Relational Parametricity for a Polymorphic Linear Lambda Calculus

Jianzhou Zhao, Qi Zhang, and Steve Zdancewic

University of Pennsylvania
{jianzhou,qzh,stevez}@cis.upenn.edu

Abstract. This paper presents a novel syntactic logical relation for a polymorphic linear λ-calculus that treats all types as linear and introduces the constructor ! to account for intuitionistic terms, and System F$^\circ$—an extension of System F that uses kinds to distinguish linear from intuitionistic types. We define a logical relation for open values under both open linear and intuitionistic contexts, then extend it for open terms with evaluation and open relation substitutions. Relations that instantiate type quantifiers are for open terms and types. We demonstrate the applicability of this logical relation through its soundness with respect to contextual equivalence, along with free theorems for linearity that are difficult to achieve by closed logical relations. When interpreting types on only closed terms, the model defaults to a closed logical relation that is both sound and complete with respect to contextual equivalence and is sufficient to reason about isomorphisms of type encodings. All of our results have been mechanically verified in Coq.

1 Introduction

In the polymorphic lambda calculus, System F [13], relational parametricity [18] is the essence of type abstraction. It asserts that a parametrically polymorphic function must use the same algorithm to compute its result, independently of the instantiated types.

Relational parametricity can be used to derive equivalences involving functional programs by observing solely the types of those programs, with no knowledge of the functions' actual definitions. Wadler [23] refers to these equivalences as the "free theorems" associated with particular types. For instance, we can conclude that there is no closed inhabitant of type $\forall\alpha.\alpha$ in a pure setting. If there were such a term, it must yield a value of any type at which it is instantiated, but there is no uniform algorithm to compute a value at any type. Therefore, $\forall\alpha.\alpha$ is an empty type. Given a closed polymorphic function B with type $\forall\alpha.\alpha \to \alpha \to \alpha$, for any type τ, $B[\tau]$ evaluates to a function F of type $\tau \to \tau \to \tau$. Since B is polymorphic, F cannot depend on its arguments. Given any closed values $V_1{:}\tau$, $V_2{:}\tau$, F must return one of the arguments directly or yield a term equivalent to one of the arguments. Therefore we can conclude that B must behave like one of $\Lambda\alpha.\ \lambda x{:}\alpha.\ \lambda y{:}\alpha.\ x$ or $\Lambda\alpha.\ \lambda x{:}\alpha.\ \lambda y{:}\alpha.\ y$.

Linearity and parametricity Bierman et al. [6] used $\top\top$-closed [17] logical relations for the parametricity principle of the programming language Lily—a dual intuitionistic linear lambda calculus [4] with polymorphic types and recursion. They proved isomorphisms of type encodings via relational parametricity properties. In the language without recursion, their model defaults to the standard logical relations (that interpret types

with pairs of closed terms), while the free theorems obtained by the standard techniques do not tell the whole story because linearity changes the set of possible well-typed programs. Consider a *linear* variant of the Boolean type used above: $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$. Here, $\multimap$ is a linear function type. The following standard parametricity holds:

Theorem 1 (Parametricity). *If* $\Gamma; \Delta \vdash M{:}\tau$, *then* $\Gamma; \Delta \vdash M \simeq^{\star}_{\mathsf{log}} M \in \tau$.

Here, $\simeq^{\star}_{\mathsf{log}}$ denotes a standard logical relation. As in the case for ordinary System F, this parametricity theorem implies that any closed term M with type $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$ must behave like either $\Lambda\alpha.\lambda a_1{:}\alpha.\lambda a_2{:}\alpha.a_1$ or $\Lambda\alpha.\lambda a_1{:}\alpha.\lambda a_2{:}\alpha.a_2$. However, the linear variables a_1 and a_2 must be used exactly once in the body of M. Therefore, *neither* of these expressions is well-typed. More generally, the free theorem of this type should say that this is an *empty* type—there are no terms that inhabit it! On the other hand, the similar type $\forall\alpha.!\alpha \multimap !\alpha \multimap !\alpha$, in which ! constructs an intuitionistic term, is still isomorphic to the Booleans. Moreover, analogous examples can be constructed in System F$^{\circ}$ [16]—a variant of polymorphic linear lambda calculus that uses the kinds, instead of ! type constructor, to distinguish linear from intuitionistic types. These observations suggest that there might be a stronger linear parametricity that distinguishes these cases.

The key idea of this paper is that linearity properties can be properly taken into account by using *open* logical relations. There is a large body of work on logical relations that interprets types as relations R, which are sets of pairs of *closed* terms with *closed* types under empty contexts. Linear type systems, however, restrict the use of variables of linear types in a *non-empty* context. Intuitively, if logical relations interpret types by relations for *open* terms under non-empty linear contexts, the free theorem associated with the above type should show that it is void.

Suppose we apply M to open values V_1 and V_2 with type τ where V_1 and V_2 are logically related to themselves under disjoint linear contexts respectively. By the free theorem we have $M[\tau]\,V_1\,V_2 \longrightarrow^{*} V_1$ or $M[\tau]\,V_1\,V_2 \longrightarrow^{*} V_2$. But neither of the above evaluations is valid, because the set of free linear variables in a well-formed expression should be preserved under reduction; in other words, the evaluation should not consume the set of free linear variables from either V_1 or V_2. With this stronger logical relation, we can conclude that there is no such closed M of type $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$. However, closed terms with type $\forall\alpha.!\alpha \multimap !\alpha \multimap !\alpha$ still behave like polymorphic boolean functions because the system ensures that intuitionistic values V_1 and V_2 cannot capture linear variables. Section 4.1 formally proves these results, and explains that the standard logical relations cannot distinguish them. Although Tait [21], Girard [13], Crary [10] and Hasegawa [14] developed logical relations for *open* terms, their works do not support both polymorphism and linearity. We discuss related work in Section 3.

The specific contributions of this paper include:

1. Section 2 gives the type system and operational semantics for PDILL that combines a term calculus for Plotkin's dual intuitionistic linear logic [4] with impredicative polymorphism. We then discuss related work, provide a novel syntactic logical relation that interprets types with pairs of open terms for PDILL in Section 3, and show that every well-typed expression is related to itself in the interpretation of its type by this logical relation. This logical relation is sound with respect to contextual

equivalence, and therefore suitable for reasoning about program equivalence. When interpreting types on only closed terms, it defaults to a standard logical relation that is both sound and complete with respect to contextual equivalence.
2. In Section 4 we give several applications of this logical relational model: free theorems for linearity that are difficult to achieve by standard logical relations, properties of contextual equivalence, and isomorphisms of type encodings.
3. The results for PDILL also carry over to System F$^\circ$ [16]. All of our results (including PDILL, System F$^\circ$ and System F) are available at `http://www.cis.upenn.edu/~jianzhou/parametricity4linf`, and are mechanically verified in Coq [9] and OTT [20]. We do not only view this formalization as supporting material, but also as an extensive formalization for polymorphic (linear) languages in Coq [3]. Section 5 briefly discusses the mechanical proofs and future work.

2 A Brief Introduction to PDILL

$$
\begin{array}{lll}
\text{Terms} & M ::= x \mid a \mid \lambda a{:}\tau.M \mid M\,M' \mid \Lambda\alpha.M \mid M[\tau] \mid !M \mid \mathbf{let}\,!x = M\,\mathbf{in}\,M' \\
& \quad \mid \langle M, M'\rangle \mid M.1 \mid M.2 \mid (M, M') \mid \mathbf{let}\,(a, a') = M\,\mathbf{in}\,M' \\
\text{Values } V, U & ::= \lambda a{:}\tau.M \mid \Lambda\alpha.M \mid !M \mid \langle M, M'\rangle \mid (V, V') \\
\text{Types } \tau, \sigma & ::= \alpha \mid \tau \multimap \tau' \mid \forall\alpha.\tau \mid !\tau \mid \tau \& \tau' \mid \tau \otimes \tau' \\
\text{Intuitionistic contexts } \Gamma, G & ::= \cdot \mid \Gamma, x{:}\tau \mid \Gamma, \alpha \qquad \text{Linear contexts } \Delta, D ::= \cdot \mid \Delta, a{:}\tau
\end{array}
$$

Fig. 1. Syntax for PDILL.

In programming languages, linear type systems check the ability to duplicate and to discard a resource. There are many variants on linear type systems to distinguish linear from intuitionistic variables [24]. PDILL combines a term calculus for Plotkin's dual intuitionistic linear logic [4] with impredicative polymorphism. Figure 1, 2 and 3 give the syntax, typing rules and evaluation relations for PDILL. Our typing rules take an intuitionistic context Γ — an ordered list of both type and intuitionistic term variables, and a linear context Δ that binds only linear term variables. Intuitionistic variables x that may be duplicated and discarded, and linear variables a that must be used exactly once are syntactically different. We encode intuitionistic lambda abstractions $\lambda x{:}\tau_1.M_2$ as $\lambda a{:}!\tau_1.\mathbf{let}\,!x = a\,\mathbf{in}\,M_2$ with type $!\tau_1 \multimap \tau_2$.

Typing rules can drop unused variables (weakening) or use a variable more than once (contraction) from an intuitionistic context Γ, but must use variables exactly once from a linear context Δ. Linear usage of Δ holds because T_IVAR allows only an empty Δ, and T_LVAR takes a linear context that consists of only the variable typechecked. Rules T_LVAR and T_IVAR permit weakening and contraction by allowing an arbitrary Γ at the leaves of typing derivations, while rule T_APP duplicates Γ but splits Δ. T_APP splits Δ using the operation $\Cup$, which allows type variables and intuitionistic expression variables to propagate to both sides of an application while restricting linear variables. We treat all types as linear, and use the ! constructor to account

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \cdot} \qquad \frac{\vdash \Gamma \quad \Gamma \vdash \tau \quad x \notin dom(\Gamma)}{\vdash \Gamma, x{:}\tau} \qquad \frac{\vdash \Gamma \quad \alpha \notin dom(\Gamma)}{\vdash \Gamma, \alpha}$$

$\boxed{\Gamma \vdash \Delta}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \cdot} \qquad \frac{\Gamma \vdash \Delta \quad \Gamma \vdash \tau \quad a \notin dom(\Delta)}{\Gamma \vdash \Delta, a{:}\tau}$$

$\boxed{\Gamma \vdash \tau}$

$$\frac{\vdash \Gamma \quad \alpha \in \Gamma}{\Gamma \vdash \alpha} \qquad \frac{\Gamma \vdash \tau_1 \quad \Gamma \vdash \tau_2}{\Gamma \vdash \tau_1 \multimap \tau_2} \qquad \frac{\Gamma, \alpha \vdash \tau}{\Gamma \vdash \forall \alpha.\tau}$$

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash !\tau} \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \tau'}{\Gamma \vdash \tau \& \tau'} \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \tau'}{\Gamma \vdash \tau \otimes \tau'}$$

$\boxed{\Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \cdot \Cup \cdot = \cdot} \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta \quad a \notin \Gamma, \Delta}{\Gamma \vdash \Delta_1, a{:}\tau \Cup \Delta_2 = \Delta, a{:}\tau} \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta \quad a \notin \Gamma, \Delta}{\Gamma \vdash \Delta_1 \Cup \Delta_2, a{:}\tau = \Delta, a{:}\tau}$$

$\boxed{\Gamma; \Delta \vdash M{:}\tau}$

$$\frac{\Gamma \vdash (a{:}\tau)}{\Gamma; a{:}\tau \vdash a{:}\tau}\ \text{T\_LVAR} \qquad \frac{\vdash \Gamma \quad x{:}\tau \in \Gamma}{\Gamma; \cdot \vdash x{:}\tau}\ \text{T\_IVAR}$$

$$\frac{\Gamma \vdash \tau_1 \quad \Gamma; \Delta, a{:}\tau_1 \vdash M_2{:}\tau_2}{\Gamma; \Delta \vdash \lambda a{:}\tau_1.M_2{:}\tau_1 \multimap \tau_2}\ \text{T\_ABS} \qquad \frac{\Gamma; \Delta_1 \vdash M_1{:}\tau_{11} \multimap \tau_{12} \quad \Gamma; \Delta_2 \vdash M_2{:}\tau_{11} \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta_3}{\Gamma; \Delta_3 \vdash M_1\ M_2{:}\tau_{12}}\ \text{T\_APP}$$

$$\frac{\Gamma, \alpha; \Delta \vdash M_1{:}\tau}{\Gamma; \Delta \vdash \Lambda\alpha.M_1{:}\forall\alpha.\tau}\ \text{T\_TABS} \qquad \frac{\Gamma; \Delta \vdash M_1{:}\forall\alpha.\tau \quad \Gamma \vdash \tau_1}{\Gamma; \Delta \vdash M_1[\tau_1]{:}\tau\{\tau_1/\alpha\}}\ \text{T\_TAPP}$$

$$\frac{\Gamma; \cdot \vdash M{:}\tau}{\Gamma; \cdot \vdash !M{:}!\tau}\ \text{T\_BANG} \qquad \frac{\Gamma; \Delta_1 \vdash M_1{:}!\tau_1 \quad \Gamma, x{:}\tau_1; \Delta_2 \vdash M_2{:}\tau_2 \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta_3}{\Gamma; \Delta_3 \vdash \mathbf{let}\ !x = M_1\ \mathbf{in}\ M_2{:}\tau_2}\ \text{T\_LET}$$

$$\frac{\Gamma; \Delta \vdash M_1{:}\tau_1 \quad \Gamma; \Delta \vdash M_2{:}\tau_2}{\Gamma; \Delta \vdash \langle M_1, M_2\rangle{:}\tau_1 \& \tau_2}\ \text{T\_APAIR} \qquad \frac{\Gamma; \Delta \vdash M{:}\tau_1 \& \tau_2}{\Gamma; \Delta \vdash M.i{:}\tau_i}\ \text{T\_PROJI}$$

$$\frac{\Gamma; \Delta_1 \vdash M_1{:}\tau_1 \quad \Gamma; \Delta_2 \vdash M_2{:}\tau_2 \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta_3}{\Gamma; \Delta_3 \vdash (M_1, M_2){:}\tau_1 \otimes \tau_2}\ \text{T\_PPAIR} \qquad \frac{\Gamma; \Delta_1 \vdash M_1{:}\tau_1 \otimes \tau_2 \quad \Gamma; \Delta_2, a_1{:}\tau_1, a_2{:}\tau_2 \vdash M_2{:}\tau \quad \Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta_3}{\Gamma; \Delta_3 \vdash \mathbf{let}\ (a_1, a_2) = M_1\ \mathbf{in}\ M_2{:}\tau}\ \text{T\_PLET}$$

Fig. 2. Kinding and typing rules for PDILL.

for terms that can be discarded or duplicated. A ! value cannot capture linear variables, and is used lazily. We have two distinct forms of pairs: (M_1, M_2) is a muplicative pair where both components are used eagerly (eliminated via pattern matching using let), and the free linear variables of M_1 and M_2 are disjoint; $\langle M_1, M_2 \rangle$ is an additive pair where only one component is used lazily (projected), and the free linear variables of M_1 and M_2 are identical. Lemma 1 states that linear contexts Δ contain exactly the free linear variables in a well-typed expression:

Lemma 1. *If $\Gamma; \Delta \vdash M{:}\tau$, then the set of free linear variables in M is $\mathsf{dom}(\Delta)$.*

By the preservation lemma and Lemma 1, we also have that the set of free linear variables in a well-typed term is preserved under reduction:

Lemma 2. *If $\Gamma; \Delta \vdash M{:}\tau$, and $M \longrightarrow^* M'$, then M and M' contain the same free linear variables.*

$$\text{Evaluation Contexts } E ::= \Box \mid E\,M \mid V\,E \mid E[\tau] \mid E.1 \mid E.2 \mid (E, M) \mid (V, E) \mid \mathbf{let}\,(a, a') = E\,\mathbf{in}\,M \mid \mathbf{let}\,!x = E\,\mathbf{in}\,M$$

$$\boxed{M \longrightarrow M'}$$

$$\frac{M \longrightarrow M'}{E[M] \longrightarrow E[M']} \qquad \frac{}{(\Lambda\alpha.M)[\tau] \longrightarrow M\{\tau/\alpha\}} \qquad \frac{}{\mathbf{let}\,!x = !M_1\,\mathbf{in}\,M_2 \longrightarrow M_2\{M_1/x\}}$$

$$\frac{}{(\lambda a{:}\tau.M)\,V' \longrightarrow M\{V'/a\}} \qquad \frac{}{\mathbf{let}\,(a_1, a_2) = (V_1, V_2)\,\mathbf{in}\,M \longrightarrow M\{V_1/a_1\}\{V_2/a_2\}}$$

Fig. 3. Evaluation relations for PDILL.

3 Parametricity for PDILL

This section presents the main contribution of this paper, namely a syntactic logical relation for PDILL. Proof by logical relations is a fundamental technique for proving properties of programming languages. This technique was first developed by Tait [21] to prove that the simply typed λ-calculus is strongly normalizing, and was adopted by Girard [13] to prove normalization for System F^{ω}.

Although their logical relations were for *open* terms via open substitutions, there are a large number of related works on logical relations that interpret types as relations for closed terms. Its basic idea is to first define an equivalence relation for closed values inductively on type structures. The case of arrow types is a typical case where two closed functions are logically related at the type $\tau_1 \to \tau_2$ if they evaluate to values that are logically related at τ_2 with closed arguments that are logically related at τ_1. Logical relations are extended to closed terms with evaluation, and then extended to open terms via closing substitutions. The fundamental theorem proves that any well-typed term, under those closing substitutions, is related to itself by the relation induced

logically by its type. To build logical relations for recursive functions and quantified types, Pitts [17] developed logical relations based on the notion of $\top\top$-closed relations. Bierman et al. [6] also used $\top\top$-closed relations for the parametricity principle of the programming language Lily—a polymorphic intuitionistic/linear lambda calculus. Ahmed [1] proposed step-indexed logical relations for recursive and stateful types. Ahmed et al. [2], Birkedal et al. [8] extended step-indexed logical relations with possible "worlds" based on Kripke logical relations for increasingly realistic languages.

Crary [10] developed the technique of Kripke logical relations for *open* terms under open contexts to show the completeness of an equivalence algorithm for terms in a simply typed λ-calculus. To ensure that a relation holds by adding variables to the current context, the logical relation is required to be monotone with respect to contexts analogous to "worlds" in Kripke models. Hasegawa [14] developed a notion of Kripke logical predicates for two fragments of intuitionistic linear logic (MILL and DILL) in terms of their category-theoretic models. However these logical relations do not support both polymorphism and linearity.

Our logical relation first interprets types as relations for pairs of *open* values, then extends to logical relations for *open* terms with evaluation and *open* relation substitutions. Open relation substitutions map term variables to related open values, and type variables to relations for open terms. The fundamental theorem shows that a well-typed term, under open substitutions and evaluation, is related to itself in the interpretation of its type. Lemma 1 and 2 also prove that the set of the free linear variables in a well-typed term is equal to its linear context, and preserved under reduction. Section 4 illustrates how these properties and logical relations eliminate possible inhabitants for a type.

3.1 Definitions

Figure 4 gives a logical relation for PDILL. In the following sections, we shaded the difference in definitions, lemmas and proofs between our open logical relation and a closed one. Without shaded parts, the formalism defaults to the setting of a closed logical relation. We abbreviate $\Gamma; \Delta \vdash M{:}\tau \wedge \Gamma; \Delta \vdash M'{:}\tau$ to $\Gamma; \Delta \vdash M, M'{:}\tau$, and $\cdot\,;\cdot \vdash J$ to $\vdash J$. A type relation substitution, θ, maps type variables to relations R for pairs of open terms with type τ and τ'. We use θ_l and θ_r to denote relation type substitutions. γ and δ are intuitionistic and linear open value relation substitutions respectively. We also use γ_l, γ_r, δ_l, and δ_r to denote relation value substitutions.

$$\begin{array}{lll} \text{Type Substitution Contexts} & \theta ::= \cdot & \mid\ \theta, \alpha{\mapsto}(R, \tau, \tau') \\ \text{Intuitionistic Substitution Contexts} & \gamma ::= \cdot & \mid\ \gamma, x{\mapsto}(V, V') \\ \text{Linear Substitution Contexts} & \delta ::= \cdot & \mid\ \delta, a{\mapsto}(V, V') \end{array}$$

To simplify the definition of substitutions, we require that the domains of substitutions are disjoint from the free variables of types and values in the ranges of substitutions. Thus types and values in substitutions do not reuse any variable in their domains. We use Γ and Δ to typecheck the domains of substitutions, and G and D to denote contexts that typecheck types, relations, and terms in the ranges of θ, γ and δ. Our definitions force the variables of G and D to be disjoint from the variables of Γ and Δ. This simplification has been chosen since all parametricity reasoning in our experience involves only disjoint variables. We consider the definition of substitutions on types and values that can reuse variables in domains of substitutions in future work.

$\boxed{G; D \vdash V \sim V' \in \tau{:}\theta}$ Related open values

$$G; D \vdash V \sim V' \in \alpha{:}\theta \triangleq \alpha{\mapsto}(R, \tau, \tau') \in \theta \wedge (V, V') \in R \qquad \text{OVR\_VAR}$$

$$G; D \vdash \lambda a{:}\tau_1.M \sim \lambda a{:}\tau_1.M' \in \tau_1 \multimap \tau_2{:}\theta \triangleq \qquad \text{OVR\_ARR}$$
$$\forall(G; \boldsymbol{D}_1 \vdash V{:}\theta_l(\tau_1) \wedge G; \boldsymbol{D}_1 \vdash V'{:}\theta_r(\tau_1) \wedge G; \boldsymbol{D}_1 \vdash V \sim V' \in \tau_1{:}\theta \wedge G \vdash D, \boldsymbol{D}_1).$$
$$G; D, \boldsymbol{D}_1 \vdash (\lambda a{:}\tau_1.M)\, V \simeq (\lambda a{:}\tau_1.M')\, V' \in \tau_2{:}\theta$$

$$G; D \vdash \Lambda\alpha.M \sim \Lambda\alpha.M' \in \forall\alpha.\tau{:}\theta \triangleq \qquad \text{OVR\_ALL}$$
$$\forall(R \in \tau_2 \leftrightarrow \tau_2' \dashv G).$$
$$G; D \vdash (\Lambda\alpha.M)[\tau_2] \simeq (\Lambda\alpha.M')[\tau_2'] \in \tau{:}(\theta, \alpha{\mapsto}(R, \tau_2, \tau_2'))$$

$$G; D \vdash !M \sim !M' \in !\tau{:}\theta \triangleq \qquad \text{OVR\_BANG}$$
$$G; D \vdash M \simeq M' \in \tau{:}\theta$$

$$G; D \vdash \langle M_1, M_2\rangle \sim \langle M_1', M_2'\rangle \in \tau_1 \& \tau_2{:}\theta \triangleq \qquad \text{OVR\_WITH}$$
$$G; D \vdash M_1 \simeq M_1' \in \tau_1{:}\theta \wedge G; D \vdash M_2 \simeq M_2' \in \tau_2{:}\theta$$

$$G; D \vdash (V_1, V_2) \sim (V_1', V_2') \in \tau_1 \otimes \tau_2{:}\theta \triangleq \qquad \text{OVR\_TENSOR}$$
$$\exists D_1\, D_2, G; D_1 \vdash V_1 \sim V_1' \in \tau_1{:}\theta \wedge G; D_2 \vdash V_2 \sim V_2' \in \tau_2{:}\theta$$
$$\wedge\, G \vdash D_1 \Cup D_2 = D$$

$\boxed{G; D \vdash M \simeq M' \in \tau{:}\theta}$ Related open terms

$$\frac{G; D \vdash M{:}\theta_l(\tau) \quad G; D \vdash M'{:}\theta_r(\tau) \quad M \longrightarrow^* V \quad M' \longrightarrow^* V' \quad G; D \vdash V \sim V' \in \tau{:}\theta}{G; D \vdash M \simeq M' \in \tau{:}\theta}$$

$\boxed{\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D}$ Open relation substitutions

$$\frac{\vdash G}{\cdot; \cdot \vdash \cdot|\cdot; \cdot{:}G; \cdot} \qquad \frac{\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D_1 \quad \Gamma \vdash \tau \quad G \vdash D_1, \boldsymbol{D}_2 \quad G; \boldsymbol{D}_2 \vdash V \sim V' \in \tau{:}\theta \quad a \notin \Gamma, \Delta, G, (D_1, \boldsymbol{D}_2)}{\Gamma; \Delta, a{:}\tau \vdash \theta|\gamma, a{\mapsto}(V, V'){:}G; D_1, \boldsymbol{D}_2}$$

$$\frac{\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D \quad \Gamma \vdash \tau \quad G; \cdot \vdash V \sim V' \in \tau{:}\theta \quad x \notin \Gamma, \Delta, G, D}{\Gamma, x{:}\tau; \Delta \vdash \theta|\gamma, x{\mapsto}(V, V'); \delta{:}G; D} \qquad \frac{\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D \quad R \in \tau \leftrightarrow \tau' \dashv G \quad \alpha \notin \Gamma, \Delta, G, D}{\Gamma, \alpha; \Delta \vdash \theta, \alpha{\mapsto}(R, \tau, \tau')|\gamma; \delta{:}G; D}$$

$\boxed{\Gamma; \Delta \vdash \theta|\gamma; \delta_1 \Cup \delta_2 = \delta{:}G; D_1 \Cup D_2 = D}$ Disjoint linear relation substitutions

$$\frac{\Gamma; \cdot \vdash \theta|\gamma; \cdot{:}G; \cdot}{\Gamma; \cdot \vdash \theta|\gamma; \cdot \Cup \cdot = \cdot{:}G; \cdot \Cup \cdot = \cdot}$$

$$\frac{\Gamma; \Delta \vdash \theta|\gamma; \delta_1 \Cup \delta_2 = \delta{:}G; D_1 \Cup D_2 = D_3 \quad \Gamma \vdash \tau \quad G; D' \vdash V \sim V' \in \tau{:}\theta \quad a \notin \Gamma, \Delta, G, (D_3, D')}{\Gamma; \Delta, a{:}\tau \vdash \theta|\gamma; (\delta_1, a{\mapsto}(V, V')) \Cup \delta_2 = (\delta, a{\mapsto}(V, V')){:}G; D_1, D' \Cup D_2 = D_3, D'}$$

$$\frac{\Gamma; \Delta \vdash \theta|\gamma; \delta_1 \Cup \delta_2 = \delta{:}G; D_1 \Cup D_2 = D_3 \quad \Gamma \vdash \tau \quad G; D' \vdash V \sim V' \in \tau{:}\theta \quad a \notin \Gamma, \Delta, G, (D_3, D')}{\Gamma; \Delta, a{:}\tau \vdash \theta|\gamma; \delta_1 \Cup (\delta_2, a{\mapsto}(V, V')) = (\delta, a{\mapsto}(V, V')){:}G; D_1 \Cup D_2, D' = D_3, D'}$$

Fig. 4. Related open values and terms for PDILL.

Two open PDILL values are related at type τ under a relation substitution θ, written $G; D \vdash V \sim V' \in \tau{:}\theta$ and defined inductively on type structures. An intuitionistic context G and a linear context D contain the free type and term variables in the values V and V'. Type variables in indexed types τ are disjoint from the domains of G and D. θ maps variables in τ to relations with types under contexts G and D.

A relation for each type variable α is defined by looking up relations in substitution contexts θ. Two functions are related if and only if they map related inputs to related outputs. When testing equivalence on two open function values under a linear context D, OVR_ARR applies the two functions to arbitrary open related values under a linear context $\boldsymbol{D}_1$, which is disjoint from D, and tests if they can normalize to two related values under a merged linear context $D, \boldsymbol{D}_1$. A standard logical relation does not necessarily require related closed arguments to be well-typed in the arrow type case. However, the logical relation for PDILL allows only well-typed related open arguments at OVR_ARR because the invariant that reduction preserves the set of free linear variables is satisfied for only typechecked terms. PDILL ensures that a ! value cannot capture linear variables in its closure; thus we conclude that $\boldsymbol{D}_1 = \cdot$ if τ_1 is a ! type, since the arguments are well-typed.

The idea of the parametricity of polymorphism is that even when passed completely different type arguments, logically related type abstractions must behave uniformly, so they may not use different algorithms for different arguments by analyzing the type argument. To formalize the independence of type arguments, Reynolds' approach is to interpret a type quantifier as an (almost) arbitrary relation that relates two type arguments at which the type quantifier can be instantiated, although a relational interpretation needs to satisfy certain admissibility conditions [17]. Relational interpretations for type variables are stored in type relation substitutions θ that parameterize logical relations. At rule OVR_ALL, two type abstractions are related if and only if two type inputs, which are related by any well-formed relation, generate related outputs with type substitutions extended by that new relation for type variable α. $R \in \tau_2 \leftrightarrow \tau_2' \dashv G$ defines a well-formed binary relation on open terms with types τ_2 and τ_2' respectively. τ_2 and τ_2' are under an intuitionistic context G.

In PDILL, the behavior of logically related type abstractions cannot depend on which linear contexts the arguments with polymorphic types will take, either. Suppose we have related values of type $\forall\alpha.\alpha \multimap \alpha$ under contexts G and D with an empty relation type substitution. Rule OVR_ALL maps α to $R \in \tau_2 \leftrightarrow \tau_2' \dashv G$. At rule OVR_ARR, we must choose a pair of related values $G; D_1 \vdash V \sim V' \in \alpha{:}(\alpha{\mapsto}(R, \tau_2, \tau_2'))$ under an arbitrary fresh linear context D_1. Rule OVR_VAR requires that $(V, V') \in R$. If R is a relation that only takes the linear context D, we cannot choose the above V and V' for an arbitrary D_1. Therefore, $R \in \tau \leftrightarrow \tau' \dashv G$ does not specify which linear context D to use. For any pair of terms related by R, it only requires that there exists a linear context that can typecheck them:

$G \vdash \tau \wedge G \vdash \tau' \wedge \forall((M, M') \in R).\, \exists D,\, G; D \vdash M{:}\tau \wedge G; D \vdash M'{:}\tau'$

Two values $!M$ and $!M'$ are logically related by type $!\tau$ if and only if the two suspended terms M and M' are related by type τ. Under a linear context D, two values are logically related by an additive product $\tau_1 \& \tau_2$ if and only if their components are logically related under D respectively; two values are logically related by a muplica-

tive product $\tau_1 \otimes \tau_2$ if and only if D can be split into disjoint D_1 and D_2, and their components are related under D_1 and D_2 respectively.

$G; D \vdash M \simeq M' \in \tau{:}\theta$ extends logical relations on open values to well-typed open terms with evaluation.The well-formedness of open terms ensures that terms use linear expressions exactly once, and preserves the invariant that reduction does not consume free linear variables. A relation substitution θ also maps free variables in τ that are disjoint to the domains of G and D to relations with types under the context G.

$\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D$ formalizes open relation substitutions. Relation substitutions θ, γ, and δ map type and term variables in Γ and term variables in Δ respectively to open types and values with free variables in G and D. The domains of Γ and Δ and the domains of G and D are disjoint. Related open values indexed by ! types must typecheck under an empty linear context and do not introduce free linear variables in D, not changing D. Adding relations with related types to relation substitutions does not change D either. However, related open values indexed by linear types can contain free linear variables, introducing new linear variables in D. When extending the existing linear context Δ with a variable a of type τ, we must add a pair of related values under a linear context $\boldsymbol{D}_2$ that is disjoint from D_1, extending D_1 to $D_1, \boldsymbol{D}_2$. The intuitionistic context G satisfies weakening and contraction properties.

Since linear variables in Δ must be used exactly once, the corresponding linear relation substitutions should satisfy a similar property. That is, a substitution on a linear variable must occur exactly once. Lemma 3 formalizes this idea, stating that, in a judgment $\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D$, because only related values in δ can contain variables in D, given that Δ can be split into Δ_1 and Δ_2, we can split the linear relation substitution δ into two linear relation substitutions δ_1 and δ_2 under D_1 and D_2 respectively, where D must be split into D_1 and D_2. Figure 4 also defines the operation $\Cup$ on δ.

Lemma 3 (Split Linear Relation Substitutions). *If $\Gamma \vdash \Delta_1 \Cup \Delta_2 = \Delta$, and $\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D$, then $\exists \delta_1, \delta_2, D_1, D_2$. $\Gamma; \Delta \vdash \theta|\gamma; \delta_1 \Cup \delta_2 = \delta{:}G; D_1 \Cup D_2 = D$, $\Gamma; \Delta_1 \vdash \theta|\gamma; \delta_1{:}G; D_1$, and $\Gamma; \Delta_2 \vdash \theta|\gamma; \delta_2{:}G; D_2$.*

$\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M' \in \tau$ defines that two well-typed terms with the same type are logically equivalent if they are related under all open relation substitutions. $\Gamma; \Delta \vdash M \simeq^{\star}_{\mathsf{log}} M' \in \tau$ defines closed logical equivalence:

$$\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M' \in \tau \triangleq \Gamma; \Delta \vdash M{:}\tau \wedge \Gamma; \Delta \vdash M'{:}\tau \wedge \forall(\Gamma; \Delta \vdash \theta|\gamma; \delta{:}G; D).\ G; D \vdash \theta_l(\gamma_l(\delta_l(M))) \simeq \theta_r(\gamma_r(\delta_r(M'))) \in \tau{:}\theta$$

$$\Gamma; \Delta \vdash M \simeq^{\star}_{\mathsf{log}} M' \in \tau \triangleq \Gamma; \Delta \vdash M{:}\tau \wedge \Gamma; \Delta \vdash M'{:}\tau \wedge \forall(\Gamma; \Delta \vdash \theta|\gamma; \delta).\ \theta_l(\gamma_l(\delta_l(M))) \simeq \theta_r(\gamma_r(\delta_r(M'))) \in \tau{:}\theta$$

3.2 Fundamental Theorem

Theorem 2 shows parametricity, which states that, independent of open relation substitutions for terms, types and relations under contexts Γ and Δ, a well-typed term, via open substitutions, is related to itself under the same contexts Γ and Δ.

As usual, the proof of parametricity depends on a compositionality lemma (Lemma 4) for logical relations. Note that R contains a *family* of logical relations under different D''s, but not only the linear context D that σ takes.

Lemma 4 (Compositionality). $G; D \vdash V \sim V' \in \sigma{:}(\theta, \alpha{\mapsto}(R, \theta_l(\tau), \theta_r(\tau)))$ *iff* $G; D \vdash V \sim V' \in \sigma\{\tau/\alpha\}{:}\theta$ *where* $R \in \theta_l(\tau) \leftrightarrow \theta_r(\tau) \dashv G$ *is a relation such that* $(U, U') \in R$ *iff* $\exists D', G; D' \vdash U \sim U' \in \tau{:}\theta$.

Theorem 2 (Parametricity). *If* $\Gamma; \Delta \vdash M{:}\tau$, *then* $\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M \in \tau$.

Proof. By induction on the typing derivation, case T_APP relies on Lemma 3, and case T_TAPP uses Lemma 4.

If the logical relation interprets types by relations for only closed terms, Theorem 2 degrades to Theorem 1. Thus Theorem 2 is a more general result than Theorem 1.

3.3 Soundness & Completeness

This section proves that the logical equivalence $\Gamma; \Delta \vdash M \simeq^{\star}_{\mathsf{log}} M \in \tau$, which interprets types on closed terms, is sound and complete with respect to contextual equivalence. We use these results to show the properties of contextual equivalence in Section 4. We also prove that the logical equivalence $\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M \in \tau$, which interprets types on open relations, is sound with respect to contextual equivalence. Here we state interesting aspects of the proofs. Further details are available online in our Coq scripts.

The definition of contextual equivalence [17] is based on identifying a type of *answers* that are observable outcomes of closed programs. To achieve this, we enrich the system with a base type, $\mathbf{Bool}$, containing two constants, $\mathbf{true}$ and $\mathbf{false}$, that serve as possible answers for a *complete computation* that is a closed expression of type $\mathbf{Bool}$. Two values are logically related with type $\mathbf{Bool}$ iff they are both $\mathbf{true}$ or both $\mathbf{false}$.

Kleene equivalence $M \simeq_{\mathsf{kleene}} M'$ is defined for complete computations M and M' by requiring that $M \longrightarrow^* \mathbf{true} \wedge M' \longrightarrow^* \mathbf{true}$ or $M \longrightarrow^* \mathbf{false} \wedge M' \longrightarrow^* \mathbf{false}$.

To define contextual equivalence, we define contexts C as expressions with a single hole $\square$. Bierman [5] and Crole [11] showed how to define a program context in a linear setting without polymorphic types. We extend their definitions with polymorphic types.

Typing judgments for contexts have the form $\Gamma'; \Delta' \vdash C{:}(\Gamma; \Delta \triangleright \tau) \rightsquigarrow \tau'$, where $\Gamma; \Delta \triangleright \tau$ indicates the type of the hole. Most typing judgments for contexts are similar to the typing rules listed in Figure 5 and 6 at [25]. Plugging an expected expression into a well-formed context produces a well-typed term with type τ' under Γ' and Δ'.

Two well-formed expressions of the same type are contextually equivalent if, given any well-formed context C that is a complete computation when an expected expression is plugged into, $C[M]$ and $C[M']$ are Kleene equivalent:

$$\Gamma; \Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau \triangleq \Gamma; \Delta \vdash M{:}\tau \wedge \Gamma; \Delta \vdash M'{:}\tau \wedge \forall(\cdot; \cdot \vdash C{:}(\Gamma; \Delta \triangleright \tau) \rightsquigarrow \mathbf{Bool}).C[M] \simeq_{\mathsf{kleene}} C[M']$$

We first prove that logical equivalence is congruent with respect to contexts:

Lemma 5 (Logical Equivalence is Congruent). *If* $\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M' \in \tau$, *and* $\Gamma'; \Delta' \vdash C{:}(\Gamma; \Delta \triangleright \tau) \rightsquigarrow \tau'$, *then* $\Gamma'; \Delta' \vdash C[M] \simeq_{\mathsf{log}} C[M'] \in \tau'$.

Theorem 3 (Soundness). *If* $\Gamma; \Delta \vdash M \simeq_{\mathsf{log}} M' \in \tau$, *then* $\Gamma; \Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau$.

Proof. By Lemma 5, and that $\mathbf{true}$ and $\mathbf{false}$ are not related.

When interpreting relations for only closed terms:

Theorem 4. *If* $\Gamma;\Delta \vdash M \simeq^{\star}_{\mathsf{log}} M' \in \tau$, *then* $\Gamma;\Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau$.

A typical approach [17] [1] to proving that a logical relation is complete with respect to contextual equivalence is via the notion of CIU-equivalence, introduced by Mason and Talcott [15]. Let evaluation contexts E be a subset of contexts C, and consist of only holes for closed terms. Two well-formed closed terms M and M' are CIU-equivalent if, given any context E, $E[M]$ and $E[M']$ are complete computations, and Kleene equivalent. Given any closing substitution $\Gamma;\Delta \vdash s|g;d$ where s maps type variables in Γ to closed types, g and d map intuitionistic and linear term variables typed in Γ and Δ to closed values respectively, we extend it to open terms as follows:

$$\Gamma;\Delta \vdash M \simeq_{\mathsf{ciu}} M' \in \tau \triangleq \Gamma;\Delta \vdash M{:}\tau \wedge \Gamma;\Delta \vdash M'{:}\tau \wedge \forall(\Gamma;\Delta \vdash s|g;d \wedge \cdot;\cdot \vdash E{:}(\cdot;\cdot \triangleright s(\tau)) \rightsquigarrow \mathbf{Bool}).\ E[s(g(d(M)))] \simeq_{\mathsf{kleene}} E[s(g(d(M')))]$$

Given a term M with closing substitutions s, g and d, we can construct a context C such that when M is placed into the hole, $C[M]$ evaluates to the same value that $s(g(d(M)))$ can reduce to. We can use the context $(\cdots(\cdots(\cdots(\Lambda\alpha_1\cdots\alpha_n.\lambda x_1 : \tau_1 \cdots x_m : \tau_m.\lambda a_1 : \tau'_1 \cdots a_l : \tau'_l.\Box)\cdots[\sigma_i]\cdots)\cdots V_i\cdots)\cdots V'_i\cdots)$ where s maps α_i to closed type σ_i, g maps intuitionistic variables x_i with type τ_i to closed terms V_i, and d maps linear variables a_i with type τ'_i to closed terms V'_i. Therefore, we have:

Lemma 6 ($\simeq_{\mathsf{ctx}} \subseteq \simeq_{\mathsf{ciu}}$). *If* $\Gamma;\Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau$, *then* $\Gamma;\Delta \vdash M \simeq_{\mathsf{ciu}} M' \in \tau$.

To ensure that our closed logical relations are complete, we will restrict attention to a certain class of *admissible* binary relations R that respect CIU-equivalence [1]:

Definition 1 (Admissibility). $R \in \tau_2 \leftrightarrow \tau'_2$ *is admissible iff it satisfies that if* $(M_1, M_2) \in R$, $\cdot;\cdot \vdash M_1 \simeq_{\mathsf{ciu}} M'_1 \in \tau_2$, $\cdot;\cdot \vdash M_2 \simeq_{\mathsf{ciu}} M'_2 \in \tau'_2$, *then* $(M'_1, M'_2) \in R$.

Thus we prove that closed logical relations respect CIU-equivalence:

Lemma 7 (Equivalence-Respecting). *If* $\Gamma;\Delta \vdash M_1 \simeq^{\star}_{\mathsf{log}} M_2 \in \tau$, $\Gamma;\Delta \vdash M_1 \simeq_{\mathsf{ciu}} M'_1 \in \tau$, *and* $\Gamma;\Delta \vdash M_2 \simeq_{\mathsf{ciu}} M'_2 \in \tau$, *then* $\Gamma;\Delta \vdash M'_1 \simeq^{\star}_{\mathsf{log}} M'_2 \in \tau$.

By Theorem 1, Lemmas 6 and Lemma 7, we have:

Theorem 5 (Completeness). *If* $\Gamma;\Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau$, *then* $\Gamma;\Delta \vdash M \simeq^{\star}_{\mathsf{log}} M' \in \tau$.

We have thus proved that the relations $\simeq^{\star}_{\mathsf{log}}$, $\simeq_{\mathsf{ctx}}$ and $\simeq_{\mathsf{ciu}}$ coincide with each other. It remains an open question as to whether $\simeq_{\mathsf{log}}$ is complete.

4 Examples

In this section we present applications of our method. Our reasoning about equivalence and definitions of relations in this section depends on properties of contextual equivalence that include reflexivity, symmetry, transitivity, $\beta\eta$-reduction, and congruence (see Figure 7 at [25]). Most of these properties are straightforward, following from

the definition of contextual equivalence and Theorems 3, 4 and 5. With closed logical relation $\simeq^{\star}_{\mathsf{log}}$, we proved that τ is isomorphic to $\forall\alpha.(\tau \multimap \alpha) \multimap \alpha$, $\forall\alpha.(\tau_1 \multimap (\tau_2 \multimap \alpha)) \multimap \alpha$ encodes $\tau_1 \otimes \tau_2$, and other type encodings in [23] [7] [16]. This section is focused on novel applications that open logical relations can prove. We abbreviate $\Gamma; \Delta \vdash M \simeq_{\mathsf{ctx}} M' \in \tau \wedge \Gamma; \Delta \vdash M' \simeq_{\mathsf{ctx}} M'' \in \tau$ to $\Gamma; \Delta \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} M'' \in \tau$.

4.1 Polymorphic Boolean

A closed term M with type $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$ does not exist, while a closed term with type $\forall\alpha.!\alpha \multimap !\alpha \multimap !\alpha$ must behave like a polymorphic boolean function.

Given $\cdot;\cdot \vdash M{:}\forall\alpha.\alpha \multimap \alpha \multimap \alpha$, Theorem 2 gives:
$\forall(R \in \tau \leftrightarrow \tau' \dashv \star).\, \forall(\cdot; D_1 \vdash V_1{:}\tau \wedge \cdot; D_1 \vdash V_1'{:}\tau' \wedge (V_1, V_1') \in R).$
$\forall(\cdot; D_2 \vdash V_2{:}\tau \wedge \cdot; D_2 \vdash V_2'{:}\tau' \wedge (V_2, V_2') \in R).$
$M[\tau]\, V_1\, V_2 \longrightarrow^* V \wedge M[\tau']\, V_1'\, V_2' \longrightarrow^* V' \wedge (V, V') \in R$
Here D_1 and D_2 are disjoint. Consider $\cdot; D_1 \vdash V_1{:}\tau$ and $\cdot; D_2 \vdash V_2{:}\tau$; let R be a relation $\{(M, M') \mid \cdot; D_1 \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot; D_2 \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_2 \in \tau\}$. By the free theorem and properties of $\simeq_{\mathsf{ctx}}$, $\cdot; D_1 \vdash (M[\tau]\, V_1\, V_2) \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot; D_2 \vdash (M[\tau]\, V_1\, V_2) \simeq_{\mathsf{ctx}} V_2 \in \tau$. By regularity of $\simeq_{\mathsf{ctx}}$, we have
(1) $\cdot; D_1 \vdash M[\tau]\, V_1\, V_2{:}\tau \vee \cdot; D_2 \vdash M[\tau]\, V_1\, V_2{:}\tau$
However, by typing rules we can also derive
(2) $\cdot; D_1, D_2 \vdash M[\tau]\, V_1\, V_2{:}\tau$
By Lemma 1 on (1) and (2), $\mathsf{dom}(D_1) = \mathsf{dom}(D_1, D_2) \vee \mathsf{dom}(D_2) = \mathsf{dom}(D_1, D_2)$. In this case, D_1 and D_2 can be non-empty and must be disjoint from each other; this contradicts the above constraint. Thus we conclude that the type $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$ is empty. The closed logical relation cannot show this contradiction because it requires that D_1 and D_2 must be empty. Similarly, we can prove that the types $\forall\alpha.\alpha \multimap \alpha \multimap (\alpha \& \alpha)$ and $\forall\alpha.\alpha \multimap (\alpha \otimes \alpha)$ do not have any closed inhabitant.

If a closed term M is of type $\forall\alpha.!\alpha \multimap !\alpha \multimap !\alpha$, its free theorem is:
$\forall(R \in \tau \leftrightarrow \tau' \dashv \star).$
$\forall(\cdot;\cdot \vdash M_1{:}\tau \wedge \cdot;\cdot \vdash M_1'{:}\tau' \wedge M_1 \longrightarrow^* V_1 \wedge M_1' \longrightarrow^* V_1' \wedge (V_1, V_1') \in R).$
$\forall(\cdot;\cdot \vdash M_2{:}\tau \wedge \cdot;\cdot \vdash M_2'{:}\tau' \wedge M_2 \longrightarrow^* V_2 \wedge M_2' \longrightarrow^* V_2' \wedge (V_2, V_2') \in R).$
$M[\tau]\, !M_1\, !M_2 \longrightarrow^* !N \wedge M[\tau']\, !M_1'\, !M_2' \longrightarrow^* !N' \wedge$
$N \longrightarrow^* V \wedge N' \longrightarrow^* V' \wedge (V, V') \in R$
which only allows us to reason about closed terms, because ! values do not capture any free linear variables. Given any $\cdot;\cdot \vdash !M_1, !M_2{:}!\tau$, by strong normalization (that is proved by Theorem 1) there exist values V_1 and V_2 such that $M_1 \longrightarrow^* V_1$ and $M_2 \longrightarrow^* V_2$. Let R be a relation $\{(M, M') \mid \cdot;\cdot \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot;\cdot \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_2 \in \tau\}$. The free theorem shows that $\cdot;\cdot \vdash (M[\tau]\, !M_1\, !M_2) \simeq_{\mathsf{ctx}} !M_1 \in !\tau$ or $\cdot;\cdot \vdash (M[\tau]\, !M_1\, !M_2) \simeq_{\mathsf{ctx}} !M_2 \in !\tau$, namely that M behaves like a polymorphic boolean function. We can also derive possible inhabitants of types $\forall\alpha.!\alpha \multimap !\alpha \multimap (!\alpha \& !\alpha)$ and $\forall\alpha.!\alpha \multimap (!\alpha \otimes !\alpha)$.

4.2 Multiplicative Selection

This section derives closed inhabitants of the type $\forall\alpha.\alpha \multimap \alpha \multimap (\alpha \otimes \alpha)$. By Theorem 2, this type gives:

$\forall(R \in \tau \leftrightarrow \tau' \dashv \cdot).\, \forall(\cdot; D_1 \vdash V_1 : \tau \wedge \cdot; D_1 \vdash V_1' : \tau' \wedge (V_1, V_1') \in R).$
$\forall(\cdot; D_2 \vdash V_2 : \tau \wedge \cdot; D_2 \vdash V_2' : \tau' \wedge (V_2, V_2') \in R).$
$M[\tau]\, V_1\, V_2 \longrightarrow^* (V_3, V_4) \wedge M[\tau']\, V_1'\, V_2' \longrightarrow^* (V_3', V_4')$
$\wedge (V_3, V_3') \in R \wedge (V_4, V_4') \in R$

Here D_1 and D_2 are disjoint. Consider $\cdot; D_1 \vdash V_1 : \tau$ and $\cdot; D_2 \vdash V_2 : \tau$; let R be a relation $\{(M, M') \mid \cdot; D_1 \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot; D_2 \vdash M \simeq_{\mathsf{ctx}} M' \simeq_{\mathsf{ctx}} V_2 \in \tau\}$. By the free theorem, $M[\tau]\, V_1\, V_2 \longrightarrow^* (V_3, V_4) \wedge (V_3, V_3) \in R \wedge (V_4, V_4) \in R$. By the definition of R, $\cdot; D_1 \vdash V_3 \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot; D_2 \vdash V_3 \simeq_{\mathsf{ctx}} V_2 \in \tau$ and $\cdot; D_1 \vdash V_4 \simeq_{\mathsf{ctx}} V_1 \in \tau \vee \cdot; D_2 \vdash V_4 \simeq_{\mathsf{ctx}} V_2 \in \tau$. By regularity of $\simeq_{\mathsf{ctx}}$,
(1) $\cdot; D_1 \vdash V_3 : \tau \vee \cdot; D_2 \vdash V_3 : \tau$ and $\cdot; D_1 \vdash V_4 : \tau \vee \cdot; D_2 \vdash V_4 : \tau$
By typing rules we also have,
(2) $\cdot; D_1, D_2 \vdash (V_3, V_4) : \tau \otimes \tau$
By Lemma 1 on (1) and (2),
$\cdot; D_1 \vdash V_3 \simeq_{\mathsf{ctx}} V_1 \in \tau \wedge \cdot; D_2 \vdash V_4 \simeq_{\mathsf{ctx}} V_2 \in \tau$ and $\mathsf{dom}(D_1, D_2) = \mathsf{dom}(D_1, D_2)$ or
$\cdot; D_2 \vdash V_3 \simeq_{\mathsf{ctx}} V_2 \in \tau \wedge \cdot; D_1 \vdash V_4 \simeq_{\mathsf{ctx}} V_1 \in \tau$ and $\mathsf{dom}(D_2, D_1) = \mathsf{dom}(D_1, D_2)$ or
$\cdot; D_1 \vdash V_3 \simeq_{\mathsf{ctx}} V_1 \in \tau \wedge \cdot; D_1 \vdash V_4 \simeq_{\mathsf{ctx}} V_1 \in \tau$ and $\mathsf{dom}(D_1, D_1) = \mathsf{dom}(D_1, D_2)$ or
$\cdot; D_2 \vdash V_3 \simeq_{\mathsf{ctx}} V_2 \in \tau \wedge \cdot; D_2 \vdash V_4 \simeq_{\mathsf{ctx}} V_2 \in \tau$ and $\mathsf{dom}(D_2, D_2) = \mathsf{dom}(D_1, D_2)$.

Because D_1 and D_2 can be non-empty in our case, M can only behave like one of $\Lambda\alpha.\lambda a_1 : \alpha.\lambda a_2 : \alpha.(a_1, a_2)$, and $\Lambda\alpha.\lambda a_1 : \alpha.\lambda a_2 : \alpha.(a_2, a_1)$. If M is of type $\forall\alpha.!\alpha \multimap !\alpha \multimap (!\alpha \otimes !\alpha)$, D_1 and D_2 must be empty. Therefore M can be equivalent to one of $\Lambda\alpha.\lambda x_1 : !\alpha.\lambda x_2 : !\alpha.(x_1, x_2)$, $\Lambda\alpha.\lambda x_1 : !\alpha.\lambda x_2 : !\alpha.(x_2, x_1)$, $\Lambda\alpha.\lambda x_1 : !\alpha.\lambda x_2 : !\alpha.(x_1, x_1)$, and $\Lambda\alpha.\lambda x_1 : !\alpha.\lambda x_2 : !\alpha.(x_2, x_2)$.

4.3 Natural Numbers

This section proves properties of type $\forall\alpha.(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$ and $\forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$. Given a closed term N of type $\forall\alpha.(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$. By Theorem 2, this type gives:
$\forall(R \in \tau \leftrightarrow \tau' \dashv \cdot).\, \forall(\cdot; Dz \vdash Z : \tau \wedge \cdot; Dz \vdash Z' : \tau' \wedge (Z, Z') \in R).$
$\forall(\cdot; Ds \vdash S : \tau \multimap \tau \wedge \cdot; Ds \vdash S' : \tau' \multimap \tau' \wedge \cdot; Ds \vdash S \sim S' \in \alpha \multimap \alpha : \theta).$
$N[\tau]\, S\, Z \longrightarrow^* V \wedge N[\tau']\, S'\, Z' \longrightarrow^* V' \wedge (V, V') \in R$

Here Ds and Dz are disjoint, and $\theta = \alpha \mapsto (R, \tau, \tau')$. Consider $\cdot; Ds \vdash S : \tau \multimap \tau$ and $\cdot; Dz \vdash Z : \tau$; let R be a relation $\{(N, N') \mid \exists n.\, \cdot; Ds^n, Dz \vdash N \simeq_{\mathsf{ctx}} N' \simeq_{\mathsf{ctx}} S^n\, Z \in \tau\}$. Here $S^0 = Id$, $S^{n+1} = \lambda a : \tau.(r_n(S))\,(S^n\, a)$, $\cdot; r_i(Ds) \vdash r_i(S) : \tau \multimap \tau$ and $Ds^n = r_1(Ds), \cdots, r_n(Ds)$, where r_i is a well-formed renaming that is a finite function from free linear variables to free linear variables. A renaming is well-formed if it is identical, or it is bijective from the domain of Ds to a range that is disjoint from Ds. We have $(Z, Z) \in R$ because n can be 0. Suppose $(V, V') \in R$, we have that there exists an n such that $\cdot; Ds^n, Dz \vdash V \simeq_{\mathsf{ctx}} V' \simeq_{\mathsf{ctx}} S^n\, Z \in \tau$ where (Ds^n, Dz) and Ds are disjoint. Let r_{n+1} to be identical, we have $\cdot; Ds, Ds^n, Dz \vdash S\, V \simeq_{\mathsf{ctx}} S\, V' \simeq_{\mathsf{ctx}} S\,(S^n\, Z) \in \tau$ for $n+1$, namely $(S\, V, S\, V') \in R$. Therefore, we proved that $\cdot; Ds \vdash S \sim S' \in \alpha \multimap \alpha : \alpha \mapsto (R, \tau, \tau)$. By the free theorem, $N[\tau]\, S\, Z \longrightarrow^* V$ and $\cdot; Ds^n, Dz \vdash V \simeq_{\mathsf{ctx}} S^n\, Z \in \tau$ for some n. By regularity, we have $\mathsf{dom}(Ds, Dz) = \mathsf{dom}(Ds^n, Dz)$.

Because Ds can be non-empty, N can only behave like $\Lambda\alpha.\lambda S : \alpha \multimap \alpha.\lambda Z : \alpha.(S\, Z)$. If N is of type $\forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$, the free theorem proves that N encodes natural numbers (it is equivalent to $\Lambda\alpha.\lambda x : !(\alpha \multimap \alpha).\lambda Z : \alpha.\mathbf{let}\, !S =$

x **in** $S^n\ Z$) because Ds must be empty. We can also prove that the closed inhabitants of $\forall\alpha.(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$ must behave like $\Lambda\alpha.\lambda F_1 : \alpha \multimap \alpha.\lambda F_2 : \alpha \multimap \alpha.\lambda a : \alpha.(F_1\ (F_2\ a))$ or $\Lambda\alpha.\lambda F_1 : \alpha \multimap \alpha.\lambda F_2 : \alpha \multimap \alpha.\lambda a : \alpha.(F_2\ (F_1\ a))$.

5 Discussion

Other formulations of linearity All of these results presented here also carry through to System F$^\circ$ [16]—an extension of System F that uses kinds to distinguish linear from intuitionistic types. The online technical appendix [25] gives all formation rules. As expected, values with intuitionistic kind do not capture any linear variables. Open logical relations can easily adapt the definitions from PDILL. At the case of arrow types, when the well-formed open input values are with type of intuitionistic kind, they must be under empty linear contexts. We proved similar applications presented in Section 4.

Formalization There have been several formalizations of logical relations in proof assistants. Schürmann and Sarnat illustrated how to use logical relation arguments in Twelf by assertion logic [19]. Donnelly and Xi proved strong normalization for System F using high-order abstract syntax in ATS/LF [12].Vytiniotis and Weirich formalized System F in Isabelle/HOL using the locally nameless representation [22].

We formalized all results (including PDILL, System F$^\circ$ [16] and System F) in Coq. The development uses the Coq metatheory libraries [3] that combined locally nameless representation of de Bruijn indices for the bound variables and cofinite quantification of free variable names in inductive definitions of relations. The whole development for PDILL represents approximately 75,000 lines (350,000 words) of Coq (excluding comments). The formalism comprises of 6% of the calculus of PDILL that is extended from the proofs of type soundness for System F$_{<:}$ [3], 10% of closed logical relations, 17% of open logical relations, 43% of properties for contextual equivalence and applications, and 24% of supporting definitions and lemmas for logical relations and contextual equivalence. Definitions account for 5% of the source, and infrastructure overhead accounts for 24% of the source. The size of the formalism for System F$^\circ$ is similar.

The logical relations are recursively defined by *Program Fixpoint* on the size of types (typ_size) indicated by the keyword *measure*:

Program Fixpoint $G; D \vdash V \sim V' \in \tau{:}\theta$ {*measure* typ_size τ} : *Prop* :=
match τ *with*
...
$| \ \tau_1 \multimap \tau_2$ => $\exists L, \forall D_1 V_1 V_1',\ (L \sharp D_1 \wedge G; D_1 \vdash V_1{:}\theta_l(\tau_1) \wedge G; D_1 \vdash V_1'{:}\theta_r(\tau_1) \wedge G; D_1 \vdash V_1 \sim V_1' \in \tau_1{:}\theta \wedge G \vdash D, D_1) \Rightarrow G; D, D_1 \vdash V\ V_1 \simeq V'\ V_1' \in \tau_2{:}\theta$
end.

At the case of arrow types, to ensure that the domains of substitutions are disjoint from the free variables of types and values in the ranges of substitutions, we first choose an L that includes all the existing free variables in the domains of substitutions, and then pick a fresh linear context D_1 that is disjoint from the L.

Aydemir et al. [3] gave the case that renaming is necessary for proofs. The proof of Theorem 2 is by induction on the typing derivation. At T_APP, we have two induction hypotheses: $G; D_1 \vdash V \sim V' \in \tau_1 \multimap \tau_2{:}\theta$ and $G; D_2 \vdash V_1 \sim V_1' \in \tau_1{:}\theta$

where $G \vdash D_1 \Cup D_2 = D$. To conclude $G; D \vdash V\,V_1 \simeq V'\,V_1' \in \tau_2{:}\theta$, we need to apply the first hypothesis to the second one. However, we cannot show that D_2 is disjoint from the existential L in the first hypothesis. The proof goes like this: we first pick a fresh linear context D_2' that is disjoint from L and other existing free variables, constructing a bijective renaming r from D_2 to D_2'; by renaming lemmas we have $G; D_2' \vdash r(V_1) \sim r(V_1') \in \tau_1{:}\theta$; then induction hypotheses give that $G; D' \vdash V\,(r(V_1)) \simeq V'\,(r(V_1')) \in \tau_2{:}\theta$ where $G \vdash D_1 \Cup D_2' = D'$; finally renaming lemmas conclude $G; D \vdash V\,V_1 \simeq V'\,V_1' \in \tau_2{:}\theta$. In the intuitionistic setting, renaming lemmas may be derived from properties such as substitution and weakening, while in the linear setting that disallows weakening, renaming lemmas for each judgment that accounts for 5% of the source must be proved directly. In future work we would like to allow substitutions on types and values that can reuse variables in domains of substitutions. That would remove the L from the formalization of the logical relations, and simplify the proof, because we do not need renaming lemmas to apply induction hypotheses.

The proof of Lemma 4 requires that the relation such that $(U, U') \in R$ iff $\exists D,\ G; D \vdash U \sim U' \in \tau{:}\theta$ has the weakening property on θ. Suppose we extend θ to $(\theta, \alpha{\mapsto}(R, \tau_2, \tau_2'))$. Here α is free to existing variables. However, we do not know which D the relation takes before picking a free α that can be captured by D because R is defined for arbitrary linear contexts. α is a type variable, and the domain of D consists of only term variables. If type and term variables are in different name spaces, we can always pick an α that is free for D. The metatheory [3] does not support name spaces separation. We define axioms that assume separate name spaces between type and term variables. Adding this feature to metatheory libraries is our future work.

An important feature of contexts formalism is that holes are place-holders into which open terms may be placed, whose free variables may be captured. This capture feature is defined by *close* that turns free variable into de Bruijn indices, and *lift* that explicitly selects correct de Bruijn indices for bound variables. The infrastructure lemmas for *close* and *lift* account for 14% of the source.

Effects As usual for parametricity properties, our results would need to be modified in the presence of effects. We expect we can adopt the TT-closed relations [6] in PDILL to support fixpoints. That paper defines TT-closed relations on closed terms with closed testing functions, to avoid explicit treatment of holes in contexts. However, to test open terms, we need testing contexts with holes that capture variables from those open terms. We also hope to apply open logical relations to other substructural type systems [24].

References

1. Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *Lecture Notes in Computer Science*, volume 3924, pages 69–83. Springer, 2006.
2. Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. *SIGPLAN Not.*, 44(1):340–353, 2009.
3. Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2008.

4. Andrew Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Edinburgh University, 1997.
5. G. M. Bierman. Program equivalence in a linear functional language. *J. Funct. Program.*, 10(2):167–190, 2000.
6. G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of lily, a polymorphic linear lambda calculus with recursion. In *Proc. of HOOTS, volume 41 of ENTCS*. Elsevier, 2000.
7. Lars Birkedal, Rasmus E. Møgelberg, and Rasmus L. Petersen. Linear Abadi and Plotkin logic. *Logical Methods in Computer Science*, 2(5:1):1–33, Nov 2006.
8. Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. Relational parametricity for references and recursive types. In *TLDI '09: Proceedings of the 4th international workshop on Types in language design and implementation*, pages 91–104. ACM, 2009.
9. The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.2)*, 2009. URL http://coq.inria.fr.
10. Karl Crary. *Advanced Topics in Types and Programming Languages*, chapter Logical Relations and a Case Study in Equivalence Checking. MIT Press, 2005.
11. Roy L. Crole. Completeness of bisimilarity for contextual equivalence in linear theories. *Logic Jnl IGPL*, 9(1):27–51, 2001.
12. Kevin Donnelly and Hongwei Xi. A formalization of strong normalization for simply-typed lambda-calculus and System F. *Electron. Notes Theor. Comput. Sci.*, 174(5):109–125, 2007.
13. Jean-Yves Girard. Interprétation fonctionnelle et élimination des coupures de l'arith mé tique d'ordre supérieur. Thèse d'état, 1972. Summary in J. E. Fenstad, editor, *Scandinavian Logic Symposium*, pp. 63–92, North-Holland, 1971.
14. Masahito Hasegawa. Logical predicates for intuitionistic linear type theories. In *Typed Lambda Calculi and Applications (TLCA '99), Lecture Notes in Computer Science 1581*, pages 198–213. Springer-Verlag, 1999.
15. Ian Mason and Carolyn Talcott. Equivalence in functional languages with effects. *J. Funct. Program.*, 1(3):245–285, 1991.
16. Karl Mazurak, Jianzhou Zhao, and Steve Zdancewic. Lightweight linear types in system F°. In *TLDI '10: Proceedings of the 4th international workshop on Types in language design and implementation*, 2010.
17. Andrew Pitts. *Advanced Topics in Types and Programming Languages*, chapter Typed Operational Reasoning. MIT Press, 2005.
18. John C. Reynolds. Types, abstraction, and parametric polymorphism. In R.E.A Mason, editor, *Information Processing*, pages 513–523. Elsevier Science Publishers B.V., 1983.
19. Carsten Schürmann and Jeffrey Sarnat. Towards a judgmental reconstruction of logical relation proofs. Technical report, Yale University, 2006.
20. Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strniša. Ott: Effective Tool Support for the Working Semanticist. *J. Funct. Program.*, 20(1):71–122, 2010.
21. W.W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.
22. Dimitrios Vytiniotis and Stephanie Weirich. Free theorems and runtime type representations. *Electron. Notes Theor. Comput. Sci.*, 173:357–373, 2007.
23. Philip Wadler. Theorems for free! In *Proceedings of the 4th International Symposium on Functional Programming and Computer Architecture*, September 1989.
24. David Walker. *Advanced Topics in Types and Programming Languages*, chapter Substructural Type Systems. MIT Press, 2005.
25. Jianzhou Zhao, Qi Zhang, and Steve Zdancewic. Relational parametricity for a polymorphic linear lambda calculus. Technical report, 2010. Available at `http:/www.cis.upenn.edu/~jianzhou/parametricity4linf`.