Logical Step-Indexed Logical Relations

Derek Dreyer
MPI-SWS
dreyer@mpi-sws.org

Amal Ahmed
TTI-Chicago
amal@tti-c.org

Lars Birkedal
IT University of Copenhagen
birkedal@itu.dk

Abstract

We show how to reason about "step-indexed" logical relations in an abstract way, avoiding the tedious, error-prone, and proof-obscuring step-index arithmetic that seems superficially to be an essential element of the method. Specifically, we define a logic LSLR, which is inspired by Plotkin and Abadi's logic for parametricity, but also supports recursively defined relations by means of the modal "later" operator from Appel et al.*'s "very modal model" paper. We encode in LSLR a logical relation for reasoning (in-)equationally about programs in call-by-value System F extended with recursive types. Using this logical relation, we derive a useful set of rules with which we can prove contextual (in-)equivalences without mentioning step indices.*

1 Introduction

Appel and McAllester [6] invented the *step-indexed model* in order to express "semantic" proofs of type safety for use in foundational proof-carrying code. The basic idea is to characterize type inhabitation as a predicate indexed by the number of steps of computation left before "the clock" runs out. If a term e belongs to a type τ for any number of steps, then it is truly semantically an inhabitant of τ.

The step-indexed characterization of type inhabitation has the benefit that it can be defined inductively on the step index k. This is especially useful when modeling semantically troublesome features like recursive and mutable reference types, whose inhabitants would be otherwise difficult to define inductively on the type structure. Moreover, the step-indexed model's reliance on very simple mathematical constructions makes it particularly convenient for use in *foundational* type-theoretic proofs, in which all mathematical machinery must be mechanized.

In subsequent work, Ahmed and coworkers have shown that the step-indexed model can also be used for *relational* reasoning about programs in languages with semantically complex type structure [4, 3, 5, 19].

However, a continual annoyance in working with step-indexed logical relations, as well as a stumbling block to their general acceptance, is the tedious, error-prone, and proof-obscuring reasoning about step indices that seems superficially to be an essential element of the method. To give a firsthand example: the first two authors (together with Andreas Rossberg) recently developed a step-indexed technique for proving representation independence of "generative" ADTs, *i.e.,* ADTs that employ, in an interdependent fashion, both local state and existential type abstraction [5]. While the technique is useful on a variety of examples, we found that our proofs using it tend to be cluttered with step-index arithmetic, to the point that their main substance is obscured. Thus, it seems clear that widespread acceptance of step-indexed logical relations will hinge on the development of abstract proof principles for reasoning about them.

The key difficulty in developing such abstract proof principles is that, in order to reason about things being *infinitely* logically related, *i.e.,* belonging to a step-indexed logical relation at *all* step levels — which is, in the end, what one cares about — one must reason about their presence in the logical relation at any *particular* step index, and this forces one into finite, step-specific reasoning.

To see a concrete example of this, consider Ahmed's step-indexed logical relation for proving (in-)equivalence of programs written in an extension of System F with recursive types [4]. One might expect to have a step-free proof principle for establishing that two function values are infinitely logically related, along the lines of: $\lambda x_1.e_1$ and $\lambda x_2.e_2$ are infinitely logically related at the type $\sigma \to \tau$ iff, whenever v_1 and v_2 are infinitely related at σ, it is the case that $e_1[v_1/x_1]$ and $e_2[v_2/x_2]$ are infinitely related at τ. Instead, in Ahmed's model we have that $\lambda x_1.e_1$ and $\lambda x_2.e_2$ are infinitely related at $\sigma \to \tau$ iff for all $n \geq 0$, whenever v_1 and v_2 are related at σ for n steps, $e_1[v_1/x_1]$ and $e_2[v_2/x_2]$ are related at τ for n steps. That is, the latter is a *stronger* property—if $\lambda x_1.e_1$ and $\lambda x_2.e_2$ map n-related arguments to n-related results (for any n), then they also map infinitely-related arguments to infinitely-related results, but the converse is not necessarily true. Thus, in proving infinite properties of the step-indexed model, it seems necessary to reason about an arbitrary finite index n.

In this paper, we present a solution to this dilemma in the form of a logic we call LSLR. Our solution involves a novel synthesis of ideas from two well-known pieces of prior work: (1) Plotkin and Abadi's logic for relational reasoning about parametric polymorphism (hereafter, PAL) [23], and (2) Appel, Melliès, Richards, and Vouillon's "very modal model" paper (hereafter, VMM) [7].

PAL is a second-order intuitionistic logic extended with axioms for equational reasoning about relational parametricity in pure System F. Plotkin and Abadi show how to define a logical relation interpretation of System F types in terms of the basic constructs of their logic. Second-order relation variables are important in defining the relational interpretation of polymorphic types.

In this paper, we adapt the basic apparatus of PAL toward a new purpose: reasoning operationally about contextual (in-)equivalence in a call-by-value language F^μ with recursive and polymorphic types. We will show how to encode in our logic LSLR a logical relation that is sound with respect to contextual equivalence, based on a step-indexed relation previously published by Ahmed [4]. Compared with Ahmed's relation, ours is more abstract: proofs using it do not require any step-index arithmetic. Furthermore, whereas step-indexed logical relations are fundamentally asymmetric, our logic enables the derivation of equational reasoning principles as well as inequational ones.

In order to adapt PAL in this way, we need in particular the ability to (1) reason about call-by-value and (2) logically interpret recursive types of F^μ. To address (1), we employ atomic predicates (and first-order axioms) related to CBV reduction instead of PAL's equational predicates and axioms. This approach is similar to earlier logics of partial terms for call-by-value with simple [21] and recursive (but not universal) types [2].

For handling recursive types, it suffices to have some way of defining recursive relations $\mu r.R$ in the logic. This can be done when R is suitably "contractive" in r; to express contractiveness, we borrow the "later" $\triangleright A$ operator from Appel *et al.*'s VMM, which they in turn borrowed from Gödel-Löb logic [18]. Hence, LSLR is in fact not only a second-order logic (like PAL) but a modal one, and the truth value of a proposition is the set of worlds (think: step levels) at which it holds. The key reasoning principle concerning the later operator is the Löb rule, which states that $(\triangleright A \supset A) \supset A$. This can be viewed as a principle of induction on step levels, but we shall see that, when it is employed in connection with logical relations, it also has a coinductive flavor reminiscent of the reasoning principles used in bisimulation methods like Sumii and Pierce's [26].

Overview In Section 2, we present our language under consideration, F^μ. In Section 3, we present our logic LSLR described above. We also give a Kripke model of LSLR with worlds being natural numbers, and "future worlds" being smaller numbers, so that semantic truth values are downward-closed sets of natural numbers.

In Section 4, we define a logical relation interpretation of

$$
\begin{array}{lll}
\textit{Types} & \tau ::= & \alpha \mid \mathsf{unit} \mid \mathsf{int} \mid \mathsf{bool} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \\
& & \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau \mid \mu\alpha.\,\tau \\
\textit{Prim Ops} & o ::= & + \mid - \mid = \mid < \mid \leq \mid \ldots \\
\textit{Terms} & e ::= & x \mid () \mid \pm n \mid o(e_1, \ldots, e_n) \mid \\
& & \mathtt{true} \mid \mathtt{false} \mid \mathtt{if}\, e\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2 \mid \\
& & \langle e_1, e_2 \rangle \mid \mathtt{fst}\, e \mid \mathtt{snd}\, e \mid \mathtt{inl}_\tau\, e \mid \mathtt{inr}_\tau\, e \mid \\
& & \mathtt{case}\, e\, \mathtt{of}\, \mathtt{inl}\, x_1 \Rightarrow e_1 \mid \mathtt{inr}\, x_2 \Rightarrow e_2 \mid \\
& & \lambda x\!:\!\tau.\, e \mid e_1\, e_2 \mid \Lambda\alpha.\, e \mid e\, [\tau] \mid \\
& & \mathtt{pack}\, \tau, e\, \mathtt{as}\, \exists\alpha.\, \tau' \mid \mathtt{unpack}\, e_1\, \mathtt{as}\, \alpha, x\, \mathtt{in}\, e_2 \mid \\
& & \mathtt{fold}_\tau\, e \mid \mathtt{unfold}\, e \\
\textit{Values} & v ::= & x \mid () \mid \pm n \mid \mathtt{true} \mid \mathtt{false} \mid \langle v_1, v_2 \rangle \mid \\
& & \mathtt{inl}_\tau\, v \mid \mathtt{inr}_\tau\, v \mid \lambda x\!:\!\tau.\, e \mid \Lambda\alpha.\, e \mid \\
& & \mathtt{pack}\, \tau_1, v\, \mathtt{as}\, \exists\alpha.\, \tau \mid \mathtt{fold}_\tau\, v
\end{array}
$$

Figure 1. Syntax of F^μ

the types of F^μ *into the logic*. Then we present a set of rules for establishing properties about the logical relation, all of which are derivable within the logic. Using these rules, it is easy to show, almost entirely within the logic, that the logical relation is sound w.r.t. contextual approximation. We also show in this section how to define a symmetric version of the logical relation for equational reasoning.

In Section 5, we give examples of contextual equivalence proofs that employ *purely logical reasoning* using the derivable rules from Section 4 (in particular, without any kind of step-index arithmetic). Finally, in Section 6, we discuss related work and conclude.

2 The Language F^μ

We consider F^μ, a call-by-value λ-calculus with impredicative polymorphism and iso-recursive types. The syntax of F^μ is shown in Figure 1. We define a small-step operational semantics as a relation on terms (written $e \leadsto e'$). We use evaluation contexts E to lift the primitive reductions to a standard left-to-right call-by-value semantics for the language. The dynamic semantics is completely standard and is given in the companion technical appendix [13].

F^μ typing judgments have the form $\Gamma \vdash e : \tau$, where the context Γ binds type variables α, as well as value variables x: $\Gamma ::= \cdot \mid \Gamma, \alpha \mid \Gamma, x : \tau$. The typing rules are also standard and are given in full in the appendix [13].

Contextual Equivalence A context C is a term with a single hole $[\cdot]$ in it. The typing judgment for contexts has the form $\vdash C : (\Gamma \vdash \tau) \Rightarrow (\Gamma' \vdash \tau')$, where $(\Gamma \vdash \tau)$ indicates the type of the hole. This judgment essentially says that if e is a term such that $\Gamma \vdash e : \tau$, then $\Gamma' \vdash C[e] : \tau'$. Its formal definition appears in the appendix [13].

We define contextual approximation ($\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$) to mean that, for any well-typed program context C with a hole of the type of e_1 and e_2, the termination of $C[e_1]$ (written $C[e_1] \Downarrow$) implies the termination of $C[e_2]$. Contextual

Rel. Var's	r, s	$\in$	*RelVar*
F^μ *Ctxt's*	Γ	$::=$	$\cdot \mid \Gamma, \alpha \mid \Gamma, x : \tau \mid \Gamma, t : \tau$
F^μ *Subst's*	γ	$::=$	$\cdot \mid \gamma, \alpha \mapsto \tau \mid \gamma, x \mapsto v \mid \gamma, t \mapsto e$
Rel. Types	ϕ	$::=$	$\mathrm{VRel}(\tau_1, \tau_2) \mid \mathrm{TRel}(\tau_1, \tau_2)$
Rel. Ctxt's	Δ	$::=$	$\cdot \mid \Delta, r : \phi$
Rel. Subst's	φ	$::=$	$\cdot \mid \varphi, r \mapsto R$
Log. Ctxt's	Θ	$::=$	$\cdot \mid \Theta, A$
Atomic Prop's	P	$::=$	$e_1 = e_2 \mid \cdots$
Propositions	A, B	$::=$	$P \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid$ $A \supset B \mid \forall\Gamma.A \mid \exists\Gamma.A \mid$ $\forall\Delta.A \mid \exists\Delta.A \mid (e_1, e_2) \in R \mid \triangleright A$
Relations	R, S	$::=$	$r \mid (x_1 : \tau_1, x_2 : \tau_2).A \mid$ $(t_1 : \tau_1, t_2 : \tau_2).A \mid \mu r.R$

Figure 2. Syntax of LSLR

equivalence ($\Gamma \vdash e_1 \approx^{ctx} e_2 : \tau$) is then defined as approximation in both directions.

Definition 2.1. (Contextual Approximation & Equivalence) Let $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$.

$$\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau \stackrel{\text{def}}{=}$$
$$\forall C, \tau'.\ (\vdash C : (\Gamma \vdash \tau) \Rightarrow (\cdot \vdash \tau') \wedge C[e_1] \Downarrow) \supset C[e_2] \Downarrow$$
$$\Gamma \vdash e_1 \approx^{ctx} e_2 : \tau \stackrel{\text{def}}{=}$$
$$\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau \ \wedge\ \Gamma \vdash e_2 \preceq^{ctx} e_1 : \tau$$

3 The Logic LSLR

LSLR is a second-order intuitionistic modal logic supporting a primitive notion of value/term relations, as well as the ability to define such relations recursively.

Syntax The syntax of LSLR is given in Figure 2. We extend F^μ contexts Γ with the ability to bind *term* variables t in addition to value variables x (and correspondingly extend the syntax of terms with t and the F^μ typing judgment with the obvious hypothesis rule for t). In LSLR, term variables can be substituted by terms e and value variables by values v. F^μ substitutions γ map variables bound in F^μ contexts to objects of the appropriate syntactic class.

Relation contexts Δ bind relation variables r with relation types ϕ, which describe relations between pairs of values or terms of type τ_1 and τ_2 ($\mathrm{VRel}(\tau_1, \tau_2)$ or $\mathrm{TRel}(\tau_1, \tau_2)$, respectively). Relation substitutions φ map relation variables to *syntactic* relations R (*i.e.,* relations expressible in the logic), which we describe below. Finally, logical contexts Θ are sets of propositions.

Propositions A fall into four categories: *atomic* propositions P, standard *first-order* propositions ($\top$, $\bot$, $A \wedge B$, $A \vee B$, $A \supset B$, $\forall\Gamma.A$, $\exists\Gamma.A$), *relational* propositions ($\forall\Delta.A$, $\exists\Delta.A$, $(e_1, e_2) \in R$), and the modal *later* operator $\triangleright A$ borrowed from VMM.

Atomic propositions P and the axioms concerning them are essentially a parameter of the logic. We assume the existence of an atomic proposition, $e_1 = e_2$, which says that e_1 and e_2 (of the same type) are syntactically equal modulo renaming of bound variables. Otherwise, the only requirement we impose is that P's are first-order in the sense that they may only depend on variables bound in Γ, not Δ. For the purpose of this paper, we will be interested in a particular set of atomic propositions having to do with F^μ's reduction semantics (see Section 4), but LSLR is in no way tied to this particular set of P's.

The first-order connectives are self-explanatory. The relational propositions provide the ability to abstract over a relation, which is important in defining logical relations for polymorphic and existential types. $(x_1 : \tau_1, x_2 : \tau_2).A$ and $(t_1 : \tau_1, t_2 : \tau_2).A$ introduce value and term relations (the x's and t's are bound variables.) Finally, we have the later modality $\triangleright A$, which enables the definition of recursive relations $\mu r.R$, as explained in the Introduction. Intuitively, $\triangleright A$ means that A is true in all strictly future worlds of the current one. For $\mu r.R$ to be well-formed, R must be *contractive* in r, *i.e.,* the variable r must only appear inside R under the $\triangleright$ modality. This ensures that the meaning of R in the current world only depends recursively on its meaning in strictly future worlds. Assuming that the notion of "strictly future" is well-founded, we can then define the semantics of recursive relations by induction on future worlds.

Monotonicity:
$$A \supset \triangleright A$$
Löb Rule:
$$(\triangleright A \supset A) \supset A$$
Distributivity Laws:
$$\begin{array}{rcl} \triangleright(A \supset B) & \equiv & \triangleright A \supset \triangleright B \\ \triangleright\forall\Gamma.A & \equiv & \forall\Gamma.\triangleright A \\ \triangleright\forall\Delta.A & \equiv & \forall\Delta.\triangleright A \end{array} \qquad \begin{array}{rcl} \triangleright(A \wedge B) & \equiv & \triangleright A \wedge \triangleright B \\ \triangleright(A \vee B) & \equiv & \triangleright A \vee \triangleright B \\ \triangleright\exists\Gamma.A & \equiv & \exists\Gamma.\triangleright A \\ \triangleright\exists\Delta.A & \equiv & \exists\Delta.\triangleright A \end{array}$$
Replacement Axioms:
$$e_1 = e_2 \supset A[e_1/t] \equiv A[e_2/t] \qquad v_1 = v_2 \supset A[v_1/x] \equiv A[v_2/x]$$
Relation Axioms:
$$(v_1, v_2) \in (x_1 : \tau_1, x_2 : \tau_2).A \equiv A[v_1/x_1, v_2/x_2]$$
$$(e_1, e_2) \in (t_1 : \tau_1, t_2 : \tau_2).A \equiv A[e_1/t_1, e_2/t_2]$$
$$(e_1, e_2) \in \mu r.R \equiv (e_1, e_2) \in R[\mu r.R/r]$$

Figure 3. Key Axioms of LSLR

Inference Rules The main judgment of LSLR is $\Gamma; \Delta; \Theta \vdash A$, which says that for any closing instantiation of Γ and Δ, if the propositions in Θ are true in a world, then so is A. Most of the inference rules of LSLR are completely standard; they appear in full, along with the auxiliary judgments, in the appendix [13]. We summarize the interesting axioms in Figure 3. For brevity, we write these axioms *sans* contexts; they hold in any contexts where the propositions are well-formed. $\equiv$ denotes bidirectional implication.

The monotonicity rule states that propositions that are true now (in the current world) are also true later (in future worlds). The Löb rule, adapted from VMM, provides a clean induction principle over future worlds. If under the assumption that A is true in all strictly future worlds we can prove that it is true in the current world, then by induction A is true in the current world. The induction argument requires no base case because all propositions are assumed true in the final world.

The remainder of the rules concerning the later operator state that the later operator distributes over all propositional connectives. Not all these distributivity laws are valid in classical Gödel-Löb logic or VMM, but they hold here due to our axiom of monotonicity.

The replacement axioms say that we can substitute equal terms (resp. values) for equal terms (resp. values) inside a proposition without affecting its meaning.

The last set of rules concern inhabitation of relations. Of these the only interesting one is the one for recursive relations, which states that a recursive relation $\mu r.R$ is equivalent to its expansion $R[\mu r.R/r]$.

"Indexed" Model We define a Kripke model for LSLR, where the worlds are natural numbers and n is a strictly future world of m if $n < m$. Thus, the set of semantic truth values is the complete Heyting algebra $\mathcal{P}^{\downarrow}(\mathbb{N})$ of downwards-closed subsets of $\mathbb{N}$, ordered by inclusion.

We interpret propositions, logical contexts, and relations under some semantic interpretation δ, which maps their free relation variables to *semantic* relations (*i.e.,* functions from pairs of values/terms to semantic truth values). We write $[\![A]\!]^{\delta}_{n}$ (resp. $[\![\Theta]\!]^{\delta}_{n}$, $[\![R]\!]^{\delta}_{n}(e_1, e_2)$) to mean that, under interpretation δ, A (resp. Θ, $R(e_1, e_2)$) is true in world n. Thus, the truth value of A w.r.t. δ is the downwards-closed set of natural numbers n such that $[\![A]\!]^{\delta}_{n}$. We write $\delta \in [\![\Delta]\!]$ to mean that δ maps all the relation variables in the domain of Δ to appropriately-typed semantic value/term relations.

The interpretation is mostly standard, *e.g.,* involving a quantification over future worlds in the interpretation of $\supset$ in order to ensure monotonicity. Details are given in the appendix [13]; most interestingly, $[\![\triangleright A]\!]^{\delta}_{n} \overset{\text{def}}{=} [\![A]\!]^{\delta}_{n-1}$ and $[\![\mu r.R]\!]^{\delta}_{n} \overset{\text{def}}{=} [\![R[\mu r.R/r]]\!]^{\delta}_{n}$. The latter can be shown to be well-defined so long as R is contractive in r.

The interpretation is parameterized over an interpretation $\mathcal{I}$ of the atomic propositions P. We assume that $\mathcal{I}$ maps closed P's to *absolute* truth values, *i.e.,* in a way that is not dependent on the world n. Furthermore, we assume that $\mathcal{I}(e_1 = e_2)$ is $\top$ iff e_1 is α-equivalent to e_2.

We define semantic entailment $\Gamma; \Delta; \Theta \models A$ to mean $\forall n \geq 0.\ \forall \gamma : \Gamma.\ \forall \delta \in [\![\gamma\Delta]\!]\ .\ [\![\gamma\Theta]\!]^{\delta}_{n} \supset [\![\gamma A]\!]^{\delta}_{n}$.

Theorem 3.1. (Soundness of LSLR w.r.t. the Model) *If* $\Gamma; \Delta; \Theta \vdash A$, *then* $\Gamma; \Delta; \Theta \models A$.

When we extend the logic with new axioms pertaining to atomic propositions (as in the next section), we must prove that they are sound with respect to the model as well. So long as such axioms are strictly first-order (*i.e.,* they do not mention relations), proving soundness will not require any world-indexed reasoning, since first-order propositions are true in all worlds iff they are true in world 1.

The Kripke model we have defined here may be viewed as a "step-indexed" model, but it is important to note that nothing in either the model or the logic mentions steps of computation! We happen to be using natural numbers as our worlds, but there is no computational meaning attached to them, and other models of LSLR are possible.

4 A Syntactic Logical Relation for F^{μ}

In this section, we show how to define a logical relation for F^{μ} that is sound with respect to contextual approximation. The relation is defined *syntactically* within the logic LSLR, using a particular set of atomic propositions concerning the F^{μ} reduction semantics.

Atomic Propositions The new atomic propositions are:

$$P \ ::= \ \cdots \mid e_1 \leadsto^{*} e_2 \mid e_1 \leadsto^{0} e_2 \mid e_1 \leadsto^{1} e_2$$

For each of these atomic propositions, we consider it well-formed iff its two constituent terms have the same type. The interpretations of these propositions, $\mathcal{I}(P)$, are:

- $\mathcal{I}(e_1 \leadsto^{*} e_2)$ means that e_1 reduces to e_2 in an arbitrary number of steps (possibly zero).
- $\mathcal{I}(e_1 \leadsto^{0} e_2)$ means that e_1 reduces to e_2 in an arbitrary number of steps (possibly zero), *none* of which is an `unfold-fold` reduction.
- $\mathcal{I}(e_1 \leadsto^{1} e_2)$ means that e_1 reduces to e_2 in an arbitrary positive number of steps, *exactly one* of which is an `unfold-fold` reduction.

The motivation for using this particular set of atomic propositions will become clear shortly. Along with these P's, we will also extend LSLR with new axioms concerning them. There are many axioms we would like to incorporate, all of which are straightforward first-order syntactic properties of reduction. For example:

$$\begin{array}{l} \forall \alpha, t : \alpha, t_1 : \alpha, t_2 : \alpha. \\ \quad (t \leadsto^{1} t_1 \ \wedge\ t \leadsto^{1} t_2) \ \supset\ (t_1 \leadsto^{0} t_2 \ \vee\ t_2 \leadsto^{0} t_1) \end{array}$$

For simplicity, instead of enumerating all such axioms, we will just assert that any first-order property of well-typed terms that we can prove valid can be made a new axiom of the logic. Formally, assuming Θ and A are first-order (*i.e.,* do not mention relations):

$$\frac{\forall \gamma : \Gamma.\ [\![\gamma\Theta]\!]_1 \supset [\![\gamma A]\!]_1}{\Gamma; \Delta; \Theta, \Theta' \vdash A}$$

$$
\begin{array}{rcl}
\mathcal{V}\llbracket\alpha\rrbracket\,\rho & \stackrel{\text{def}}{=} & R,\ \text{ where } \rho(\alpha) = (\tau_1, \tau_2, R) \\
\mathcal{V}\llbracket\tau_b\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \tau_b, x_2 : \tau_b).\ x_1 = x_2,\ \text{ where } \tau_b \in \{\mathsf{unit}, \mathsf{int}, \mathsf{bool}\} \\
\mathcal{V}\llbracket\tau' \times \tau''\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \rho_1(\tau' \times \tau''), x_2 : \rho_2(\tau' \times \tau'')). \\
 & & \quad \exists x_1', x_1'', x_2', x_2''.\ x_1 = \langle x_1', x_1''\rangle \ \wedge\ x_2 = \langle x_2', x_2''\rangle \ \wedge\ (x_1', x_2') \in \mathcal{V}\llbracket\tau'\rrbracket\,\rho \ \wedge\ (x_1'', x_2'') \in \mathcal{V}\llbracket\tau''\rrbracket\,\rho \\
\mathcal{V}\llbracket\tau' + \tau''\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \rho_1(\tau' + \tau''), x_2 : \rho_2(\tau' + \tau'')). \\
 & & \quad (\exists x_1', x_2'.\ x_1 = \mathtt{inl}\, x_1' \ \wedge\ x_2 = \mathtt{inl}\, x_2' \ \wedge\ (x_1', x_2') \in \mathcal{V}\llbracket\tau'\rrbracket\,\rho) \ \vee \\
 & & \quad (\exists x_1'', x_2''.\ x_1 = \mathtt{inr}\, x_1'' \ \wedge\ x_2 = \mathtt{inr}\, x_2'' \ \wedge\ (x_1'', x_2'') \in \mathcal{V}\llbracket\tau''\rrbracket\,\rho)) \\
\mathcal{V}\llbracket\tau' \to \tau''\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \rho_1(\tau' \to \tau''), x_2 : \rho_2(\tau' \to \tau'')).\ \forall y_1, y_2.\ (y_1, y_2) \in \mathcal{V}\llbracket\tau'\rrbracket\,\rho \supset (x_1 y_1, x_2 y_2) \in \mathcal{E}\llbracket\tau''\rrbracket\,\rho \\
\mathcal{V}\llbracket\forall\alpha.\,\tau\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \rho_1(\forall\alpha.\,\tau), x_2 : \rho_2(\forall\alpha.\,\tau)).\ \forall\alpha_1, \alpha_2.\ \forall r : \mathrm{VRel}(\alpha_1, \alpha_2).\ (x_1\,[\alpha_1], x_2\,[\alpha_2]) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho, \alpha \mapsto (\alpha_1, \alpha_2, r) \\
\mathcal{V}\llbracket\exists\alpha.\,\tau\rrbracket\,\rho & \stackrel{\text{def}}{=} & (x_1 : \rho_1(\exists\alpha.\,\tau), x_2 : \rho_2(\exists\alpha.\,\tau)).\ \exists\alpha_1, \alpha_2, y_1, y_2.\ \exists r : \mathrm{VRel}(\alpha_1, \alpha_2). \\
 & & \quad x_1 = \mathtt{pack}\,\alpha_1, y_1\,\mathtt{as}\,\exists\alpha.\,\rho_1\tau \ \wedge\ x_2 = \mathtt{pack}\,\alpha_2, y_2\,\mathtt{as}\,\exists\alpha.\,\rho_2\tau \ \wedge\ (y_1, y_2) \in \mathcal{V}\llbracket\tau\rrbracket\,\rho, \alpha \mapsto (\alpha_1, \alpha_2, r) \\
\mathcal{V}\llbracket\mu\alpha.\,\tau\rrbracket\,\rho & \stackrel{\text{def}}{=} & \mu r.(x_1 : \rho_1(\mu\alpha.\,\tau), x_2 : \rho_2(\mu\alpha.\,\tau)). \\
 & & \quad \exists y_1, y_2.\ x_1 = \mathtt{fold}\, y_1 \ \wedge\ x_2 = \mathtt{fold}\, y_2 \ \wedge\ \triangleright(y_1, y_2) \in \mathcal{V}\llbracket\tau\rrbracket\,\rho, \alpha \mapsto (\rho_1(\mu\alpha.\,\tau), \rho_2(\mu\alpha.\,\tau), r) \\
\mathcal{E}\llbracket\tau\rrbracket\,\rho & \stackrel{\text{def}}{=} & \mu r.(t_1 : \rho_1\tau, t_2 : \rho_2\tau).(\forall x_1.\ t_1 \rightsquigarrow^0 x_1 \supset \exists x_2.\ t_2 \rightsquigarrow^* x_2 \wedge (x_1, x_2) \in \mathcal{V}\llbracket\tau\rrbracket\,\rho) \wedge (\forall t_1'.\ t_1 \rightsquigarrow^1 t_1' \supset \triangleright(t_1', t_2) \in r)
\end{array}
$$

Figure 4. Syntactic Logical Relation for F^{μ}

In other words, our goal here is not to use LSLR to formalize *entire* proofs, just the part of the proof that involves relational reasoning. We are happy to make use of first-order syntactic properties proved by other means.

Logical Relation Figure 4 defines two logical relations for F^{μ}, one for values ($\mathcal{V}\llbracket\tau\rrbracket\,\rho$) and one for terms ($\mathcal{E}\llbracket\tau\rrbracket\,\rho$). For brevity, the definition omits type annotations on variable bindings. These are syntactic LSLR relations R, defined by induction on τ. Here, ρ is a *syntactic* relational interpretation of the free type variables of τ, *i.e.,* a mapping from each $\alpha \in \mathrm{FV}(\tau)$ to a triple (τ_1, τ_2, R), where R has type $\mathrm{VRel}(\tau_1, \tau_2)$. We write ρ_i to mean the type substitution mapping each α to the corresponding τ_i. Thus, $\mathcal{V}\llbracket\tau\rrbracket\,\rho$ has type $\mathrm{VRel}(\rho_1\tau, \rho_2\tau)$, and $\mathcal{E}\llbracket\tau\rrbracket\,\rho$ has type $\mathrm{TRel}(\rho_1\tau, \rho_2\tau)$. Except for the last two cases ($\mathcal{V}\llbracket\mu\alpha.\tau\rrbracket\,\rho$ and $\mathcal{E}\llbracket\tau\rrbracket\,\rho$), the definition of the logical relation is entirely straightforward.

First, let us consider $\mathcal{V}\llbracket\mu\alpha.\tau\rrbracket\,\rho$. The basic idea here is to give the relational interpretation of a recursive type using a recursive relation $\mu r.R$. Recall, though, that references to r in R must only appear under "later" propositions. Thus, we have that $\mathtt{fold}\, v_1$ and $\mathtt{fold}\, v_2$ are related by $\mathcal{V}\llbracket\mu\alpha.\tau\rrbracket\,\rho$ "now" iff v_1 and v_2 are related by $\mathcal{V}\llbracket\tau[\mu\alpha.\tau/\alpha]\rrbracket\,\rho$ "later".

Next, consider $\mathcal{E}\llbracket\tau\rrbracket\,\rho$. Ideally, we would like to say that two terms e_1 and e_2 are related if, whenever e_1 evaluates to a value v_1, e_2 also evaluates to some value v_2, and $(v_1, v_2) \in \mathcal{V}\llbracket\tau\rrbracket\,\rho$. In fact, the definition of $\mathcal{E}\llbracket\tau\rrbracket\,\rho$ says precisely this, in the case that e_1 evaluates to v_1 without incurring any `unfold`-`fold` reductions (*i.e.,* when $e_1 \rightsquigarrow^0 v_1$).

However, the interpretation of recursive types forces us to require something weaker in the case that e_1 incurs an `unfold`-`fold` reduction. Specifically, in order to prove that the logical relation is sound with respect to contextual approximation, we must prove that it is *compatible* in the sense of Pitts [20]. Compatibility for `unfold` demands that if $\mathtt{fold}\, v_1$ and $\mathtt{fold}\, v_2$ are logically related, then $\mathtt{unfold}\,(\mathtt{fold}\, v_1)$ and $\mathtt{unfold}\,(\mathtt{fold}\, v_2)$ are related, too. By definition of $\mathcal{V}\llbracket\mu\alpha.\tau\rrbracket\,\rho$, knowing $\mathtt{fold}\, v_1$ and $\mathtt{fold}\, v_2$ are related only tells us that v_1 and v_2 are related "later". We need to be able to derive from that that $\mathtt{unfold}\,(\mathtt{fold}\, v_1)$ and $\mathtt{unfold}\,(\mathtt{fold}\, v_2)$ are related "now". Thus, in defining whether $(e_1, e_2) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho$, in the case that e_1 makes an `unfold`-`fold` reduction (*i.e.,* $e_1 \rightsquigarrow^1 e_1'$), we only require that e_1' and e_2 be related *later* (*i.e.,* $\triangleright(e_1', e_2) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho$).

For the reader who is familiar with prior work on step-indexed models and logical relations, our formulation here may seem familiar and yet somewhat unusual. Our use of the later operator corresponds to where one would "go down a step" in the construction of a step-indexed model. However, in prior work, step-indexed models typically go down a step *everywhere* (*i.e.,* in every case of the logical relation), not just in one or two places, and "count" every step, not just `unfold`-`fold` reductions. If one is working with *equi-recursive* types, this may be the only option, but here we are working with *iso-recursive* types, and our present formulation serves to isolate the use of the later operator to the few places where it is absolutely needed. While we do not believe there is a fundamental difference between what one can prove using this logical relation vs. previous accounts, our formulation enables more felicitous statements of certain properties, such as the extensionality principle for functions (see discussion of Rule 9 below).

Finally, it is worth noting that, like step-indexed models, LSLR imposes no "admissibility" requirement on candidate relations. Intuitively, the reason admissibility is unnecessary is that it is an infinitary property. In LSLR, we only ever reason about finitary properties, *i.e.,* propositions that hold true in the "current" world; we do not even have the ability (within the logic) to talk about truth in all worlds.

$$\frac{\Gamma;\Delta;\Theta \vdash (v_1,v_2) \in \mathcal{V}\llbracket\tau\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (v_1,v_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}\ (1) \qquad \frac{\Gamma;\Delta;\Theta_1,\Theta_2 \vdash A}{\Gamma;\Delta;\Theta_1,\triangleright\Theta_2 \vdash \triangleright A}\ (2) \qquad \frac{\Gamma;\Delta;\Theta \vdash \forall x.\, e_2' \leadsto^* x \supset e_2 \leadsto^* x \quad \Gamma;\Delta;\Theta \vdash (e_1,e_2') \in \mathcal{E}\llbracket\tau\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}\ (3)$$

$$\frac{\Gamma;\Delta;\Theta \vdash e_1 \leadsto^* e_1' \vee e_1' \leadsto^0 e_1 \quad \Gamma;\Delta;\Theta \vdash (e_1',e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}\ (4) \qquad \frac{\Gamma;\Delta;\Theta \vdash e_1 \leadsto^1 e_1' \quad \Gamma;\Delta;\Theta \vdash \triangleright(e_1',e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho}\ (5)$$

$$\frac{\Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\tau\rrbracket\rho \qquad \Gamma,x_1,x_2;\Delta;\Theta,(x_1,x_2) \in \mathcal{V}\llbracket\tau\rrbracket\rho, e_1 \leadsto^* x_1, e_2 \leadsto^* x_2 \vdash (E[x_1],f) \in \mathcal{E}\llbracket\tau'\rrbracket\rho'}{\Gamma;\Delta;\Theta \vdash (E[e_1],f) \in \mathcal{E}\llbracket\tau'\rrbracket\rho'}\ (6)$$

$$\frac{\Gamma;\Delta;\Theta \vdash (f_1,f_2) \in \mathcal{E}\llbracket\tau' \to \tau''\rrbracket\rho \quad \Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\tau'\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (f_1\,e_1, f_2\,e_2) \in \mathcal{E}\llbracket\tau''\rrbracket\rho}\ (7) \qquad \frac{\Gamma;\Delta;\Theta \vdash (e_1,e_2) \in \mathcal{E}\llbracket\mu\alpha.\,\tau\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (\mathtt{unfold}\,e_1, \mathtt{unfold}\,e_2) \in \mathcal{E}\llbracket\tau[\mu\alpha.\,\tau/\alpha]\rrbracket\rho}\ (8)$$

$$\frac{\Gamma,x_1,x_2;\Delta;\Theta,(x_1,x_2) \in \mathcal{V}\llbracket\tau'\rrbracket\rho \vdash (v_1x_1,v_2x_2) \in \mathcal{E}\llbracket\tau''\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (v_1,v_2) \in \mathcal{V}\llbracket\tau' \to \tau''\rrbracket\rho}\ (9) \qquad \frac{\Gamma;\Delta;\Theta \vdash \triangleright(e_1,e_2) \in \mathcal{E}\llbracket\tau[\mu\alpha.\,\tau/\alpha]\rrbracket\rho}{\Gamma;\Delta;\Theta \vdash (\mathtt{fold}\,e_1, \mathtt{fold}\,e_2) \in \mathcal{E}\llbracket\mu\alpha.\,\tau\rrbracket\rho}\ (10)$$

$$\frac{F_i = \mathtt{fix}\,f(x_i).\,e_i \quad \Gamma,x_1,x_2;\Delta;\Theta_1,\Theta_2,(F_1,F_2) \in \mathcal{V}\llbracket\tau' \to \tau''\rrbracket\rho,(x_1,x_2) \in \mathcal{V}\llbracket\tau'\rrbracket\rho \vdash (e_1[F_1/f],e_2[F_2/f]) \in \mathcal{E}\llbracket\tau''\rrbracket\rho}{\Gamma;\Delta;\Theta_1,\triangleright\Theta_2 \vdash (F_1,F_2) \in \mathcal{V}\llbracket\tau' \to \tau''\rrbracket\rho}\ (11)$$

Figure 5. Some Useful Derivable Rules

Derivable Rules Figure 5 shows a number of useful inference rules that are derivable in the logic. To be clear, by saying a rule is "derivable" we mean that if one adds the premises of the rule as axioms of the logic, then one can construct a derivation of the conclusion. (The proof of derivability may rely, however, on techniques of the meta-logic, such as induction on the structure of types.) In all these rules, we assume implicitly that all propositions are well-formed. For the rules concerning $\mathcal{V}\llbracket\tau\rrbracket\rho$ and $\mathcal{E}\llbracket\tau\rrbracket\rho$, we assume that ρ binds the free variables of τ and maps them to triples (τ_1,τ_2,R) where $R : \mathrm{VRel}(\tau_1,\tau_2)$.

Rule 1 says that $\mathcal{E}\llbracket\tau\rrbracket\rho$ coincides with $\mathcal{V}\llbracket\tau\rrbracket\rho$ when restricted to values.

Rule 2 is a weakening property that is easy to derive from the distributivity laws for the $\triangleright$ operator. Assuming $\Theta = A_1,\ldots,A_n$, the notation $\triangleright\Theta$ used here denotes $\triangleright A_1,\ldots,\triangleright A_n$. The rule says that if we want to show A is true later, given some assumptions (Θ_1) that are true now, and others (Θ_2) that are true later, then we can just prove that A is true now given that all the assumptions are true now. This rule is particularly useful in conjunction with the Löb rule. Specifically, thanks to the Löb rule, a frequently effective approach to proving two terms e_1 and e_2 related is to assume inductively that they are related *later* and then prove that they are related now. Eventually, we may reduce our proof goal (via, *e.g.,* Rule 5 or 10) to showing that two other terms e_1' and e_2' are related *later.* At that point, Rule 2 allows us to un-$\triangleright$ both our new proof goal (relatedness of e_1' and e_2') and our original inductive assumption (relatedness of e_1 and e_2) simultaneously. We will see an instance of this proof pattern in Example 2 in Section 5.

Rules 3–5 allow one to prove that two terms e_1 and e_2 are related by converting one of the terms to something else. Rule 3 allows one to replace e_2 with some e_2' that approximates it, and then show that e_1 is related to e_2'. (For instance, this rule applies if e_2 and e_2' are interconvertible by some sequence of reductions and expansions.) Rule 4 allows one to reduce or expand e_1 to some e_1' according to the $\leadsto^0$ relation and then show that e_1' is related to e_2. Rule 5 is similar, but addresses the case when e_1 incurs an `unfold`-`fold` reduction on the way to e_1'. In this case, unrolling the definition of $\mathcal{E}\llbracket\tau\rrbracket\rho$, all we have to show is that e_1' and e_2 are related *later.*

The aforementioned rules are all useful when we know what the terms in question reduce/expand to. Rule 6 is important because it handles the case when a term is "stuck". For instance, suppose we want to show that e and f are related, where e is of the form $E[e_1]$ (*i.e.,* e_1 is in evaluation position in e, and E is the evaluation context surrounding it). Perhaps e_1 is something like $y_1(v_1)$, in which case there is no way to reduce it. However, if we can prove that $y_1(v_1)$ is logically related to some other expression e_2, then there are two cases to consider. In the case that they both terminate, we can bind their unknown values as x_1 and x_2, assume x_1 and x_2 are related by $\mathcal{V}\llbracket\tau\rrbracket\rho$, and reduce the goal to showing that $E[x_1]$ is related to f. In the case that e_1 diverges, there is nothing to show, since $E[e_1]$ will diverge, too. Proving this rule derivable, while not difficult, is slightly less obvious than for the other rules, and thus a good exercise for the reader: the proof is given in the appendix [13].

Rule 6 is also useful in deriving *compatibility* properties [20], which are necessary in order for the logical relation to be a precongruence (and hence contained in contextual approximation). Rules 7 and 8 are examples of such properties (for function application and `unfold`, respectively), and Rule 6 reduces their derivations to the case where the e's and f's are values. Rule 8 is also invertible, thus providing a kind of *extensionality* property for recursive types, which we exploit in the proof of the second example in Section 5.

Rule 9 demonstrates a similar extensionality property for function *values*. (The property does not hold for arbitrary terms in our call-by-value semantics.) It is worth noting that, in prior step-indexed models, this extensionality property is not quite so clean to state. For example, if one were to encode Ahmed's relation [4] in our logic directly, the assumption $(x_1, x_2) \in \mathcal{V}\llbracket\tau'\rrbracket\,\rho$ would have to be $\triangleright$'d. We find our present formulation more convenient to work with.

Rule 10 shows a *strong* compatibility rule for `fold`. That is, the relatedness of $\texttt{fold}\, e_1$ and $\texttt{fold}\, e_2$ only requires that e_1 and e_2 be related *later*. Combining this with both the Löb rule and Rule 2, we may obtain the following (co-)inductive rule for `fold` that does not mention the $\triangleright$ modality at all: to show that $\texttt{fold}\, e_1$ and $\texttt{fold}\, e_2$ are related, we may *assume* they are related while showing that e_1 and e_2 are related.

Along the same lines, Rule 11 gives the rule for recursive functions, which are encodable in a well-known way in terms of recursive types. We formalize the encoding in the appendix [13]; it has the property that if $F = \texttt{fix}\, f(x).\, e$, then $F(v) \rightsquigarrow^1 e[F/f, v/x]$. Consequently, to show two recursive functions related, we may (co-)inductively assume they are related while proving that their bodies are related.

Soundness of the Logical Relation We now state some key theorems concerning the logical relation, the primary one being that it is sound w.r.t. contextual approximation.

Lemma 4.1. (Type Substitution)

1. $\mathcal{V}\llbracket\tau[\sigma/\alpha]\rrbracket\rho = \mathcal{V}\llbracket\tau\rrbracket\rho, \alpha \mapsto (\rho_1\sigma, \rho_2\sigma, \mathcal{V}\llbracket\sigma\rrbracket\rho)$.

2. $\mathcal{E}\llbracket\tau[\sigma/\alpha]\rrbracket\rho = \mathcal{E}\llbracket\tau\rrbracket\rho, \alpha \mapsto (\rho_1\sigma, \rho_2\sigma, \mathcal{V}\llbracket\sigma\rrbracket\rho)$.

Definition 4.2. (Logical Approximation Judgment)
Suppose $\Gamma = \alpha_1, \ldots, \alpha_n, x_1 : \tau_1, \ldots, x_m : \tau_m$. Let

$$\Gamma' = \alpha_1^1, \alpha_1^2, \ldots, \alpha_n^1, \alpha_n^2, x_1^1 : \tau_1^1, x_1^2 : \tau_1^2, \ldots, x_m^1 : \tau_m^1, x_m^2 : \tau_m^2$$
$$\Delta = r_1 : \mathrm{VRel}(\alpha_1^1, \alpha_1^2), \ldots, r_n : \mathrm{VRel}(\alpha_n^1, \alpha_n^2)$$
$$\Theta = (x_1^1, x_1^2) \in \mathcal{V}\llbracket\tau_1\rrbracket\,\rho, \ldots, (x_m^1, x_m^2) \in \mathcal{V}\llbracket\tau_m\rrbracket\,\rho$$
where $\rho = \{\alpha_1 \mapsto (\alpha_1^1, \alpha_1^2, r_1), \ldots, \alpha_n \mapsto (\alpha_n^1, \alpha_n^2, r_n)\}$, $\tau_i^1 = \rho_1\tau_i$ and $\tau_i^2 = \rho_2\tau_i$.

Also, let $\gamma_j = \{x_1 \mapsto x_1^j, \ldots, x_m \mapsto x_m^j\}$ for $j \in \{1, 2\}$. Then

$$\Gamma \vdash e_1 \preceq^{log} e_2 : \tau \overset{\text{def}}{=} \Gamma'; \Delta; \Theta \vdash (\rho_1(\gamma_1(e_1)), \rho_2(\gamma_2(e_2))) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho$$

Theorem 4.3. (Fundamental Property)
If $\Gamma \vdash e : \tau$ *then* $\Gamma \vdash e \preceq^{log} e : \tau$.

The proof of Theorem 4.3 is by induction on typing derivations, in each case using the appropriate compatibility rules (as discussed above), which are all derivable in the logic.

Theorem 4.4. (Soundness w.r.t. Contextual Approx.)
If $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$ *then* $\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$.

The proof of Theorem 4.4 is by induction on contexts C. As for the previous theorem, most of the cases follow from the compatibility rules. There is only one piece of the proof that requires interpreting $\mathcal{E}\llbracket\tau\rrbracket\,\rho$ into the model — namely, the proof that the logical relation is *adequate* (in Pitts' terminology [20]), *i.e.,* that if two closed terms are related in the empty context, then termination of the first implies termination of the second. For more details, see the appendix [13].

As in Ahmed [4], the converse of Theorem 4.4 does not hold. To make the logical relation complete w.r.t. contextual approximation is not difficult, however. It involves closing the definition of $\mathcal{V}\llbracket\tau\rrbracket\,\rho$, in the cases where τ is a type variable or an existential type, w.r.t. ciu-approximation on the right [20]. Assuming one introduces ciu-approximation as an atomic proposition, this is easy to do inside the logic. Space considerations preclude further discussion here.

Symmetric Version of the Logical Relation We have shown that our logical relation supports sound *inequational* reasoning, but we would like to support *equational* reasoning as well. Of course, one can prove two terms equivalent by proving that each approximates the other, but often this results in a tedious duplication of work. Fortunately, we can define a symmetric version of our logical relation directly in terms of the asymmetric one.

First, some notation: for a term relation R of type $\mathrm{TRel}(\tau_1, \tau_2)$, let R^{op} denote $(t_2 : \tau_2, t_1 : \tau_1).(t_1, t_2) \in R$, and similarly for value relations. Also, let ρ^{op} denote the mapping with domain equal to that of ρ such that if $\rho(\alpha) = (\tau_1, \tau_2, R)$, then $\rho^{\mathrm{op}}(\alpha) = (\tau_2, \tau_1, R^{\mathrm{op}})$.

Now, perhaps the most natural way of defining a symmetric version of our logical relation would be to say that two terms/values are symmetrically related if they are *logically equivalent*, *i.e.,* asymmetrically related (by $\mathcal{E}\llbracket\tau\rrbracket$) in both directions. Interestingly, this does not work. In particular, there are a variety of properties (described below) that we would like our symmetric relation to enjoy, one of them being the property that symmetrically-related function values f_1 and f_2 (of type $\tau' \to \tau''$) are precisely those that map symmetrically-related arguments (of type τ') to symmetrically-related results (of type τ''). However, just knowing that f_1 and f_2 map *equivalent* arguments to *equivalent* results does not imply that they map $\mathcal{V}\llbracket\tau'\rrbracket$-related arguments to $\mathcal{E}\llbracket\tau''\rrbracket$-related results, which must hold in order for f_1 and f_2 to be logically related in even one direction.

Thus, instead, we define the symmetric relation to be:

$$\mathcal{E}^{\approx}\llbracket\tau\rrbracket\,\rho \overset{\text{def}}{=} (t_1 : \rho_1\tau, t_2 : \rho_2\tau).\; (d = \texttt{true} \wedge (t_1, t_2) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho) \vee (d = \texttt{false} \wedge (t_2, t_1) \in \mathcal{E}\llbracket\tau\rrbracket\,\rho^{\mathrm{op}})$$

and similarly for $\mathcal{V}^{\approx}\llbracket\tau\rrbracket\,\rho$. Here, d is a value variable of type bool that we assume is bound in the context Γ in which

$$\frac{\Gamma;\Delta;\Theta\vdash e_1 \rightsquigarrow^* e_1' \vee e_1' \rightsquigarrow^0 e_1 \quad \Gamma;\Delta;\Theta\vdash e_2 \rightsquigarrow^* e_2' \vee e_2' \rightsquigarrow^0 e_2 \quad \Gamma;\Delta;\Theta\vdash (e_1',e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho}{\Gamma;\Delta;\Theta\vdash (e_1,e_2)\in\mathcal{E}^{\approx}[\![\tau]\!]\,\rho}\ (4S)$$

$$\frac{\Gamma;\Delta;\Theta\vdash e_1 \rightsquigarrow^1 e_1' \quad \Gamma;\Delta;\Theta\vdash e_2 \rightsquigarrow^1 e_2' \quad \Gamma;\Delta;\Theta\vdash \triangleright(e_1',e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho}{\Gamma;\Delta;\Theta\vdash (e_1,e_2)\in\mathcal{E}^{\approx}[\![\tau]\!]\,\rho}\ (5S)$$

$$\frac{\Gamma;\Delta;\Theta\vdash (e_1,e_2)\in\mathcal{E}^{\approx}[\![\tau]\!]\,\rho \qquad \Gamma,x_1,x_2;\Delta;\Theta,(x_1,x_2)\in\mathcal{V}^{\approx}[\![\tau]\!]\,\rho, e_1\rightsquigarrow^* x_1, e_2\rightsquigarrow^* x_2 \vdash (E_1[x_1],E_2[x_2])\in\mathcal{E}^{\approx}[\![\tau']\!]\,\rho'}{\Gamma;\Delta;\Theta\vdash (E_1[e_1],E_2[e_2])\in\mathcal{E}^{\approx}[\![\tau']\!]\,\rho'}\ (6S)$$

Figure 6. Symmetric Versions of Derivable Rules 4–6

these symmetric relations appear. When d is `true`, $\mathcal{E}^{\approx}[\![\tau]\!]\,\rho$ and $\mathcal{V}^{\approx}[\![\tau]\!]\,\rho$ are equivalent to the asymmetric logical relation in one direction; and when d is `false`, they are equivalent to the asymmetric relation in the other direction. Thus, by proving two terms to be symmetrically-related in a context where d's identity is unknown, we can effectively prove logical approximation in both directions simultaneously.

This formulation has several nice properties. First, it is straightforward to show that if we take each case of the definition of $\mathcal{V}[\![\tau]\!]\,\rho$ in Figure 4, replace all occurrences of $\mathcal{V}[\![\tau]\!]\,\rho$ and $\mathcal{E}[\![\tau]\!]\,\rho$ with their symmetric versions, and substitute $\equiv$ for $\overset{\text{def}}{=}$, we have a set of valid relational equivalences. The same goes for the relational equivalences in Lemma 4.1. (The same is not true, however, for the definition of $\mathcal{E}[\![\tau]\!]\,\rho$, because it is inherently asymmetric.)

Furthermore, we can derive symmetric versions of most of our derived rules. In most cases, the symmetric rule looks like the asymmetric one, except with $\mathcal{E}^{\approx}$ and $\mathcal{V}^{\approx}$ in place of $\mathcal{E}$ and $\mathcal{V}$. Exceptions to this pattern include Rules 3–6. Figure 6 gives symmetric versions for the last three of these. To give the reader a concrete sense of how these rules work, we present in the next section two detailed examples of how to use them to prove contextual equivalences.

Since LSLR is inspired by Plotkin and Abadi's logic for parametricity one might expect to see some axiom corresponding to "identity extension." In fact, we do not have an identity extension axiom since, as we have discovered in the course of carrying out this work, identity extension does not hold for the step-indexed model! For identity extension to hold, one would need that contextual equivalence at τ should *equal* the semantics of $\mathcal{E}^{\approx}[\![\tau]\!]$, but it only equals the subset of $\mathcal{E}^{\approx}[\![\tau]\!]$ for which the relation holds for all n, *i.e.,* roughly, the subset $\{(e_1,e_2) \mid \forall n.(e_1,e_2)\in[\![\mathcal{E}^{\approx}[\![\tau]\!]]\!]_n\}$. In spite of this, we are still able to prove Wadler-style *free theorems* [27]; see the appendix [13] for an example.

5 Examples

We now show two examples of how to use our LSLR-based logical relation to prove contextual equivalences.

The first example is from Crary and Harper [12] (who adapted it from one in Sumii and Pierce [26]) and concerns representation independence of "objects" with existential recursive type. The second example, from Sumii and Pierce [26], is concerned with proving that the syntactic projection function associated with a general recursive type is equivalent to the identity [9]. We reason informally in LSLR but present the proofs in some detail to emphasize the use of the derivable rules from Section 4. Observe that the proofs do not involve any step-indexed reasoning!

Example 1 Consider the following type for flag objects, which have an instance variable (with abstract type α) and two methods. The first method returns a new object whose flag is reversed, while the second method returns the current state of the flag.

$$\begin{aligned}\mathsf{fld}_\alpha &= \mu\beta.\,\alpha\times((\beta\to\beta)\times(\beta\to\mathsf{bool}))\\ \mathsf{flag} &= \exists\alpha.\,\mathsf{fld}_\alpha\end{aligned}$$

We consider two implementations of flags, in which the hidden flag state is represented by a bool and an int, respectively. We assume that $\texttt{not}:\mathsf{bool}\to\mathsf{bool}$ and $\texttt{even}:\mathsf{int}\to\mathsf{bool}$ are implemented in the obvious way.

$$\begin{aligned}\texttt{bflag} &= \texttt{pack}\,\mathsf{bool},(\texttt{fold}\,\langle\texttt{true},\langle\texttt{bflip},\texttt{bret}\rangle\rangle)\ \texttt{as}\ \mathsf{flag}\\ \texttt{bflip} &= \lambda x:\mathsf{fld}_{\mathsf{bool}}.\,\texttt{fold}\,\langle\texttt{not}\,(\texttt{fst}\,(\texttt{unfold}\,x)),\\ &\qquad\qquad \texttt{snd}\,(\texttt{unfold}\,x)\rangle\\ \texttt{bret} &= \lambda x:\mathsf{fld}_{\mathsf{bool}}.\,\texttt{fst}\,(\texttt{unfold}\,x)\end{aligned}$$

$$\begin{aligned}\texttt{iflag} &= \texttt{pack}\,\mathsf{int},(\texttt{fold}\,\langle 0,\langle\texttt{iflip},\texttt{iret}\rangle\rangle)\ \texttt{as}\ \mathsf{flag}\\ \texttt{iflip} &= \lambda x:\mathsf{fld}_{\mathsf{int}}.\,\texttt{fold}\,\langle 1+(\texttt{fst}\,(\texttt{unfold}\,x)),\\ &\qquad\qquad \texttt{snd}\,(\texttt{unfold}\,x)\rangle\\ \texttt{iret} &= \lambda x:\mathsf{fld}_{\mathsf{int}}.\,\texttt{even}\,(\texttt{fst}\,(\texttt{unfold}\,x))\end{aligned}$$

To prove contextual equivalence of `bflag` and `iflag`, it suffices to show $d:\mathsf{bool}\vdash(\texttt{bflag},\texttt{iflag})\in\mathcal{E}^{\approx}[\![\mathsf{flag}]\!]\,\emptyset$. Equivalently, by Rule 1, since both terms are values, we must show $d:\mathsf{bool}\vdash(\texttt{bflag},\texttt{iflag})\in\mathcal{V}^{\approx}[\![\mathsf{flag}]\!]\,\emptyset$. Following the definition of $\mathcal{V}^{\approx}[\![\exists\alpha.\,\mathsf{fld}_\alpha]\!]\,\emptyset$, we substitute $\alpha_1\mapsto\mathsf{bool}$, $\alpha_2\mapsto\mathsf{int}$, $y_1\mapsto v_1$, $y_2\mapsto v_2$, and $r\mapsto R$ where:

$$\begin{aligned}v_1 &= \texttt{fold}\,\langle\texttt{true},\langle\texttt{bflip},\texttt{bret}\rangle\rangle\\ v_2 &= \texttt{fold}\,\langle 0,\langle\texttt{iflip},\texttt{iret}\rangle\rangle\\ R &= (x_1\!:\mathsf{bool},x_2:\mathsf{int}).\,\exists y:\mathsf{int}.\,(x_1=\texttt{true}\wedge 2y\rightsquigarrow^* x_2)\\ &\qquad\vee\ (x_1=\texttt{false}\wedge 2y+1\rightsquigarrow^* x_2)\end{aligned}$$

Let $\rho=\alpha\mapsto(\mathsf{bool},\mathsf{nat},R)$. It now suffices to show $(v_1,v_2)\in\mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha]\!]\rho$, or equivalently (using the compatibility rules and several applications of Rule 1):

1. Show $(\texttt{true},0)\in\mathcal{V}^{\approx}[\![\alpha]\!]\,\rho$. This is immediate from the definition of R by substituting $y\mapsto 0$.

2. Show $(\mathtt{bflip}, \mathtt{iflip}) \in \mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha \to \mathsf{fld}_\alpha]\!]\,\rho$. Following Rule 9, we assume that $x_1 : \mathsf{fld}_{\mathsf{bool}}$, $x_2 : \mathsf{fld}_{\mathsf{int}}$, and $(x_1, x_2) \in \mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha]\!]\,\rho$, and are required to show:

$$\begin{array}{l}(\mathtt{fold}\,\langle \mathtt{not}\,(\mathtt{fst}\,(\mathtt{unfold}\,x_1)), \mathtt{snd}\,(\mathtt{unfold}\,x_1)\rangle,\\ \ \mathtt{fold}\,\langle 1 + (\mathtt{fst}\,(\mathtt{unfold}\,x_2)), \mathtt{snd}\,(\mathtt{unfold}\,x_2)\rangle)\\ \qquad \in \mathcal{E}^{\approx}[\![\mathsf{fld}_\alpha]\!]\,\rho\end{array}$$

By compatibility and Rule 1, we can show that $(\mathtt{fst}\,(\mathtt{unfold}\,x_1), \mathtt{fst}\,(\mathtt{unfold}\,x_2)) \in \mathcal{E}^{\approx}[\![\alpha]\!]\rho$. Thus, by Rule 6, we can assume that they evaluate to some z_1 and z_2, and that $(z_1, z_2) \in \mathcal{V}^{\approx}[\![\alpha]\!]\,\rho \equiv R$, and show:

$$\begin{array}{l}(\mathtt{fold}\,\langle \mathtt{not}\,z_1, \mathtt{snd}\,(\mathtt{unfold}\,x_1)\rangle,\\ \ \mathtt{fold}\,\langle 1 + z_2, \mathtt{snd}\,(\mathtt{unfold}\,x_2)\rangle) \in \mathcal{E}^{\approx}[\![\mathsf{fld}_\alpha]\!]\,\rho\end{array}$$

By compatibility and Rule 1, this reduces to showing that $(\mathtt{not}\,z_1, 1 + z_2) \in \mathcal{E}^{\approx}[\![\alpha]\!]\,\rho$. By Rule 4, we can reduce this to showing the following:

$$\begin{array}{l}\forall z_1 : \mathsf{bool}, z_2 : \mathsf{int}.\,(z_1, z_2) \in R \supset \exists z_1' : \mathsf{bool}, z_2' : \mathsf{int}.\\ \ \mathtt{not}\,z_1 \leadsto^* z_1' \wedge 1 + z_2 \leadsto^* z_2' \wedge (z_1', z_2') \in R\end{array}$$

Expanding out the definition of membership in R, we arrive at a strictly first-order statement that is provable by straightforward means in the meta-logic and can thus be included as an axiom (as per the discussion in the first part of Section 4).

3. Show $(\mathtt{bret}, \mathtt{iret}) \in \mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha \to \mathsf{bool}]\!]\,\rho$. This is similar to part 2, with the proof boiling down to the first-order statement:

$$\begin{array}{l}\forall z_1 : \mathsf{bool}, z_2 : \mathsf{int}.\,(z_1, z_2) \in R \supset\\ \ \exists z_2' : \mathsf{bool}.\,\mathtt{even}\,z_2 \leadsto^* z_2' \wedge z_1 = z_2'\end{array}$$

Example 2 Let $\tau = \mu\alpha.\,\mathsf{unit} + (\alpha \to \alpha)$. We show that the identity function $\mathtt{id} = \lambda x : \tau.\,x$ is equivalent to

$$\begin{array}{l}v = \mathtt{fix}\,f(x : \tau).\,\mathtt{case}\,(\mathtt{unfold}\,x)\,\mathtt{of}\,\mathtt{inl}\,\_ \Rightarrow \mathtt{fold}\,(\mathtt{inl}())\\ \qquad\qquad\qquad \mid \mathtt{inr}\,g \Rightarrow \mathtt{fold}\,(\mathtt{inr}\,(\lambda y : \tau.\,f(g(f y))))\end{array}$$

By Rule 1, to prove contextual equivalence of $\mathtt{id}$ and v, we can show $d : \mathsf{bool} \vdash (\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]\,\emptyset$. Our proof will be parametric in d, and we will omit the $\emptyset$ on the logical relation hereafter. By the Löb rule, we assume $\triangleright(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$ and proceed to prove $(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$. Now, by Rules 9 and 4S, we assume $x_1 : \tau$, $x_2 : \tau$, $(x_1, x_2) \in \mathcal{V}^{\approx}[\![\tau]\!]$, and it suffices to show:

$$(x_1, \mathtt{case}\,(\mathtt{unfold}\,x_2)\,\mathtt{of}\,\ldots) \in \mathcal{E}^{\approx}[\![\tau]\!]$$

By Rule 8 (extensionality for $\mathtt{unfold}$), it suffices to show:

$$\begin{array}{l}(\mathtt{unfold}\,x_1, \mathtt{unfold}\,(\mathtt{case}\,(\mathtt{unfold}\,x_2)\,\mathtt{of}\,\ldots))\\ \qquad \in \mathcal{E}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]\end{array}$$

From $(x_1, x_2) \in \mathcal{V}^{\approx}[\![\tau]\!]$, it follows that there exist y_1, y_2 such that $x_1 = \mathtt{fold}\,y_1$, $x_2 = \mathtt{fold}\,y_2$, and $\triangleright(y_1, y_2) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$. Thus, by expanding out the definitions of x_1 and x_2 and applying Rules 2 and 5S, our proof goal reduces to showing that

$$(y_1, \mathtt{unfold}\,(\mathtt{case}\,y_2\,\mathtt{of}\,\ldots)) \in \mathcal{E}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$$

under a strengthened context where the $\triangleright$ modalities have been removed from the earlier assumptions $(y_1, y_2) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$ and $(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, thus allowing us to use them (co-)inductively. Now, from $(y_1, y_2) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$, there are two cases:

Case 1 $y_1 = \mathtt{inl}\,y_1'$, $y_2 = \mathtt{inl}\,y_2'$, and $(y_1', y_2') \in \mathcal{V}^{\approx}[\![\mathsf{unit}]\!]$. Hence, $y_1 = y_2 = \mathtt{inl}\,()$. Since y_1 and $\mathtt{unfold}\,(\mathtt{case}\,y_2\,\mathtt{of}\ldots)$ both $\leadsto^*$ $\mathtt{inl}\,()$, the result follows by Rule 4S.

Case 2 $y_1 = \mathtt{inr}\,y_1'$, $y_2 = \mathtt{inr}\,y_2'$, and $(y_1', y_2') \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$.
Since $\mathtt{unfold}\,(\mathtt{case}\,y_2\,\mathtt{of}\ldots)$

$$\begin{array}{l}\leadsto^* \mathtt{unfold}\,(\mathtt{fold}\,(\mathtt{inr}\,(\lambda y : \tau.\,v\,(y_2'\,(v\,y)))))\\ \leadsto^* \mathtt{inr}\,(\lambda y : \tau.\,v\,(y_2'\,(v\,y))),\end{array}$$

to complete the proof, it suffices to show

$$(y_1', \lambda y : \tau.\,v\,(y_2'\,(v\,y))) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$$

Applying Rule 9 (followed by Rule 4S), we assume $z_1 : \tau$, $z_2 : \tau$, and $(z_1, z_2) \in \mathcal{V}^{\approx}[\![\tau]\!]$, and have to show:

$$(y_1'\,z_1, v\,(y_2'\,(v\,z_2))) \in \mathcal{E}^{\approx}[\![\tau]\!]$$

From $(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, together with relatedness of z_1 and z_2, we may conclude by Rules 7 and 4S that $(z_1, v\,z_2) \in \mathcal{E}^{\approx}[\![\tau]\!]$. By relatedness of y_1' and y_2' and Rule 7, we have that $(y_1'\,z_1, (y_2'\,(v\,z_2))) \in \mathcal{E}^{\approx}[\![\tau]\!]$. Thus, by Rule 6S, choosing as the evaluation contexts of interest $[\cdot]$ and $v\,[\cdot]$, our proof goal reduces to showing that for any $z_1' : \tau, z_2' : \tau$ such that $(z_1', z_2') \in \mathcal{V}^{\approx}[\![\tau]\!]$, it is the case that $(z_1', v\,z_2') \in \mathcal{E}^{\approx}[\![\tau]\!]$. As before, this follows from $(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, together with Rules 7 and 4S.

6 Related Work and Conclusion

As explained in the Introduction, LSLR is greatly indebted to (1) Plotkin and Abadi's logic for parametricity, and (2) Appel et al.'s "very modal model". However, there are also significant differences between our work and theirs.

Plotkin and Abadi's logic was originally developed for pure System F, as was Abadi, Cardelli and Curien's System R [1]. (The latter is less expressive, in that the only relations definable in the logic are those that are maps of System F functions.) In recent years, several extensions of PAL to richer languages with effects have been proposed. Plotkin [22] suggested a variant for a second-order linear type theory with a polymorphic fixed-point combinator to combine polymorphism with recursion; it relies on an abstract notion of admissible relations (see also [10]), whereas our logic LSLR does not. Bierman, Pitts and Russo [8] equipped the language suggested by Plotkin with an operational semantics, resulting in a programming language called Lily. Here instead we consider a standard call-by-value language with impredicative polymorphism and recursive types and show how to define a logic for reasoning about that language's operational semantics.

Our work differs from Appel *et al.*'s very modal model in the following ways. The main difference is in the application of the Kripke model: whereas Appel *et al.* use the later operator $\triangleright A$ to reason about type safety (a unary property) in a low-level language, we use it to reason about contextual approximation and equivalence (binary properties) in a high-level language. Certain issues, such as the development of both symmetric and asymmetric reasoning principles, do not arise in the unary setting. There are other concerns that do not apply to our setting, such as the desire for non-monotone predicates (hence our monotonicity axiom, which simplifies matters). Second, unlike Appel *et al.*, who define logical operators and types directly in terms of their model-theoretic interpretation and then prove lemmas about them, we define our logic axiomatically and then prove it sound w.r.t. a Kripke model. Although this is perhaps in practice a minor difference, we adopted our approach to make clear that our proofs of contextual (in-)equivalences are completely free of step-indexed reasoning.

As already mentioned, our application of the Löb rule in connection with a logical-relations method results in coinductive-style reasoning principles reminiscent of those used in bisimulation-based methods like Sumii and Pierce's [26], or Lassen and Levy's [16]. Bisimulations have also been developed for (in-)equational reasoning in languages with general references and/or control operators [15, 25, 24]. We hope that the present work will help to illuminate the relationship between step-indexed logical relations and bisimulation techniques, perhaps leading to a more unifying account.

Also related to our use of the Löb rule is the work of Brandt and Henglein [11], who gave a coinductive axiomatization of recursive type equality and subtyping via a coinduction-like rule. They also defined the semantic interpretation of their subtyping judgment using a stratified, essentially step-indexed, interpretation.

Finally, a number of logical-relations-based reasoning methods have been proposed for languages with parametric polymorphism, recursion, and/or recursive types, *e.g.,* [20, 14, 17, 4, 12]. We do not claim that the method presented in this paper is *per se* more powerful than prior approaches. Rather, our goal is to show how to reason about *step-indexed* logical relations in a more abstract way, because step-indexed relations have proven more easily adaptable than other logical-relations methods to languages with effects (particularly state) [3, 5, 19]. We believe that the work presented here makes an important first step toward *logical* step-indexed logical relations for effectful programs.

References

[1] Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121(1–2):9–58, 1993.

[2] Martín Abadi and Marcelo P. Fiore. Syntactic considerations on recursive types. In *LICS*, 1996.

[3] Umut A. Acar, Amal Ahmed, and Matthias Blume. Imperative self-adjusting computation. In *POPL*, 2008.

[4] Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *ESOP*, 2006. Extended/corrected version available as Harvard University TR-01-06.

[5] Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *POPL*, 2009.

[6] Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *Transactions on Programming Languages and Systems*, 23(5):657–683, 2001.

[7] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *POPL*, 2007.

[8] Gavin Bierman, Andrew Pitts, and Claudio Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *HOOTS*, volume 41 of *ENTCS*, 2000.

[9] Lars Birkedal and Robert W. Harper. Constructing interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63, 1999.

[10] Lars Birkedal, Rasmus E. Møgelberg, and Rasmus L. Petersen. Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 2(5:2):1–48, 2006.

[11] Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. In *TLCA*, 1997.

[12] Karl Crary and Robert Harper. Syntactic logical relations for polymorphic and recursive types. In *Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin*. ENTCS, 2007.

[13] Derek Dreyer, Amal Ahmed, and Lars Birkedal. Logical step-indexed logical relations (technical appendix), 2009. Available at: `http://www.mpi-sws.org/~dreyer/papers/lslr`.

[14] Patricia Johann and Janis Voigtlaender. The impact of seq on free theorems-based program transformations. *Fundamenta Informaticae*, 69(1–2):63–102, 2006.

[15] Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL*, 2006.

[16] Soren B. Lassen and Paul B. Levy. Normal form bisimulation for parametric polymorphism. In *LICS*, 2008.

[17] Paul-André Melliès and Jérôme Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS*, 2005.

[18] Hiroshi Nakano. A modality for recursion. In *LICS*, 2000.

[19] Georg Neis, Derek Dreyer, and Andreas Rossberg. Non-parametric parametricity. In *ICFP*, 2009.

[20] Andrew Pitts. Typed operational reasoning. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7. MIT Press, 2005.

[21] Gordon D. Plotkin. Denotational semantics with partial functions. Lecture at C.S.L.I. Summer School, 1985.

[22] Gordon D. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, February 1993.

[23] Gordon D. Plotkin and Martín Abadi. A logic for parametric polymorphism. In *TLCA*, 1993.

[24] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *LICS*, 2007.

[25] Kristian Støvring and Soren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *POPL*, 2007.

[26] Eijiro Sumii and Benjamin Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5):1–43, 2007.

[27] Philip Wadler. Theorems for free! In *FPCA*, 1989.