Fresh Parametricity

ERIC BOND, University of Michigan, USA
MAX NEW, University of Michigan, USA

Recent works combining parametric polymorphism with language features relying on intensional type analysis have lead to non-standard definitions of parametricity. The relative strength of traditional vs these type-world notions of parametricity remains an open question. We shed some light on this issue by framing this question as a matter of translation between logics. To this end, we define two logics for proving theorems about parametricity, one traditional and one based on fresh parametricity, and we demonstrate a translation which preserves logical equivalence.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Eric Bond and Max New. 2018. Fresh Parametricity. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 20 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

<eric-rephrase "type instantiations"> <eric-rephrase "variant of System F"> <eric-fresh param instead of non-standard?> <eric-consistency in inclusion/exclusion of stoup>

1.1 Parametricity

<eric-this is word vomit. decide how you want to introduce parametrictiy, then work on flow>
Parametricity states <eric-wording> that any term of a universal or existential type must behave uniformly for any possible type instantiation. Existential types can be used to encode abstract data types like Queues or Graphs. If we know our language enjoys parametric polymorphism, then we have a guarantee that any two instance of a Queue, as encoded by an existential type, are observationally equivalent. Meaning, we can swap implementations of the Queue data type without changing the correctness of the overall program. (ignoring resource constraints, time/space) Universal types in a parametrically polymorphic language allow for a sound implementation of church encoded data. Theorems for free <eric- Address this first, then provide examples?>

Authors' Contact Information: Eric Bond, bonderic@umich.edu, University of Michigan, Ann Arbor, USA; Max New, University of Michigan, Ann Arbor, USA.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1.2 Parametricity vs Intensional Type Analysis

<eric-Dyn or ?> While parametricity is a desirable property, it is hard to maintain with certain language features that use *intensional type analysis* [5]. By Intensional type analysis, we mean the ability to inspect the run time representaiton of a type and dispatch based on the result. <eric-wording> An example of this behavior is the casting rules in a gradual typed language[11].

1.2.1 Gradual Typing. Gradual typing allows programmers gradually migrate from <eric-dynamically/untyped> code to typed code by including a type *Dyn* representing a dynamically checked term. The interface between dynamically typed code and statically typed code is handled by casting. A value of any type A can be up cast $\uparrow^?_A$ to the dynamic type. Downcasting $\downarrow^?_A$ from the dynamic type can introduce errors if the value stored in Dyn is not of the requested type. Naively adding Dyn, up casting, and down casting to a parametrically polymorphic language breaks parametricity. Consider a polymorphic gradually typed language which can error.

$$t : \forall X.X \to \mathbb{B} := (\Lambda X.\lambda(x : X).\ \downarrow^?_{\mathbb{B}} \uparrow^?_X (x))$$

<eric-$\mathbb{B}$ and Bool used here> If we assume that t behaved uniformly for all possible type instantiations, then t should either error or return a boolean that is not dependent on the argument of type X. But this is not the case when instantiated at type Bool, t will return the given boolean value $t[\mathbb{B}](b : \mathbb{B}) \rightsquigarrow^* b$ for any other type A it will error $t[A](x : A) \rightsquigarrow^* \Omega$ Therefore t does not behave uniformly for all types. The issue here is that casting inspects the type tag used in the values stored in dyn, a case of intensional type analysis.

1.3 Preserving Parametricity

The difficulties of combining parametric polymorphism and language features relying on type analysis are well known[10][11][13][1]. Solutions that preserve parametricity usually involve a form of *dynamic sealing* or generation of *fresh type tags*. Reason being, the ability to inspect a type and dispatch based on the result is in direct conflict with the information hiding principle provided in a language with parametric polymorphism. By hiding information about the runtime representation of a type, we can restore the property that polymorphic types behave uniformly across all type instantiations.

While such techniques have demonstrated they can preserve the expected information hiding properties, they do so using a *non-standard* formulation of parametricity. Specifically, the approach used by Niels[10] and New[11] invoke a *Type-World* logical relation where the freshly allocated type variables/tags, α, are associated with concrete types, (A, A'), and a relation, $R : Rel[A, A']$, on those types. This mapping, $\alpha \mapsto (A, A', R)$, of dynamically allocated type variable to concrete types and a relation are threaded through the logical relation via a Kripke world. A question remains, what is the relative strength of this non-standard formulation of parametricity compared to traditional definitions? In particular, does the *type-world* formulation allows us to prove the *free-theorems* and *data-abstraction* properties we would should expect?<eric-introduce vocab? non-starndard param as type world param?>

1.4 Failure of Full Abstraction

This was partially answered in the affirmative by Nies et al[10]. The argument proceeds by introducing two languages: An effectful variant of CBV System F with a standard notion of parametricity, and System G, an extension of the former language with type casts and fresh type name allocation supporting a *non-standard*, type world, formulation of parametricity. To compare the languages, they provide a type preserving embedding, $[\![\_]\!]$, from System F to System G.

$$\vdash_F V : A \implies \vdash_G [\![V]\!] : [\![A]\!]$$

The embedding, $[\![\vdash_F V : A]\!] = \mathcal{W}_A \circ [\![\vdash_F V : A]\!]^L$, is decomposed into a lifting, $[\![\_]\!]^L$, and a type directed wrapping function $\mathcal{W}_A$. The lifting preserves syntactic equality of terms and types, so $V \equiv [\![V]\!]^L$ and $A \equiv [\![A]\!]^L$. The wrapping function is a deep η expansion which strategically inserts fresh type allocation in the elimination forms of the universal and existential types. The purpose of the wrapping is to ensure that lifted polymorphic terms of System F are simultaneously safe from non-parametric uses and forced to behave parametrically in the presence of System G's type cast operation. More details can be found in Section 3.3.

Conjecture 1.1. ***False**: Full Abstraction of* $[\![\_]\!]$

$$\forall V_1, V_2. V_1 \cong_F V_2 \iff [\![V_1]\!] \cong_G [\![V_2]\!]$$

To demonstrate the relative strengths of the different notions of parametricity, they conjecture that the embeddings is *fully-abstract*, preserving and reflecting all contextual equivalences. However, this conjecture was later disproven[3] [4]. In particular, it was shown that contextual equivalence is not preserved by the translation. Specifically, the type $Univ := \exists Y. \forall X. (X \to Y) \times (Y \to X)$ must be *degenerate* in effectful System F, but there are non-degenerate inhabitants of this type in System G. <eric-provide explicit counter?>

1.5 Our Result

<eric-rename section, more connective tissue> <eric-TODO: more precision is needed here. Equivalence vs preservation of equivalence vs logical full abstraction which requires Reflection!> Our aim is to demonstrate that, while the transformation does not preserve contextual equivalence, it preserves a weaker notion we call *Logical Equivalence*. By Logical Equivalence, we mean to say that any results proven in a *Parametricity Logic*[12][2][8] for effectful System F can be transfered to a parametricity logic backed by the non-standard, type-world, notion of parametricity. By limiting our reasoning principles to a well-defined parametricity logic, we rule out metatheoretical reasoning which was used in prior work[4] to demonstrate that the type $Univ$ is degenerate. We demonstrate that the lack of metatheoretical reasoning principles does not preclude our logic from proving useful properties about data abstraction and representation independence. <eric-hrm.. but we don't include existentials here..>

Theorem 1.2. *Preservation of Logical Equivalence (simplified):*

$$\Gamma \vdash_{PL} M = N \implies [\![\Gamma]\!] \vdash_{FPL} [\![M]\!] = [\![N]\!]$$

1.6 Overview

<eric-bullet format or paragraph?> To demonstrate the preservation of logical equivalence we first define $\mathbf{F}^{\forall\to}_{\mu,\Omega}$, an effectful CBV variant of System F in Section 2.1 for which we want our theorem to apply. We then define two logics: **Parametricity Logic**(**PL**) and **Fresh Parametricity Logic**(**FPL**). The *term languages* of these logics, defined in Section 2.2, are effectful, CBPV [6] variants of System F. Embeddings of $\mathbf{F}^{\forall\to}_{\mu,\Omega}$ into **PL**and **FPL**are covered in <eric-cref> . The embeddings largely follow the usual[6] CBV to CBPV translation, with a few exceptions, notably the parametricity preserving wrapping in Section 3.3. In Section 4, we will introduce our parametricity logics<eric-cref> and the translation <eric-cref> between the logics. <eric-compositionality issues.. oblivious predicate..adequacy..example proof in Logic..discussion/relatedwork.. order TBD here.. come back to this>

2 LANGUAGES

- section overview.

- why have a source language
- why CBPV

<eric-redo> In this section we introduce $\mathbf{F}^{\forall\to}_{\mu,\Omega}$, an effectful variant of CBV System F, two versions of an effectful CBPV variant of System F with and without fresh case allocation. <eric-more exposition>

2.1 Source Language

- why polymorphic multi-arg function type?
- why general recursion and error? (OSum can encode these)

$\mathbf{F}^{\forall\to}_{\mu,\Omega}$ is mostly a standard polymorphic, CBV language with general recursion and the ability to raise errors. One major difference being that we replace the universal quantification and function type with a polymorphic, multi-argument function type, similar to that of λ^K[9]. Notably, this type does not support partial application; a trade-off we accept for finer control over equational reasoning, as discussed in Section 3.1.

$$\frac{}{\Gamma, X \vdash X\ Type} \qquad \frac{}{\Gamma \vdash \mathbb{B}\ Type} \qquad \frac{}{\Gamma \vdash \mathbb{N}\ Type} \qquad \frac{\Gamma \vdash A\ Type \qquad \Gamma \vdash A'\ Type}{\Gamma \vdash A \times A'\ Type}$$

$$\frac{\Gamma, \vec{X}_n \vdash A_i\ Type \quad \forall i \in \{m\} \qquad \Gamma, \vec{X}_n \vdash A\ Type}{\Gamma \vdash \forall[\vec{X}_n].(\vec{A}_m) \to A\ Type}$$

Fig. 1. Type Formers: $\mathbf{F}^{\forall\to}_{\mu,\Omega}$

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash true : \mathbb{B}} \qquad \frac{}{\Gamma \vdash false : \mathbb{B}} \qquad \frac{\Gamma \vdash b : \mathbb{B} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash rec_{\mathbb{B}}\ b\ \{M \mid N\} : A} \qquad \frac{}{\Gamma \vdash z : \mathbb{N}}$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash s\ n : \mathbb{N}} \qquad \frac{\Gamma \vdash n : \mathbb{N} \qquad \Gamma \vdash M : A \qquad \Gamma, x : \mathbb{N} \vdash N : A}{\Gamma \vdash rec_{\mathbb{N}}\ n\ \{M \mid x.\ N\} : A}$$

$$\frac{\Gamma, \vec{X}_n \vdash A_i\ Type \quad \forall i \in \{m\} \qquad \Gamma, \vec{X}_n \vdash A'\ Type \qquad \Gamma, \vec{X}_n, x_1 : A_1, \cdots, x_m : A_m \vdash M : A'}{\Gamma \vdash \Lambda[\vec{X}_n](x_1 : A_1, \cdots, x_m : A_m).M : \forall[\vec{X}_n].(\vec{A}_m) \to A'}$$

$$\frac{\Gamma \vdash A_i\ Type \quad \forall i \in \{n\} \qquad \Gamma \vdash M : \forall[\vec{X}_n].(\vec{A}_m) \to A' \qquad \Gamma \vdash N_i : B_i[\vec{A}/\vec{X}]\ \forall i \in \{m\}}{\Gamma \vdash M[\vec{A}_n](N_1 : B_1, \cdots N_m : B_m) : A'}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash (M, N) : A \times A} \qquad \frac{\Gamma \vdash M : A \times A' \qquad \Gamma, x : A, y : A' \vdash N : A}{\Gamma \vdash rec_{\times}\ M\ \{x, y.\ N\} : A''} \qquad \frac{}{\Gamma \vdash \Omega_A : A} \qquad \frac{\Gamma, x : A \vdash V : A'}{\Gamma \vdash \mu x.V : A'}$$

Fig. 2. Typing Rules:$\mathbf{F}^{\forall\to}_{\mu,\Omega}$

2.2 PL & FPL Term Languages

- remark: mixed term/type context
- remark: how *new* is presented as an effect? (its church encoding)
- remark: exclusion of $\underline{B} \multimap \underline{B}'$ type
- <eric-TODO: rephrase "object language">

Here we introduce the term languages for **PL** and **FPL**. These are based on a polymorphic CBPV calculus with an error effect and general recursion. We include an observation type, *Obs*, with no elimination form. Two notable difference are the inclusion of an extensible sum type *OSum* and absence of the computation type *FA*.

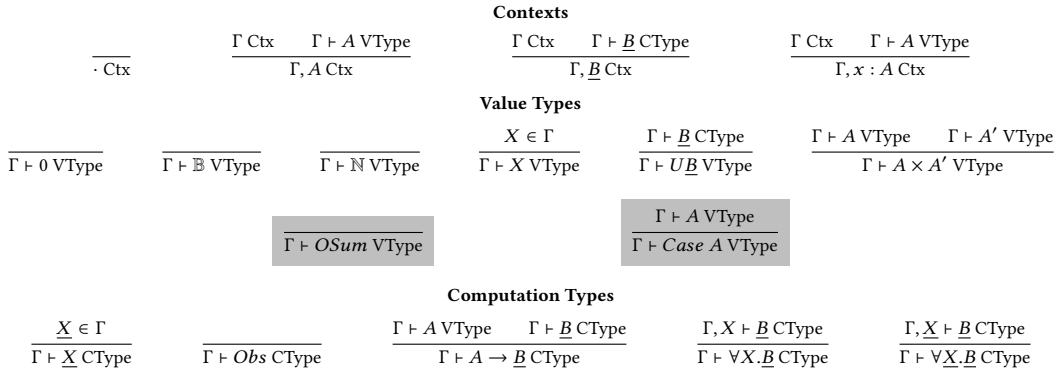

Fig. 3. Contexts, Value Types, and Computation Types: **PL** & **FPL**

2.2.1 Extensible Sum Type. In order to study parametricity in the presence of type analysis, we include an extensible sum type, *OSum*, along with the type *Case A* as a proxy for the dynamic type and first class representaiton of type tags. Thus, instead of including a type case operation ala System G[10], we adopt a form of type analysis used in gradual typing where type tags of the dynamic type can be inspected. Following prior work on combining gradual typing and parametricity [11], we reinterpret the universal type $\forall X.\underline{B}$ as the *fresh universal type* $\forall X.Case\ X \to \underline{B}$ in our type translation(**??**) between **PL** and **FPL**.

$$\sigma \leftarrow new_{\mathbb{B}}; ret\ inj_{\sigma}(true) : F(OSum)$$

In order to inject a value $V : A$ into *OSum*, we must first have a case symbol $\sigma : Case\ A$ for type A, which can be dynamically allocated using new_A. To eliminate $V : OSum$, we compare a given $\sigma : Case\ A$ to the case symbol which was used to construct the value V. When the cases match, we have a continuation, $x.\ M$, which receives the value which was stored in the sum. Otherwise, we proceed to a default computation N.

2.2.2 Church Encoded F. Following prior work on parametricity logics with effects[8], our calculi do not contain a first class computation type *FA*. Instead, *FA* is church encoded so that its relational interpretation is inherited from its component types. Unlike typical church encoded data, *FA* uses universal quantification over computation types rather than value types. Relational interpretation of types can be found in **??**

$$FA := \forall \underline{X}.U(A \to \underline{X}) \to \underline{X}$$
$$ret\ V : FA := \Lambda \underline{X}.\lambda k : U(A \to \underline{X}).(force\ k)V$$
$$(x \leftarrow M; N) : \underline{B} := M[\underline{B}](thunk(\lambda x : A.N))$$

Fig. 4. Church Encoding of *F*

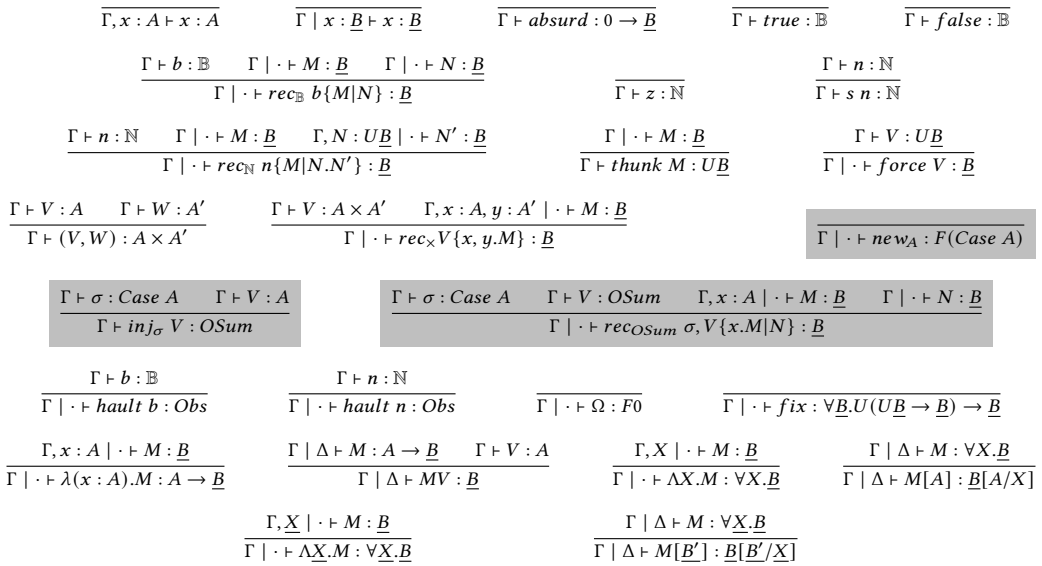

Fig. 5. Typing Rules: **PL** & **FPL** Term Languages

2.2.3 Equational Theory. By embedding $\mathbf{F}^{\forall\to}_{\mu,\Omega}$ into a CBPV term language, we gain access to a rich equational theory (Fig. 6) which can be used by our parametricity logics. The reader may notice that, while we do not include F, ret and $bind$ as primitives in the term language, we do list their equational rules. These rules are derivable *within the logic*, as demonstrated in prior work [7], using parametricity. An equation of critical importance to our result is *Drop*, stating: $M = \sigma \leftarrow new_X; M$. As we will see in later sections, preservation of parametricity in our setting requires inserting case allocations at each type applicaition in our term language. The translation of proofs from **PL**to **FPL** relies heavily on removing unused allocations symbols. <eric-bind rules proved internally to the logic, drop rule is critical, we also rely heavily on complex value equations>

β Laws

$$force(thunk\ M) = M$$
$$x \leftarrow ret\ V; M = M[V/x]$$
$$rec_{\mathbb{B}}\ true\ \{M \mid N\} = M$$
$$rec_{\mathbb{B}}\ false\ \{M \mid N\} = N$$
$$rec_{OSum}\ \sigma, (inj_{\sigma'}\ V)\ \{x.M \mid N\} = M[V/x] \text{ where } \sigma = \sigma'$$
$$rec_{OSum}\ \sigma, (inj_{\sigma'}\ V)\ \{x.M \mid N\} = N \text{ where } \sigma \neq \sigma'$$
$$rec_{\times}(V, W)\ \{x, y.\ M\} = M[V/x, W/y]$$
$$rec_{\mathbb{N}}\ z\ \{M \mid x.N\} = M$$
$$rec_{\mathbb{N}}\ (s\ n)\ \{M \mid x.N\} = N[thunk(rec_{\mathbb{N}}\ n\ \{M \mid x.N\})/x]$$
$$fix[\underline{B}](thunk(\lambda x : U\underline{B}.M)) = M[thunk(fix[\underline{B}](thunk(\lambda x.M)))/x]$$
$$(\lambda x.M)V = M[V/x]$$
$$(\Lambda X.M)[A] = M[A/X]$$
$$(\Lambda \underline{X}.M)[\underline{B}] = M[\underline{B}/\underline{X}]$$

η Laws

$$thunk(force\ V) = V \text{ where } V : U\underline{B}$$
$$x \leftarrow M; ret\ x = M \text{ where } M : FA$$
$$rec_{\mathbb{B}}\ V\ \{M[true/x] \mid M[false/x]\} = M[V/x]$$
$$rec_{\times} V\ \{x, y.\ M[(x, y)/z]\} = M[V/z]$$
$$\lambda(x : A).Mx = M \text{ where } M : A \to \underline{B}, x \notin fv(M)$$
$$\Lambda X.M[X] = M \text{ where } M : \forall X.\underline{B}, X \notin ftv(M)$$
$$\Lambda \underline{X}.M[\underline{X}] = M \text{ where } M : \forall \underline{X}.\underline{B}, \underline{X} \notin ftv(M)$$

Sequencing Laws

$$y \leftarrow (x \leftarrow M; N); P = x \leftarrow M; y \leftarrow N; Px \notin fv(P)$$
$$x \leftarrow M; (\lambda y.N) = \lambda y.(x \leftarrow M; N)\ y \notin fv(M)$$

Effect Laws

$$M = \sigma \leftarrow new_A; M\ \sigma \notin fv(M)$$

Fig. 6. Equational Theory: **PL** & **FPL**

3 TRANSLATIONS

In this section we cover the translations between the languages introduced in the prior section. We also introduce the concept of *wrapping* a term to ensure it behaves parametrically.<eric-more> <eric-placeholder figure, refactor>

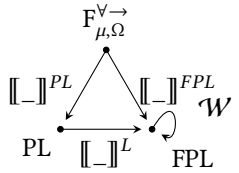

Fig. 7. Translations

3.1 Embedding $F^{\forall\to}_{\mu,\Omega}$ in PL & FPL

- explain: $[\![\_]\!]^B := [\![\_]\!]^L \circ [\![\_]\!]^A$

We embed $F^{\forall\to}_{\mu,\Omega}$ in **PL** using the standard CBV-to-CBPV translation[6] as a guide. One notable difference is the handling of our polymorphic, multi-argument function type. Rather than introducing intermediate thunks at every type and term abstraction, we only introduce one top level thunk. <eric-TODO: rethink motivation for the polymorphic multi-arg type and its translation> <eric-omit the full figure, provide a fragment>

$$
\begin{aligned}
[\![\Gamma \vdash X]\!] &= X \\
[\![\Gamma \vdash \mathbb{B}]\!] &= \mathbb{B} \\
[\![\Gamma \vdash \mathbb{N}]\!] &= \mathbb{N} \\
[\![\Gamma \vdash A \times A']\!] &= [\![\Gamma \vdash A]\!] \times [\![\Gamma \vdash A']\!]
\end{aligned}
$$

$$
[\![\Gamma \vdash \forall[\overrightarrow{X}_n].(\overrightarrow{A}_m) \to A']\!] = U\Big(\forall X_1 \cdots X_n.\, ([\![\Gamma, \overrightarrow{X} \vdash A_1]\!], \ldots, [\![\Gamma, \overrightarrow{X} \vdash A_m]\!]) \to F[\![\Gamma, \overrightarrow{X} \vdash A']\!]\Big)
$$

Fig. 8. Type Translation: $\mathbf{F}^{\forall \to}_{\mu,\Omega}$ to **PL**

$$
\begin{aligned}
[\![\Gamma \vdash x]\!]_c &= ret\ x \\
[\![\Gamma \vdash true]\!]_c &= ret\ true \\
[\![\Gamma \vdash fasle]\!]_c &= ret\ false \\
[\![\Gamma \vdash rec_{\mathbb{B}}\ M\ \{N, N'\}]\!]_c &= b \leftarrow [\![\Gamma \vdash M]\!]_c;\ rec_{\mathbb{B}}\ b\ \{\ [\![\Gamma \vdash N]\!]_c | [\![\Gamma \vdash N']\!]_c\} \\
[\![\Gamma \vdash z]\!]_c &= ret\ z \\
[\![\Gamma \vdash s\ M]\!]_c &= x \leftarrow [\![\Gamma \vdash M]\!]_c;\ ret(s\ x) \\
[\![\Gamma \vdash rec_{\mathbb{N}}\ M\ \{N \mid x : A.\ N'\} : A]\!]_c &= n \leftarrow [\![\Gamma \vdash M]\!]_c; \\
&\quad rec_{\mathbb{N}}\ n\ \{[\![\Gamma \vdash N]\!]_c \mid r : U[\![\Gamma \vdash A]\!].x \leftarrow force\ r;\ [\![\Gamma, x \vdash N']\!]_c\} \\
[\![\Gamma \vdash \Omega_A : A]\!]_c &= \Omega_{FA} \\
[\![\Gamma \vdash \mu x.M : A]\!]_c &= fix[FA](thunk(\lambda r : UF[\![\Gamma \vdash A]\!].x \leftarrow force\ r;\ [\![\Gamma, x \vdash M]\!]_c) \\
[\![\Gamma \vdash (M, N)]\!]_c &= \\
&\quad x \leftarrow [\![\Gamma \vdash M]\!]_c; \\
&\quad y \leftarrow [\![\Gamma \vdash N]\!]_c; \\
&\quad ret(x, y) \\
[\![\Gamma \vdash rec_{\times}\ M\ \{x, y.\ N\}]\!]_c &= m \leftarrow [\![\Gamma \vdash M]\!]_c;\ rec_{\times}\ m\ \{x, y.\ [\![\Gamma, x, y \vdash N]\!]_c\} \\
[\![\Gamma \vdash \Lambda[\overrightarrow{X}_n](\overrightarrow{x}_m).M]\!]_c &= ret(thunk(\\
&\quad \Lambda X_1.\Lambda X_2. \cdots \Lambda X_n. \\
&\quad \lambda(x : [\![\Gamma, \overrightarrow{X}_n \vdash A_1]\!]). \cdots \lambda(x_m : [\![\Gamma, \overrightarrow{X}_n \vdash A_m]\!]). \\
&\quad [\![\Gamma, \overrightarrow{X}_n, \overrightarrow{x}_m \vdash M]\!]_c)) \\
[\![\Gamma \vdash M[\overrightarrow{A}_n](\overrightarrow{M}_m)]\!]_c &= \\
&\quad m \leftarrow [\![\Gamma \vdash M]\!]_c; \\
&\quad m_1 \leftarrow [\![\Gamma \vdash M_1]\!]_c; \\
&\quad \cdots \\
&\quad m_m \leftarrow [\![\Gamma \vdash M_m]\!]_c; \\
&\quad (force\ m)[\![\Gamma \vdash A_1]\!] \cdots [\![\Gamma \vdash A_n]\!](m_1) \cdots (m_m)
\end{aligned}
$$

Fig. 9. Term Translation: $\mathbf{F}^{\forall \to}_{\mu,\Omega}$ to **PL**

3.2 Term Language Translation: PL to FPL

<eric-only present fragment of translaiton, full translation in appendix> Here we introduce the backbone of our proof translation which is the term language translation (Fig. 10) between **PL** and **FPL**.

3.2.1 Translating $\forall X.\underline{B}$. At the core of this translation, as alluded to in Section 2.2.1, is interpretation of $\forall X.\underline{B}$ as *fresh universal quantification* $\forall X.Case\ X \to \underline{B}$. By freshness, we mean to enforce an invariant that all case symbols passed into a universal type are freshly allocated. To demonstrate how this invariant enforces parametric behavior in the presence of *OSum* and *Case*, consider the following:

$$eq_{\sigma_{\mathbb{B}}} : \forall X.Case\ X \to F\mathbb{B}$$
$$\sigma_{\mathbb{B}} : Case\ \mathbb{B} \vdash eq_{\sigma_{\mathbb{B}}} := \Lambda X.\lambda \sigma_X : Case\ X.rec_{OSum}\ (inj_{\sigma_{\mathbb{B}}} true), \sigma_X\ \{x.\ ret\ true \mid ret\ false\}$$

The term $eq_{\sigma_{\mathbb{B}}}$ is a case equality test comparing any given $\sigma_X : Case\ X$ to a fixed $\sigma_{\mathbb{B}} : Case\ \mathbb{B}$. As any case symbol is only associated with one type, it is clear to see that $eq_{\sigma_{\mathbb{B}}}$ does not behave parametrically. Note that by ensuring $eq_{\sigma_{\mathbb{B}}}[\mathbb{B}]$ is only ever passed fresh cases, uniformity is restored.

$$\begin{aligned} eq_{\sigma_{\mathbb{B}}}[A](\sigma_A) &\rightsquigarrow_\beta ret\ false \quad \text{where } A \neq \mathbb{B} \\ eq_{\sigma_{\mathbb{B}}}[\mathbb{B}](\sigma_{\mathbb{B}}) &\rightsquigarrow_\beta ret\ true \\ \sigma'_{\mathbb{B}} \leftarrow new_{\mathbb{B}}; eq_{\sigma_{\mathbb{B}}}[\mathbb{B}](\sigma'_{\mathbb{B}}) &\rightsquigarrow_\beta ret\ false \end{aligned}$$

Via $[\![\_]\!]^L$, any type application in **PL** allocates a new case symbol, any type abstraction additionally abstracts over a case symbol, and any value type variable in the context has an associated case symbol. It is crucial to note that the case symbol introduced in the translation of type abstraction is not used in the body of the type lambda, a property we will use in our notion of *obliviousness*. Computation type abstraction and application, which are used to church encode F and are not modified. There is no need to do so as they are not within the codomain of $[\![\_]\!]^{PL}$. As the next section will demonstrate, $[\![\_]\!]^L$ alone is not sufficient to enforce the fresh case invariant.

Context:

$$\begin{aligned}
[\![\emptyset]\!]^L_{ctx} &= \emptyset \\
[\![\Gamma, X]\!]^L_{ctx} &= [\![\Gamma]\!]^L_{ctx}, X, \sigma_X : Case\ X \\
[\![\Gamma, \underline{X}]\!]^L_{ctx} &= [\![\Gamma]\!]^L_{ctx}, \underline{X} \\
[\![\Gamma, x : A]\!]^L_{ctx} &= [\![\Gamma]\!]^L_{ctx}, x : [\![\Gamma \vdash A]\!]^L_{ty}
\end{aligned}$$

Types:

$$\begin{aligned}
[\![\Gamma \vdash \mathbb{B}]\!]^L_{ty} &= \mathbb{B} \\
[\![\Gamma \vdash \mathbb{N}]\!]^L_{ty} &= \mathbb{N} \\
[\![\Gamma \vdash X]\!]^L_{ty} &= X \\
[\![\Gamma \vdash U\underline{B}]\!]^L_{ty} &= U[\![\Gamma \vdash \underline{B}]\!]^L_{ty} \\
[\![\Gamma \vdash A \times A']\!]^L_{ty} &= [\![\Gamma \vdash A]\!]^L_{ty} \times [\![\Gamma \vdash A']\!]^L_{ty} \\
[\![\Gamma \vdash \underline{X}]\!]^L_{ty} &= \underline{X} \\
[\![\Gamma \vdash Obs]\!]^L_{ty} &= Obs \\
[\![\Gamma \vdash A \to \underline{B}]\!]^L_{ty} &= [\![\Gamma \vdash A]\!]^L_{ty} \to [\![\Gamma \vdash \underline{B}]\!]^L_{ty} \\
[\![\Gamma \vdash \forall X.\underline{B}]\!]^L_{ty} &= \forall X.Case\ X \to [\![\Gamma, X \vdash \underline{B}]\!]^L_{ty} \\
[\![\Gamma \vdash \forall \underline{X}.\underline{B}]\!]^L_{ty} &= \forall \underline{X}.[\![\Gamma, \underline{X} \vdash \underline{B}]\!]^L_{ty}
\end{aligned}$$

Values:

$$\begin{aligned}
[\![\Gamma \vdash x]\!] &= x \\
[\![\Gamma \vdash true]\!] &= true \\
[\![\Gamma \vdash false]\!] &= false \\
[\![\Gamma \vdash z]\!] &= z \\
[\![\Gamma \vdash sV]\!] &= s[\![\Gamma \vdash V]\!] \\
[\![\Gamma \vdash thunk\ M]\!] &= thunk\ [\![\Gamma \mid \cdot \vdash M]\!] \\
[\![\Gamma \vdash (V, W)]\!] &= ([\![\Gamma \vdash V]\!], [\![\Gamma \vdash W]\!])
\end{aligned}$$

Computations:

$$\begin{aligned}
[\![\Gamma \vdash rec_{\mathbb{B}}\ V\{M \mid N\}]\!] &= rec_{\mathbb{B}}\ [\![\Gamma \vdash V]\!]\ \{[\![\Gamma \vdash M]\!] \mid [\![\Gamma \vdash N]\!]\} \\
[\![\Gamma \vdash rec_{\mathbb{N}}\ V\ \{M \mid x.N\}]\!] &= rec_{\mathbb{N}}\ [\![\Gamma \vdash V]\!]\ \{[\![\Gamma \vdash M]\!] \mid x.[\![\Gamma, x \vdash N]\!]\} \\
[\![\Gamma \vdash fix]\!] &= fix \\
[\![\Gamma \vdash force\ V]\!] &= force\ [\![\Gamma \vdash V]\!] \\
[\![\Gamma \vdash hault\ V]\!] &= hault\ [\![\Gamma \vdash V]\!] \\
[\![\Gamma \vdash \lambda x : A.M]\!] &= \lambda x : [\![\Gamma \vdash A]\!].[\![\Gamma, x : A \vdash M]\!] \\
[\![\Gamma \vdash MV]\!] &= [\![\Gamma \vdash M]\!][\![\Gamma \vdash V]\!] \\
[\![\Gamma \vdash \Lambda X.M]\!] &= \Lambda X.\lambda\sigma : Case\ X.[\![\Gamma, X \mid \Delta \vdash M]\!] \\
[\![\Gamma \vdash M[A]]\!] &= \sigma \leftarrow new_{[\![\Gamma \vdash A]\!]}; [\![\Gamma \mid \Delta \vdash M]\!][[\![\Gamma \vdash A]\!]](\sigma) \\
[\![\Gamma \vdash \Lambda \underline{X}.M]\!] &= \Lambda \underline{X}.[\![\Gamma, \underline{X} \vdash M]\!] \\
[\![\Gamma \vdash M[\underline{B}]]\!] &= [\![\Gamma \vdash M]\!][[\![\Gamma \vdash \underline{B}]\!]] \\
[\![\Gamma \vdash rec_{\times}\ V\ \{x, y.\ M\}]\!] &= rec_{\times}\ [\![\Gamma \vdash V]\!]\ \{x, y.\ [\![\Gamma, x, y \vdash M]\!]\}
\end{aligned}$$

Fig. 10. Translation of Contexts, Types, Values, and Computations: **PL**to **FPL**

3.3 Wrapping

The reader may notice that $eq_{\sigma_{\mathbb{B}}}$ will never be in the codomain of $[\![\_]\!]^{FPL}$ and question why we should be concerned about similar misbehaved terms. To answer this, let us consider the interaction between the ambient *environment* and open terms in **PL** and **FPL**.

$$f : U(\forall X.F\mathbb{B}) \vdash_{PL} (force\ f) = \Lambda X.(force\ f)[X]$$

In **PL**, the equation above is simply proved using the η equality for type $\forall X.F\mathbb{B}$.

$$f : U(\forall X.Case\ X \to F\mathbb{B}) \vdash_{FPL} (force\ f) = \Lambda X.\lambda\sigma_X.\sigma \leftarrow new_X; (force\ f)[X]\sigma$$

However, in **FPL**, if we try to use $\eta\forall$, $\eta\lambda$, and congruence rules, we arrive at:

$$f : U(\forall X.Case\ X \to F\mathbb{B}), X, \sigma_X \vdash_{FPL} (force\ f)[X]\sigma_X = \sigma \leftarrow new_X; (force\ f)[X]\sigma$$

Note that variable f is not limited to ranging over terms in the codomain of $[\![\_]\!]^{FPL}$. If we consider the environment an adversary, it can provide values which do not behave parametrically. In this case, take $X \mapsto \mathbb{B}, \sigma_X \mapsto \sigma_{\mathbb{B}}, f \mapsto thunk(eq_{\sigma_{\mathbb{B}}})$ which, as we saw in Section 3.2.1, results in inequal terms. Generally, unguarded type applications, or failures of our fresh case invariant, can result in inequalities and broken proofs.

Niels et al.[10] solve this issue using a specialized deep η expansion, which they call *wrapping*, to surface and guard all type applications. We adapt a simplified version(Fig. 11) of their wrapping strategy to preserve equalities in our proof translation. However, wrapping introduces complications in our proof translation as we demonstrate in Section 5.1.

$$\mathcal{W}_X(t) := t$$
$$\mathcal{W}_{\mathbb{B}}(t) := t$$
$$\mathcal{W}_{\mathbb{N}}(t) := t$$
$$\mathcal{W}_{Obs}(t) := t$$
$$\mathcal{W}_{A\times A'}(t) := rec_\times\ t\ \{x, y.\ (\mathcal{W}_A(x), \mathcal{W}_{A'}(y))\}$$
$$\mathcal{W}_{U\underline{B}}(t) := thunk(\mathcal{W}_{\underline{B}}(force\ t))$$

$$\mathcal{W}_{\underline{X}}(t) := t$$
$$\mathcal{W}_{A\to\underline{B}}(t) := \lambda(x : A).\mathcal{W}_{\underline{B}}(t(\mathcal{W}_A(x)))$$
$$\mathcal{W}_{\forall X.\underline{B}}(t) := \Lambda X.\lambda\sigma_X.\mathcal{W}_{\underline{B}}(\sigma'_X \leftarrow new_X;\ t[X](\sigma'_X))$$
$$\mathcal{W}_{\forall\underline{X}.\underline{B}}(t) := \Lambda\underline{X}.\mathcal{W}_{\underline{B}}(t[\underline{X}])$$

$$\mathcal{W}_\Gamma(t) := t[\mathcal{W}_{A_1}(x_1)/x_1, \cdots, \mathcal{W}_{A_n}(x_n)/x_n]$$
where $x_1 : A_1, \cdots, x_n : A_n \in \Gamma$

Fig. 11. Wrapping

4 LOGICS

- explain: rules of Logic A
- explain: relational interpretation of types (including Case,OSum)
- explain: IEP axiom

<eric-this section needs more exposition. Nothing fancy, just explain the parts of the logic and provide an example to conclude> Here we present the parametricity logics **PL** and **FPL**.

4.1 Core

<eric-cite PE, mention influence> Both **PL** and **FPL** are based on a higher order logic, permitting quantification over predicates and relations(Fig. 13). The distinction between value and computations types is extended to our logics as well. Here we have separate classes of value and computation predicates and relations. Propositions (Fig. 14) and relations (Fig. 15) are judged to be well-formed with respect to a context, Γ, of types and terms and a context, Θ, of value and computation relations. Proof sequents are of the form $\Gamma; \Theta \mid \Phi \vdash \phi$ where Φ is a context of well-formed propositions in context $\Gamma; \Theta$. A fragment of the derivation rules appear in Fig. 16.

In addition to the derivation rules, our logic provides a number of rules that are useful in proving parametricity theorems. First, many parametricy proofs boil down to creatively picking a relation. Frequently, these relations are based on the graph of some function. As an example, take the function $even : \mathbb{N} \to F\mathbb{B}$ used to establish a relation $\langle even\rangle : Rel_v[\mathbb{N}, \mathbb{B}]$ between $\mathbb{N}$ and $\mathbb{B}$. This can be constructed using the graph rule.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \mid \cdot \vdash f : A \to FA'}\,\text{asm}}{\Gamma, x : A \mid \cdot \vdash f : A \to FA'}\,\text{weaken} \quad \dfrac{}{\Gamma, x : A \vdash x : A}}{\Gamma, x : A \mid \cdot \vdash f(x) : FA'} \quad \dfrac{\dfrac{}{\Gamma, y : A' \vdash y : A'}}{\Gamma, y : A' \mid \cdot \vdash ret\ y : FA'}}{\Gamma; \Theta \vdash (x : A, y : A').f(x) =_{FA} ret\ y : Rel_v[A, A']}}{\Gamma; \Theta \vdash \langle f\rangle : Rel_v[A, A']} \quad \rightsquigarrow \quad \dfrac{\Gamma \mid \cdot \vdash f : A \to FA'}{\Gamma; \Theta \vdash \langle f\rangle : Rel_v[A, A']}\,\text{graph}\lambda_c$$

Fig. 12. Graph Rule

We provide induction principles for $\mathbb{B}$ and $\mathbb{N}$ which are useful in proving propositions such as $\forall n : \mathbb{N}, b : \mathbb{B}.\langle even\rangle(n, b) \implies \langle even\rangle(2 + n, b)$. As **PL** and **FPL** are both programming logics, we include the equational theories (Fig. 6) of the term languages, including congruence rules and derivable rules about function extensionality.

4.2 Parametricity Axiom

Thus far, we have set up a higher order programming logic. In order to reason about parametricity, we need to define a relational interpretation of types and add an axiom schema.

The relational interpretation of types is given in Fig. 17. The synatx "$\mathcal{V}[A]_{\rho,\underline{\rho}}$" can be seen as a *marco* which, given a value type A, unfolds to a value relation $Rel_v[A', A'']$. Relation environments, $\rho, \underline{\rho}$, consists of a mapping $X \mapsto (Y, Z, R : Rel[Y, Z])$ from type variable X to a tuple of two types Y, Z and a relation $R : Rel[Y, Z]$. We use $A[\rho_L]$ to denote a type substitution which replaces every type variable X in A with the corresponding type given by the first element of tuple $\rho(X)$. <eric-more context, why introduce this, what are the interesting cases>

In order to make use of the relational interpretation of types, we employ an axiom schema for the *Identity Extension Principle*.

Definition 4.1. Identity Extension Principle. For all types A, we have:

$$\Gamma; \Theta \mid \Phi \vdash \forall x, y : A. \mathcal{V}[A]_{\vec{eq},\underline{\vec{eq}}}(x, y) \iff x =_A y$$

4.3 Parametricity Proof

<eric-rename section, demonstrate that this logic can actually prove parametricty theorems> To demonstrate the utility of **PL** we demonstrate a simple parametricity proof. Consider the $\mathbf{F}^{\forall \to}_{\mu,\Omega}$ term:

$$M : \forall [X, Y].(X \times Y) \to (Y \times X)$$

By informal parametricity reasoning, we would expect M to either, swap the given values, error, or diverge. That is to say, given

$$swap : \forall [X, Y].(X \times Y) \to (Y \times X) := \Lambda [X, Y](p).rec_\times\ p\ \{x, y.\ (y, x)\}$$

we should expect

$$swap[A', A](M[A, A'](a, a')) =_{A \times A'} M[A', A](a', a)$$

We state this claim formally in **PL**:

Theorem 4.2.

$$\vdash_{PL} \forall M : U(\forall X. \forall Y.(X \times Y) \to F(Y \times X)), a : [\![A]\!], a' : [\![A']\!].$$
$$r \leftarrow (force\ M)[\![A]\!][\![A']\!](a, a'); swap[\![A']\!][\![A]\!](r) = r \leftarrow (force\ M)[\![A']\!][\![A]\!](a', a); ret\ r$$

Proof. We begin by introducing M, a, a' to the context. Then, we use congruence for bind which is derivable within the logic given that bind is church encoded. We then have two goals.

- $M, a, a' \vdash_{PL} C[F(Y \times X)]_{?,?}((force\ M)[\![A]\!][\![A']\!](a, a'), (force\ M)[\![A']\!][\![A]\!](a', a))$
- $M, a, a' \vdash_{PL} C[(Y \times X) \to \underline{X}]_{?,?,EQ_{F([\![A]\!] \times [\![A']\!])}}(\lambda x. swap[\![A']\!][\![A]\!]x, \lambda x. ret\ x)$

These goals are not fuly specified. We need to determine what relations we want to assign to type variables X and Y. Here we make use of the IEP axiom for type $U(\forall X. \forall Y.(X \times Y) \to F(Y \times X))$.

$$M, a, a' \vdash_{PL} \forall X', Y', R : Rel_v[X', Y'], Z, W, R' : Rel_v[Z, W], (p_1 : X' \times Z), (p_2 : Y' \times W).$$
$$\mathcal{V}[X \times Y]_{R,R'}(p_1, p_2) \implies C[F(Y \times X)]_{R,R'}((force\ M)[X'][Z](p_1), (force\ M)[Y'][W](p_2))$$

From our first goal, it is clear that we should pick

$$X' \mapsto [\![A]\!], Y; \mapsto [\![A']\!], Z \mapsto [\![A']\!], W \mapsto [\![A]\!], p_1 \mapsto (a, a'), p_2 \mapsto (a', a)$$

We still need to pick relations $R : Rel_v[[\![A]\!], [\![A']\!]], R' : Rel_v[[\![A']\!], [\![A]\!]]$ such that our proof goes through. It suffices to say that the easiest relations to pick for this proof are:

- $R := (x : [\![A]\!], y : [\![A']\!]).x = a \wedge y = a'$
- $R' := (x : [\![A']\!], y : [\![A']\!]).x = a' \wedge y = a$

Then, it suffices to show $R(a, a') \wedge R'(a', a)$ in order to prove our first goal. This trivially holds given our choice of relations. The second goal boils down to showing $swap[\![A']\!][\![A]\!](a', a) = ret(a, a')$ □

<eric-predicates>

$$\phi := \bot \mid V =_A W \mid M =_{\underline{B}} N \mid R(V, W) \mid \underline{R}(M, N) \mid$$
$$\phi \wedge \psi \mid \phi \vee \psi \mid \phi \implies \psi \mid \exists \square.\phi \mid \forall \square.\phi$$
$$\square \in \{x : A, X, \underline{X}, R : Rel_v[A, A'], \underline{R} : Rel_c[\underline{B}, \underline{B}']\}$$

Fig. 13. Propositions

$$\frac{\Gamma \vdash V : A \qquad \Gamma \vdash W : A}{\Gamma; \Theta \vdash V =_A W : Prop} \qquad \frac{\Gamma \mid - \vdash M : \underline{B} \qquad \Gamma \mid - \vdash N : \underline{B}}{\Gamma; \Theta \vdash M =_{\underline{B}} N : Prop} \qquad \frac{\Gamma \vdash V : A \qquad \Gamma \vdash W : A' \qquad R : Rel_v[A, A'] \in \Theta}{\Gamma; \Theta \vdash R(V, W) : Prop}$$

$$\frac{\Gamma \mid - \vdash M : \underline{B} \qquad \Gamma \mid - \vdash N : \underline{B'} \qquad \underline{R} : Rel_c[\underline{B}, \underline{B'}] \in \Theta}{\Gamma; \Theta \vdash \underline{R}(M, N) : Prop} \qquad \frac{\Gamma; \Theta \vdash \phi : Prop \qquad \Gamma; \Theta \vdash \psi : Prop}{\Gamma; \Theta \vdash \phi \square \psi} \quad (\square \in \{\wedge, \vee, \implies\})$$

$$\frac{\Gamma, x : A; \Theta \vdash \phi : Prop}{\Gamma; \Theta \vdash \forall (x : A).\phi : Prop} \qquad \frac{\Gamma; \Theta \vdash \phi : Prop}{\Gamma; \Theta \vdash \forall X.\phi : Prop} \qquad \frac{\Gamma; \Theta, R \vdash \phi : Prop}{\Gamma; \Theta \vdash \forall (R : Rel_v[A, B]).\phi : Prop}$$

Fig. 14. Proposition Formation Rules Fragment

$$\frac{\Gamma, x : A \mid \cdot \vdash V : C \qquad \Gamma, y : A' \mid \cdot \vdash W : C}{\Gamma; \Theta \vdash (x : A, y : A').V =_C W : Rel_v[A, A']} \qquad \frac{\Gamma \mid x : \underline{B} \vdash M : \underline{C} \qquad \Gamma \mid y : \underline{B'} \vdash N : \underline{C}}{\Gamma; \Theta \vdash (x : \underline{B}, y : \underline{B'}).M =_{\underline{C}} N : Rel_c[\underline{B}, \underline{B'}]}$$

$$\frac{\Gamma; \Theta \vdash (x : A, y : C).\phi : Rel_=[A, C] \qquad \Gamma; \Theta \vdash (x : A, y : C).\psi : Rel_=[A, C]}{\Gamma; \Theta \vdash (x : A, y : C).\phi \wedge \psi : Rel_=[A, C]}$$

$$\frac{\Gamma; \Theta \vdash \phi : Prop \qquad \Gamma; \Theta \vdash (x : A, y : C).\psi : Rel_=[A, C]}{\Gamma; \Theta \vdash (x : A, y : C).\phi \implies \psi : Rel_=[A, C]} \qquad \frac{\Gamma, x : C; \Theta \vdash (x : A, y : B).\phi : Rel_=[A, B]}{\Gamma; \Theta \vdash (x : A, y : B).\forall x : C.\phi : Rel_=[A, B]}$$

$$\frac{\Gamma; \Theta \vdash (x : A, y : B).\phi : Rel_=[A, B]}{\Gamma; \Theta \vdash (x : A, y : B).\forall X.\phi : Rel_=[A, B]} \qquad \frac{\Gamma, \Theta, R : Rel_=[C, C'] \vdash (x : A, y : B).\phi : Rel\_[A, B]}{\Gamma; \Theta \vdash (x : A, y : B).\forall R : Rel_=[C, C'].\phi : Rel\_[A, B]}$$

Fig. 15. Relation Formation Rules Fragment

$$\frac{\Gamma; \Theta \mid \Phi \vdash \bot}{\Gamma; \Theta \mid \Phi \vdash \phi} \qquad \frac{\Gamma \vdash V : A}{\Gamma; \Theta \mid \Phi \vdash V =_A V} \qquad \frac{\Gamma; \Theta \mid \Phi \vdash V =_A W \qquad \Gamma; \Theta \mid \Phi \vdash \phi[V/x]}{\Gamma; \Theta \mid \Phi \vdash \phi[W/x]} \qquad \frac{\Gamma; \Theta \mid \Phi \vdash \phi \qquad \Gamma; \Theta \mid \Phi \vdash \psi}{\Gamma; \Theta \mid \Phi \vdash \phi \wedge \psi}$$

$$\frac{\Gamma; \Theta \mid \Phi \vdash \phi \wedge \psi}{\Gamma; \Theta \mid \Phi \vdash \phi} \qquad \frac{\Gamma, X; \Theta \mid \Phi \vdash \phi}{\Gamma; \Theta \mid \Phi \vdash \forall X.\phi} \quad (X \notin ftv(\Theta, \Gamma, \Phi)) \qquad \frac{\Gamma; \Theta \mid \Phi \vdash \forall X.\phi}{\Gamma; \Theta \mid \Phi \vdash \phi[A/X]} \qquad \frac{\Gamma; \Theta, R \mid \Phi \vdash \phi}{\Gamma; \Theta \mid \Phi \vdash \forall (R : Rel_v[A, B]).\phi}$$

$$\frac{\Gamma; \Theta \mid \Phi \vdash \forall (R : Rel_v[A, B]).\phi \qquad \Gamma; \Theta \vdash (x : A, y : B).\psi : Rel_v[A, B]}{\Gamma; \Theta \mid \Phi \vdash \phi[\psi[M/x, N/y]/R(M, N)]}$$

Fig. 16. Derivation Rules Fragment

<eric-Obs> <eric-thunk computation bindings> <eric-highlight Case, OSum>

$$\mathcal{V}[X]_{\rho,\underline{\rho}} = \rho(X)$$
$$\mathcal{V}[\mathbb{B}]_{\rho,\underline{\rho}} = (x : \mathbb{B}, y : \mathbb{B}).\ x = y$$
$$\mathcal{V}[\mathbb{N}]_{\rho,\underline{\rho}} = (x : \mathbb{N}, y : \mathbb{N}).\ x = y$$
$$\mathcal{V}[U\underline{B}]_{\rho,\underline{\rho}} = (x : U\underline{B}[\rho_L, \underline{\rho_L}], y : U\underline{B}[\rho_R, \underline{\rho_R}]).C[\underline{B}]_{\rho,\underline{\rho}}(force\ x, force\ y)$$
$$\mathcal{V}[A \times A']_{\rho,\underline{\rho}} = (p_1 : (A \times A')[\rho_L, \underline{\rho_L}], p_2 : (A \times A')[\rho_R, \underline{\rho_R}]).$$
$$\exists e_1 : A[_L], e_2 : A'[_L], e_3 : A[_R], e_4 : A'[_R].$$
$$p_1 = (e_1, e_2) \wedge p_2 = (e_3, e_4) \wedge \mathcal{V}[A]_{\rho,\underline{\rho}}(e_1, e_3) \wedge \mathcal{V}[A']_{\rho,\underline{\rho}}(e_2, e_4)$$

$$C[\underline{X}]_{\rho,\underline{\rho}} = \underline{\rho}(\underline{X})$$
$$C[Obs_{\mathbb{B}}]_{\rho,\underline{\rho}} = (x : Obs_{\mathbb{B}}, y : Obs_{\mathbb{B}}).$$
$$\exists b_1, b_2.x = hault\ b_1 \wedge y = hault\ b_2 \wedge \mathcal{V}[\mathbb{B}](b_1, b_2)$$
$$C[A \to \underline{B}]_{\rho,\underline{\rho}} = (f : (A \to \underline{B})[\rho_L, \underline{\rho_L}], g : (A \to \underline{B})[\rho_R, \underline{\rho_R}]).$$
$$\forall x : A[\rho_L, \underline{\rho_L}], y : A[\rho_R, \underline{\rho_R}].\mathcal{V}[A]_{\rho,\underline{\rho}}(x, y) \implies C[\underline{B}]_{\rho,\underline{\rho}}(fx, gy)$$
$$C[\forall X.\underline{B}]_{\rho,\underline{\rho}} = (f : \forall X.\underline{B}[\rho_L, \underline{\rho_L}], g : \forall X.\underline{B}[\rho_R, \underline{\rho_R}]).$$
$$\forall Y, Z, R : Rel_v[Y, Z].C[\underline{B}]_{\rho,R,\underline{\rho}}(f[Y], g[Z])$$
$$C[\forall \underline{X}.\underline{B}]_{\rho,\underline{\rho}} = (f : \forall \underline{X}.\underline{B}[\rho_L, \underline{\rho_L}], g : \forall \underline{X}.\underline{B}[\rho_R, \underline{\rho_R}]).$$
$$\forall \underline{Y}, \underline{Z}, \underline{R} : Rel_c[\underline{Y}, \underline{Z}].C[\underline{B}]_{\rho,\underline{\rho},\underline{R}}(f[\underline{Y}], g[\underline{Z}])$$

Fig. 17. Relational Interpretation of Types

5 PRESERVING LOGICAL EQUIVALENCE

In this section Here we state our theorem in full detail. The remainder of this section will be dedicated to defining the proof translation and presenting lemmas we think are necessary to prove our result.

5.1 Obliviousness

Modifying the terms in a proof has consequences. Without any adjustment to our interpretation of propositions, proofs which use η expansion for value type abstraction break. Considering the following convoluted proof of $2 = 2$ in **PL**[1].

$$\dfrac{\dfrac{\dfrac{}{\vdash 2 : \mathbb{N}}\ \text{refl}}{\vdash 2 = 2} \qquad \dfrac{\dfrac{f \vdash (force\ f) : \forall X.\underline{B}}{f \vdash (force\ f) = \Lambda X.(force\ f)[X]}\ \eta\forall}{\vdash \forall f.(force\ f) = \Lambda X.(force\ f)[X]}\ \text{I}\forall_{vtm}}{\dfrac{\vdash 2 = 2 \wedge \forall f : U(\forall X.\underline{B}).(force\ f) = \Lambda X.(force\ f)[X]}{\vdash 2 = 2}\ \text{E}\wedge_1}\ \text{I}\wedge$$

The proof statement $2 = 2$ could simply be solved using reflexivity, but our logical equivalence theorem states that we should be able to handle any possible proof of $2 = 2$, even one which introduces superfluous derivations. Assuming the proposition and derivation translation from **PL** to **FPL** do not introduce extra hypotheses, we run into a problem. Specifically, we do not have enough information to prove the rule $\eta\forall$ under translation. Given $[\![\Gamma]\!] \vdash [\![M]\!] : [\![\forall X.\underline{B}]\!]$, show $[\![\Gamma]\!]; [\![\Theta]\!] \mid [\![\Phi]\!] \vdash [\![M]\!] = [\![\Lambda X.M[X]]\!]$ in **FPL**. Or, as in our concrete example above:

[1]Note that $\mathcal{W}[\mathbb{B}]$ is the identity function, so wrapping does not play a role in the translated proof.

$$\dfrac{\dfrac{f : [\![U(\forall X.\underline{B})]\!] \vdash [\![(force\ f)]\!] : [\![\forall X.\underline{B}]\!]}{f : U(\forall X.Case\ X \to [\![\underline{B}]\!]) \vdash (force\ f) : \forall X.Case\ X \to [\![\underline{B}]\!]}}{\dfrac{?}{f \vdash force\ f = \Lambda X.\lambda\sigma_X.\sigma \leftarrow new_X; (force\ f)[X]\sigma}}$$
$$f \vdash [\![force\ f]\!] = [\![\Lambda X.(force\ f)[X]]\!]$$

Using the rules of **FPL** to η expanding lambdas on the left-hand-side of the equation as well as congruence rules for lambdas yields:

$$f, X, \sigma_X \vdash (force\ f)[X]\sigma_X = \sigma \leftarrow new_X; (force\ f)[X]\sigma \tag{1}$$

Note that if the variable f was wrapped, then this equation would hold. But, as we saw in the example above, the wrapping *did not cover* f. To patch over this issue, we introduce the concept of *obliviousness*.

We define obliviousness (Fig. 18) as a **PL** type-indexed *logical predicate* over terms in **FPL**. An oblivious term is one which cannot use case symbols in any meaningful way. In particular, it is not able to distinguish between a fresh case symbol and one which has already been allocated. Formally, this property is encoded in the obliviousness predicate as an equation in case $O[\forall X.\underline{B}]$.

$$\begin{aligned}
&O[X]_\rho(V : \rho_{ty}(X)) \coloneqq \rho_{rel}(X)(V)\\
&O[\mathbb{B}]_\rho(V : \mathbb{B}) \coloneqq \top\\
&O[\mathbb{N}]_\rho(V : \mathbb{N}) \coloneqq \top\\
&O[A \times A']_\rho(V : [\![A \times A']\!]^L_{ty}[\rho]) \coloneqq \exists x_1 : [\![A]\!]^L_{ty}[\rho], x_2 : [\![A']\!]^L_{ty}[\rho].\\
&\quad V = (x_1, x_2) \wedge O[A]_\rho(x_1) \wedge O[A']_\rho(x_2)\\
&O[U\underline{B}]_\rho(V : [\![\underline{B}]\!]^L_{ty}[\rho])) \coloneqq O[\underline{B}]_\rho(force\ V)\\
\\
&O[\underline{X}]_\rho(V : \rho_{ty}(\underline{X})) \coloneqq \rho_{rel}(\underline{X})(V)\\
&O[A \to \underline{B}]_\rho(M : [\![A \to \underline{B}]\!]^L_{ty}[\rho]) \coloneqq \forall\ V : [\![A]\!]^L_{ty}[\rho].O[A]_\rho(V) \implies O[\underline{B}]_\rho(MV)\\
&O[\forall X.\underline{B}]_\rho(M : [\![\forall X.\underline{B}]\!]^L_{ty}[\rho]) \coloneqq\\
&\quad \forall X, P : Pred[X], \sigma_X : Case\ X.\\
&\qquad O[\underline{B}]_{\rho,P}(M[X](\sigma_X))\\
&\qquad \wedge\ M[X](\sigma_X) = (\sigma \leftarrow new_X; M[X](\sigma))\\
&O[\forall \underline{X}.\underline{B}]_\rho(M : [\![\forall \underline{X}.\underline{B}]\!]^L_{ty}[\rho]) \coloneqq \forall \underline{X}, \underline{P} : Pred[\underline{X}].O[\underline{B}]_{\rho,\underline{P}}(M[\underline{X}])
\end{aligned}$$

Fig. 18. Obliviousness Predicate

To demonstrate how obliviousness solves our problem, we reconsider Eq. (1) with an added hypothesis, H_f, about the obliviousnes of f in Eq. (2). Our hypothesis H_f applied to X, P_X, σ_X gives us a proof of obliviousness $O[\underline{B}]((force\ f)[X]\sigma_X)$ and the exact equation we are trying to prove.

$$f, X, \sigma_X;\ P_X \mid H_f : O[U(\forall X.\underline{B})](f) \vdash (force\ f)[X]\sigma_X = \sigma \leftarrow new_X; (force\ f)[X]\sigma \tag{2}$$

We define our proof translation making usage of obliviousness. In order to justify our approach, we prove an *adequacy* theorem in Section 5.4 demonstrating that our usage of obliviousness is sound. While wrapping **FPL** terms enforces the freshness invariant, it complicates the proof translation by potentially introducing η expansions and allocation of fresh cases which modify the terms in

a proof. Obliviousness helps here as well by enabling a more compositional approach to proof translation, as we will see in section Section 5.3.

5.2 Proposition Translation

As we saw in the last section, in order to preserve **PL** proofs we needed terms to be *oblivious* to case symbols. We introduce obliviousness via the proof translation from **PL** to **FPL**. Here we define a translation $[\![\_]\!]^O$ which enhances the term translation $[\![\_]\!]^L$ by pairing a translated term with a proof of its obliviousness. Additionally, $[\![\_]\!]^O$ adds data to the context required by the definition of obliviousness. Specifically, any term variable $x : A$ in the context has an associated obliviousness hypothesis $H_x : O[A](x)$ in the proof context and type variables $X/\underline{X}$ are associated with predicates $P/\underline{P}$ so that type variables can be interpreted in the obliviousness predicate.

$$[\![\Gamma, x : A]\!]^O := [\![\Gamma]\!]^O, x : [\![A]\!]^L \mid H_x : O[A](x)$$
$$[\![\Gamma, X]\!]^O := [\![\Gamma]\!]^O, X, \sigma_X : Case\ X; \ P : Pred[X]$$
$$[\![\Gamma, \underline{X}]\!]^O := [\![\Gamma]\!]^O, \underline{X}; \ \underline{P} : Pred[\underline{X}]$$

$$[\![\Gamma \vdash_{PL} M : A]\!]^O := [\![\Gamma \vdash_{PL} M : A]\!]^L, \ O[A]([\![M]\!])$$

Fig. 19. Obliviousness Context and Term Translation

For the invariants on contexts to be preserved, we must adjust the translation of propositions. Quantification over a value $x : A$ introduces an obliviousness hypothesis $O[A](x)$. Quantification over computation types $\underline{X}$ introduces a predicate $\underline{P}_{\underline{X}}$. Quantification over value types X introduces a predicate P as well as a case symbol $\sigma_X : Case\ X$.

$$\begin{aligned}
[\![\Gamma; \Theta \vdash_A \bot]\!]^L_{prop} &= \bot \\
[\![\Gamma; \Theta \vdash_A V =_A W]\!]^L_{prop} &= [\![\Gamma \vdash_A V]\!]^O =_{[\![\Gamma \vdash A]\!]^L_{ty}} [\![\Gamma \vdash W]\!]^O \\
[\![\Gamma; \Theta \vdash_A M =_{\underline{B}} N]\!]^L_{prop} &= [\![\Gamma \vdash M]\!]^O =_{[\![\Gamma \vdash \underline{B}]\!]^L_{ty}} [\![\Gamma \vdash N]\!]^O \\
[\![\Gamma; \Theta \vdash_A R(V, W)]\!]^L_{prop} &= [\![\Gamma; \Theta \vdash_A R]\!]^L_{rel_v}([\![\Gamma \vdash V]\!]^O, [\![\Gamma \vdash W]\!]^O) \\
[\![\Gamma; \Theta \vdash_A \phi \wedge \psi]\!]^L_{prop} &= [\![\Gamma; \Theta \vdash_A \phi]\!]^L_{prop} \wedge [\![\Gamma; \Theta \vdash_A \psi]\!]^L_{prop} \\
[\![\Gamma; \Theta \vdash_A \forall (x : A).\phi]\!]^L_{prop} &= \forall (x : [\![\Gamma \vdash A]\!]^L_{ty}). \ O[A](x) \implies [\![\Gamma, x : A\ \Theta \vdash_A \phi]\!]^L_{prop} \\
[\![\Gamma; \Theta \vdash_A \forall X.\phi]\!]^L_{prop} &= \forall X, \ P : Pred[X], \sigma_X : Case\ X \ .[\![\Gamma, X; \Theta \vdash_A \phi]\!]^L_{prop} \\
[\![\Gamma; \Theta \vdash_A \forall \underline{X}.\phi]\!]^L_{prop} &= \forall \underline{X}, \ \underline{P} : Pred[\underline{X}] \ .[\![\Gamma, \underline{X}; \Theta \vdash_A \phi]\!]^L_{prop} \\
[\![\Gamma; \Theta \vdash_A \forall R : Rel_v[A, A'].\phi]\!]^L_{prop} &= \forall R : Rel_v[[\![\Gamma \vdash A]\!]^L_{ty}, [\![\Gamma \vdash A']\!]^L_{ty}].[\![\Gamma; \Theta, R : Rel_v[A, A'] \vdash_A \phi]\!]^L_{prop}
\end{aligned}$$

Fig. 20. Proposition Translation Fragment

5.3 Oblivious Proof Translation

Here we demonstrate the first key theorem (Theorem 5.3) in our preservation of logical equivalence. This theorem relies on auxiliary lemmas demonstrating that our term and type translations are sound, such as the preservation of typing.

Lemma 5.1. $[\![\_]\!]^L$ *Preserves well-formedness of types and well-typedness of terms*

Proof. This is mostly routine. The case for type application is slightly complicated by the need to use typing rules for church encoded bind. □

Here we demonstrate that for any $\Gamma \vdash_{PL} V : A$ we can provide a proof $[\![\Gamma]\!]^O \vdash_{FPL} O[A]([\![V]\!]^L)$ as demanded by our updated term translation. In other words, obliviousness is a congruence with respect to the typing rules of **PL**. <eric-TODO: this typesetting looks like shit>

Lemma 5.2. $\Gamma \vdash_{PL} V : A \implies [\![\Gamma]\!]^O \vdash_{FPL} O[A]([\![V]\!])$

Proof. By induction on the typing rules of **PL**. We consider the interesting cases here.

Case : $\Lambda X.M : \forall X.\underline{B}$.
Given: $[\![\Gamma]\!]^O, X\sigma_X; P_X \vdash O[\underline{B}]([\![M]\!]^L)$
Prove: $[\![\Gamma]\!]^O \vdash O[\forall X.\underline{B}]([\![\Lambda X.M]\!]^L)$
We have to show $\forall X, P_X, \sigma_X.O[\underline{B}]([\![\Lambda X.M]\!]^L[X]\sigma_X) \wedge [\![\Lambda X.M]\!]^L[X]\sigma_X = \sigma \leftarrow new_X; [\![\Lambda X.M]\!]^L[X]\sigma$. Here we use the observation that the translation of a type lambda is oblivious, the body $[\![X \vdash M]\!]^L$ does not use σ_X. Thus, we have $[\![\Lambda X.M]\!]^L[X]\sigma_X = (\Lambda X.\lambda\sigma_X.[\![X \vdash M]\!]^L)[X]\sigma_X = [\![M]\!]^L$. We have the first conjuct by assumption. The second conjunct simplifies to $[\![M]\!]^L = \sigma \leftarrow new_X; [\![M]\!]^L$, but we know that $[\![M]\!]^L$ does not use σ. Therefore, the second conjunct is solved by the drop rule.

Case : $M[A]$.
Given: $[\![\Gamma]\!]^O \vdash O[\forall X.\underline{B}]([\![M]\!]^L)$
Prove: $[\![\Gamma]\!]^O \vdash O[\underline{B}]([\![M[A]]\!]^L)$
We have to show $O[\underline{B}](\sigma \leftarrow new_{[\![A]\!]^L}; [\![M]\!]^L[\![A]\!]^L\sigma)$. In order to use the hypothesis, we need a case symbol in scope. For this we look at the unary relational interpretation of $new_{[\![A]\!]^L}$.

$$C[F(Case\ [\![A]\!])](new_{[\![A]\!]}) = \forall \underline{Y}, \underline{P_Y}.k : U(Case\ [\![A]\!] \to \underline{Y}).$$
$$(\forall \sigma : Case\ [\![A]\!].\mathcal{V}[Case\ [\![A]\!]](\sigma) \implies \underline{P_Y}((force\ k)\sigma)) \implies \underline{P_Y}(new_{[\![A]\!]}[\underline{Y}]k)$$

Picking $\underline{Y} := [\![\underline{B}]\!], \underline{P_Y} := O[\underline{B}], k := thunk(\lambda\sigma.[\![M]\!][\![A]\!]\sigma)$, we have

$$(\forall \sigma : Case\ [\![A]\!].\mathcal{V}[Case\ [\![A]\!]](\sigma) \implies O[\underline{B}]([\![M]\!][\![A]\!]\sigma)) \implies O[\underline{B}](\sigma \leftarrow new_{[\![A]\!]^L}; [\![M]\!]^L[\![A]\!]^L\sigma)$$

In which case, it suffices to show

$$\forall \sigma : Case\ [\![A]\!].\mathcal{V}[Case\ [\![A]\!]](\sigma) \implies O[\underline{B}]([\![M]\!][\![A]\!]\sigma)$$

This, along with our hypothesis, is sufficient to prove the goal.

□

<eric-TODO: this typesetting looks like shit, use lists?>

Theorem 5.3. ***(INCOMPLETE)*** $\Gamma \vdash_{PL} M = N \implies [\![\Gamma]\!]^O \vdash_{FPL} [\![M]\!]^O = [\![N]\!]^O$

Proof. Proceed by induction on the proof rules and axioms of **PL**.
Derivation Rules

Case: $\eta\forall$. This case, which motivated our definition of obliviousness, was demonstrated in Section 5.1

Case: $E\forall_{vtm}$.
Given: $[\![\Gamma]\!]^O \vdash_{FPL} [\![\forall x : A.\phi]\!]^L_{prop}$ and $[\![\Gamma]\!]^L \vdash_{FPL} [\![V]\!]^L : [\![A]\!]^L$
Prove: $[\![\Gamma]\!]^O \vdash_{FPL} [\![\phi[V/x]]\!]^L_{prop}$
We are given $\forall x : [\![A]\!].O[A](x) \implies [\![x \vdash \phi]\!]^L_{prop}$ and, via substitution lemmas, our goal is $[\![\phi]\!]^L_{prop}[[\![V]\!]^L/x]$. We instantiate our hypothesis with $x \mapsto [\![V]\!]$.By Lemma 5.2 we have $O[A]([\![V]\!])$ and so our goal is solved.

Case: $cong_\forall$. <eric-broken>
Axioms
<eric-IEP, demonstrate a case?>

□

Additionally, we need some lemmas about substitution which we omit.

5.4 Adequacy

Here we demonstrate that our usage of obliviousness is sound.

Lemma 5.4. *Wrapping of oblivious terms is identity.* $\forall M : [\![A]\!]^L.O[A](M) \implies \mathcal{W}[A](M) = M$

Proof. By induction on the type formation rules of **PL**. The base cases where wrapping is identity are trivially true. The only case that is not simply an application of an η rule is $\forall X.\underline{B}$ where we use the equation baked into the obliviousness predicate. □

Lemma 5.5. ***(INCOMPLETE)*** *Wrapped terms are Oblivious.* $\forall M : [\![A]\!]^L.O[A](\mathcal{W}[A](M))$

Proof. <eric-> this one is sus. type variables are the problematic case. Consider $f : \forall \underline{X}.U\underline{X} \to \underline{X}$. How would we prove..

$$
\begin{aligned}
&O[\forall \underline{X}.U\underline{X} \to \underline{X}](\mathcal{W}[\forall \underline{X}.U\underline{X} \to \underline{X}](f)) \\
&= \\
&O[\forall \underline{X}.U\underline{X} \to \underline{X}](\Lambda \underline{X}.\lambda x : U\underline{X}.f[\underline{X}](force\ x)) \\
&= \\
&\forall \underline{X}, P_{\underline{X}}, x : U\underline{X}.O[U\underline{X}](x) \implies O[\underline{X}]((\Lambda \underline{X}.\lambda x : U\underline{X}.f[\underline{X}](force\ x))[\underline{X}]x) \\
&= \\
&\forall \underline{X}, P_{\underline{X}}, x : U\underline{X}.P_{\underline{X}}(force\ x) \implies P_{\underline{X}}(f[\underline{X}](force\ x))
\end{aligned}
$$

□

Theorem 5.6. *Adequacy:*

$$\frac{[\![\Gamma]\!]^O \vdash_{FPL} [\![M]\!]^O = [\![N]\!]^O}{[\![\Gamma]\!]^L; \emptyset \mid \emptyset \vdash_{FPL} \mathcal{W}_{\Gamma \vdash A}([\![M]\!]^L) = \mathcal{W}_{\Gamma \vdash A}([\![N]\!]^L)}$$

Proof. <eric-sketchy proof sketch> By assumption, we have the obliviousness of $[\![M]\!]^L, [\![N]\!]^L$. Combine these with Lemma 5.4 to get $\mathcal{W}[A]([\![M]\!]^L) = [\![M]\!]^L$ and use this fact with our hypothesis. Next, we define a substitution $[\![\Gamma]\!]^L \xrightarrow{\sigma} [\![\Gamma]\!]^O$ by:

$$X \mapsto X, \underline{X} \mapsto \underline{X}, x : [\![A]\!] \mapsto \mathcal{W}[A](x), P_X := (x : X).\top, P_{\underline{X}} := (x : \underline{X}).\top$$

Then, using substitutivity:

$$\frac{[\![\Gamma]\!]^L \xrightarrow{\sigma} [\![\Gamma]\!]^O \qquad [\![\Gamma]\!]^O \vdash_{FPL} \mathcal{W}[A][\![M]\!]^L = \mathcal{W}[A][\![N]\!]^L}{[\![\Gamma]\!]^L \mid H_{x_1} : O[A_1](x_1)[\sigma], \cdots H_{x_n} : O[A_n](x_n)[\sigma] \vdash \mathcal{W}_{\Gamma \vdash A}([\![M]\!]^L) = \mathcal{W}_{\Gamma \vdash A}([\![N]\!]^L)}$$

Noting that $\mathcal{W}[A]([\![M]\!]^L)[\sigma] = \mathcal{W}_{\Gamma \vdash A}([\![M]\!]^L)$. The remaining obligation is to prove all the obliviousness hypotheses H_x using Lemma 5.5. □

5.5 Grand Finale

<eric-rename section, add exposition>

Theorem 5.7. *Preservation of Logical Equivalence:* $\forall\ \Gamma \vdash_{F^{\forall \to}_{\mu,\Omega}} M, N : A.$

$$[\![\Gamma]\!]^{PL}; \emptyset \mid \emptyset \vdash_{PL} [\![M]\!]^{PL} = [\![N]\!]^{PL} \implies [\![\Gamma]\!]^{FPL}; \emptyset \mid \emptyset \vdash_{FPL} \mathcal{W}_{\Gamma \vdash A}([\![M]\!]^{FPL}) = \mathcal{W}_{\Gamma \vdash A}([\![N]\!]^{FPL})$$

Proof. <eric-sketch - combine theorem Theorem 5.3 and Theorem 5.6> □

6 CONCLUSION/DISCUSSION/RELATED WORK/FUTURE WORK

- Holes in proof, friction with current definitions
- Model!
- Iris style primitives in the logic or directly to Iris?

<eric-rehash the abstract/intro>

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation CISE Graduate Fellowships (CSGrad4US) under Grant No. 2313998. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. 2017. Theorems for free for free: parametricity, with and without types. *Proc. ACM Program. Lang.* 1, ICFP, Article 39 (Aug. 2017), 28 pages. https://doi.org/10.1145/3110283

[2] Lars Birkedal, Rasmus Ejlers Møgelberg, and Rasmus Lerchedahl Petersen. 2006. Linear Abadi and Plotkin Logic. *Log. Methods Comput. Sci.* 2 (2006). https://api.semanticscholar.org/CorpusID:14086681

[3] Dominique Devriese, Marco Patrignani, and Frank Piessens. 2017. Parametricity versus the universal type. *Proc. ACM Program. Lang.* 2, POPL, Article 38 (Dec. 2017), 23 pages. https://doi.org/10.1145/3158126

[4] Dominique Devriese, Marco Patrignani, and Frank Piessens. 2022. Two Parametricities Versus Three Universal Types. *ACM Trans. Program. Lang. Syst.* 44, 4, Article 23 (Sept. 2022), 43 pages. https://doi.org/10.1145/3539657

[5] Robert Harper and Greg Morrisett. 1995. Compiling polymorphism using intensional type analysis. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '95)*. Association for Computing Machinery, New York, NY, USA, 130–141. https://doi.org/10.1145/199448.199475

[6] Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis (Semantics Structures in Computation, V. 2)*. Kluwer Academic Publishers, USA.

[7] Rasmus Ejlers Mogelberg and Alex Simpson. 2007. Relational Parametricity for Computational Effects. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*. 346–355. https://doi.org/10.1109/LICS.2007.40

[8] Rasmus Ejlers Møgelberg and Alex Simpson. 2008. A Logic for Parametric Polymorphism with Effects. In *Types for Proofs and Programs*, Marino Miculan, Ivan Scagnetto, and Furio Honsell (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 142–156.

[9] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. 1999. From system F to typed assembly language. *ACM Trans. Program. Lang. Syst.* 21, 3 (May 1999), 527–568. https://doi.org/10.1145/319301.319345

[10] Georg Neis, Derek Dreyer, and Andreas Rossberg. 2009. Non-parametric parametricity. *SIGPLAN Not.* 44, 9 (Aug. 2009), 135–148. https://doi.org/10.1145/1631687.1596572

[11] Max S. New, Dustin Jamner, and Amal Ahmed. 2019. Graduality and parametricity: together again for the first time. *Proc. ACM Program. Lang.* 4, POPL, Article 46 (Dec. 2019), 32 pages. https://doi.org/10.1145/3371114

[12] Gordon D. Plotkin and Martín Abadi. 1993. A Logic for Parametric Polymorphism. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*. Springer-Verlag, Berlin, Heidelberg, 361–375.
[13] Matías Toro, Elizabeth Labrada, and Éric Tanter. 2019. Gradual parametricity, revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 17 (Jan. 2019), 30 pages. https://doi.org/10.1145/3290330

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009