Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types

Amal Ahmed

Harvard University
Cambridge, MA
amal@eecs.harvard.edu

Abstract. We present a proof technique based on syntactic logical relations for showing contextual equivalence of expressions in a λ-calculus with recursive types and impredicative universal and existential types. Our development builds on the step-indexed PER model of recursive types presented by Appel and McAllester. We have discovered that a direct proof of transitivity of that model does not go through, leaving the "PER" status of the model in question. We show how to augment the Appel-McAllester model to obtain a logical relation that we can prove is transitive, as well as sound and complete with respect to contextual equivalence. We then extend this model to support relational reasoning in the presence of quantified types.
Step-indexed relations are indexed not just by types, but also by the number of steps available for future evaluation. This stratification is essential for handling various circularities, from recursive functions, to recursive types, to impredicative polymorphism. The resulting construction is more elementary than existing logical relations which require complex machinery such as domain theory, syntactic minimal invariance, admissibility, and $\top\top$-closure.

1 Introduction

Proving equivalence of programs is important for verifying the correctness of compiler optimizations and other program transformations, as well as for establishing that program behavior is independent of the representation of an abstract type. This representation independence principle guarantees that if one implementation of an abstraction is exchanged for another, client modules will not be able to detect a difference.

Program equivalence is generally defined in terms of *contextual equivalence*. We say that two programs are contextually equivalent if they have the same observable behavior when placed in any program context C. Unfortunately, proving contextual equivalence is difficult in general, since it involves quantification over *all* possible contexts. As a result, there's been much work on finding tractable techniques for proving contextual equivalence. Many of these techniques are based on the method of *logical relations*.

Logical relations specify relations on well-typed terms via structural induction on the syntax of types. Thus, for instance, logically related functions take

logically related arguments to logically related results, while logically related pairs consist of components that are logically related pairwise. Logical relations may be based on denotational models (e.g. [1–3]) or on the operational semantics of a language [4–7]. The latter are also known as syntactic logical relations [8] and it is this flavor that is the focus of this paper.

To prove the soundness of a logical relation, one must prove the Fundamental Property (also called the Basic Lemma) which says that any well-typed term is related to itself. For simple type systems, it is relatively easy to prove the Fundamental Property for languages without nontermination, but for a language with recursive functions, establishing the Fundamental Property requires proving additional unwinding lemmas [9, 6, 7, 10].

For type systems with recursive and quantified types, the definition of the logical relation (by induction on types) must be more involved in order to deal with contravariant recursive types and with impredicative quantified types.[1] Establishing the existence of a relational interpretation of recursive types requires proving a nontrivial *minimal invariance* property [3, 10, 8, 11, 12]. Furthermore, to prove contextual equivalence using these logical relations, one usually has to prove the admissibility [10, 8], calculate the $\top\top$-closure [9, 6, 7], or show biorthogonality [12] of particular relations.

Appel and McAllester [13] proposed a radically different solution to the problem of recursive types. They defined *intensional* types, based on the operational semantics of the language, that are indexed by the number of available (future) execution steps. This extra information is sufficient to solve recursive equations on types. Step-indexed (unary) models have scaled well to features like mutable references and impredicative quantified types [14, 15]. Appel and McAllester also presented a PER (relational) model of recursive types, which we build on in this paper. The advantage of step-indexed logical relations is that they allow one to prove contextual equivalence without proving admissibility or calculating the $\top\top$-closure of relations.

Appel and McAllester proved the Fundamental Property for their PER model of equi-recursive types, and conjectured that their model was sound with respect to contextual equivalence. We show that their claim is correct — to be precise we show soundness for a calculus with iso-recursive types, but the essence of the model is the same.

We discovered, however, that the expected proof of transitivity for the Appel-McAllester model does not go through. To definitively show that their model is not transitive, we tried at length to find a counterexample for transitivity, but could not find one. Thus, we note that the question of transitivity of the Appel-McAllester model remains an open problem.

In Section 2 we consider a λ-calculus with iso-recursive types and present a sound and complete logical relation for the language. We also show how a direct proof of transitivity of the Appel-McAllester model fails, and discuss some of the peculiarities of the step-indexed approach. Section 3 extends the logical relation

[1] A quantified type such as $\forall\alpha.\,\tau$ is impredicative if α may be instantiated with *any* type, including $\forall\alpha.\,\tau$ itself.

for recursive types to support quantified types. Proofs of all lemmas in the paper and several examples to illustrate the uses of our logical relation are given in the accompanying technical report [16].

2 Recursive Types

We consider a call-by-value λ-calculus with iso-recursive types (dubbed the λ^{rec}-calculus). Figure 1 presents the syntax and small-step operational semantics for the language, which supports booleans and pairs in addition to recursive types. We define the operational semantics for λ^{rec} as a relation between closed terms e. We use evaluation contexts to lift the primitive rewriting rules to a standard, left-to-right, innermost-to-outermost, call-by-value interpretation of the language.

$$
\begin{array}{lll}
\textit{Types} & \tau & ::= \mathsf{bool} \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2 \mid \alpha \mid \mu\alpha.\,\tau \\
\textit{Expressions} & e & ::= x \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}\, e_0, e_1, e_2 \mid \langle e_1, e_2\rangle \mid \mathtt{let}\, \langle x_1, x_2\rangle = e_1\, \mathtt{in}\, e_2 \mid \\
 & & \quad \lambda x.\, e \mid e_1\, e_2 \mid \mathtt{fold}\, e \mid \mathtt{unfold}\, e \\
\textit{Values} & v & ::= \mathtt{tt} \mid \mathtt{ff} \mid \langle v_1, v_2\rangle \mid \lambda x.\, e \mid \mathtt{fold}\, v
\end{array}
$$

$$
\textit{Eval. Contexts}\ E ::= [\cdot] \mid \mathtt{if}\, E, e_1, e_2 \mid \mathtt{let}\, \langle x_1, x_2\rangle = E\, \mathtt{in}\, e \mid E\, e \mid v\, E \mid \mathtt{fold}\, E \mid \mathtt{unfold}\, E
$$

$$
\begin{array}{ll}
(\mathsf{iftrue}) & \mathtt{if}\, \mathtt{tt}, e_1, e_2 \longmapsto e_1 \\
(\mathsf{iffalse}) & \mathtt{if}\, \mathtt{ff}, e_1, e_2 \longmapsto e_2 \\
(\mathsf{letpair}) & \mathtt{let}\, \langle x_1, x_2\rangle = \langle v_1, v_2\rangle\, \mathtt{in}\, e \longmapsto e[v_1/x_1][v_2/x_2] \\
(\mathsf{app}) & (\lambda x.\, e)\, v \longmapsto e[v/x] \\
(\mathsf{unfold}) & \mathtt{unfold}\, (\mathtt{fold}\, v) \longmapsto v \\
(\mathsf{ctxt}) & \dfrac{e \longmapsto e'}{E[e] \longmapsto E[e']}
\end{array}
$$

Fig. 1. λ^{rec} Syntax and Operational Semantics

We say that a term e is irreducible ($\mathit{irred}(e)$) if e is a value ($\mathit{val}(e)$) or if e is a "stuck" expression to which no operational rule applies. We also use the following abbreviation:

$$
e \Downarrow \stackrel{\text{def}}{=} \exists e'.\, e \longmapsto^* e' \wedge \mathit{val}(e')
$$

λ^{rec} typing judgments have the form $\Gamma \vdash e : \tau$ where the context Γ is defined as follows:

$$
\textit{Value Context}\ \Gamma ::= \bullet \mid \Gamma, x{:}\tau
$$

Thus, Γ is used to track the set of variables in scope, along with their (closed) types. There may be at most one occurrence of a variable x in Γ. The λ^{rec} static semantics is entirely conventional (see, e.g., [17]) and is given in Figure 2. We use the abbreviated judgment $\vdash e : \tau$ when the variable context is empty.

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

$$\text{(True)}\ \frac{}{\Gamma \vdash \mathtt{tt} : \mathsf{bool}} \qquad \text{(False)}\ \frac{}{\Gamma \vdash \mathtt{ff} : \mathsf{bool}} \qquad \text{(If)}\ \frac{\Gamma \vdash e_0 : \mathsf{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathtt{if}\, e_0, e_1, e_2 : \tau}$$

$$\text{(Pair)}\ \frac{\Gamma \vdash v_1 : \tau_1 \quad \Gamma \vdash v_2 : \tau_2}{\Gamma \vdash \langle v_1, v_2 \rangle : \tau_1 \times \tau_2} \qquad \text{(LetPair)}\ \frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \quad \Gamma, x_1{:}\tau_1, x_2{:}\tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathtt{let}\, \langle x_1, x_2 \rangle = e_1\, \mathtt{in}\, e_2 : \tau}$$

$$\text{(Var)}\ \frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \text{(Fn)}\ \frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.\, e : \tau_1 \to \tau_2} \qquad \text{(App)}\ \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2}$$

$$\text{(Fold)}\ \frac{\Gamma \vdash e : \tau[\mu\alpha.\, \tau/\alpha]}{\Gamma \vdash \mathtt{fold}\, e : \mu\alpha.\, \tau} \qquad \text{(Unfold)}\ \frac{\Gamma \vdash e : \mu\alpha.\, \tau}{\Gamma \vdash \mathtt{unfold}\, e : \tau[\mu\alpha.\, \tau/\alpha]}$$

Fig. 2. λ^{rec} Static Semantics

Theorem 1 (λ^{rec} Safety). *If $\bullet \vdash e : \tau$ and $e \longmapsto^* e'$, then either e' is a value, or there exists an e'' such that $e' \longmapsto e''$.*

2.1 λ^{rec}: Contextual Equivalence

A context C is an expression with a single hole $[\cdot]$ in it. Typing judgments for contexts have the form $\Gamma_1 \vdash C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$, where $(\Gamma \triangleright \tau)$ indicates the type of the hole — that is, if $\Gamma \vdash e : \tau$, then $\Gamma_1 \vdash C[e] : \tau_1$.

Definition 1 (λ^{rec} Contextual Approximation ($\preceq^{ctx}$) & Equivalence ($\simeq^{ctx}$)). *If $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, we write $\Gamma \vdash e \preceq^{ctx} e' : \tau$ to mean*

$$\forall C, \tau_1.\ \bullet \vdash C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1 \ \wedge\ C[e] \Downarrow \implies C[e'] \Downarrow$$

Two terms are contextually equivalent if they contextually approximate one another:

$$\Gamma \vdash e \simeq^{ctx} e' : \tau \stackrel{\text{def}}{=} \Gamma \vdash e \preceq^{ctx} e' : \tau \ \wedge\ \Gamma \vdash e' \preceq^{ctx} e : \tau$$

2.2 λ^{rec}: Logical Relation

Our step-indexed logical relation for λ^{rec} is based on the PER model for equirecursive types presented by Appel and McAllester [13] (henceforth AM). The latter claimed, but did not prove, that their PER model was sound with respect to contextual equivalence. We have proved that this is indeed the case. However, "PER" may be somewhat of misnomer for the AM model since the status of transitivity is unclear, as we shall show.

In both models, the relational interpretation $\mathcal{RV}[\![\tau]\!]$ of a type τ is a set of triples of the form (k, v, v') where k is a natural number (called the *approximation index* or *step index*), and v and v' are (closed) values. Intuitively, $(k, v, v') \in \mathcal{RV}[\![\tau]\!]$ says that in any computation running for no more than k steps, v approximates v' at the type τ. Our model differs from the AM model

$$Rel_\tau \stackrel{\text{def}}{=} \{\chi \in 2^{Nat \times CValues \times CValues} \mid \quad \forall (j,v,v') \in \chi.\ \ \bullet \vdash v' : \tau\ \ \wedge\ \ \forall i \leq j.\ (i,v,v') \in \chi\}$$

$$\lfloor \chi \rfloor_k \stackrel{\text{def}}{=} \{(j,v,v') \mid\ j < k\ \wedge\ (j,v,v') \in \chi\}$$

$$\mathcal{RV}\,[\![\alpha]\!]\,\rho\ =\ \rho^{\mathsf{sem}}(\alpha)$$

$$\mathcal{RV}\,[\![\mathsf{bool}]\!]\,\rho\ =\ \{(k,v,v') \mid\ \vdash v' : \mathsf{bool}\ \wedge\ (v = v' = \mathtt{tt}\ \vee\ v = v' = \mathtt{ff})\}$$

$$\mathcal{RV}\,[\![\tau_1 \times \tau_2]\!]\,\rho\ =\ \{(k,\langle v_1,v_2\rangle,\langle v_1',v_2'\rangle) \mid\ \vdash \langle v_1',v_2'\rangle : (\tau_1 \times \tau_2)^{[\rho]}\ \wedge\ (k,v_1,v_1') \in \mathcal{RV}\,[\![\tau_1]\!]\,\rho\ \wedge\ (k,v_2,v_2') \in \mathcal{RV}\,[\![\tau_2]\!]\,\rho\}$$

$$\mathcal{RV}\,[\![\tau_1 \to \tau_2]\!]\,\rho\ =\ \{(k,\lambda x.\,e,\lambda x.\,e') \mid\ \vdash \lambda x.\,e' : (\tau_1 \to \tau_2)^{[\rho]}\ \wedge\ \forall j < k, v, v'.\ (j,v,v') \in \mathcal{RV}\,[\![\tau_1]\!]\,\rho \implies (j, e[v/x], e'[v'/x]) \in \mathcal{RC}\,[\![\tau_2]\!]\,\rho\}$$

$$\mathcal{RV}\,[\![\mu\alpha.\,\tau]\!]\,\rho\ =\ \{(k,\mathtt{fold}\,v,\mathtt{fold}\,v') \mid\ \vdash \mathtt{fold}\,v' : (\mu\alpha.\,\tau)^{[\rho]}\ \wedge\ \forall j < k.\ \text{let } \chi = \lfloor \mathcal{RV}\,[\![\mu\alpha.\,\tau]\!]\,\rho \rfloor_{j+1} \text{ in } (j,v,v') \in \mathcal{RV}\,[\![\tau]\!]\,\rho[\alpha \mapsto (\chi,\ (\mu\alpha.\,\tau)^{[\rho]})]\}$$

$$\mathcal{RC}\,[\![\tau]\!]\,\rho\ =\ \{(k,e,e') \mid\ \forall j < k, e_f.\ e \longmapsto^j e_f \wedge \mathit{irred}(e_f) \implies \exists e_f'.\ e' \longmapsto^* e_f'\ \wedge\ (k-j, e_f, e_f') \in \mathcal{RV}\,[\![\tau]\!]\,\rho\}$$

$$\mathcal{RG}\,[\![\bullet]\!]\ =\ \{(k,\emptyset,\emptyset)\}$$
$$\mathcal{RG}\,[\![\Gamma, x:\tau]\!]\ =\ \{(k,\gamma[x \mapsto v],\gamma'[x \mapsto v']) \mid\ (k,\gamma,\gamma') \in \mathcal{RG}\,[\![\Gamma]\!]\ \wedge\ (k,v,v') \in \mathcal{RV}\,[\![\tau]\!]\,\emptyset\}$$

$$\Gamma \vdash e \leq e' : \tau \stackrel{\text{def}}{=} \Gamma \vdash e : \tau\ \wedge\ \Gamma \vdash e' : \tau\ \wedge\ \forall k \geq 0.\ \forall \gamma, \gamma'.\ (k,\gamma,\gamma') \in \mathcal{RG}\,[\![\Gamma]\!] \implies (k, \gamma(e), \gamma'(e')) \in \mathcal{RC}\,[\![\tau]\!]\,\emptyset$$

$$\Gamma \vdash e \sim e' : \tau \stackrel{\text{def}}{=} \Gamma \vdash e \leq e' : \tau\ \wedge\ \Gamma \vdash e' \leq e : \tau$$

Fig. 3. λ^{rec} Relational Model (Shaded $\notin$ Appel-McAllester)

in that whenever $(k,v,v') \in \mathcal{RV}\,[\![\tau]\!]$, we additionally require that $\bullet \vdash v' : \tau$. This additional constraint enables us to prove the transitivity of our logical relation. Moreover, restricting the model to terms that are both safe and well-typed seems essential for completeness with respect to contextual equivalence, as others have also noted [12]. We defer an explanation of why we don't also require $\bullet \vdash v : \tau$ till Section 2.3.

Figure 3 gives the definition of our logical relation; shaded parts of the definitions have no analog in the AM model. We use the meta-variable χ to denote sets of tuples of the form (k, v, v'), where v and v' are closed values ($v, v' \in \mathit{CValues}$). For any set χ, we define the k-approximation of the set (written $\lfloor \chi \rfloor_k$) as the subset of its elements whose indices are less than k.

We define Rel_τ (where τ is a closed syntactic type) as the set of those sets $\chi \in 2^{\mathit{Nat} \times \mathit{CValues} \times \mathit{CValues}}$ that have the following two properties: if $(k, v, v') \in \chi$, then v' must be well-typed with type τ, and χ must be closed with respect to a decreasing step-index.

We use the meta-variable ρ to denote type substitutions. These are partial maps from type variables α to pairs (χ, τ) where χ is the semantic substitution for α and τ (a closed syntactic type) is the syntactic substitution for α. We note that our definitions ensure that if $\rho(\alpha) = (\chi, \tau)$ then $\chi \in \mathit{Rel}_\tau$. Since types in λ^{rec} may contain free type variables, the interpretation of a type τ is parametrized by a type substitution ρ such that $\mathit{FTV}(\tau) \subseteq \mathit{dom}(\rho)$. We use the following abbreviations:

- Let $\rho(\alpha) = (\chi, \tau)$. Then $\rho^{\mathsf{sem}}(\alpha) = \chi$ and $\rho^{\mathsf{syn}}(\alpha) = \tau$.
- Let $\rho = \{\alpha_1 \mapsto (\chi_1, \tau_1), \ldots, \alpha_n \mapsto (\chi_n, \tau_n)\}$.
  Then $\tau^{[\rho]}$ is an abbreviation for $\tau[\tau_1/\alpha_1, \tau_2/\alpha_2, \ldots, \tau_n/\alpha_n]$.

Next, we consider the relational interpretation $\mathcal{RV}[\![\tau]\!]\,\rho$ of each type τ. In each case, note that if $(k, v, v') \in \mathcal{RV}[\![\tau]\!]\,\rho$ then $\vdash v' : (\tau)^{[\rho]}$.

Booleans Two values are related at the type bool for any number of steps $k \geq 0$, if they are both $\mathtt{tt}$ or both $\mathtt{ff}$.

Pairs The pairs $\langle v_1, v_2 \rangle$ and $\langle v_1', v_2' \rangle$ are related at type $\tau_1 \times \tau_1$ for k steps if v_i and v_i' are related for k steps at the type τ_i (for $i \in \{1, 2\}$).

Functions Since functions are suspended computations, their interpretation is given in terms of the interpretation of types as computations (see below). Two functions are related if they map related arguments to related results. Specifically, $\lambda x.\, e$ and $\lambda x.\, e'$ are related at the type $\tau_1 \to \tau_2$ for k steps if, at some point in the future, when there are $j < k$ steps left to execute, and there are arguments v_a and v_a' that are related at the type τ_1 for j steps, then $e[v_a/x]$ and $e'[v_a'/x]$ are related as computations of type τ_2 for j steps.

Recursive Types One would expect the values $\mathtt{fold}\, v$ and $\mathtt{fold}\, v'$ to be related at the type $\mu\alpha.\, \tau$ for k steps if v and v' are related at the type $\tau[\mu\alpha.\, \tau/\alpha]$ for $j < k$ steps. We show that the latter is equivalent to what is required by the definition in Figure 3. Note that by the definition of $\lfloor \cdot \rfloor_k$

$$(j, v, v') \in \mathcal{RV}[\![\tau[\mu\alpha.\, \tau/\alpha]]\!]\,\rho \;\Leftrightarrow\; (j, v, v') \in \lfloor \mathcal{RV}[\![\tau[\mu\alpha.\, \tau/\alpha]]\!]\,\rho \rfloor_{j+1}$$

We prove a type substitution lemma (see [16]) that allows us to conclude that if $\chi = \lfloor \mathcal{RV}[\![\mu\alpha.\, \tau]\!]\,\rho \rfloor_{j+1}$ then:

$$\lfloor \mathcal{RV}[\![\tau[\mu\alpha.\, \tau/\alpha]]\!]\,\rho \rfloor_{j+1} \;=\; \lfloor \mathcal{RV}[\![\tau]\!]\,\rho[\alpha \mapsto (\chi, (\mu\alpha.\, \tau)^{[\rho]})] \rfloor_{j+1}$$

Hence,

$$\begin{array}{ll}
(j, v, v') \in \mathcal{RV}\,[\![\tau[\mu\alpha.\,\tau/\alpha]]\!]\,\rho & \\
\quad \Leftrightarrow (j, v, v') \in \lfloor \mathcal{RV}\,[\![\tau[\mu\alpha.\,\tau/\alpha]]\!]\,\rho \rfloor_{j+1} & \text{by } \lfloor\cdot\rfloor_k \\
\quad \Leftrightarrow (j, v, v') \in \lfloor \mathcal{RV}\,[\![\tau]\!]\,\rho[\alpha \mapsto (\chi, (\mu\alpha.\,\tau)^{[\rho]})] \rfloor_{j+1} & \text{by type subst} \\
\quad \Leftrightarrow (j, v, v') \in \mathcal{RV}\,[\![\tau]\!]\,\rho[\alpha \mapsto (\chi, (\mu\alpha.\,\tau)^{[\rho]})] & \text{by } \lfloor\cdot\rfloor_k
\end{array}$$

which is exactly what is required by the definition of $\mathcal{RV}\,[\![\mu\alpha.\,\tau]\!]\,\rho$.

Computations Two closed expressions e and e' are related as computations of type τ for k steps as follows. If e steps to an irreducible term e_f in $j < k$ steps, then e' must also step to some irreducible e'_f. Furthermore, both e_f and e'_f must be values that are related for the remaining $k - j$ steps.

What is surprising about this definition is that e must terminate in $j < k$ steps, while e' may terminate in *any* number of steps, say i. That is, i may be less than or greater than k. This leads to problems with proving transitivity in the AM model and we shall return to this point shortly.

Logical Relation If $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, then we write $\Gamma \vdash e \leq e' : \tau$ to mean that for all $k \geq 0$, if γ and γ' are mappings from variables x to closed values that are related for k steps at Γ, then $\gamma(e)$ and $\gamma'(e')$ are related for k steps as computations of type τ. We say e and e' are logically equivalent, written $\Gamma \vdash e \sim e' : \tau$, if they logically approximate one another.

We now have to prove that each type τ is a valid type — that is, that the relational interpretation of τ belongs to Rel_τ (i.e., $\mathcal{RV}\,[\![\tau]\!]\,\rho \in Rel_{\tau[\rho]}$). This involves showing well-typedness and closure under decreasing step-index.

Next, we prove a number of nontrivial lemmas (see the technical report [16]). Specifically, we prove that the logical relation defined in Figure 3 has the compatibility and substitutivity properties (see e.g., [9]). These allow us to show that the λ^{rec} typing rules preserve the logical relation, and hence prove the following lemma.

Lemma 1 (λ^{rec} Fundamental Property / Reflexivity).
If $\Gamma \vdash e : \tau$, then $\Gamma \vdash e \leq e : \tau$.

2.3 Transitivity and the Appel-McAllester Model

Let us ignore the shaded parts of Figure 3 and try to prove the following lemma with the resulting definitions.

Proposed Lemma (Transitivity: Appel-McAllester)
If $\Gamma \vdash e_1 \leq e_2 : \tau$ and $\Gamma \vdash e_2 \leq e_3 : \tau$, then $\Gamma \vdash e_1 \leq e_3 : \tau$.
Proof Attempt: Suppose $k \geq 0$ and $(k, \gamma, \gamma') \in \mathcal{RG}\,[\![\Gamma]\!]$.
Show $(k, \gamma(e_1), \gamma'(e_3)) \in \mathcal{RC}\,[\![\tau]\!]\,\emptyset$. Suppose $j < k$, $\gamma(e_1) \longmapsto^j e_{f_1}$, and $\mathit{irred}(e_{f_1})$.
Show $\exists e_{f_3}.\gamma'(e_3) \longmapsto^* e_{f_3} \wedge (k - j, e_{f_1}, e_{f_3}) \in \mathcal{RV}\,[\![\tau]\!]\,\emptyset$.
Instantiate $\Gamma \vdash e_1 \leq e_2 : \tau$ with $k \geq 0$ and $(k, \gamma, \gamma') \in \mathcal{RG}\,[\![\Gamma]\!]$.
Hence, $(k, \gamma(e_1), \gamma'(e_2)) \in \mathcal{RC}\,[\![\tau]\!]\,\emptyset$.

Instantiate this with $j < k$, $\gamma(e_1) \longmapsto^j e_{f_1}$, and $\mathit{irred}(e_{f_1})$.
Hence, there exist e_{f_2} and i such that $i \geq 0$, $\gamma'(e_2) \longmapsto^i e_{f_2}$, and $(k-j, e_{f_1}, e_{f_2}) \in \mathcal{RV}[\![\tau]\!]\emptyset$.
Now we need to use the premise $\Gamma \vdash e_2 \leq e_3 : \tau$. But what index should we instantiate this with? There are 2 ways to proceed.

Use index k: Instantiate $\Gamma \vdash e_2 \leq e_3 : \tau$ with $k \geq 0$ and $(k, \gamma', \gamma') \in \mathcal{RG}[\![\Gamma]\!]$.
Hence, $(k, \gamma'(e_2), \gamma'(e_3)) \in \mathcal{RC}[\![\tau]\!]\emptyset$.
We want to instantiate this with i and e_{f_2} since we have $\gamma'(e_2) \longmapsto^i e_{f_2}$.
Problem: We cannot show $i < k$ since i may be greater than k.

Use any $z > i$: We want to instantiate $\Gamma \vdash e_2 \leq e_3 : \tau$ with z, γ', γ'.
Problem: We cannot show $(z, \gamma', \gamma') \in \mathcal{RG}[\![\Gamma]\!]$. All we know is that $(k, \gamma, \gamma') \in \Gamma$, but $z > i$ and i may be greater than k.

We note that if we restrict our attention to closed terms e_1, e_2, e_3, then the above lemma can be proved. In the case of open terms, however, the status of transitivity of the AM model is unclear as we have been unable to find a counterexample.

There are several things one could attempt in order to rectify the above problem with the AM model (unshaded parts of Figure 3). One possibility may be to try to redefine $\Gamma \vdash e \leq e' : \tau$ as follows:

$$\Gamma \vdash e \leq e' : \tau \stackrel{\text{wrong}}{=} \forall \gamma, \gamma'.\ (\forall z \geq 0.\ (z, \gamma, \gamma') \in \mathcal{RG}[\![\Gamma]\!]) \implies \forall k \geq 0.\ (k, \gamma(e), \gamma'(e')) \in \mathcal{RC}[\![\tau]\!]\emptyset$$

But now the compatibility lemma for functions[2] (and hence, the Fundamental Property) cannot be proved.

Another possibility is to change the definition of $(k, e, e') \in \mathcal{RC}[\![\tau]\!]$ to require that e' must terminate in less than k steps. Unfortunately, if we step back and examine the resulting meaning of $\Gamma \vdash e_1 \sim e_2 : \tau$, we see that the latter now requires that both e_1 and e_2 must terminate in *exactly the same number of steps*. Clearly such a logical relation would not be very useful (unless we are concerned with reasoning about timing leaks in an information-flow setting).[3]

In fact, we believe that the definition of $\mathcal{RC}[\![\tau]\!]$, though it seems asymmetric, is exactly the right definition since we want a logical relation that considers programs equivalent modulo the number of steps they take.

The right way then to fix the problem with transitivity is to move to a typed setting where $(k, v, v') \in \mathcal{RV}[\![\tau]\!]\emptyset$ implies $\vdash v' : \tau$. Assuming the definitions in Figure 3, including the shaded parts, let us again try to prove transitivity.

Lemma 2 (λ^{rec} : Transitivity). *(Our model: Figure 3, including shaded parts) If $\Gamma \vdash e_1 \leq e_2 : \tau$ and $\Gamma \vdash e_2 \leq e_3 : \tau$, then $\Gamma \vdash e_1 \leq e_3 : \tau$.*

[2] Compatibility (Fun): If $\Gamma, x : \tau_1 \vdash e \leq e' : \tau_2$ then $\Gamma \vdash \lambda x.\, e \leq \lambda x.\, e' : \tau_1 \to \tau_2$.

[3] Other formulations involving the use of not one, but two step-indices (where the second bounds the number of steps in which e' must terminate), also lead to models where both terms are required to terminate in exactly the same number of steps.

Proof. We start at the point where we got stuck before. Now from $(k, \gamma, \gamma') \in \mathcal{RG}[\![\Gamma]\!]$ we can conclude that $\vdash \gamma' : \Gamma$. By reflexivity (Fundamental Property, Lemma 1) it follows that $\vdash \gamma' \leq \gamma' : \Gamma$. Hence, we can show that for all $z \geq 0$, $(z, \gamma', \gamma') \in \mathcal{RG}[\![\Gamma]\!]$ holds. Now we may instantiate $\Gamma \vdash e_2 \leq e_3 : \tau$ above with $i+1$ since we know that $(i+1, \gamma', \gamma') \in \mathcal{RG}[\![\Gamma]\!]$. The rest of the proof follows easily and is given in the accompanying technical report [16].

Seemingly Asymmetric Well-Typedness Requirement The definitions in Figure 3 may have left the reader with the impression that we only require terms on one side of our logical relation to be well-typed. This, however, is not the case. First, notice that in the definition of $\Gamma \vdash e \leq e' : \tau$, we require that *both* e and e' be well-typed. Second, notice that once we have picked a step index k (i.e., once we have moved under the $\forall k$ quantifier), there is an asymmetry in the model in that when $(k, e, e') \in \mathcal{RC}[\![\tau]\!]$, k pertains (as a bound) only to e and not to e'. As a result, for any $(k, v, v') \in \mathcal{RV}[\![\tau]\!]$, once we have considered all k (i.e., in the limit), we do know that v has type τ – just as in the unary step-indexed model [13]. But when working with a specific k (in the definition of $\mathcal{RV}[\![\tau]\!]$) we do not need to know that v has type τ in the limit, and so we chose not to require that $\bullet \vdash v : \tau$ at the value interpretation level $\mathcal{RV}[\![\tau]\!]$. One could add such a requirement in the interest of symmetry, but that would just lead to additional proof obligations being shuffled around, and would also complicate definitions when we get to quantified types as Rel_τ would have to be replaced by Rel_{τ_1,τ_2}.

2.4 λ^{rec}: Soundness

To prove that our logical relation is sound with respect to contextual equivalence, we first define what it means for two contexts to be logically related.

Definition 2 (λ^{rec} Logical Relation: Contexts).
$\Gamma_1 \vdash C \leq C' : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1 \stackrel{\text{def}}{=} \forall e, e'.\ \Gamma \vdash e \leq e' : \tau \implies \Gamma_1 \vdash C[e] \leq C'[e'] : \tau_1$

Next, we prove the compatibility lemmas for contexts, which allows us to prove the following.

Lemma 3 (λ^{rec} Reflexivity: Contexts).
If $\Gamma_1 \vdash C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$, *then* $\Gamma_1 \vdash C \leq C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$.

Lemma 4 (λ^{rec} Soundness: $\leq \subseteq \preceq^{ctx}$).
If $\Gamma \vdash e \leq e' : \tau$ *then* $\Gamma \vdash e \preceq^{ctx} e' : \tau$.

Proof. Suppose $\bullet \vdash C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$ and $C[e] \Downarrow$. Hence, there exist v_f, k such that $C[e] \longmapsto^k v_f$. We must show $C[e'] \Downarrow$.
Applying Lemma 3 to $\bullet \vdash C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$, we have $\bullet \vdash C \leq C : (\Gamma \triangleright \tau) \rightsquigarrow \tau_1$.
Instantiate this with $\Gamma \vdash e \leq e' : \tau$. Hence, $\bullet \vdash C[e] \leq C[e'] : \tau_1$.
Instantiate this with $k+1 \geq 0$ and $(k+1, \emptyset, \emptyset) \in \mathcal{RG}[\![\bullet]\!]$.
Hence, $(k+1, C[e], C[e']) \in \mathcal{RC}[\![\tau_1]\!]\emptyset$.
Instantiate this with $k < k+1$, $C[e] \longmapsto^k v_f$, and $\mathit{irred}(v_f)$.
Hence, exists v_f' such that $C[e'] \longmapsto^* v_f'$. Hence, $C[e'] \Downarrow$.

2.5 λ^{rec}: Completeness

To show that our logical relation is complete with respect to contextual equivalence, we make use of the notion of *ciu-equivalence* introduced by Mason and Talcott [18]. Two closed terms of the same closed type are said to be ciu-equivalent if they have the same termination behavior in any evaluation context E (a use of the term). The relation is extended to open terms via closing substitutions (i.e., closed instantiations). We note that evaluation contexts E are a simply a subset of general contexts C and that only closed terms can be placed in an evaluation context.

Definition 3 (λ^{rec} Ciu Approximation ($\preceq^{ciu}$) and Equivalence ($\simeq^{ciu}$)). *Let $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$.*

$$\Gamma \vdash e \preceq^{ciu} e' : \tau \stackrel{\text{def}}{=} \forall \gamma, E, \tau_1.\ \bullet \vdash \gamma : \Gamma \ \wedge\ \bullet \vdash E : (\bullet \triangleright \tau) \rightsquigarrow \tau_1 \ \wedge\ E[\gamma(e)] \Downarrow \implies E[\gamma(e')] \Downarrow$$

$$\Gamma \vdash e \simeq^{ciu} e' : \tau \stackrel{\text{def}}{=} \Gamma \vdash e \preceq^{ciu} e' : \tau \ \wedge\ \Gamma \vdash e' \preceq^{ciu} e : \tau$$

Lemma 5 (λ^{rec} : $\preceq^{ctx} \ \subseteq\ \preceq^{ciu}$). *If $\Gamma \vdash e \preceq^{ctx} e' : \tau$ then $\Gamma \vdash e \preceq^{ciu} e' : \tau$.*

Lemma 6 (λ^{rec} : $\preceq^{ciu} \ \subseteq\ \leq$). *If $\Gamma \vdash e \preceq^{ciu} e' : \tau$ then $\Gamma \vdash e \leq e' : \tau$.*

Proof. Suppose $k \geq 0$ and $(k, \gamma, \gamma') \in \mathcal{RG}[\![\Gamma]\!]$. We must show $(k, \gamma(e), \gamma'(e')) \in \mathcal{RC}[\![\tau]\!]\emptyset$. Suppose $j < k$, $\gamma(e) \longmapsto^j e_f$, and $\mathit{irred}(e_f)$.
Show $\exists e_f''.\ \gamma'(e') \longmapsto^* e_f'' \ \wedge\ (k-j, e_f, e_f'') \in \mathcal{RV}[\![\tau]\!]\emptyset$.
From $\Gamma \vdash e \preceq^{ciu} e' : \tau$, we have $\Gamma \vdash e : \tau$. Applying Lemma 1 to $\Gamma \vdash e : \tau$, we have $\Gamma \vdash e \leq e : \tau$. Instantiate this with $k \geq 0$ and $(k, \gamma, \gamma') \in \mathcal{RG}[\![\Gamma]\!]$. Hence, $(k, \gamma(e), \gamma'(e)) \in \mathcal{RC}[\![\tau]\!]\emptyset$. Instantiate this with $j < k$, $\gamma(e) \longmapsto^j e_f$, and $\mathit{irred}(e_f)$. Hence, there exists e_f' such that $\gamma'(e) \longmapsto^* e_f'$ and $(k-j, e_f, e_f') \in \mathcal{RV}[\![\tau]\!]\emptyset$. Hence, $e_f \equiv v_f$ and $e_f' \equiv v_f'$. Hence, $\gamma'(e) \Downarrow v_f'$.
Instantiate $\Gamma \vdash e \preceq^{ciu} e' : \tau$ with $\vdash \gamma' : \Gamma$ (follows from $(k, \gamma, \gamma') \in \mathcal{RG}[\![\Gamma]\!]$), and $\bullet \vdash [\cdot] : (\bullet \triangleright \tau) \rightsquigarrow \tau$, and $\gamma'(e) \Downarrow$.
Hence, there exists v_f'' such that $\gamma'(e') \longmapsto^* v_f''$.
Remains to show: $(k-j, v_f, v_f'') \in \mathcal{RV}[\![\tau]\!]\emptyset$.
This follows from Lemma 7 below applied to $(k-j, v_f, v_f') \in \mathcal{RV}[\![\tau]\!]\emptyset$ and $v_f' \preceq^{ciu} v_f'' : \tau$ (which follows from $\Gamma \vdash e \preceq^{ciu} e' : \tau$ and $\gamma'(e) \Downarrow v_f'$ and $\gamma'(e') \Downarrow v_f''$).

Lemma 7 (λ^{rec} Transitivity-Ciu: Closed Values).
If $(k, v_1, v_2) \in \mathcal{RV}[\![\tau]\!]\emptyset$ and $\bullet \vdash v_2 \preceq^{ciu} v_3 : \tau$, then $(k, v_1, v_3) \in \mathcal{RV}[\![\tau]\!]\emptyset$.

Proof. By induction on k and nested induction on the structure of the (closed) type τ.

3 Type Abstraction

We now extend λ^{rec} with impredicative univeral and existential types; we call the extended language the $\lambda^{\forall\exists}$-calculus. The syntactic extensions to support quantified types are as follows:

$$\begin{array}{lll} \textit{Types} & \tau ::= \ldots \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau \\ \textit{Values} & v ::= \ldots \mid \Lambda.\,e \mid \mathtt{pack}\,v \\ \textit{Expressions} & e ::= \ldots \mid e\,[\,] \mid \mathtt{unpack}\,e_1\,\mathtt{as}\,x\,\mathtt{in}\,e_2 \end{array}$$

Note that terms are not decorated with types (which was also the case for λ^{rec}). Here we let the vestigial operators remain in the untyped syntax in order to preserve the operational semantics. For instance, the term $\Lambda.\,e$ is a suspended computation (normally written $\Lambda\alpha.e$); $e\,[\,]$ runs the suspended computation.

We extend the λ^{rec} operational semantics as follows:

$$\textit{Evaluation Contexts } E ::= \ldots \mid E\,[\,] \mid \mathtt{unpack}\,E\,\mathtt{as}\,x\,\mathtt{in}\,e$$

$$\begin{array}{ll} (\mathsf{inst}) & (\Lambda.\,e)\,[\,] \longmapsto e \\ (\mathsf{unpack}) & \mathtt{unpack}\,(\mathtt{pack}\,v)\,\mathtt{as}\,x\,\mathtt{in}\,e \longmapsto e[v/x] \end{array}$$

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

$$(\mathsf{All})\ \frac{\Delta,\alpha;\Gamma \vdash e : \tau}{\Delta;\Gamma \vdash \Lambda.\,e : \forall\alpha.\,\tau} \qquad (\mathsf{Inst})\ \frac{\Delta;\Gamma \vdash e : \forall\alpha.\,\tau \qquad \Delta \vdash \tau_1}{\Delta;\Gamma \vdash e\,[\,] : \tau[\tau_1/\alpha]}$$

$$(\mathsf{Pack})\ \frac{\Delta \vdash \tau_1 \qquad \Delta;\Gamma \vdash e : \tau[\tau_1/\alpha]}{\Delta;\Gamma \vdash \mathtt{pack}\,e : \exists\alpha.\,\tau}$$

$$(\mathsf{Unpack})\ \frac{\Delta;\Gamma \vdash e_1 : \exists\alpha.\,\tau_1 \qquad \Delta \vdash \tau_2 \qquad \Delta,\alpha;\Gamma,x:\tau_1 \vdash e_2 : \tau_2}{\Delta;\Gamma \vdash \mathtt{unpack}\,e_1\,\mathtt{as}\,x\,\mathtt{in}\,e_2 : \tau_2}$$

Fig. 4. $\lambda^{\forall\exists}$ Static Semantics

$\lambda^{\forall\exists}$ typing judgments have the form $\Delta;\Gamma \vdash e : \tau$, where the context Γ is as before, and the context Δ is defined as follows:

$$\textit{Type Context } \Delta ::= \bullet \mid \Delta,\alpha$$

The type context Δ is used to track the set of type variables in scope. We modify the typing rules in Figure 2 by adding Δ to each typing judgment. Figure 4 gives the typing rules for the additional terms in $\lambda^{\forall\exists}$. We prove soundness of the $\lambda^{\forall\exists}$ typing rules, show that value and type substitution hold, and prove type safety.

Lemma 8 ($\lambda^{\forall\exists}$ Safety). *If $\bullet;\bullet \vdash e : \tau$ and $e \longmapsto^* e'$, then either e' is a value, or there exists an e'' such that $e' \longmapsto e''$.*

3.1 $\lambda^{\forall\exists}$: Contextual Equivalence

Typing judgments for contexts C now have the form $\Delta_1; \Gamma_1 \vdash C : (\Delta; \Gamma \triangleright \tau) \rightsquigarrow \tau_1$ (where $(\Delta; \Gamma \triangleright \tau)$ represents the type of the hole) indicating that whenever $\Delta; \Gamma \vdash e : \tau$, then $\Delta_1; \Gamma_1 \vdash C[e] : \tau_1$.

Definition 4 ($\lambda^{\forall\exists}$ Contextual Approximation ($\preceq^{ctx}$)).
If $\Delta; \Gamma \vdash e : \tau$ and $\Delta; \Gamma \vdash e' : \tau$, then we write $\Delta; \Gamma \vdash e \preceq^{ctx} e' : \tau$ to mean

$$\forall C, \tau_1.\ \bullet; \bullet \vdash C : (\Delta; \Gamma \triangleright \tau) \rightsquigarrow \tau_1 \ \wedge\ C[e] \Downarrow \implies C[e'] \Downarrow$$

3.2 $\lambda^{\forall\exists}$: Logical Relation

As in the case of λ^{rec}, the relational interpretation of a type $\mathcal{RV}\,[\![\tau]\!]\,\rho$ in $\lambda^{\forall\exists}$ is a set of triples of the form (k, v, v'). However, there is now one additional property (in addition to well-typedness of the second value of each tuple and closure under decreasing step-index) that every set χ in Rel_τ must satisfy. To motivate this property, we take the reader back to the proof of completeness of λ^{rec}, specifically to Lemma 7 which establishes a form of transitivity: if $(k, v_1, v_2) \in \mathcal{RV}\,[\![\tau]\!]\,\emptyset$ and $\bullet \vdash v_2 \preceq^{ciu} v_3 : \tau$, then $(k, v_1, v_3) \in \mathcal{RV}\,[\![\tau]\!]\,\emptyset$. The proof of that lemma required induction on k and nested induction on the structure of the closed type τ. In the case of $\lambda^{\forall\exists}$, when we get to the proof of the corresponding lemma, τ may have free type variables. Thus, one of the cases we must consider for the inner induction is $\tau = \alpha$. Assuming that $\rho(\alpha) = (\chi, \tau_\alpha)$, we will be required to show that if $(k, v_1, v_2) \in \mathcal{RV}\,[\![\alpha]\!]\,\rho \equiv \rho^{\mathsf{sem}}(\alpha) \equiv \chi$ and $\vdash v_2 \preceq^{ciu} v_3 : \alpha^{[\rho]}$ (where $\alpha^{[\rho]} \equiv \tau_\alpha$), then $(k, v_1, v_3) \in \chi$. Note that $\chi \in Rel_{\tau_\alpha}$. Thus, we must add this requirement directly to the definition of Rel_τ.

A more informal justification is that in the presence of quantified types, we can instantiate a type variable with a relational interpretation of our own choosing. Thus, we have to show that the relation we pick satisfies certain properties; transitivity is one of these properties.

The modified definition of Rel_τ is given below. It makes use of a notion of ciu-equivalence that applies just to closed values.

$$v \prec^{ciu} v' : \tau \stackrel{\text{def}}{=} \forall E, \tau_1.\ \bullet; \bullet \vdash E : (\bullet; \bullet \triangleright \tau) \rightsquigarrow \tau_1 \ \wedge\ E[v] \Downarrow \implies E[v'] \Downarrow$$

$$\begin{aligned} Rel_\tau \stackrel{\text{def}}{=} \{\chi \in 2^{Nat \times CValues \times CValues} \mid\ & \forall (j, v, v') \in \chi.\ \vdash v' : \tau \ \wedge \\ & \forall i \leq j.\ (i, v, v') \in \chi \ \wedge \\ & (\forall v''.\ v' \prec^{ciu} v'' : \tau \implies (j, v, v'') \in \chi)\} \end{aligned}$$

As we have noted before, when $(k, v, v') \in \mathcal{RV}\,[\![\tau]\!]\,\rho$, we require that $\vdash v' : \tau^{[\rho]}$. Meanwhile, notice that v may have any type. Hence, in the presence of quantified types, our relational model allows us to relate values of different types.

The relational interpretations of universal and existential types are given below.

$$\begin{aligned}\mathcal{RV}\left[\!\left[\forall\alpha.\,\tau\right]\!\right]\rho=\{(k,\Lambda.\,e,\Lambda.\,e') \mid\ & \vdash \Lambda.\,e' : (\forall\alpha.\,\tau)^{[\rho]} \ \wedge \\ & \forall\tau_2,\chi. \\ & \quad \chi \in \mathit{Rel}_{\tau_2} \implies \\ & \quad \forall j<k.\ (j,e,e') \in \mathcal{RC}\left[\!\left[\tau\right]\!\right]\rho[\alpha\mapsto(\chi,\tau_2)]\}\end{aligned}$$

$$\begin{aligned}\mathcal{RV}\left[\!\left[\exists\alpha.\,\tau\right]\!\right]\rho=\{(k,\mathtt{pack}\,v,\mathtt{pack}\,v') \mid\ & \vdash \mathtt{pack}\,v' : (\exists\alpha.\,\tau)^{[\rho]} \ \wedge \\ & \exists\tau_2,\chi. \\ & \quad \chi \in \mathit{Rel}_{\tau_2} \ \wedge \\ & \quad \forall j<k.\ (j,v,v') \in \mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho[\alpha\mapsto(\chi,\tau_2)]\}\end{aligned}$$

Two values $\mathtt{pack}\,v$ and $\mathtt{pack}\,v'$ are related at the type $\exists\alpha.\,\tau$ for k steps if there exists a syntactic type τ_2 and a semantic interpretation $\chi \in \mathit{Rel}_{\tau_2}$ such that for all $j < k$, $(j,v,v') \in \mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho[\alpha \mapsto (\chi,\tau_2)]$. In a setting with logical equivalence relations (instead of logical approximation relations) we pick two types, τ_1 and τ_2, when we wish to instantiate α. In the current setting we only pick a type for the second value v' and the type of v is left unrestricted. We restrict the type of v only when we take the intersection of the relations (i.e., $\Delta;\Gamma\vdash e\le e':\tau \wedge \Delta;\Gamma\vdash e'\le e:\tau$) in $\Delta;\Gamma\vdash e\sim e':\tau$. The relational interpretation of universal types is the dual of existential types.

The relational interpretation of types as computations is defined exactly as before. The definition of the logical relation $\Delta;\Gamma \vdash e \le e' : \tau$ is given in Figure 5.

$$\begin{aligned}\mathcal{RD}\left[\!\left[\bullet\right]\!\right] &= \{\emptyset\} \\ \mathcal{RD}\left[\!\left[\Delta,\alpha\right]\!\right] &= \{\rho[\alpha\mapsto(\chi,\tau_2)]) \mid\ \rho\in\mathcal{RD}\left[\!\left[\Delta\right]\!\right] \ \wedge\ \chi\in\mathit{Rel}_{\tau_2}\}\end{aligned}$$

$$\begin{aligned}\mathcal{RG}\left[\!\left[\bullet\right]\!\right]\rho &= \{(k,\emptyset,\emptyset)\} \\ \mathcal{RG}\left[\!\left[\Gamma,x:\tau\right]\!\right]\rho &= \{(k,\gamma[x\mapsto v],\gamma'[x\mapsto v']) \mid\ (k,\gamma,\gamma')\in\mathcal{RG}\left[\!\left[\Gamma\right]\!\right]\rho \ \wedge\ (k,v,v')\in\mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho\}\end{aligned}$$

$$\begin{aligned}\Delta;\Gamma\vdash e\le e':\tau \stackrel{\text{def}}{=}\ & \Delta;\Gamma\vdash e:\tau \ \wedge\ \Delta;\Gamma\vdash e':\tau \ \wedge \\ & (\forall k\ge 0.\ \forall\rho,\gamma,\gamma'. \\ & \quad \rho\in\mathcal{RD}\left[\!\left[\Delta\right]\!\right] \ \wedge\ (k,\gamma,\gamma')\in\mathcal{RG}\left[\!\left[\Gamma\right]\!\right]\rho \implies (k,\gamma(e),\gamma'(e'))\in\mathcal{RC}\left[\!\left[\tau\right]\!\right]\rho)\end{aligned}$$

Fig. 5. $\lambda^{\forall\exists}$ Relational Model

We prove that each type τ is a valid type: $\mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho \in \mathit{Rel}_{\tau^{[\rho]}}$. Specifically, we have to show well-typedness, closure under decreasing step-index, and the following lemma.

Lemma 9 ($\lambda^{\forall\exists}$ Rel Transitivity). *Let $\rho \in \mathcal{RD}\left[\!\left[\Delta\right]\!\right]$ and $\Delta \vdash \tau$. If $(k,v_1,v_2) \in \mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho$ and $v_2 \prec^{ciu} v_3 : \tau^{[\rho]}$, then $(k,v_1,v_3) \in \mathcal{RV}\left[\!\left[\tau\right]\!\right]\rho$.*

To show the Fundamental Property of the logical relation, we prove the new set of compatibility lemmas, as well as value and type substitutivity.

Lemma 10 ($\lambda^{\forall\exists}$ Fundamental Property / Reflexivity).
If $\Delta; \Gamma \vdash e : \tau$ then $\Delta; \Gamma \vdash e \leq e : \tau$.

3.3 $\lambda^{\forall\exists}$ Soundness and Completeness

We prove that the logical relation in Figure 5 is sound with respect to contextual equivalence. The overall proof structure is the same as for λ^{rec}.

Lemma 11 ($\lambda^{\forall\exists}$: $\leq \subseteq \preceq^{ctx}$). *If $\Delta; \Gamma \vdash e \leq e' : \tau$ then $\Delta; \Gamma \vdash e \preceq^{ctx} e' : \tau$.*

To establish completeness, we again rely on the notion of ciu-equivalence, which we define for $\lambda^{\forall\exists}$ as follows.

Definition 5 ($\lambda^{\forall\exists}$ Ciu Approximation ($\preceq^{ciu}$)).
Let $\Delta; \Gamma \vdash e : \tau$ and $\Delta; \Gamma \vdash e' : \tau$. If δ is a mapping from type variables α to closed syntactic types τ, we write $\delta \models \Delta$ whenever $dom(\delta) = \Delta$.

$$\begin{aligned}\Delta; \Gamma \vdash e \preceq^{ciu} e' : \tau \stackrel{\text{def}}{=} \forall \delta, \gamma, E, \tau_1.\ & \delta \models \Delta \ \wedge \ \vdash \gamma : \delta(\Gamma) \ \wedge \\ & \bullet; \bullet \vdash E : (\bullet; \bullet \triangleright \delta(\tau)) \rightsquigarrow \tau_1 \ \wedge \\ & E[\gamma(e)] \Downarrow \implies E[\gamma(e')] \Downarrow\end{aligned}$$

We can now prove completeness as before.

Lemma 12 ($\lambda^{\forall\exists}$: $\preceq^{ctx} \subseteq \preceq^{ciu} \subseteq \leq$).
If $\Delta; \Gamma \vdash e \preceq^{ctx} e' : \tau$ then $\Delta; \Gamma \vdash e \preceq^{ciu} e' : \tau$.
If $\Delta; \Gamma \vdash e \preceq^{ciu} e' : \tau$ then $\Delta; \Gamma \vdash e \leq e' : \tau$.

4 Related Work

Logical relations were first developed for denotational semantics of typed λ-calculi (e.g., [1, 2]). Early examples of the use of logical relations based on operational semantics include Tait's [4] proof of strong normalization of the typed λ-calculus, and Girard's method of reducibility candidates [5] used to prove normalization for System F.

Pitts [6] developed syntactic logical relations based on the operational semantics of the polymorphic λ-calculus, where, to support recursive functions without using denotional techniques, he introduced the notion of $\top\top$-closure. He proved the resulting relation complete with respect to contextual equivalence in a call-by-name polymorphic λ-calculus with recursive functions. Pitts [7] also developed syntactic logical relations for a call-by-value polymorphic λ-calculus with recursive functions and both universal and existential types, with a value restriction on abstract types. The resulting relations were complete for contextual equivalence, except in the case of existential types.

For recursive types, Birkedal and Harper [10] and Crary and Harper [8] showed how to adapt Pitts' minimal invariance [3] technique for use in a purely syntactic setting. Crary and Harper [8] also showed how to extend these logical relations with polymorphic types.

In recent work, Melliès and Vouillon [12, 11] showed how to construct a realizability model of a language with recursive types and polymorphism based on intuitions from the ideal model of types [19], and showed how to extend their ideas to a relational setting. Their relational model is based on an orthogonality relation between quadruples of terms and contexts [12]. We note that in order to show completeness, they too have to move to a typed setting.

There has been much work on logical relations for state, from Algol-like languages with stack-allocated local variables [20], to languages with dynamic allocated references (e.g., [21–23]).

Abramsky [24] introduced applicative bisimulations for showing contextual equivalence; Gordon and Rees [25, 26] extended these to universal types, recursive types, objects, and subtyping. Applicative bisimulations, however, are not very useful for proving contextual equivalence of existential packages.

Sumii and Pierce [27] presented a bisimulation for reasoning about recursive types and quantified types, that does not suffer from the same weakness as applicative bisimulations in the case of existential types. Using their examples as a point of comparison (see [16]), we have shown that our logical relations are somewhat easier to use when proving contextual equivalence. Koutavas and Wand [28] recently proposed bisumulations for proving contextual equivalence of terms in an untyped higher-order language with dynamically allocated mutable cells.

We note that our logical relations may be used to prove parametricity properties [29], which is not true of bisimulations.

5 Conclusion

We have presented step-indexed logical relations for λ-calculus with full universal, existential, and recursive types. Our logical relations provide an elementary proof technique for proving contextual equivalence.

The most important direction for future work is to develop logical relations for a language with dynamically allocated mutable references that may store values of any (closed) type, including functions, recursive types, and other references. We have made some progress in this direction, but currently we have a logical relation that is sound but not complete with respect to contextual equivalence.

References

1. Plotkin, G.D.: Lambda-definability and logical relations. Memorandum SAI–RM–4, University of Edinburgh, Edinburgh, Scotland (1973)
2. Statman, R.: Logical relations and the typed λ-calculus. Information and Control **65** (1985) 85–97
3. Pitts, A.M.: Relational properties of domains. Information and Computation **127** (1996) 66–90
4. Tait, W.W.: Intensional interpretations of functionals of finite type i. Journal of Symbolic Logic **32** (1967) 198–212

5. Girard, J.Y.: Interprétation Fonctionnelle et Élimination des Coupures de l'Arithmétique d'Ordre Supérieur. Thèse de doctorat d'état, Université Paris VII, Paris, France (1972)
6. Pitts, A.M.: Parametric polymorphism and operational equivalence. Mathematical Structures in Computer Science **10** (2000) 321–359
7. Pitts, A.M.: Existential types: Logical relations and operational equivalence. Lecture Notes in Computer Science **1443** (1998) 309–326
8. Crary, K., Harper, R.: Syntactic logical relations over polymorphic and recursive types. Draft (2000)
9. Pitts, A.M.: Typed operational reasoning. In Pierce, B.C., ed.: Advanced Topics in Types and Programming Languages. MIT Press (2005)
10. Birkedal, L., Harper, R.: Relational interpretations of recursive types in an operational setting. In: Theoretical Aspects of Computer Software (TACS). (1997)
11. Melliès, P.A., Vouillon, J.: Semantic types: A fresh look at the ideal model for types. In: ACM Symposium on Principles of Programming Languages (POPL), Venice, Italy. (2004)
12. Melliès, P.A., Vouillon, J.: Recursive polymorphic types and parametricity in an operational framework. In: IEEE Symposium on Logic in Computer Science (LICS), Chicago, Illinois. (2005)
13. Appel, A.W., McAllester, D.: An indexed model of recursive types for foundational proof-carrying code. ACM Transactions on Programming Languages and Systems **23** (2001) 657–683
14. Ahmed, A., Appel, A.W., Virga, R.: An indexed model of impredicative polymorphism and mutable references. Available at `http:// www.cs.princeton.edu/~appel/papers/impred.pdf` (2003)
15. Ahmed, A.J.: Semantics of Types for Mutable State. PhD thesis, Princeton University (2004)
16. Ahmed, A.: Step-indexed syntactic logical relations for recursive and quantified types. Available at `http:// www.eecs.harvard.edu/~amal/papers/esop06-tr.pdf` (2005)
17. Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
18. Mason, I.A., Talcott, C.L.: Equivalence in functional languages with effects. Journal of Functional Programming **1** (1991) 287–327
19. MacQueen, D., Plotkin, G., Sethi, R.: An ideal model for recursive polymophic types. Information and Computation **71** (1986) 95–130
20. Pitts, A.M.: Reasoning aobout local variables with operationally-based logical relations. In: IEEE Symposium on Logic in Computer Science (LICS), New Brunswick, New Jersey, IEEE Computer Society Press (1996)
21. Pitts, A.M., Stark, I.D.B.: Observable properties of higher order functions that dynamically create local names, or: What's *new*? In Borzyszkowski, A.M., Sokołowski, S., eds.: Mathematical Foundations of Computer Science. Volume 711 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (1993) 122–141
22. Stark, I.D.B.: Names and Higher-Order Functions. Ph. D. dissertation, University of Cambridge, Cambridge, England (1994)
23. Benton, N., Leperchey, B.: Relational reasoning in a nominal semantics for storage. In: TLCA. (2005) 86–101
24. Abramsky, S.: The lazy lambda calculus. In Turner, D.A., ed.: Research topics in functional programming. Addison-Wesley, Boston, MA, USA (1990) 65–117
25. Gordon, A.D.: Operational equivalences for untyped and polymorphic object calculi. In: Higher Order Operational Techniques in Semantics. (1995) 9–54

26. Gordon, A.D., Rees, G.D.: Bisimilarity for a first-order calculus of objects with subtyping. In: ACM Symposium on Principles of Programming Languages (POPL), St. Petersburg Beach, Florida, ACM Press (1996) 386–395
27. Sumii, E., Pierce, B.C.: A bisimulation for type abstraction and recursion. In: ACM Symposium on Principles of Programming Languages (POPL), Long Beach, California. (2005) 63–74
28. Koutavas, V., Wand, M.: Smaller bisimulations for reasoning about higher-order imperative programs. In: ACM Symposium on Principles of Programming Languages (POPL), Charleston, South Carolina. (2006)
29. Wadler, P.: Theorems for free! In: ACM Symposium on Functional Programming Languages and Computer Architecture (FPCA), London (1989)