\![A]\!]^n_\xi \ddagger^\pi_\xi = [\![A]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi$; *and*

ii) $(a \in [\![A]\!]^n_\xi) \ddagger^\pi_\xi = (a \in [\![A]\!]^n_\xi) \ddagger^{\mathrm{normal}_n(\pi)}_\xi$.

Proof. By simultaneous induction on the structure of A.

Variable (1) $[\![x_i]\!]^m \dagger^\rho, x \in \xi$

$$\begin{aligned}
[\![[\![x_i]\!]^m \dagger^\rho]\!]^n_\xi \ddagger^\pi_\xi &= [\![x_{1\dots1i}]\!]^m \dagger^{n+(0\dots m)} \ddagger^{n+\rho} \ddagger^\pi_\xi \\
&= [\![x_{1\dots1i}]\!]^m \dagger^{n+(0\dots m)} \ddagger^{n+\rho} \\
&= [\![[\![x_i]\!]^m \dagger^\rho]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi
\end{aligned}$$

Variable (2) $[\![x_i]\!]^m \dagger^\rho, x \notin \xi$

$$\begin{aligned}
[\![[\![x_i]\!]^m \dagger^\rho]\!]^n_\xi \ddagger^\pi_\xi &= [\![x_i]\!]^{n+m} \dagger^{n+\rho} \ddagger^\pi_\xi \\
&= [\![x_i]\!]^{n+m} \dagger^{\mathrm{normal}_{n+m}((n+\rho)\circ\pi)} \\
&\quad \text{since } n+\rho \text{ and } (0\dots n-1) \text{ are disjoint} \\
&= [\![x_i]\!]^{n+m} \dagger^{\mathrm{normal}_{n+m}((n+\rho)\circ\mathrm{normal}_n(\pi))} \\
&= [\![[\![x_i]\!]^m \dagger^\rho]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi
\end{aligned}$$

Lambda (Lambda• is similar) $\lambda\bar{x} : \bar{A}. B$

$$\begin{aligned}
[\![\lambda\bar{x} : \bar{A}. B]\!]^n_\xi \ddagger^\pi_\xi &= \lambda\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^\pi_\xi . [\![B]\!]^n_{\xi,x} [\bar{x} \dagger^\pi / \bar{x}] \ddagger^\pi_{\xi,x} \\
&= \lambda\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi . [\![B]\!]^n_{\xi,x} [\bar{x} \dagger^\pi / \bar{x}] \ddagger^{\mathrm{normal}_n(\pi)}_{\xi,x} && \text{by IH} \\
&= \lambda\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi . [\![B]\!]^n_{\xi,x} [\bar{x} \dagger^{\mathrm{normal}_n(\pi)} / \bar{x}] \ddagger^{\mathrm{normal}_n(\pi)}_{\xi,x} \\
&= [\![\lambda\bar{x} : \bar{A}. B]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi
\end{aligned}$$

Application (Application• is similar) $F\bar{a}$

$$\begin{aligned}
[\![F\,\bar{a}]\!]^n_\xi \ddagger^\pi_\xi &= \lambda\bar{x} : [\![\bar{F}]\!]^n_\xi \ddagger^\pi_\xi . [\![a]\!]^n_\xi \ddagger^\pi_\xi \\
&= \lambda\bar{x} : [\![\bar{F}]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi . [\![a]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_{\xi,x} && \text{by IH} \\
&= [\![\lambda\bar{x} : \bar{A}. B]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi
\end{aligned}$$

Product (Arrow• is similar) $\forall\bar{x} : \bar{A}. B$

$$\begin{aligned}
&(C \in [\![\forall\bar{x} : \bar{A}. B]\!]^n_\xi) \ddagger^\pi_\xi \\
&= \forall\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^\pi_\xi . (C\,\bar{x} \in [\![B]\!]^n_{\xi,x}) [\bar{x} \dagger^\pi / \bar{x}] \ddagger^\pi_{\xi,x} \\
&= \forall\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi . (C\,\bar{x} \in [\![B]\!]^n_{\xi,x}) [\bar{x} \dagger^\pi / \bar{x}] \ddagger^{\mathrm{normal}_n(\pi)}_{\xi,x} && \text{by IH} \\
&= \forall\bar{x} : [\![\bar{A}]\!]^n_\xi \ddagger^{\mathrm{normal}_n(\pi)}_\xi . (C\,\bar{x} \in [\![B]\!]^n_{\xi,x}) [\bar{x} \dagger^{\mathrm{normal}_n(\pi)} / \bar{x}] \ddagger^{\mathrm{normal}_n(\pi)}_{\xi,x} \\
&= (C \in [\![\forall\bar{x} : \bar{A}. B]\!]^n_\xi) \ddagger^{\mathrm{normal}_n(\pi)}_\xi
\end{aligned}$$

Sort s
The result $(C \in [\![s]\!]^n_\xi) \ddagger^\pi_\xi = (C \in [\![s]\!]^n_\xi) \ddagger^{\mathrm{normal}_n(\pi)}_\xi$ stems from an easy induction on n. □

Lemma 12 (Generalized abstraction)**.** *Assuming that ξ conforms to Γ,*

i) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash [\![A]\!]_\xi : \{A\}_\xi \in [\![B]\!]_\xi$

ii) $\Gamma \vdash A : B \;\Rightarrow\; [\![\Gamma]\!]_\xi \vdash \{A\}_\xi : \{B\}_\xi$

iii) $\Gamma \vdash B : s^n \;\Rightarrow\; [\![\Gamma]\!]_\xi, x : B \vdash x \in [\![B]\!]_\xi : s^{n+1}$

Proof. The lemmas are proved by transforming derivation trees. They mutually depend on each other, (but only for structurally smaller statements, hence the recursion is sound). For each lemma, each rule is treated. The rule being handled is written before the corresponding part of the resulting derivation.

In the proofs, the application of each sub-lemma to an arbitrary derivation $\Gamma \vdash A : B$ are written as follows:

i) $[\![\Gamma \vdash A : B]\!]_\xi$

ii) $|\Gamma \vdash A : B|_\xi$

iii) $\{\Gamma \vdash A : B\}_\xi$

We only give further details for the two first items in the following; iii) stemming from simple application of induction hypotheses.

i) $\Gamma \vdash A : B \Rightarrow [\![\Gamma]\!]_{\tilde{\xi}} \vdash [\![A]\!]_{\tilde{\xi}} : \{A\}_{\tilde{\xi}} \in [\![B]\!]_{\tilde{\xi}}$

Axiom; Rel-Elim; Rel-Form; Product In this case, the definition of $[\![A]\!]_{\tilde{\xi}}$ falls through: a new relation is introduced. The proof relies on the next sub-lemma.

$$\Gamma \vdash A : s^n$$

$$\frac{\dfrac{\dfrac{\vdots\ |\Gamma \vdash A : s^n|_\xi \qquad \vdots\ \{\Gamma \vdash A : s^n\}_\xi}{\Gamma_\xi, z_0 : A \vdash z_0 \in A_\xi : s^{n+1} \quad \Gamma_\xi \vdash A : s^n}\,\text{Rel-I}}{\Gamma_\xi \vdash \lambda^\bullet \check{z} : A.\ z_0 \in A_\xi : A \bullet s^{n+1}}\,\text{DEF}}{\Gamma_\xi \vdash A_\xi : A \in s^n{}_\xi}$$

Weakening

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s^n}{\Gamma, x : C \vdash A : B}\,\text{WK}$$

-$x \notin \tilde{\xi}$

$$\frac{\dfrac{\dfrac{\vdots\ \Gamma \vdash A : B_\xi \qquad \vdots\ \{\Gamma \vdash C : s^n\}_\xi}{\Gamma_\xi \vdash A_\xi : A \in B_\xi \qquad \Gamma_\xi \vdash C : s^n}\,\text{WK}}{\Gamma_\xi, x : C \vdash A_\xi : A \in B_\xi}\,\text{DEF}}{\Gamma, x : C_\xi \vdash A_\xi : A \in B_\xi}$$

-$x \in \xi$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\Gamma \vdash C : s^n{}_\xi \qquad \vdots\,\{\Gamma \vdash C : s^n\}_\xi}{\Gamma_\xi \vdash C_\xi : C \in s^n{}_\xi \qquad \Gamma_\xi \vdash C : s^n}\,\text{WK} \qquad \vdots\,\{\Gamma \vdash C : s^n\}_\xi}{\Gamma_\xi, x_0 : C \vdash C_\xi : C \in s^n{}_\xi}\,\text{DEF} \qquad \dfrac{\Gamma_\xi \vdash C : s^n}{\Gamma_\xi, x_0 : C \vdash x_0 : C}\,\text{ST}}{\Gamma_\xi, x_0 : C \vdash C_\xi \bullet x_0 : C \bullet s^{n+1}}\,\text{APP}}{\Gamma_\xi, x_0 : C \vdash C_\xi \bullet x_0 : s^{n+1}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\Gamma \vdash A : B_\xi \qquad \vdots\,\{\Gamma \vdash C : s^n\}_\xi}{\Gamma_\xi \vdash A_\xi : A \in B_\xi \qquad \Gamma_\xi \vdash C : s^n}\,\text{WK}}{\Gamma_\xi, x_0 : C \vdash A_\xi : A \in B_\xi}\,\text{WK}}{\Gamma_\xi, \bar{x} : \begin{pmatrix} C \\ C_\xi \end{pmatrix} \vdash A_\xi : A \in B_\xi}\,\text{DEF}}{\Gamma, x : C_\xi \vdash A_\xi : A \in B_\xi}$$

Rel-Intro

$$\frac{\Gamma, \check{z} : \check{A} \vdash B : s^n \quad \Gamma \vdash \check{A} : s^n}{\Gamma \vdash (\lambda^\bullet \check{z} : \check{A}.\, B) : \check{A} \bullet s^n}\,\text{Rel-I}$$

$$\dfrac{\dfrac{\dfrac{\vdots\ |\Gamma, \check{z} : \check{A} \vdash B : s^n|_\xi}{\Gamma_\xi, \check{z} : \check{A}, z_{011} : B \vdash z_{011} \in B_\xi : s^{n+1}}\,\text{WK}}{\Gamma_\xi, \check{z} : (\check{A}_\xi \oplus (\lambda^\bullet \check{z} : \check{A}.\, B)) \vdash z_{011} \in B_\xi : s^{n+1}} \quad \Gamma_\xi \vdash (\check{A}_\xi \oplus (\lambda^\bullet \check{z} : \check{A}.\, B)) : s^{n+1}}{\dfrac{\Gamma_\xi \vdash (\lambda^\bullet \check{z} : (\check{A}_\xi \oplus (\lambda^\bullet \check{z} : \check{A}.\, B)).\, z_{011} \in B_\xi) : ((\check{A}_\xi \oplus (\lambda^\bullet \check{z} : \check{A}.\, B)) \bullet s^{n+1})}{\Gamma_\xi \vdash \lambda^\bullet \check{z} : \check{A}.\, B_\xi : (\lambda^\bullet \check{z} : \check{A}.\, B \in \check{A} \bullet s^n{}_\xi)}\,\text{DEF}}\,\text{Rel-I}$$

Application

$$\frac{\Gamma \vdash F : (\bar{x} : \bar{A}.\, B) \quad \Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash F\, \bar{a} : B[\bar{a}/x]}\,\text{APP}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \Gamma \vdash F : (\bar{x} : \bar{A}.\, B)_\xi}{\Gamma_\xi \vdash F_\xi : F \in \bar{x} : \bar{A}.\, B_\xi}\,\text{DEF} \quad \dfrac{\vdots\ \Gamma \vdash \bar{a} : \bar{A}_\xi}{\Gamma_\xi \vdash \bar{a}_\xi : \bar{A}_\xi}}{\Gamma_\xi \vdash F_\xi : (\bar{x} : \bar{A}_\xi.\, (F\, \bar{x}) \in B_{\xi,x})}\,\text{APP}}{\Gamma_\xi \vdash F_\xi\, \bar{a}_\xi : ((F\, \bar{x}) \in B_{\xi,x})[\bar{a}/\bar{x}]}\,\text{Lem. 4}}{\Gamma_\xi \vdash F_\xi\, \bar{a}_\xi : (F\, \bar{a}) \in B[\bar{a}/\bar{x}]_\xi}\,\text{DEF}}{\Gamma_\xi \vdash F\, \bar{a}_\xi : (F\, \bar{a}) \in B[\bar{a}/\bar{x}]_\xi}\,\text{DEF}$$

Abstraction

$$\frac{\Gamma, z : \bar{A} \vdash b : B \quad \Gamma \vdash (\bar{x} : \bar{A}.\, B) : s^n}{\Gamma \vdash (\lambda \bar{z} : \bar{A}.\, b) : (\bar{x} : \bar{A}.\, B)}\,\text{ABS}$$

$$\dfrac{\dfrac{\dfrac{\vdots\ \Gamma, z : \bar{A} \vdash b : B_{\xi,z}}{\Gamma, z : \bar{A}_{\xi,z} \vdash b_{\xi,z} : b \in B_{\xi,z}}\,\text{DEF} \qquad \vdots\ \Gamma_\xi \vdash (\bar{z} : \bar{A}_\xi.\, b \in B_{\xi,z}) : s^{n+1}}{\Gamma_\xi, z : \bar{A}_\xi \vdash b_{\xi,z} : b \in B_{\xi,z}}\,\text{ABS}}{\Gamma_\xi \vdash (\lambda \bar{z} : \bar{A}_\xi.\, b_{\xi,z}) : (\bar{z} : \bar{A}_\xi.\, b \in B_{\xi,z})}\,\text{DEF}}{\Gamma_\xi \vdash \lambda \bar{z} : \bar{A}.\, b_\xi : (\lambda \bar{z} : \bar{A}.\, b) \in \bar{z} : \bar{A}.\, B_\xi}\,\text{DEF}$$

Conversion

$$\dfrac{\Gamma \vdash A : B' \quad \Gamma \vdash B : s^n \quad B' =_\beta B}{\Gamma \vdash A : B}\,\text{CONV}$$

$$\dfrac{\dfrac{\vdots\ |\Gamma \vdash B : s^n|_\xi \qquad \dfrac{\vdots\ \{\Gamma \vdash A : B'\}_\xi \qquad \dfrac{\Gamma_\xi \vdash A : B' \qquad \vdots\ \{\Gamma \vdash B : s^n\}_\xi \quad \Gamma_\xi \vdash B : s^n \quad B' =_\beta B}{\Gamma_\xi \vdash A : B}\,\text{CONV}}{\Gamma_\xi, x : B \vdash x \in B_\xi : s^{n+1}}\,\text{subst}}{\Gamma_\xi \vdash A \in B_\xi : s^{n+1}}$$

$$\dfrac{\dfrac{\vdots\ \Gamma \vdash A : B'_\xi \qquad \dfrac{B' =_\beta B}{A \in B'_\xi =_\beta A \in B_\xi}\,\text{Lem. 9}}{\Gamma_\xi \vdash A_\xi : A \in B'_\xi}\,\text{CONV}}{\Gamma_\xi \vdash A_\xi : A \in B_\xi}$$

Start

$$\dfrac{\Gamma \vdash A : s^n}{\Gamma, x : A \vdash x : A}\,\text{ST}$$

-$x \notin \xi$ Since ξ conforms to Γ, no variable of ξ is in Γ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A : s^n}{\Gamma, x : A \vdash x : A}\text{ST}}{\Gamma, x : A \vdash [\![x]\!]^d : x \in A}\text{PARAM}}{\Gamma_\xi, x : A \vdash [\![x]\!]^d : x \in A_\xi}(\text{conforms})}{\Gamma, x : A_\xi \vdash x_\xi : x \in A_\xi}\text{DEF}$$

-$x \in \xi$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, |\Gamma \vdash A : s^n|_\xi \\ \Gamma_\xi, x_0 : A \vdash x_0 \in A_\xi : s^{n+1}\end{array}}{\Gamma_\xi, x_0 : A, x_1 : x_0 \in A_\xi \vdash x_1 : x_0 \in A_\xi}\text{ST}}{\Gamma, x : A_\xi \vdash x_\xi : x \in A_\xi}\text{DEF}$$

Param

$$\dfrac{\Gamma \vdash x : A}{\Gamma \vdash [\![x]\!]^d : x \in A}\text{PARAM}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_\xi \vdash \llbracket x \rrbracket^{\,d} \in x \in A_\xi : s^{n+1}}}{\Gamma_\xi \vdash \llbracket x \rrbracket^{\,d+1}_{\,\xi} : \llbracket x \rrbracket^{\,d} \in x \in A_\xi}\,\text{DEF}}{\Gamma_\xi \vdash \llbracket x \rrbracket^{\,d}_{\,\xi} : (x_\xi \in x \in A_\xi)}\,\text{CONV}}{\Gamma_\xi \vdash \llbracket x \rrbracket^{\,d}_{\,\xi} : \llbracket x \rrbracket^{\,d} \in x \in A_\xi}$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash x : A_\xi}}{\Gamma_\xi \vdash x_\xi : x \in A_\xi}\,\text{PARAM} \qquad \dfrac{\dfrac{\text{Eq. (7)}}{A_\xi =_\beta A_\xi}}{A_\xi \bullet \begin{pmatrix} x & x \\ x_\xi & \cdot \end{pmatrix} =_\beta (A_\xi \bullet \begin{pmatrix} x & x \\ x_\xi & \cdot \end{pmatrix})}\,\text{-Rel-Elim}}{(\llbracket x \rrbracket^{\,d} \in x \in A_\xi) =_\beta (x_\xi \in x \in A_\xi)}\,\text{DEF}$$

ii) $\Gamma \vdash B : s^n \ \llbracket \Gamma \rrbracket_\xi, x : \{B\}_\xi \vdash x \in \llbracket B \rrbracket_\xi : s^{n+1}$

Axiom

$$\dfrac{}{\vdash {s_1}^n : {s_2}^n}\,\text{AX}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash {s_1}^n : {s_2}^n}\,\text{AX}}{x : {s_1}^n \vdash x : {s_1}^n}\,\text{ST}}{x : {s_1}^n \vdash x : {s_1}^{n+1}}\,\text{DEF}}{x : {s_1}^n \vdash x \bullet {s_1}^{n+1} : {s_2}^{n+1}}\,\text{Rel-F}}{x : {s_1}^n \vdash x \in {s_1}^n{}_\xi : {s_2}^{n+1}}\,\text{DEF}$$

Start

$$\dfrac{\Gamma \vdash s^n : {s_2}^n}{\Gamma, y : s^n \vdash y : s^n}\,\text{ST}$$

$y \notin \xi$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash s^n : {s_2}^n}{\Gamma, y : s^n \vdash y : s^n}\,\text{ST}}{\Gamma, y : s^n \vdash [\![y]\!]^d : y \in s^n}\,\text{PARAM} \quad \dfrac{\Gamma \vdash s^n : {s_2}^n}{\Gamma, y : s^n \vdash y : s^n}\,\text{ST}}{\Gamma, y : s^n, x : y \vdash [\![y]\!]^d : y \in s^n}\,\text{WK}}{\Gamma, y : s^n, x : y \vdash [\![y]\!]^d : y \bullet s^{n+1}}\,\text{DEF} \quad \dfrac{\dfrac{\Gamma \vdash s^n : {s_2}^n}{\Gamma, y : s^n \vdash y : s^n}\,\text{ST}}{\Gamma, y : s^n, x : y \vdash x : y}\,\text{ST}}{\Gamma, y : s^n, x : y \vdash [\![y]\!]^d \bullet x : s^{n+1}}\,\text{Rel-E}}{\Gamma_\xi, y : s^n, x : y \vdash [\![y]\!]^d \bullet x : s^{n+1}}\,\text{(conforms)}}{\Gamma, y : s^n{}_\xi, x : y \vdash x \in y_\xi : s^{n+1}}\,\text{DEF}$$

-$y \in \xi$

$$\dfrac{\dfrac{\Gamma, y : s^n, y_1 : y \bullet s^{n+1}, x : y \vdash y_1 : y \bullet s^{n+1} \quad \dfrac{\Gamma, y : s^n, y_1 : y \bullet s^{n+1} \vdash y : s^n}{\Gamma, y : s^n, y_1 : y \bullet s^{n+1}, x : y \vdash x : y}\,\text{ST}}{\Gamma_\xi, y : s^n, y_1 : y \bullet s^{n+1}, x : y \vdash y_1 \bullet x : s^{n+1}}\,\text{Rel-E}}{\Gamma, y : s^n{}_\xi, x : y \vdash x \in y_\xi : s^{n+1}}\,\text{DEF}$$

Weakening

$$\dfrac{\Gamma \vdash B : s^n \quad \Gamma \vdash C : s^m}{\Gamma, y : C \vdash B : s^n}\,\text{WK}$$

-$y \notin \xi$

$$\dfrac{\dfrac{\Gamma_\xi, x : B \vdash x \in B_\xi : s^{n+1} \quad \Gamma_\xi \vdash C : s^m}{\Gamma_\xi, y : C, x : B \vdash x \in B_\xi : s^{n+1}}\,\text{THINNING}}{\Gamma, y : C_\xi, x : B \vdash x \in B_\xi : s^{n+1}}\,\text{DEF}$$

-$y \in \xi$

$$\dfrac{\dfrac{\Gamma_\xi, x : B \vdash A_\xi : A \in B_\xi \quad \Gamma_\xi \vdash C : s^m \quad \Gamma_\xi, y : C \vdash y \in C_\xi : s^{m+1}}{\Gamma_\xi, y : C, y_1 : y \in C_\xi, x : B \vdash x \in B_\xi : s^{n+1}}\,\text{THINNING}}{\Gamma, y : C_\xi, x : B \vdash x \in B_\xi : s^{n+1}}\,\text{DEF}$$

Rel-Elim

$$\frac{\Gamma \vdash F : \check{A} \bullet s^n \quad \Gamma \vdash \check{a} : \check{A}}{\Gamma \vdash F{\bullet}\check{a} : s^n}\,\text{Rel-E}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\Gamma \vdash F : \check{A} \bullet s^n{}_\xi \qquad \vdots\,\{\Gamma \vdash F{\bullet}\check{a} : s^n\}_\xi}{\Gamma_\xi \vdash F_\xi : F \in \check{A} \bullet s^n{}_\xi \qquad \Gamma_\xi \vdash F{\bullet}\check{a} : s^n}\,\text{WK}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash F_\xi : F \in \check{A} \bullet s^n{}_\xi}\,\text{DEF}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash F_\xi : (\check{A}_\xi \oplus F) \bullet s^{n+1}}\,\text{DEF} \qquad \dfrac{\dfrac{\vdots\,\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash \check{a} : \check{A}_\xi \qquad \dfrac{\dfrac{\vdots\,\{\Gamma \vdash F{\bullet}\check{a} : s^n\}_\xi}{\Gamma_\xi, z_0 \vdash F{\bullet}\check{a} : s^n}\,\text{ST}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash z_0 : F{\bullet}\check{a}}\,\text{DEF}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash z_0 : F{\bullet}(\check{a}_\xi/01...1)}\,\text{DEF}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash (\check{a}_\xi \oplus z_0) : (\check{A}_\xi \oplus F)}\,\text{Rel-E}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash F_\xi{\bullet}(\check{a}_\xi \oplus z_0) : s^{n+1}}\,\text{DEF}}{\Gamma_\xi, z_0 : F{\bullet}\check{a} \vdash z_0 \in F{\bullet}\check{a}_\xi : s^{n+1}}\,\text{DEF}$$

Rel-Intro Absurd: the type is a relation ($\check{A} \overset{\bullet}{\to} s^n$), which cannot be a sort.

Rel-Form

$$\frac{\Gamma \vdash \check{A} : {s_1}^n}{\Gamma \vdash (\check{A} \bullet {s_1}^n) : {s_2}^n}\,\text{Rel-F}$$

$$\dfrac{\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma_\xi, z_0 : (\check{A} \bullet {s_1}^n) \vdash \check{A} : {s_1}^n{}_\xi \quad \Gamma_\xi, z_0 : (\check{A} \bullet {s_1}^n) \vdash z_0 : \check{A} \bullet {s_1}^n}\,\text{DEF}}{\Gamma_\xi, z_0 : (\check{A} \bullet {s_1}^n) \vdash (\check{A}_\xi \oplus z_0) : {s_1}^{n+1}}}{\Gamma_\xi, z_0 : (\check{A} \bullet {s_1}^n) \vdash (\check{A}_\xi \oplus z_0) \bullet {s_1}^{n+1} : {s_2}^{n+1}}\,\text{Rel-F}}{\Gamma_\xi, z_0 : (\check{A} \bullet {s_1}^n) \vdash z_0 \in \check{A} \bullet {s_1}^n{}_\xi : {s_2}^{n+1}}\,\text{DEF}$$

Application

$$\dfrac{\dfrac{\Gamma \vdash \bar{A} : {s_1}^m}{\Gamma \vdash F : \bar{A} s^n}\,\text{gen} \qquad \Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash F\,\bar{A} : s^n}\,\text{APP}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots \vdash F : \bar{A} s^n : {s^{n+1}}_\xi}{\Gamma_\xi \vdash F_\xi : F \in \bar{A} s^n{}_\xi}\,\text{DEF} \qquad \vdots \vdash \bar{a} : \bar{A} : {s^n}_\xi}{\Gamma_\xi \vdash F_\xi : \bar{a} \in \bar{A}_\xi F\,\bar{a} \in {s^n}_\xi}\,\text{DEF} \qquad \Gamma_\xi \vdash \bar{a}_\xi : \bar{a} \in \bar{A}_\xi}{\Gamma_\xi \vdash F_\xi\,\bar{a}_\xi : F\,\bar{a} \in {s^n}_\xi}\,\text{APP} \qquad \dfrac{\vdots \{\vdash F\,\bar{a} : s^n\}_\xi}{\Gamma_\xi \vdash F\,\bar{a} : s^n}\,\text{ST}}{\Gamma_\xi \vdash F_\xi\,\bar{a}_\xi : F\,\bar{a} \bullet s^{n+1}}\,\text{DEF}}{\Gamma_\xi, x : F\,\bar{a} \vdash F_\xi\,\bar{a}_\xi\,x : s^{n+1}}\,\text{Rel-E}}{\Gamma_\xi, x : F\,\bar{a} \vdash x \in F\,\bar{a}_\xi : s^{n+1}}\,\text{DEF}$$

Abstraction Absurd.

Product

$$\dfrac{\Gamma \vdash \bar{A} : {s_1}^m \quad \Gamma, x : \bar{A} \vdash B : {s_2}^n}{\Gamma \vdash (\bar{x} : \bar{A}.\,B) : {s_3}^{mn}}\,(s_1, s_2, s_3)$$

$$\dfrac{\dfrac{\dfrac{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B) \vdash \bar{A}_\xi : {s_1}^{1+m} \qquad \dfrac{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B), x : \bar{A}_\xi \vdash f\,x_0 : B \qquad \dfrac{\vdots |\Gamma, x : \bar{A} \vdash B : {s_2}^n|_{\xi,x}}{\Gamma_\xi, x : \bar{A}_\xi, z : B \vdash z \in B_{\xi,x} : {s_2}^{1+n}}\,\text{WK}}{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B), x : \bar{A}_\xi, z : B \vdash z \in B_{\xi,x} : {s_2}^{1+n}}\,\text{subst}}{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B), x : \bar{A}_\xi \vdash (f\,x_0) \in B_{\xi,x} : {s_2}^{1+n}}\,(s_1, s_2, s_3)}{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B) \vdash (\bar{x} : \bar{A}_\xi.\,(f\,x_0) \in B_{\xi,x}) : {s_3}^{1+mn}}\,\text{DEF}}{\Gamma_\xi, f : (\bar{x} : \bar{A}.\,B) \vdash f \in \bar{x} : \bar{A}.\,B_\xi : {s_3}^{1+mn}}\,\text{DEF}$$

Conversion

$$\frac{\Gamma \vdash B : s^n \quad \Gamma \vdash s^n : s^{n+1} \quad s^n =_\beta s^n}{\Gamma \vdash B : s^n}\ \textsc{conv}$$

Trivial.

Param Absurd: the type of the parametricity witness is $z \in [\![B]\!]_{\xi}$, which cannot be a sort s^n. □

Theorem 7 (Soundness)**.** *If* $\Gamma \vdash_{\mathscr{P}} A : B$*, then*

$$\langle\!|\Gamma|\!\rangle \vdash_{\mathscr{O}} \langle\!|A|\!\rangle : \langle\!|B|\!\rangle.$$

Proof. We proceed by induction on the derivation; however the proof requires a stronger induction hypothesis when the derivation $\Gamma \vdash_{\mathscr{P}} A : B$ starts with the Application rule, hence we generalize the statement as follows:

> Let $\Gamma \vdash_{\mathscr{P}} A : B : s^n$, k such that $k \leq \epsilon(x)$ for each free variable x, and $\pi \in \mathfrak{S}_{n+k}$. Then
>
> $$\langle\!|\Gamma|\!\rangle \vdash_{\mathscr{O}} \langle\!|[\![A]\!]^k \ddagger^\pi|\!\rangle : \langle\!|(A \in [\![B]\!]^k) \ddagger^\pi|\!\rangle \qquad (1)$$

However, for the sake of readability we only prove the specialized statement

$$\Gamma \vdash A : B \Longrightarrow \langle\!|\Gamma|\!\rangle \vdash_{\mathscr{O}} \langle\!|A|\!\rangle : \langle\!|B|\!\rangle$$

(The proof for (1) stems from an additional decreasing induction on $k \leq \cap_{x \text{ free}} \epsilon(x)$.)

Axiom
Trivial.

Weakening

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle : \langle\!|B|\!\rangle} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash C : s^n}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|C|\!\rangle : s}}{\langle\!|\Gamma|\!\rangle, \langle\!|x_i : C|\!\rangle \text{ legal}}\ \text{Lemma 12}_{\text{iii}}}{\langle\!|\Gamma|\!\rangle, \langle\!|x_i : C|\!\rangle \vdash \langle\!|A|\!\rangle : \langle\!|B|\!\rangle}\ \text{Thinning}$$

Application (Rel-Elim **is similar**)

$$\text{by def.}\ \dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash F : (\forall \bar{x} : \bar{A}.\, B)}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F|\!\rangle : (\forall \langle\!|\bar{x} : \bar{A}|\!\rangle .\, \langle\!|B|\!\rangle)}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F|\!\rangle : (\forall \{x^{\pi}_{ji} : \ldots\}.\, \langle\!|B|\!\rangle)} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \bar{a} : \bar{A}}{\Gamma \vdash a_i : A_i\bullet(\bar{x}/i)}\ \text{by def.}}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|[\![a_i]\!]^{|j|} \ddagger^{\pi}|\!\rangle : \langle\!|(a_i \in [\![A_i\bullet(\bar{x}/i)]\!]^{|j|}) \ddagger^{\pi}|\!\rangle}\ \text{(many-)App.}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F|\!\rangle \{\langle\!|[\![a_i]\!]^{|j|} \ddagger^{\pi}|\!\rangle \mid \ldots\} : \langle\!|B|\!\rangle[\langle\!|[\![a_i]\!]^{|j|} \ddagger^{\pi}|\!\rangle / x^{\pi}_{ji}, \ldots]}\ \text{by def., Lem. 15}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|F\,\bar{a}|\!\rangle : \langle\!|B[\bar{a}/\bar{x}]|\!\rangle}$$

Abstraction (Rel-Intro **is similar**)

$$\text{by def.}\ \dfrac{\dfrac{\dfrac{\Gamma, \bar{x} : \bar{A} \vdash b : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \langle\!|\bar{x} : \bar{A}|\!\rangle \vdash \langle\!|b|\!\rangle : \langle\!|B|\!\rangle} \quad \vdots}{\langle\!|\Gamma|\!\rangle \vdash (\lambda \langle\!|\bar{x} : \bar{A}|\!\rangle .\, \langle\!|b|\!\rangle : (\forall \langle\!|\bar{x} : \bar{A}|\!\rangle .\, \langle\!|B|\!\rangle)}\ \text{(many-)Abs.}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|\lambda \bar{x} : \bar{A}.\, b|\!\rangle : \langle\!|\forall \bar{x} : \bar{A}.\, B|\!\rangle}$$

PRODUCT **(REL-FORM is similar)**

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \bar{A} : s_1^m}{\Gamma, \ldots \vdash A_i\bullet(\bar{x}/i) : s_1^m}}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \ldots \vdash \langle\!|(x_i \in [\![A_i\bullet(\bar{x}/i)]\!]^{|j|}) \ddagger^{\pi}|\!\rangle : s_1} \qquad \dfrac{\dfrac{\Gamma, \bar{x} : \bar{A} \vdash B : s_2^n}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \langle\!|\bar{x} : \bar{A}|\!\rangle \vdash \langle\!|B|\!\rangle : s_2}}{\dfrac{\langle\!|\Gamma|\!\rangle \vdash (\forall\{x_{ji}^{\pi} : \langle\!|(x_i \in [\![A_i\bullet(\bar{x}/i)]\!]^{|j|}) \ddagger^{\pi}|\!\rangle \mid \ldots\}.\, \langle\!|B|\!\rangle) : s_3}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|\forall \bar{x} : \bar{A}.\, B|\!\rangle : s_3}\ \text{by def.}}\ \text{(many-)PROD.}$$

CONVERSION

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A : B}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle : \langle\!|B|\!\rangle} \qquad \dfrac{\dfrac{\Gamma \vdash B' : s^n}{\text{induction}}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|B'|\!\rangle : s} \qquad \dfrac{\dfrac{B =_\beta B'}{\text{Theorem 3, Lemma 16}}}{\langle\!|B|\!\rangle =_\beta \langle\!|B'|\!\rangle}}{\langle\!|\Gamma|\!\rangle \vdash \langle\!|A|\!\rangle : \langle\!|B'|\!\rangle}\ \text{CONV.}$$

START, PARAM, EXCHANGE

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A : s^m}{\text{induction}}}{\langle\!|\Gamma|\!\rangle, \langle\!|x_i : A|\!\rangle \text{ legal}}}{\langle\!|\Gamma|\!\rangle, \langle\!|x_i : A|\!\rangle \vdash x_{ji}^{\pi} : \langle\!|(x_i \in [\![A]\!]^n) \ddagger^{\pi}|\!\rangle}\ \text{Thinning, START}}{\langle\!|\Gamma|\!\rangle, \langle\!|x_i : A|\!\rangle \vdash \langle\!|[\![x_i]\!]^n \dagger^{\pi}|\!\rangle : \langle\!|(x_i \in [\![A]\!]^n) \ddagger^{\pi}|\!\rangle}\ \text{by def.}$$

Appendix B

Definition of CCCC

Definition 1 (Syntax)**.**

Variable	$\ni x, y, z$			
Color	$\ni i, j$			
Sort	$\ni s$	$\stackrel{\text{def}}{=}$	$\star_\theta \mid \Box_\theta$	
Taint	$\ni \theta, \iota$	$\stackrel{\text{def}}{=}$	$\emptyset$	*empty*
		$\mid$	θ, i	*tainted*
Modality	$\ni \psi, \varphi$	$\stackrel{\text{def}}{=}$	(θ, ι)	
Term	$\ni A, \ldots, Z$	$\stackrel{\text{def}}{=}$	x	*variable*
	a, b, c, t, u	$\mid$	s	*sort*
		$\mid$	$(x :_\psi A) \to B$	*product*
		$\mid$	$\lambda x :_\psi A.b$	*abstraction*
		$\mid$	$F \bullet_\psi a$	*application*
Context	$\ni \Gamma, \Delta$	$\stackrel{\text{def}}{=}$	$\diamond$	*empty*
		$\mid$	$\Gamma, x :_\psi A$	*binding*
		$\mid$	Γ, i	*color*

Definition 2 (Typing rules $\boxed{\Gamma \vdash A :_\theta B}$)**.**

$\Gamma \vdash A :_\theta B$ *is well-formed only if* $i \in \Gamma$ *for each* $i \in \theta$.

$$\textsc{Conv}\ \frac{\Gamma \vdash a :_\theta A \qquad A =_\beta A'}{\Gamma \vdash a :_\theta A'} \qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star_\theta :_\theta \Box_\theta} \qquad \textsc{Var}\ \frac{\vdash \Gamma \qquad x :_\theta A \in \Gamma}{\Gamma \vdash x :_\theta A}$$

$$\textsc{Prod}\ \frac{\Gamma, x :_\psi A \vdash B :_\theta s}{\Gamma \vdash (x :_\psi A) \to B :_\theta s} \qquad \textsc{Abs}\ \frac{\Gamma, x :_\psi A \vdash b :_\theta B}{\Gamma \vdash (\lambda x :_\psi A.b) :_\theta (x :_\psi A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F :_\theta (x :_\psi A) \to B \qquad \Gamma \vdash u :_\psi A}{\Gamma \vdash F \bullet_\psi u :_\theta B[u/x]} \qquad \textsc{Param}\ \frac{\Gamma \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash A :_{\theta,i} B \bullet_{\theta,\dot{\imath}} \lfloor A \rfloor_i}$$

(To limit clutter we omit the well-sorted conditions of types A and B in the rule Abs*.) We also have*

$$\textit{If } \psi = (\theta, \iota) \textit{ then } \Gamma \vdash A :_\psi B \stackrel{\text{def}}{=} \lfloor \Gamma \rfloor_\iota \vdash A :_\theta B$$

Definition 3 (Accessible variable $\boxed{x :_\theta A \in \Gamma}$)**.**

$$\textsc{Start}\ \frac{}{x :_\theta A \in \Gamma, x :_{(\theta,\iota)} A} \qquad \textsc{Col. Wk}\ \frac{x :_\theta A \in \Gamma \qquad i \notin \theta}{x :_\theta A \in \Gamma, i} \qquad \textsc{Wk}\ \frac{x :_\theta A \in \Gamma}{x :_\theta A \in \Gamma, y :_\psi B}$$

Definition 4 (Well-formed contexts $\boxed{\vdash \Gamma}$)**.**

$$\textsc{Empty}\ \frac{}{\vdash \diamond} \qquad \textsc{Color}\ \frac{\vdash \Gamma}{\vdash \Gamma, i} \qquad \textsc{Bind}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A :_\psi s}{\vdash \Gamma, x :_\psi A}$$

Definition 5 (erasure $\boxed{\lfloor T \rfloor_i}$)**.** *The definition of erasure depends on the actual modality used. We write all the cases on the same line; the condition is written above each column.*

	$i \notin \psi$	$i \in \psi$	$\psi = \varphi, \dot{\imath}$
$\lfloor x \rfloor_i$	$= x$		
$\lfloor s \rfloor_i$	$= s$		
$\lfloor (x :_\psi A) \to B \rfloor_i$	$= (x :_\psi \lfloor A \rfloor_i) \to \lfloor B \rfloor_i$	$\lfloor B \rfloor_i$	$(x :_\varphi A) \to \lfloor B \rfloor_i$
$\lfloor \lambda x :_\psi A.b \rfloor_i$	$= \lambda x :_\psi \lfloor A \rfloor_i . \lfloor b \rfloor_i$	$\lfloor b \rfloor_i$	$\lambda x :_\varphi A . \lfloor b \rfloor_i$
$\lfloor F \bullet_\psi a \rfloor_i$	$= (\lfloor F \rfloor_i) \bullet_\psi \lfloor a \rfloor_i$	$\lfloor F \rfloor_i$	$(\lfloor F \rfloor_i) \bullet_\varphi a$
$\lfloor \Gamma, x :_\psi A \rfloor_i$	$= \lfloor \Gamma \rfloor_i, x :_\psi \lfloor A \rfloor_i$	$\lfloor \Gamma \rfloor_i$	$\lfloor \Gamma \rfloor_i, x :_\varphi A$
$\lfloor \Gamma, j \rfloor_i$	$= \lfloor \Gamma \rfloor_i, j$		
$\lfloor \Gamma, i \rfloor_i$	$= \Gamma$		

Erasure is extended to taints as follows:

Definition 6 (erasure $\boxed{\lfloor T \rfloor_\iota}$)**.**

$$\lfloor T \rfloor_\emptyset = T$$
$$\lfloor T \rfloor_{\iota,i} = \lfloor \lfloor T \rfloor_i \rfloor_\iota$$

Definition 7 (Reduction $\boxed{t \longrightarrow u}$)**.**

$$
\begin{aligned}
s_\theta \bullet_\varphi t &\longrightarrow (z :_\varphi t) \to s_{\theta \cup \iota} && (1)\\
&\quad \textit{where } \varphi = (\theta, \iota) \\
((x :_\psi A) \to B) \bullet_\varphi t &\longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) && (2)\\
&\quad \textit{if } \exists i \textit{ such that } i \in \psi \textit{ and } \not{i} \in \varphi \\
((x :_\psi A) \to B) \bullet_\varphi t &\longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) && (3)\\
&\quad \textit{otherwise} \\
(\lambda x :_\psi A.b) \bullet_\psi t &\longrightarrow b[t/x] && (4)
\end{aligned}
$$

and congruences.

Definition 8 (Substitution)**.** *The substitution of variables in i-oblivious contexts erases i from the substitutees.*

$$
\begin{aligned}
(F \bullet_\psi a)[u/x] &= F[u/x] \bullet_\psi a[u\{\psi\}/x] \\
((y :_\psi A) \to B)[u/x] &= (y :_\psi A[u\{\psi\}/x]) \to B[u/x] \\
(\lambda y :_\psi A.b)[u/x] &= \lambda y :_\psi A[u\{\psi\}/x].b[u/x]
\end{aligned}
$$

Where

$$u\{(\theta, \iota)\} = \lfloor u \rfloor_\iota$$

Appendix C

Proof details for chapter 2

Lemma 1 (Substitution)**.** *For any term A, u, and v and variables $x \neq y$ such that x is not free in v,*

$$A[u/x][v/y] = A[v/y][u[v/y]/x]$$

Proof. By structural induction on the raw term A. CCCC does not support abstraction over colors, therefore we can ignore the case where x or y are color variables, and the proof below follows exactly the structure of the usual proof of substitution lemma for PTSs. We show only the variable and abstraction cases; other cases are similar.

Variable z As usual, we have the three following cases:

- $z = x$:

$$x[u/x][v/y] = u[v/y] = x[u[v/y]/x] = x[v/y][u[v/y]/x]$$

- $z = y$: $y[u/x][v/y] = v = y[v/y][u[v/y]/x]$
- otherwise: $z[u/x][v/y] = z = z[v/y][u[v/y]/x]$

Abstraction $\lambda z :_\psi A.b$

$$\begin{aligned}
& (\lambda z :_\psi A.b)[u/x][v/y] \\
= \;& (\lambda z :_\psi A[u\{\psi\}/x].b[u/x])[v/y] \\
= \;& \lambda z :_\psi A[u\{\psi\}/x][v\{\psi\}/y].b[u/x][v/y] \\
& \text{by IH} \\
= \;& \lambda z :_\psi A[v\{\psi\}/y][u\{\psi\}[v\{\psi\}/y]/x]. \\
& \qquad\qquad b[v/y][u[v/y]/x] \\
= \;& (\lambda z :_\psi A.b)[v/y][u[v/y]/x]
\end{aligned}$$

□

We proceed to show the confluence of the reduction relation. To do this, we use the Tait/Martin-Löf technique of parallel reduction.

Definition 1 (Parallel nested reduction)**.**

$$\textsc{Refl}\ \frac{}{A \rhd A} \qquad \beta\ \frac{b \rhd b' \qquad a \rhd a'}{(\lambda z :_\psi A.b) \bullet_\psi a \rhd b'[a'/z]}$$

$$\textsc{AppSort}\ \frac{t \rhd t' \qquad \varphi = (\theta, \iota)}{s_\theta \bullet_\varphi t \rhd (z :_\varphi t') \to s_{\theta \cup \iota}}$$

$$\textsc{AppAll}_1\ \frac{\exists i \text{ such that } i \in \psi \text{ and } \dot{\iota} \in \varphi \qquad A \rhd A' \qquad B \rhd B' \qquad t \rhd t}{((z :_\psi A) \to B) \bullet_\varphi t \rhd (z :_\psi A') \to (B' \bullet_\varphi t')}$$

$$\textsc{AppAll}_2\ \frac{\nexists i \text{ such that } i \in \psi \text{ and } \dot{\iota} \in \varphi \qquad A \rhd A' \qquad B \rhd B' \qquad t \rhd t'}{((z :_\psi A) \to B) \bullet_\varphi t \rhd (z :_\psi A') \to (B' \bullet_\varphi (t' z))}$$

$$\textsc{App-Cong}\ \frac{F \rhd F' \qquad a \rhd a'}{F \bullet_\psi a \rhd F' \bullet_\psi a'} \qquad \textsc{Abs-Cong}\ \frac{A \rhd A' \qquad b \rhd b'}{\lambda z :_\psi A.b \rhd \lambda z :_\psi A'.b'}$$

$$\textsc{All-Cong}\ \frac{A \rhd A' \qquad B \rhd B'}{(z :_\psi A) \to B \rhd (z :_\psi A') \to B'}$$

Since one needs to erase the substitutee when substituting under an oblivious binding (see Definition 8), we use the fact that erasure preserves parallel reduction.

Lemma 2. *For each A,A' such that $A \rhd A'$, we have $\lfloor A \rfloor_i \rhd \lfloor A' \rfloor_i$ for all i.*

Proof. By induction on the derivation $A \rhd A'$. □

We can now prove that substitution preserves parallel reduction.

Lemma 3. *For each A,A' and u,u' such that $A \rhd A'$ and $u \rhd u'$, we have $A[u/x] \rhd A'[u'/x]$.*

Proof. By induction on the derivation $A \rhd A'$. The proof is almost completely standard, except for the use of Lemma 2. In addition of the β case which uses it, we show the Refl case for reference. Other cases are similar or standard.

Refl: $A \rhd A$. We get $A[u/x] \rhd A[u'/x]$ by structural induction on A.

β: $(\lambda z :_\psi A.b) \bullet_\psi a \rhd b'[a'/z]$.

$$
\begin{array}{ll}
 & ((\lambda z :_\psi A.b) \bullet_\psi a)[u/x] \\
= & (\lambda z :_\psi A[u\{\psi\}/x].b[u/x]) \bullet_\psi a[u\{\psi\}/x] \\
\text{by IH and Lemma 2} & \\
\rhd & b[u'/x][a'[u'\{\psi\}/x]/z] \\
= & b'[u'/x][a'[u'/x]/z] \\
\text{by Lemma 1} & \\
= & b'[a'/z][u'/x]
\end{array}
$$

AppSort: $s_\theta \bullet_\varphi t \rhd (z :_\varphi t') \to s_{\theta \cup \iota}$.

$$
\begin{array}{ll}
 & (s_\theta \bullet_\varphi t)[u/x] \\
= & s_\theta \bullet_\varphi t[u\{\varphi\}/x] \\
\text{by IH and Lemma 2} & \\
\rhd & (z :_\varphi t'[u'\{\varphi\}/x]) \to s_{\theta \cup \iota} \\
= & ((z :_\varphi t') \to s_{\theta \cup \iota})[u'/x]
\end{array}
$$

AppAll$_1$: $((z :_\psi A) \to B) \bullet_\varphi t \rhd (z :_\psi A') \to (B' \bullet_\varphi t')$.

$$
\begin{array}{ll}
 & (((z :_\psi A) \to B) \bullet_\varphi t)[u/x] \\
= & ((z :_\psi A[u\{\psi\}/x]) \to B[u/x]) \bullet_\varphi t[u\{\varphi\}/x] \\
\text{by IH and Lemma 2} & \\
\rhd & (z :_\psi A'[u'\{\psi\}/x]) \to B'[u'/x] \bullet_\varphi t'[u'\{\varphi\}/x] \\
= & (z :_\psi A'[u'\{\psi\}/x]) \to (B' \bullet_\varphi t')[u'\{\varphi\}/x] \\
= & ((z :_\psi A') \to (B' \bullet_\varphi t'))[u'/x]
\end{array}
$$

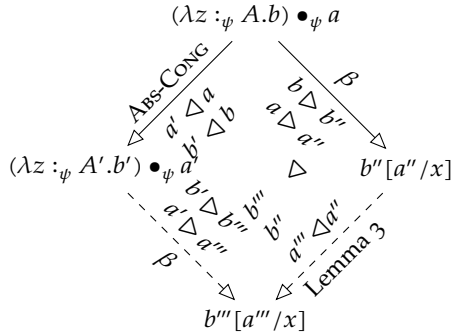
AppAll$_2$ is similar.

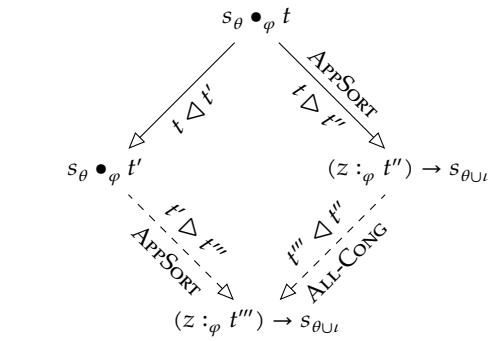
$\star$-cong Trivial. □

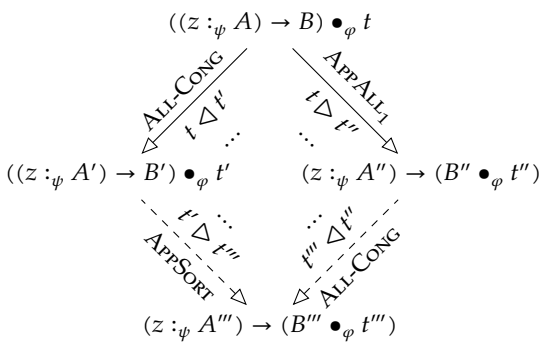
Theorem 1 (Diamond)**.** *The rewriting system ($\rhd$) has the diamond property. That is, for each A, B, B' such that $B \lhd A \rhd B'$, there exists C such that $B \rhd C \lhd B'$*

Proof. By induction on the derivations.

- If one of the derivations ends with Refl, one has either $A = B$, or $A = B'$. We pick $C = B'$ in the former case and $C = B$ in the latter.
- If one of the derivations ends with App-Cong, the other one has to end with App-Cong, β, AppSort, AppAll$_1$, or with AppAll$_2$. The first case is straightforward. In the other cases, the diverging reductions meet as shown below:







(The case of AppAll$_2$ is similar.)

- If both derivations end with the same rule β, AppSort, AppAll$_1$, or with AppAll$_2$, the result is a straightforward use of the induction hypothesis (using Lemma 3 in the case of β).
- If one of the derivations ends with Abs-Cong or All-Cong, the other one has to end with the same rule, and the result is a straightforward use of the induction hypothesis. □