Logical Methods in Computer Science
Vol. 7 (2:16) 2011, pp. 1–37
www.lmcs-online.org

Submitted Jan. 8, 2010
Published Jun. 7, 2011

LOGICAL STEP-INDEXED LOGICAL RELATIONS *

DEREK DREYER [a], AMAL AHMED [b], AND LARS BIRKEDAL [c]

[a] MPI-SWS, Germany
e-mail address: dreyer@mpi-sws.org

[b] Indiana University, USA
e-mail address: amal@cs.indiana.edu

[c] IT University of Copenhagen, Denmark
e-mail address: birkedal@itu.dk

Abstract. Appel and McAllester's "step-indexed" logical relations have proven to be a simple and effective technique for reasoning about programs in languages with semantically interesting types, such as general recursive types and general reference types. However, proofs using step-indexed models typically involve tedious, error-prone, and proof-obscuring step-index arithmetic, so it is important to develop clean, high-level, equational proof principles that avoid mention of step indices.

In this paper, we show how to reason about binary step-indexed logical relations in an abstract and elegant way. 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, Melliès, Richards, and Vouillon's "very modal model" paper. We encode in LSLR a logical relation for reasoning relationally about programs in call-by-value System F extended with general recursive types. Using this logical relation, we derive a set of useful rules with which we can prove contextual equivalence and approximation results without counting steps.

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 (*i.e.,* for an arbitrarily wound-up clock), 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

1998 ACM Subject Classification: ??
Key words and phrases: ??

* This is an expanded and revised version of a paper that appeared at LICS'09. In addition to presenting improved results, this version corrects a technical flaw in the earlier paper (see Section 7).

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-7 (2:16) 2011

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 interesting types, such as general recursive types and general reference types [4, 3, 5, 24].

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 proved useful on a variety of examples, we found that our proofs using it tended to be cluttered with step-index arithmetic, to the point that their main substance was 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 what one ultimately 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 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 show how to alleviate this problem by reasoning inside a logic we call LSLR. Our approach 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) [30], and (2) Appel, Melliès, Richards, and Vouillon's "very modal model" paper (hereafter, AMRV) [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 quantification over abstract relation variables is 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 equivalence and approximation 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 and complete with respect to contextual approximation, 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 Ahmed's relation is fundamentally asymmetric, our logic enables the derivation of both equational and inequational reasoning principles.

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 calculi with simple [28] 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" $\triangleright P$ operator from AMRV, which they in turn borrowed from Gödel-Löb logic [23]. 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 $(\triangleright P \Rightarrow P) \Rightarrow P$. 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 [34].

1.1. **Overview.** In Section 2, we present our language under consideration, F^μ.

In Section 3, we present our logic LSLR described above. We 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. We also present a set of basic axioms that are sound with respect to this model, and which are useful in deriving more complex rules later in the paper.

In Section 4, we define a logical relation interpretation of F^μ types directly in terms of the syntactic relations of LSLR. Then we derive a set of useful rules for establishing properties about the logical relation. Using these rules, it is easy to show that the logical relation is sound and complete w.r.t. contextual approximation. We also show in this section how to define a symmetric version of the logical relation, which enables direct equational reasoning about F^μ programs.

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).

In Section 6, we demonstrate how our LSLR proofs improve on previous step-indexed proofs by comparing our proof for one of the examples from Section 5 to a proof of that example in the style of Ahmed [4].

In Section 7, we explain how the present version of LSLR improves on (and corrects a technical flaw in) the version we published previously in LICS 2009 [14].

Finally, in Section 8, 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. Sum and recursive type injections are

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

Figure 1: F^μ Syntax

$$
\begin{array}{llll}
\textit{Eval. Contexts} & E & ::= & [\cdot] \mid o(v_1, \ldots, v_{i-1}, E, e_{i+1}, \ldots, e_n) \mid \\
& & & \mathtt{if}\, E\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2 \mid \langle E, e_2\rangle \mid \langle v_1, E\rangle \mid \mathtt{fst}\, E \mid \mathtt{snd}\, E \mid \\
& & & \mathtt{inl}_\tau\, E \mid \mathtt{inr}_\tau\, E \mid \mathtt{case}\, E\, \mathtt{of}\, \mathtt{inl}\, x_1 \Rightarrow e_1 \mid \mathtt{inr}\, x_2 \Rightarrow e_2 \mid \\
& & & E\, e \mid v\, E \mid E\,\tau \mid \mathtt{pack}\,\tau_1, E\,\mathtt{as}\,\exists\alpha.\,\tau \mid \mathtt{unpack}\, E\,\mathtt{as}\,\alpha, x\,\mathtt{in}\, e_2 \mid \\
& & & \mathtt{roll}_\tau\, E \mid \mathtt{unroll}\, E
\end{array}
$$

$\boxed{e \rightsquigarrow e'}$

$$
\begin{array}{rl}
\mathtt{if}\,\mathtt{true}\,\mathtt{then}\, e_1\,\mathtt{else}\, e_2 & \rightsquigarrow e_1 \\
\mathtt{if}\,\mathtt{false}\,\mathtt{then}\, e_1\,\mathtt{else}\, e_2 & \rightsquigarrow e_2 \\
\mathtt{fst}\,\langle v_1, v_2\rangle & \rightsquigarrow v_1 \\
\mathtt{snd}\,\langle v_1, v_2\rangle & \rightsquigarrow v_2 \\
\mathtt{case}\,(\mathtt{inl}_\tau\, v)\,\mathtt{of}\,\mathtt{inl}\, x_1 \Rightarrow e_1 \mid \mathtt{inr}\, x_2 \Rightarrow e_2 & \rightsquigarrow e_1[v/x_1] \\
\mathtt{case}\,(\mathtt{inr}_\tau\, v)\,\mathtt{of}\,\mathtt{inl}\, x_1 \Rightarrow e_1 \mid \mathtt{inr}\, x_2 \Rightarrow e_2 & \rightsquigarrow e_2[v/x_2] \\
(\lambda x\!:\!\tau.\, e)\, v & \rightsquigarrow e[v/x] \\
(\Lambda\alpha.\, e)\,\tau & \rightsquigarrow e[\tau/\alpha] \\
\mathtt{unpack}\,(\mathtt{pack}\,\tau, v\,\mathtt{as}\,\exists\alpha.\,\tau_1)\,\mathtt{as}\,\alpha, x\,\mathtt{in}\, e & \rightsquigarrow e[v/x][\tau/\alpha] \\
\mathtt{unroll}\,(\mathtt{roll}_\tau\, v) & \rightsquigarrow v
\end{array}
$$

$$\frac{e \rightsquigarrow e'}{E[e] \rightsquigarrow E[e']}$$

Figure 2: F^μ Dynamic Semantics

type-annotated to ensure unique typing, but we will often omit the annotations when they are obvious from context. Figure 2 shows the left-to-right call-by-value dynamic semantics for the language, defined as a small-step relation on terms (written $e \rightsquigarrow e'$), which employs evaluation contexts E in the standard way. Note that the reduction relation is deterministic.

F^μ typing judgments have the form $\Gamma \vdash e : \tau$, where the context Γ binds type variables α, as well as term variables x: $\Gamma ::= \cdot \mid \Gamma, \alpha \mid \Gamma, x : \tau$. The typing rules are also standard and are given in full in Appendix A (Figure 10).

2.1. **Contextual Approximation and Equivalence.** A context C is a term with a single hole $[\cdot]$ in it. The typing judgment for contexts has the form $\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')$,

Relation Variables	r	$\in$	RelVar
F^μ *Variable Contexts*	$\mathcal{X}$	$::=$	$\cdot \mid \mathcal{X}, \alpha \mid \mathcal{X}, x$
F^μ *Variable Substitutions*	γ	$::=$	$\cdot \mid \gamma, \alpha \mapsto \tau \mid \gamma, x \mapsto e$
Relation Contexts	$\mathcal{R}$	$::=$	$\cdot \mid \mathcal{R}, r$
Relation Substitutions	φ	$::=$	$\cdot \mid \varphi, r \mapsto R$
Proposition Contexts	$\mathcal{P}$	$::=$	$\cdot \mid \mathcal{P}, P$
Combined Contexts	$\mathcal{C}$	$::=$	$\mathcal{X}; \mathcal{R}; \mathcal{P}$
Atomic Relations	A, B	$::=$	$e_1 = e_2 \mid \cdots$
Relations	P, Q, R, S	$::=$	$r \mid A \mid \top \mid \bot \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid$ $\forall \mathcal{X}.P \mid \exists \mathcal{X}.P \mid \forall \mathcal{R}.P \mid \exists \mathcal{R}.P \mid$ $\overline{x}.P \mid \overline{e} \in R \mid \mu r.R \mid \triangleright P$

Figure 3: Syntax of Core LSLR

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 Appendix A (Figures 11 and 12).

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 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 $\Gamma \vdash e_2 : \tau$.

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

$$\Gamma \vdash e_1 \approx^{ctx} e_2 : \tau \stackrel{\text{def}}{=} \Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau \ \wedge\ \Gamma \vdash e_2 \preceq^{ctx} e_1 : \tau$$

3. The Logic LSLR

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

3.1. Syntax.

The core syntax of LSLR is given in Figure 3.

F^μ variable contexts $\mathcal{X}$ are similar to F^μ contexts Γ, except that they omit type annotations on term variables. Instead, well-typedness of variables is modeled through explicit typing hypotheses in the proposition context $\mathcal{P}$ (see below). F^μ variable substitutions γ map variables bound in F^μ variable contexts to objects of the appropriate syntactic class.

As a matter of notation, we will use y and t as term variables in addition to x. Often, we write x or y to denote values, whereas t stands for arbitrary terms. (This is merely a mnemonic, however. The fact that x or y is a value will always be guaranteed by some separate, explicit assumption.)

Relation contexts $\mathcal{R}$ bind relation variables r, which stand for relations of arbitrary arity between F^μ terms. For ease of notation, we assume that relation variables r come equipped implicitly with a particular arity (namely, $\mathrm{arity}(r)$). Relation substitutions φ map relation variables to *relations* R of the appropriate arity, which we describe below.

Proposition contexts $\mathcal{P}$ are sets of *propositions*, which are just nullary relations that we typically denote using P and Q. (Note: We treat all three kinds of contexts as unordered sets, and use comma to denote disjoint union of such sets.)

We write $\mathcal{C}$ to denote a combined context $\mathcal{X};\mathcal{R};\mathcal{P}$. Correspondingly, we also define $\mathcal{C},\mathcal{X}'$ to mean $\mathcal{X},\mathcal{X}';\mathcal{R};\mathcal{P}$ (and similarly for $\mathcal{C},\mathcal{R}'$ and $\mathcal{C},\mathcal{P}'$).

Relations R (of which propositions P are a subset) fall into several categories: *variable* relations (r), *atomic* relations (A), *first-order* propositions ($\top$, $\bot$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $\forall\mathcal{X}.P$, $\exists\mathcal{X}.P$), *second-order* propositions ($\forall\mathcal{R}.P$, $\exists\mathcal{R}.P$), relation introduction and elimination ($\overline{x}.P$, $\overline{e} \in R$), recursive relations ($\mu r.R$), and the *later* modality ($\triangleright P$) borrowed from AMRV [7].

Atomic propositions A and the axioms concerning them are essentially orthogonal to the other components of the logic. We have listed in Figure 3 one particularly central atomic proposition, $e_1 = e_2$, which says that e_1 and e_2 are syntactically equal modulo renaming of bound variables. In Section 4.2, we will introduce several other atomic propositions related to the reduction semantics of F^μ. The only common requirement we impose on all of these atomic propositions is that they are first-order, in the sense that they only depend on type and term variables, not relation variables.

The first-order connectives are self-explanatory. The second-order ones provide the ability to abstract over a relation, which is critical in defining logical relations for polymorphic and existential types. As for the relational introduction and elimination forms: $\overline{x}.P$, which we sometimes write as $(\overline{x}).P$, introduces the term relation that one would write in set notation as $\{(\overline{x}) \mid P\}$, and $\overline{e} \in R$ says that the tuple of terms $(\overline{e})$ belong to the relation R. In general, we use the overbar notation to denote a possibly nullary tuple of objects.

A recursive relation $\mu r.R$ denotes the relation R that may refer to itself recursively via the variable r. In order to ensure that such relations are well-founded, we require that R be *contractive* in r, a notion that we make precise (following AMRV) using the modal $\triangleright$ operator. Specifically, we define R to be contractive in r if r may only appear in R underneath the $\triangleright$ operator (*i.e.,* inside propositions of the form $\triangleright P$). Intuitively (and formally), $\triangleright P$ means that P is true in all *strictly* future worlds of the current one. As a result, the meaning of $\mu r.R$ only depends recursively on its own meaning in strictly future worlds. Thus, assuming that the "strictly future world" ordering is well-founded, we can define the meaning of $\mu r.R$ by induction on strictly future worlds.

3.2. **A "Step-Indexed" Model of LSLR.** Figure 4 defines a Kripke model for LSLR, where the worlds are natural numbers and m is a strictly future world of n if $m < n$. The model enjoys *monotonicity*, meaning that if a proposition is true in world n, it is true in all strictly future worlds as well. Thus, the set of semantic truth values is the complete Heyting algebra $\mathcal{P}^{\downarrow}(\mathbb{N})$ of downward-closed subsets of $\mathbb{N}$, ordered by inclusion (or, isomorphically, the complete Heyting algebra $\vec{\omega}$ of vertical natural numbers with infinity).

We interpret relations and proposition contexts under some semantic interpretation δ, which maps their free relation variables to *semantic* (*i.e.,* world-indexed, monotone) relations of the appropriate arity. We write $[\![R]\!]\delta n\overline{e}$ (resp. $[\![\mathcal{P}]\!]\delta n$) to mean that, under interpretation δ, $\overline{e} \in R$ (resp. $\mathcal{P}$) is true in world n. The interpretations refer to $[\![\mathcal{X}]\!]$ and $[\![\mathcal{R}]\!]$. The semantic interpretation of a variable context, $[\![\mathcal{X}]\!]$, is the set of closing variable substitutions γ whose domains equal $\mathcal{X}$. The semantic interpretation of a relation context, $[\![\mathcal{R}]\!]$, is the set of semantic relation substitutions δ whose domains equal $\mathcal{R}$.

If $n = 0$, then:

$$\begin{array}{rcl} [\![R]\!]\delta n\overline{e} & \stackrel{\text{def}}{=} & \top \\ [\![\mathcal{P}]\!]\delta n & \stackrel{\text{def}}{=} & \top \end{array}$$

If $n > 0$, then:

$$\begin{array}{rcl} [\![r]\!]\delta n\overline{e} & \stackrel{\text{def}}{=} & \delta r n\overline{e} \\ [\![A]\!]\delta n\overline{e} & \stackrel{\text{def}}{=} & \mathcal{I}(A)\overline{e} \\ [\![\top]\!]\delta n & \stackrel{\text{def}}{=} & \top \\ [\![\bot]\!]\delta n & \stackrel{\text{def}}{=} & \bot \\ [\![P \wedge Q]\!]\delta n & \stackrel{\text{def}}{=} & [\![P]\!]\delta n \wedge [\![Q]\!]\delta n \\ [\![P \vee Q]\!]\delta n & \stackrel{\text{def}}{=} & [\![P]\!]\delta n \vee [\![Q]\!]\delta n \\ [\![P \Rightarrow Q]\!]\delta n & \stackrel{\text{def}}{=} & \forall k \leq n.\ [\![P]\!]\delta k \Rightarrow [\![Q]\!]\delta k \\ [\![\forall \mathcal{X}.P]\!]\delta n & \stackrel{\text{def}}{=} & \forall \gamma \in [\![\mathcal{X}]\!]\,.\ [\![\gamma P]\!]\delta n \\ [\![\exists \mathcal{X}.P]\!]\delta n & \stackrel{\text{def}}{=} & \exists \gamma \in [\![\mathcal{X}]\!]\,.\ [\![\gamma P]\!]\delta n \\ [\![\forall \mathcal{R}.P]\!]\delta n & \stackrel{\text{def}}{=} & \forall \delta' \in [\![\mathcal{R}]\!]\,.\ [\![P]\!](\delta, \delta')n \\ [\![\exists \mathcal{R}.P]\!]\delta n & \stackrel{\text{def}}{=} & \exists \delta' \in [\![\mathcal{R}]\!]\,.\ [\![P]\!](\delta, \delta')n \\ [\![\overline{x}.P]\!]\delta n\overline{e} & \stackrel{\text{def}}{=} & [\![P[\overline{e}/\overline{x}]]\!]\delta n \\ [\![\overline{e} \in R]\!]\delta n & \stackrel{\text{def}}{=} & [\![R]\!]\delta n\overline{e} \\ [\![\mu r.R]\!]\delta n\overline{e} & \stackrel{\text{def}}{=} & [\![R[\mu r.R/r]]\!]\delta n\overline{e} \\ [\![\triangleright P]\!]\delta n & \stackrel{\text{def}}{=} & [\![P]\!]\delta(n-1) \\ \\ [\![\mathcal{P}]\!]\delta n & \stackrel{\text{def}}{=} & \forall P \in \mathcal{P}.\ [\![P]\!]\delta n \end{array}$$

Figure 4: Kripke "Step-Indexed" Model of LSLR

The interpretations in Figure 4 are defined by a double induction, first on the world n (in world 0, everything is true), and second on the "size" of the relation being interpreted. The *size* of a relation is defined to equal the number of logical/relational connectives in it, *ignoring* all connectives appearing inside a proposition of the form $\triangleright P$ (*i.e.,* $\triangleright P$ has constant size, no matter what P is). This size metric makes it possible to interpret a recursive relation $\mu r.R$ directly in terms of its expansion $R[\mu r.R/r]$. Assuming the relation is well-formed, this interpretation is well-defined because the expansion has a smaller size. (Specifically, since R is contractive in r, we know that r may only appear inside constant-size propositions in R, so the size of $R[\mu r.R/r]$ equals the size of R, which is smaller than the size of $\mu r.R$.)

Since $\triangleright P$ may have smaller size than P, it is critical that the interpretation of $\triangleright P$ in world n be defined in terms of the interpretation of P in strictly future worlds (*i.e.,* worlds strictly less than n). Fortunately, this is no problem since, as explained above, $\triangleright P$ means precisely that P is true in all strictly future worlds. Thanks to the built-in monotonicity restriction, it suffices to say that $\triangleright P$ is true in world n iff P is true in world $n-1$.

Otherwise, the interpretation is mostly standard. One point of note is the interpretation of implication $P \Rightarrow Q$, which quantifies over all future worlds in order to ensure monotonicity. Another is the interpretation of atomic relations A. We assume an interpretation function $\mathcal{I}$, which maps closed atomic relations A to *absolute* (*i.e.,* world-independent) relations. As one instance, we define $\mathcal{I}(e_1 = e_2)$ to be true ($\top$) iff e_1 is α-equivalent to e_2.

Using this model, we can define our main logical judgment, $\mathcal{X}; \mathcal{R}; \mathcal{P} \vdash P$. Assuming that $\mathcal{P}$ and P are well-formed in $\mathcal{X}; \mathcal{R}$ (see Appendix B for the definition of proposition/relation

$$\frac{\mathcal{C} \vdash P}{\mathcal{C} \vdash \triangleright P}\ (\textsc{mono}) \qquad \frac{\mathcal{C}, \triangleright P \vdash P}{\mathcal{C} \vdash P}\ (\textsc{löb})$$

$$\frac{\mathcal{C} \vdash \triangleright(P \wedge Q)}{\mathcal{C} \vdash \triangleright P \wedge \triangleright Q}\ (\triangleright\wedge) \qquad \frac{\mathcal{C} \vdash \triangleright(P \vee Q)}{\mathcal{C} \vdash \triangleright P \vee \triangleright Q}\ (\triangleright\vee) \qquad \frac{\mathcal{C} \vdash \triangleright(P \Rightarrow Q)}{\mathcal{C} \vdash \triangleright P \Rightarrow \triangleright Q}\ (\triangleright\Rightarrow)$$

$$\frac{\mathcal{C} \vdash \triangleright\forall\mathcal{X}.P}{\mathcal{C} \vdash \forall\mathcal{X}.\triangleright P}\ (\triangleright\forall 1) \qquad \frac{\mathcal{C} \vdash \triangleright\exists\mathcal{X}.P}{\mathcal{C} \vdash \exists\mathcal{X}.\triangleright P}\ (\triangleright\exists 1) \qquad \frac{\mathcal{C} \vdash \triangleright\forall\mathcal{R}.P}{\mathcal{C} \vdash \forall\mathcal{R}.\triangleright P}\ (\triangleright\forall 2) \qquad \frac{\mathcal{C} \vdash \triangleright\exists\mathcal{R}.P}{\mathcal{C} \vdash \exists\mathcal{R}.\triangleright P}\ (\triangleright\exists 2)$$

$$\frac{\mathcal{C} \vdash e_1 = e_2 \quad \mathcal{C} \vdash P[e_1/x]}{\mathcal{C} \vdash P[e_2/x]}\ (\textsc{replace}1) \qquad \frac{\mathcal{C} \vdash R_1 \equiv R_2 \quad \mathcal{C} \vdash P[R_1/r]}{\mathcal{C} \vdash P[R_2/r]}\ (\textsc{replace}2)$$

$$\frac{\mathcal{C} \vdash \overline{e} \in \overline{x}.P}{\mathcal{C} \vdash P[\overline{e}/\overline{x}]}\ (\textsc{elem}) \qquad \frac{\mathcal{C} \vdash \overline{e} \in \mu r.R}{\mathcal{C} \vdash \overline{e} \in R[\mu r.R/r]}\ (\textsc{elem}\text{-}\mu)$$

Figure 5: Core Inference Rules of LSLR

well-formedness), the judgment is interpreted as follows:

$$\mathcal{X}; \mathcal{R}; \mathcal{P} \vdash P \quad \stackrel{\text{def}}{=} \quad \forall n \geq 0.\ \forall \gamma \in [\![\mathcal{X}]\!]\,.\ \forall \delta \in [\![\mathcal{R}]\!]\,.\ [\![\gamma\mathcal{P}]\!]\delta n \ \Rightarrow\ [\![\gamma P]\!]\delta n$$

Note that we interpret the judgment directly as a statement in the model, rather than inductively defining it via a set of inference rules. This allows us to prove new inference rules sound whenever needed. In the next section, however, we will establish a core set of sound inference rules that will enable us to reason about the judgment (in most cases) without having to appeal directly to the model.

The judgment asserts that under any closing substitution γ for $\mathcal{X}$ and any semantic interpretation δ for $\mathcal{R}$, and in any world n, the hypotheses $\mathcal{P}$ imply the conclusion P. The key here is that, while n is universally quantified and thus not explicitly mentioned in the logical judgment, the hypotheses $\mathcal{P}$ and the conclusion P are both interpreted in the *same* world (*i.e.,* step level) n. This is what allows us to prove something like "f_1 and f_2 map n-related arguments to n-related results" (as discussed in the introduction) without having to talk about a specific step level n.

Finally, it is worth noting that, while the Kripke model we have defined here may be viewed as a "step-indexed" model, nothing in the model mentions steps of computation. We happen to be using natural numbers as our worlds, but there is no computational meaning attached to them at this point. The connection between worlds and (certain) steps of computation will be made later on, when we define the logical relation for F^{μ} in Section 4.

3.3. **Core Inference Rules.** We now present the core inference rules of LSLR, all of which are easy to prove sound directly in the model. The most interesting ones are shown in Figure 5; the remainder, all of which are standard rules for second-order intuitionistic logic, appear in Appendix B.

Rule MONO is the axiom of monotonicity, stating that propositions that are true now (in the current world) are also true later (in future worlds). The LÖB rule, adapted from AMRV, provides a clean induction principle over future worlds. If under the assumption that A is true later (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 (*i.e.,* world 0).

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 AMRV, but they hold here due to our axiom of monotonicity. For example, we give here the proof of Rule $\triangleright\Rightarrow$:

Proposition 3.1. *Rule* $\triangleright\Rightarrow$ *is admissible.*

Proof. First, the forwards direction. Suppose $[\![\triangleright(P \Rightarrow Q)]\!]\delta n$ and $[\![\triangleright P]\!]\delta n$. We want to show $[\![\triangleright Q]\!]\delta n$. If $n = 0$, the proof is trivial, so assume $n > 0$. By the interpretation of $\triangleright$, we know $[\![P \Rightarrow Q]\!]\delta(n-1)$ and $[\![P]\!]\delta(n-1)$. Thus, by the interpretation of $\Rightarrow$, we know $[\![Q]\!]\delta(n-1)$, which is equivalent to our goal.

Next, the backwards direction. Suppose $[\![\triangleright P \Rightarrow \triangleright Q]\!]\delta n$; we want to show $[\![\triangleright(P \Rightarrow Q)]\!]\delta n$. If $n = 0$, the proof is trivial, so assume $n > 0$. Our goal is equivalent to $[\![P \Rightarrow Q]\!]\delta(n-1)$, so suppose $k \leq n-1$ and $[\![P]\!]\delta k$, and we will prove $[\![Q]\!]\delta k$. By the interpretation of $\triangleright$, we know $[\![\triangleright P]\!]\delta(k+1)$. Since $k+1 \leq n$, by the interpretation of $\Rightarrow$ we obtain $[\![\triangleright Q]\!]\delta(k+1)$, which is equivalent to $[\![Q]\!]\delta k$, our desired goal.

Note that the backwards direction relies critically on monotonicity. In the absence of monotonicity, the premise $[\![\triangleright P \Rightarrow \triangleright Q]\!]\delta n$ is only applicable if $[\![P]\!]\delta k$ for *all* $k < n$, but in the proof we only assume $[\![P]\!]\delta k$ for *some* $k < n$. □

The replacement axioms (REPLACE1 and REPLACE2) say that we can substitute equals for equals inside a proposition without affecting its meaning. For terms, equality is just syntactic equality. For relations, equivalence is definable as

$$R_1 \equiv R_2 \quad \stackrel{\text{def}}{=} \quad \forall \overline{x}.\ (\overline{x} \in R_1 \Rightarrow \overline{x} \in R_2)\ \wedge\ (\overline{x} \in R_2 \Rightarrow \overline{x} \in R_1)$$

The last two rules concern inhabitation of relations. The key interesting point here is that recursive relations are equivalent to their expansions.

Lastly, when we introduce atomic propositions in the next section related to F^μ reduction, we will want to also import into LSLR various first-order theorems about those propositions, *e.g.,* preservation, progress, canonical forms, etc. Fortunately, this can be done easily, without requiring any stepwise reasoning.

Formally, assuming P is a first-order proposition (*i.e.,* it does not involve relation variables, recursive relations, second-order quantification, or the $\triangleright$ operator), then it is easy to show that P is true in all worlds n iff it is true in world 1 (the "latest" nontrivial world). Consequently, the following rule is sound:

$$\frac{\forall \gamma \in [\![\mathcal{X}]\!]\,.\ \forall \delta \in [\![\mathcal{R}]\!]\,.\ [\![\gamma\mathcal{P}]\!]\,\delta(1) \Rightarrow [\![\gamma P]\!]\,\delta(1)}{\mathcal{X}; \mathcal{R}; \mathcal{P} \vdash P}$$

Thus, in particular, if P is closed:

$$\frac{[\![P]\!]\,1}{\vdash P}$$

For first-order P, the interpretation of $[\![P]\!]\,1$ in our model is tantamount to the standard step-free interpretation of P in first-order logic.

In other words, our goal here is not to use LSLR to formalize *entire* proofs, just the parts of the proofs that involve interesting relational reasoning. We are happy to make use of first-order syntactic properties proved by other means in the meta-logic.

4. A Syntactic Logical Relation for F^μ

In this section, we show how to define a logical relation for F^μ that coincides with contextual approximation, as well as a symmetric version thereof that coincides with contextual equivalence. The relation is defined *syntactically* within the logic LSLR, using a particular set of atomic propositions concerning the F^μ reduction semantics, as we explain below.

4.1. Roadmap and Preliminaries.

Eventually, we are going to define a logical relation on open terms, which we denote $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$, and prove that it is sound and complete w.r.t. contextual approximation, $\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$, as defined in Section 2. In order to prove this, we will follow Pitts [26] in employing an intermediate form of approximation, often referred to as *ciu approximation*.

Ciu approximation, due to Mason and Talcott [21], is a superficially coarser version of contextual approximation in which (1) attention is restricted to evaluation contexts E instead of arbitrary program contexts, and (2) the "closing" of open terms is handled by an explicit substitution γ instead of relying on λ-abstractions in a closing context C. We say that ciu approximation is only *superficially* coarser because ultimately we will prove that it too coincides with contextual approximation. In the meantime, ciu approximation turns out to be an easier notion of approximation to work with.

First, a bit of notation: we will write $\vdash \gamma : \Gamma$ to mean that (1) $\mathrm{dom}(\gamma) = \mathrm{dom}(\Gamma)$, (2) $\forall \alpha \in \Gamma.\ \mathrm{FV}(\gamma\alpha) = \emptyset$, and (3) $\forall x : \tau \in \Gamma.\ \exists v.\ \gamma x = v \wedge \ \vdash v : \gamma\tau$. We will also write $\vdash E : \tau \rightsquigarrow \tau'$ to mean $\vdash E : (\cdot \vdash \tau) \rightsquigarrow (\cdot \vdash \tau')$, thus defining the typing of evaluation contexts in terms of the typing judgment for general contexts C (introduced in Section 2.1).

Definition 4.1 (Ciu Approximation for Closed Terms)**.** Let $\cdot \vdash e_1 : \tau$ and $\cdot \vdash e_2 : \tau$.

$$\vdash e_1 \preceq^{ciu} e_2 : \tau \ \stackrel{\text{def}}{=}\ \forall E, \tau'.\ (\vdash E : \tau \rightsquigarrow \tau' \wedge E[e_1]\Downarrow)\ \Rightarrow E[e_2]\Downarrow$$

Definition 4.2 (Ciu Approximation for Open Terms)**.** Let $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$.

$$\Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau \ \stackrel{\text{def}}{=}\ \forall \gamma.\ \vdash \gamma : \Gamma\ \Rightarrow\ \vdash \gamma e_1 \preceq^{ciu} \gamma e_2 : \gamma\tau$$

Definition 4.3 (Ciu Equivalence)**.** Let $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$.

$$\Gamma \vdash e_1 \approx^{ciu} e_2 : \tau \ \stackrel{\text{def}}{=}\ \Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau\ \wedge\ \Gamma \vdash e_2 \preceq^{ciu} e_1 : \tau$$

One of the main reasons to use ciu approximation instead of contextual approximation is that it is immediately obvious that the F^μ reduction relation is contained in ciu equivalence (part (3) of the following proposition).

Proposition 4.4 (Useful Properties of Ciu Approximation)**.**
(1) *If* $\Gamma \vdash e : \tau$*, then* $\Gamma \vdash e \preceq^{ciu} e : \tau$*.*
(2) *If* $\Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau$ *and* $\Gamma \vdash e_2 \preceq^{ciu} e_3 : \tau$*, then* $\Gamma \vdash e_1 \preceq^{ciu} e_3 : \tau$*.*
(3) *If* $\Gamma \vdash e_1 : \tau$ *and* $e_1 \rightsquigarrow^* e_2$*, then* $\Gamma \vdash e_1 \approx^{ciu} e_2 : \tau$*.*
(4) *If* $\vdash e_1 \preceq^{ciu} e_2 : \tau$ *and* $\vdash E : \tau \rightsquigarrow \tau'$*, then* $\vdash E[e_1] \preceq^{ciu} E[e_2] : \tau'$*.*

Again following Pitts [26], we will show that contextual, ciu, and logical approximation all coincide by showing that $\preceq^{ctx} \subseteq \preceq^{ciu} \subseteq \preceq^{log} \subseteq \preceq^{ctx}$. The first link of that chain is easy.

Theorem 4.5 (Contextual Approximation ⇒ Ciu Approximation)**.**
If $\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$*, then* $\Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau$*.*

Proof. Suppose $\vdash \gamma : \Gamma$, $\vdash E : \gamma\tau \rightsquigarrow \tau'$, and $E[\gamma e_1] \Downarrow$. We want to show $E[\gamma e_2] \Downarrow$. Say that $\Gamma = \alpha_1, \ldots, \alpha_m, x_1 : \tau_1, \ldots, x_n : \tau_n$ and that $\gamma\alpha_i = \sigma_i$ and $\gamma x_i = v_i$ for some σ_i's and v_i's. Then, let $C = (\Lambda\alpha_1. \cdots \Lambda\alpha_m. \lambda x_1 : \tau_1. \cdots \lambda x_n : \tau_n. [\cdot])\, \sigma_1 \cdots \sigma_m v_1 \cdots v_n$. It is easy to show that $\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\cdot \vdash \gamma\tau)$, and thus that $\vdash E[C] : (\Gamma \vdash \tau) \rightsquigarrow (\cdot \vdash \tau')$. It is also easy to show that $E[C[e_i]] \rightsquigarrow^* E[\gamma e_i]$, and thus that $E[C[e_i]] \Downarrow$ iff $E[\gamma e_i] \Downarrow$. So the goal is reduced to showing that $E[C[e_1]] \Downarrow$ implies $E[C[e_2]] \Downarrow$, which follows from $\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$. □

4.2. **Atomic Relations.** In order to define our logical relation, we introduce the following new atomic relations:

$$A \ ::= \ \cdots \mid \mathrm{Val} \mid e : \tau \mid C : \tau \rightsquigarrow \tau' \mid e_1 \rightsquigarrow^* e_2 \mid e_1 \rightsquigarrow^0 e_2 \mid e_1 \rightsquigarrow^1 e_2 \mid e_1 \preceq e_2$$

Except for the first, which is a unary relation, the rest are all nullary (*i.e.,* propositions). The interpretations of these propositions, $\mathcal{I}(A)$, are as follows:

- $\mathcal{I}(\mathrm{Val})(e) \stackrel{\text{def}}{=} \exists v.\ e = v.$
- $\mathcal{I}(e : \tau) \stackrel{\text{def}}{=} \vdash e : \tau.$
- $\mathcal{I}(C : \tau \rightsquigarrow \tau') \stackrel{\text{def}}{=} \exists E.\ C = E \ \wedge \vdash E : \tau \rightsquigarrow \tau'.$
- $\mathcal{I}(e_1 \rightsquigarrow^* e_2) \stackrel{\text{def}}{=} e_1 \rightsquigarrow^* e_2.$
- $\mathcal{I}(e_1 \rightsquigarrow^0 e_2) \stackrel{\text{def}}{=} e_1 \rightsquigarrow^* e_2$ and *none* of the reductions in the reduction sequence is an `unroll-roll` reduction.
- $\mathcal{I}(e_1 \rightsquigarrow^1 e_2) \stackrel{\text{def}}{=} e_1 \rightsquigarrow^* e_2$ and *exactly one* of the reductions in the reduction sequence is an `unroll-roll` reduction.
- $\mathcal{I}(e_1 \preceq e_2) \stackrel{\text{def}}{=} \exists \tau.\ \vdash e_1 \preceq^{ciu} e_2 : \tau.$

The motivation for using this particular set of atomic propositions will become clear shortly. One point of note is that the $e_1 \preceq e_2$ proposition lacks a type; this is simply for brevity, since F^μ enjoys unique typing. Another is that, although the proposition $C : \tau \rightsquigarrow \tau'$ permits an arbitrary context C, the proposition only holds when C takes the form of an evaluation context, and we will only use it when C is an evaluation context. The reason that we do not syntactically write E here instead of C is simply that the syntaxes of values v and evaluation contexts E are not closed under substitution of arbitrary terms for variables—they assume that variables are values—and we want proposition well-formedness to be preserved under arbitrary term substitutions. All this means, practically speaking, is that something like $x\,[\cdot] : \tau \rightsquigarrow \tau'$ cannot hold categorically, but only in a context where $x \in \mathrm{Val}$ is also provable.

As explained in Section 3.3, along with these new atomic propositions, we will also make use of various first-order theorems about them, which are provable straightforwardly in the meta-logic without requiring any stepwise reasoning. For example,

$$\frac{\mathcal{C} \vdash e \rightsquigarrow^1 e_1 \quad \mathcal{C} \vdash e \rightsquigarrow^1 e_2}{\mathcal{C} \vdash e_1 \rightsquigarrow^0 e_2 \ \vee \ e_2 \rightsquigarrow^0 e_1}$$

and

$$\frac{\mathcal{C} \vdash E : \tau \rightsquigarrow \tau' \quad \mathcal{C} \vdash e_1 : \tau \quad \mathcal{C} \vdash e_2 : \tau \quad \mathcal{C} \vdash e_1 \preceq e_2}{\mathcal{C} \vdash E[e_1] \preceq E[e_2]}$$

See the proofs in subsequent sections for more examples.

$$
\begin{array}{rcl}
\mathcal{V}\llbracket\alpha\rrbracket\rho & \stackrel{\text{def}}{=} & R, \quad \text{where } \rho(\alpha) = (\tau_1, \tau_2, R) \\
\mathcal{V}\llbracket\tau_b\rrbracket\rho & \stackrel{\text{def}}{=} & (x_1 \downarrow \tau_b, x_2 \downarrow \tau_b).\ x_1 = x_2, \quad \text{where } \tau_b \in \{\mathsf{unit}, \mathsf{int}, \mathsf{bool}\} \\
\mathcal{V}\llbracket\tau' \times \tau''\rrbracket\rho & \stackrel{\text{def}}{=} & (x_1 \downarrow \rho_1(\tau' \times \tau''), x_2 \downarrow \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 \\
& & \qquad (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 \downarrow \rho_1(\tau' + \tau''), x_2 \downarrow \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 \downarrow \rho_1(\tau' \to \tau''), x_2 \downarrow \rho_2(\tau' \to \tau'')). \\
& & \quad \forall y_1, y_2.\ (y_1, y_2) \in \mathcal{V}\llbracket\tau'\rrbracket\rho \Rightarrow (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 \downarrow \rho_1(\forall\alpha.\,\tau), x_2 \downarrow \rho_2(\forall\alpha.\,\tau)). \\
& & \quad \forall\alpha_1, \alpha_2.\ \forall r.\ r : \mathrm{VRel}(\alpha_1, \alpha_2) \Rightarrow (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 \downarrow \rho_1(\exists\alpha.\,\tau), x_2 \downarrow \rho_2(\exists\alpha.\,\tau)). \\
& & \quad \exists\alpha_1, \alpha_2, y_1, y_2.\ \exists r.\ r : \mathrm{VRel}(\alpha_1, \alpha_2) \ \wedge \\
& & \qquad 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 \\
& & \qquad (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 \downarrow \rho_1(\mu\alpha.\,\tau), x_2 \downarrow \rho_2(\mu\alpha.\,\tau)). \\
& & \quad \exists y_1, y_2.\ x_1 = \mathtt{roll}\, y_1 \ \wedge\ x_2 = \mathtt{roll}\, y_2 \ \wedge \\
& & \qquad \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). \\
& & \quad (\forall x_1.\ t_1 \Downarrow^0 x_1 \Rightarrow \exists x_2.\ x_2 \preceq t_2 \wedge (x_1, x_2) \in \mathcal{V}\llbracket\tau\rrbracket\rho) \ \wedge \\
& & \quad (\forall t_1'.\ t_1 \rightsquigarrow^1 t_1' \Rightarrow \triangleright(t_1', t_2) \in r)
\end{array}
$$

Figure 6: Syntactic Logical Relation for F^μ

Finally, we will make use of some additional notation, which is definable in terms of the atomic propositions we have introduced:

$$
\begin{array}{rcl}
e \downarrow \tau & \stackrel{\text{def}}{=} & e : \tau \ \wedge\ e \in \mathrm{Val} \\
e_1 \Downarrow e_2 & \stackrel{\text{def}}{=} & e_1 \rightsquigarrow^* e_2 \ \wedge\ e_2 \in \mathrm{Val} \\
e_1 \Downarrow^0 e_2 & \stackrel{\text{def}}{=} & e_1 \rightsquigarrow^0 e_2 \ \wedge\ e_2 \in \mathrm{Val} \\
R : \mathrm{TRel}(\tau_1, \tau_2) & \stackrel{\text{def}}{=} & \forall x_1, x_2.\ (x_1, x_2) \in R \ \Rightarrow\ x_1 : \tau_1 \ \wedge\ x_2 : \tau_2 \\
R : \mathrm{VRel}(\tau_1, \tau_2) & \stackrel{\text{def}}{=} & \forall x_1, x_2.\ (x_1, x_2) \in R \ \Rightarrow\ x_1 \downarrow \tau_1 \ \wedge\ x_2 \downarrow \tau_2 \\
(x_1 : \tau_1, x_2 : \tau_2).\ P & \stackrel{\text{def}}{=} & (x_1, x_2).\ x_1 : \tau_1 \ \wedge\ x_2 : \tau_2 \ \wedge\ P \\
(x_1 \downarrow \tau_1, x_2 \downarrow \tau_2).\ P & \stackrel{\text{def}}{=} & (x_1, x_2).\ x_1 \downarrow \tau_1 \ \wedge\ x_2 \downarrow \tau_2 \ \wedge\ P
\end{array}
$$

4.3. **Logical Relation.** Figure 6 defines two logical relations for F^μ, one for values ($\mathcal{V}\llbracket\tau\rrbracket\rho$) and one for terms ($\mathcal{E}\llbracket\tau\rrbracket\rho$). These are syntactic LSLR relations, defined by induction on τ. Here, ρ is assumed to be 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) such that $R : \mathrm{VRel}(\tau_1, \tau_2)$. We write ρ_i to mean the type substitution mapping each α to the corresponding τ_i. Thus, it is trivial to prove that $\mathcal{V}\llbracket\tau\rrbracket\rho : \mathrm{VRel}(\rho_1\tau, \rho_2\tau)$ and $\mathcal{E}\llbracket\tau\rrbracket\rho : \mathrm{TRel}(\rho_1\tau, \rho_2\tau)$. 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, following Plotkin and Abadi [30], with each type constructor being modeled by its corresponding logical connective via the Curry-Howard isomorphism.

First, let us consider $\mathcal{V}[\![\mu\alpha.\tau]\!]\rho$. 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 $\texttt{roll}\, v_1$ and $\texttt{roll}\, v_2$ are related by $\mathcal{V}[\![\mu\alpha.\tau]\!]\rho$ "now" iff v_1 and v_2 are related by $\mathcal{V}[\![\tau]\!]\rho, \alpha \mapsto (\ldots, \mathcal{V}[\![\mu\alpha.\tau]\!]\rho) = \mathcal{V}[\![\tau[\mu\alpha.\tau/\alpha]]\!]\rho$ "later".

Next, consider $\mathcal{E}[\![\tau]\!]\rho$. Intuitively, we would like to say that two terms e_1 and e_2 are related if, whenever e_1 evaluates to some value v_1, we have that e_2 also evaluates to some value v_2 such that $(v_1, v_2) \in \mathcal{V}[\![\tau]\!]\rho$. In fact, in the case that e_1 evaluates to v_1 without incurring any `unroll-roll` reductions (*i.e.,* when $e_1 \Downarrow^0 v_1$), the definition of $\mathcal{E}[\![\tau]\!]\rho$ *almost* says this—the only difference is that instead of saying "e_2 evaluates to some value v_2 such that…", it says that "e_2 is ciu-approximated by some value v_2 such that…" Of course, by definition of ciu approximation, this also implies that e_2 terminates, but it is somewhat more liberal in that it does not require the value that e_2 produces to be directly related to v_1 by $\mathcal{V}[\![\tau]\!]\rho$. This extra freedom is not strictly necessary if we just want to define a logical relation that is *sound* w.r.t. contextual approximation—as we did in the previous version of this paper [14]—but it is key to ensuring *completeness* (see Theorems 4.24 and 4.25 in Section 4.6). An alternative approach to ensuring completeness would be to employ $\top\top$-closure, as Pitts does [26]. We discuss this alternative in Section 8.

However, in the case that the evaluation of e_1 incurs an `unroll-roll` reduction, the interpretation of recursive types forces us to require something still weaker. 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 [26]. Compatibility for `unroll` demands that if $\texttt{roll}\, v_1$ and $\texttt{roll}\, v_2$ are logically related, then $\texttt{unroll}\,(\texttt{roll}\, v_1)$ and $\texttt{unroll}\,(\texttt{roll}\, v_2)$ are related, too. By definition of $\mathcal{V}[\![\mu\alpha.\tau]\!]\rho$, knowing $\texttt{roll}\, v_1$ and $\texttt{roll}\, v_2$ are related only tells us that v_1 and v_2 are related "later". We need to be able to derive from that that $\texttt{unroll}\,(\texttt{roll}\, v_1)$ and $\texttt{unroll}\,(\texttt{roll}\, v_2)$ are related "now". Thus, in defining whether $(e_1, e_2) \in \mathcal{E}[\![\tau]\!]\rho$, in the case that e_1 makes an `unroll-roll` reduction (*i.e.,* $e_1 \rightsquigarrow^1 e_1'$), we only require that e_1' and e_2 be related *later* (*i.e.,* $\triangleright(e_1', e_2) \in \mathcal{E}[\![\tau]\!]\rho$).

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 `unroll-roll` 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 FUNEXT 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.

$$\frac{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}\ (\textsc{val}) \qquad \frac{\triangleleft\mathcal{C} \vdash P}{\mathcal{C} \vdash \triangleright P}\ (\textsc{weak-}\triangleright)$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash e_1 : \rho_1\tau \\ \mathcal{C} \vdash e_1 \rightsquigarrow^* e_1' \quad \mathcal{C} \vdash (e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho\end{array}}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}\ (\textsc{exp}) \quad \frac{\begin{array}{c}\mathcal{C} \vdash e_1 : \rho_1\tau \quad \mathcal{C} \vdash e_2 : \rho_2\tau \\ \mathcal{C} \vdash e_1 \rightsquigarrow^1 e_1' \quad \mathcal{C} \vdash \triangleright(e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho\end{array}}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}\ (\textsc{exp-}\triangleright)$$

$$\frac{\mathcal{C} \vdash e_1' \rightsquigarrow^0 e_1 \quad \mathcal{C} \vdash (e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}\ (\textsc{red}) \quad \frac{\mathcal{C} \vdash (e_1, e_2') \in \mathcal{E}\,[\![\tau]\!]\,\rho \quad \mathcal{C} \vdash e_2' \preceq e_2}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho}\ (\textsc{ciu})$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash E : \rho_1\tau \rightsquigarrow \rho_1'\tau' \quad \mathcal{C} \vdash f : \rho_2'\tau' \quad \mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho \\ \mathcal{C}, x_1, x_2, (x_1, x_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho, e_1 \rightsquigarrow^* x_1, x_2 \preceq e_2 \vdash (E[x_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'\end{array}}{\mathcal{C} \vdash (E[e_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'}\ (\textsc{bind})$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash E_1 : \rho_1\tau \rightsquigarrow \rho_1'\tau' \quad \mathcal{C} \vdash E_2 : \rho_2\tau \rightsquigarrow \rho_2'\tau' \quad \mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho \\ \mathcal{C}, x_1, x_2, (x_1, x_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho, e_1 \rightsquigarrow^* x_1, x_2 \preceq e_2 \vdash (E_1[x_1], E_2[x_2]) \in \mathcal{E}\,[\![\tau']\!]\,\rho'\end{array}}{\mathcal{C} \vdash (E_1[e_1], E_2[e_2]) \in \mathcal{E}\,[\![\tau']\!]\,\rho'}\ (\textsc{bind}2)$$

$$\frac{\mathcal{C} \vdash (f_1, f_2) \in \mathcal{E}\,[\![\tau' \to \tau'']\!]\,\rho \quad \mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau']\!]\,\rho}{\mathcal{C} \vdash (f_1\,e_1, f_2\,e_2) \in \mathcal{E}\,[\![\tau'']\!]\,\rho}\ (\textsc{app})$$

$$\frac{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}\,[\![\mu\alpha.\,\tau]\!]\,\rho}{\mathcal{C} \vdash (\mathtt{unroll}\,e_1, \mathtt{unroll}\,e_2) \in \mathcal{E}\,[\![\tau[\mu\alpha.\,\tau/\alpha]]\!]\,\rho}\ (\textsc{unroll})$$

$$\frac{\begin{array}{c}\mathcal{C} \vdash e_1 \downarrow \rho_1(\tau' \to \tau'') \quad \mathcal{C} \vdash e_2 \downarrow \rho_2(\tau' \to \tau'') \\ \mathcal{C}, x_1, x_2, (x_1, x_2) \in \mathcal{V}\,[\![\tau']\!]\,\rho \vdash (e_1x_1, e_2x_2) \in \mathcal{E}\,[\![\tau'']\!]\,\rho\end{array}}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho}\ (\textsc{funext})$$

$$\frac{\begin{array}{c}F_i = \mathtt{fix}\,f(x_i).\,e_i \quad \mathcal{C} \vdash F_1 : \rho_1(\tau' \to \tau'') \quad \mathcal{C} \vdash F_2 : \rho_2(\tau' \to \tau'') \\ \triangleleft\mathcal{C}, x_1, x_2, (x_1, x_2) \in \mathcal{V}\,[\![\tau']\!]\,\rho, (F_1, F_2) \in \mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho \vdash (e_1[F_1/f], e_2[F_2/f]) \in \mathcal{E}\,[\![\tau'']\!]\,\rho\end{array}}{\mathcal{C} \vdash (F_1, F_2) \in \mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho}\ (\textsc{fix})$$

Figure 7: Some Useful Derivable Rules

4.4. **Derivable Rules.** Figure 7 shows a number of useful inference rules that are derivable in the logic. To be clear, by "derivable" we mean that the proofs of these rules' soundness (given below in Section 4.5) is done just using the inference rules we have established so far, without needing to appeal directly to the model and perform stepwise reasoning. In all these rules, we assume implicitly that all propositions are well-formed. For the rules concerning $\mathcal{V}\,[\![\tau]\!]\,\rho$ and $\mathcal{E}\,[\![\tau]\!]\,\rho$, we assume that ρ binds the free variables of τ and maps them to triples (τ_1, τ_2, R), where $R : \mathrm{VRel}(\tau_1, \tau_2)$ is provable in the ambient context.

Rule VAL says that $\mathcal{E}\,[\![\tau]\!]\,\rho$ contains $\mathcal{V}\,[\![\tau]\!]\,\rho$. This rule is so fundamental and ubiquitously useful that we will often elide mention of it in our proofs.

Rule WEAK-$\triangleright$ is a weakening property that is easy to derive from the distributivity laws for the $\triangleright$ operator. The rule employs an $\triangleleft$ operator (pronounced "earlier") on propositions/contexts, defined as follows:

$$\begin{aligned} \triangleleft(\mathcal{X};\mathcal{R};\overline{P}) &\stackrel{\text{def}}{=} \mathcal{X};\mathcal{R};\overline{\triangleleft P} \\ \triangleleft(\triangleright P) &\stackrel{\text{def}}{=} P \\ \triangleleft P &\stackrel{\text{def}}{=} P \quad (\text{if } P \neq \triangleright P') \end{aligned}$$

This $\triangleleft$ operator has the effect of "un-$\triangleright$-ing" (*i.e.,* stripping the $\triangleright$ off of) any $\triangleright P$ hypotheses in the context. Note that this is purely a shallow syntactic operation; it does not un-$\triangleright$ any hypotheses that are propositionally equivalent to some $\triangleright P$ but not syntactically of that form. (The reader may wonder why we define $\triangleleft$ in this syntactic way instead of building it in as a primitive modality with the seemingly natural interpretation $[\![\triangleleft P]\!]\delta n = [\![P]\!]\delta(n+1)$. The trouble is that this interpretation is not well-founded, since it defines the meaning of $\triangleleft P$ in terms of the meaning of P at a *higher* step level. And indeed, our syntactic $\triangleleft$ does not satisfy this interpretation.)

Consequently, Rule WEAK-$\triangleright$ says that if we want to show P is true later, given some assumptions that are true now, and others that are true later, then we can just prove that P is true now given that all the assumptions are true now. This is a weakening property because, applying the rule backwards, we forget the fact that some of the hypotheses in $\mathcal{C}$ (namely, those that are *not* of the form $\triangleright P$) are true at an earlier world than the others.

The WEAK-$\triangleright$ 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 EXP-$\triangleright$, explained below) to showing that two other terms e_1' and e_2' are related *later*. At that point, Rule WEAK-$\triangleright$ allows us to un-$\triangleright$ both our new proof goal (relatedness of e_1' and e_2') and our original LÖB-inductive hypothesis (relatedness of e_1 and e_2) simultaneously. We will see an instance of this proof pattern in the example in Section 5.2.

The next four rules in Figure 7 allow one to prove that two terms e_1 and e_2 are related by converting one of the terms to something else. Rule EXP (closure of the logical relation under expansion) allows one to reduce e_1 to some e_1' according to the $\rightsquigarrow^*$ relation and then show that e_1' is related to e_2. Rule RED (closure of the logical relation under $\rightsquigarrow^0$ reduction) allows one to expand e_1 to some e_1' according to the $\rightsquigarrow^0$ relation and then show that e_1' is related to e_2. Rule CIU allows one to replace e_2 with some e_2' that ciu-approximates it, and then show that e_1 is related to e_2'. Rule EXP-$\triangleright$ is similar to Rule EXP, but addresses the case when e_1 incurs an `unroll-roll` reduction on the way to e_1'. In this case, unfolding the definition of $\mathcal{E}\,[\![\tau]\!]\,\rho$, 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 BIND 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 assume that there are some values x_1 and x_2 such that e_1 evaluates to x_1, e_2 is ciu-approximated by x_2, and x_1

and x_2 are related by $\mathcal{V}[\![\tau]\!]\rho$, and the goal is reduced 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.

The BIND rule may seem at first glance a bit peculiar in that the term e_2 does not necessarily have any relationship to f, and the variable x_2 does not appear anywhere on the r.h.s. of the last premise. This peculiarity is a consequence of the rule being as general as possible. In the specific (if common) case that f is in fact of the form $E_2[e_2]$ (*i.e.,* that e_2 is in evaluation position in f), an easy corollary of Rules BIND and CIU is Rule BIND2. In addition to being more intuitive, this more symmetric-looking variant of the BIND rule is very useful in deriving *compatibility* properties [26], such as Rules APP and UNROLL; these compatibility properties are necessary in order to establish that the logical relation is a precongruence (and hence contained in contextual approximation), and Rule BIND2 helps to reduce the derivations of these properties to the case where the e's and f's are values. Rule BIND2 does not subsume Rule BIND, however: the general and distinctly asymmetric nature of the original Rule BIND renders it suitable for reasoning about logical approximation in cases where the more symmetric Rule BIND2 does not apply—for instance, see the proof of the "free theorem" example in Section 5.3.

Rule FUNEXT demonstrates a clean extensionality property for function values, which was one of our key motivations for LSLR in the first place. (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 $(x_1, x_2) \in \mathcal{V}[\![\tau']\!]\rho$ would have to be $\triangleright$'d. The key to our cleaner formulation is simply that we confine the use of $\triangleright$ in $\mathcal{V}[\![\tau]\!]\rho$ to the case when τ is a recursive type. Thus, in particular, one need not mention $\triangleright$ when reasoning purely about functions and β-reduction.

Finally, Rule FIX gives the rule for recursive functions, which are encodable in a well-known way in terms of recursive types. We formalize the encoding as follows:

$$\begin{array}{rcl} \texttt{fix}\, f(x).\, e & \stackrel{\text{def}}{=} & \lambda y.(\texttt{unroll}\, v)\, v\, y \\ \quad \text{where } v & = & \texttt{roll}\, (\lambda z.(\lambda f.\lambda x.e)(\lambda y.(\texttt{unroll}\, z)\, z\, y)) \\ & & \quad \text{for } y, z \notin \mathrm{FV}(e) \end{array}$$

This encoding 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 LÖB-inductively assume they are related while proving that their bodies are related. (For the proof that the bodies are related, we may also un-$\triangleright$ any other $\triangleright$ hypotheses in the ambient context $\mathcal{C}$.) The implicit use of LÖB induction in this rule gives it a distinctively coinductive flavor.

4.5. **Proofs of Derivability.** In this section, we show how to derive the rules in Figure 7.

Proposition 4.6 (Type Substitution)**.**
(1) $\mathcal{V}[\![\tau[\sigma/\alpha]]\!]\rho = \mathcal{V}[\![\tau]\!]\rho, \alpha \mapsto (\rho_1\sigma, \rho_2\sigma, \mathcal{V}[\![\sigma]\!]\rho)$.
(2) $\mathcal{E}[\![\tau[\sigma/\alpha]]\!]\rho = \mathcal{E}[\![\tau]\!]\rho, \alpha \mapsto (\rho_1\sigma, \rho_2\sigma, \mathcal{V}[\![\sigma]\!]\rho)$.

Proof. By straightforward induction on the structure of τ. □

Proposition 4.7. *Rule* VAL *is derivable.*

Proof. Immediate, since $\preceq$ is reflexive. □

Proposition 4.8. *Rule* WEAK-▷ *is derivable.*

Proof. Suppose $\mathcal{C} = \mathcal{X}; \mathcal{R}; \mathcal{P}$. Then, $\triangleleft\mathcal{C} \vdash P$ implies $\mathcal{X}; \mathcal{R}; \cdot \vdash (\bigwedge_{Q \in \mathcal{P}} \triangleleft Q) \Rightarrow P$. By Rule MONO and the distributivity axioms, $\mathcal{X}; \mathcal{R}; \cdot \vdash (\bigwedge_{Q \in \mathcal{P}} \triangleright\!\!\triangleleft Q) \Rightarrow \triangleright P$. Since $Q \Rightarrow \triangleright\!\!\triangleleft Q$, we have $\mathcal{X}; \mathcal{R}; \cdot \vdash (\bigwedge_{Q \in \mathcal{P}} Q) \Rightarrow \triangleright P$, and thus $\mathcal{C} \vdash \triangleright P$. □

Proposition 4.9. *Rule* RED *is derivable.*

Proof. First, suppose that $e_1 \Downarrow^0 x_1$ for some value x_1. Then, $e_1' \rightsquigarrow^0 e_1$ implies that $e_1' \Downarrow^0 x_1$ as well, and the rest follows immediately from $(e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$.

Second, suppose that $e_1 \rightsquigarrow^1 t_1$ for some term t_1. Then, $e_1' \rightsquigarrow^0 e_1$ implies that $e_1' \rightsquigarrow^1 t_1$ as well, so again the rest follows immediately from $(e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. □

Proposition 4.10. *Rule* EXP *is derivable given the additional premise that* $\mathcal{C} \vdash e_1 \rightsquigarrow^0 e_1'$.

Proof. The proof is very similar to the proof of Rule RED. The key bits are: (1) if $e_1 \Downarrow^0 x_1$ and $e_1 \rightsquigarrow^0 e_1'$, then $e_1' \Downarrow^0 x_1$ by determinacy of reduction, and (2) if $e_1 \rightsquigarrow^1 t_1$ and $e_1 \rightsquigarrow^0 e_1'$, then $e_1' \rightsquigarrow^1 t_1$, again by determinacy of reduction. □

Proposition 4.11. *Rule* EXP-▷ *is derivable.*

Proof. First, suppose that $e_1 \Downarrow^0 x_1$ for some value x_1. Then, $e_1 \rightsquigarrow^1 e_1'$ yields a contradiction.

Second, suppose that $e_1 \rightsquigarrow^1 t_1$ for some term t_1. Then, since $e_1 \rightsquigarrow^1 e_1'$, we have by determinacy of reduction that either $e_1' \rightsquigarrow^0 t_1$ or $t_1 \rightsquigarrow^0 e_1'$. Thus, by either Proposition 4.9 or 4.10, $\triangleright(e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$ implies $\triangleright(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$, which is what we needed to show. □

Proposition 4.12. *Rule* EXP *is derivable.*

Proof. Assume the premises of Rule EXP. We will prove the following proposition and then instantiate t_1 with e_1 to obtain the desired result.

$$\forall t_1.\ (t_1 : \rho_1\tau \wedge t_1 \rightsquigarrow^* e_1') \Rightarrow (t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$$

The proof is by LÖB *induction*, *i.e.,* we use the LÖB rule to assume the above proposition is true "later" (under a ▷ modality) and then prove it true "now". So assume $t_1 : \rho_1\tau$ and $t_1 \rightsquigarrow^* e_1'$, and we want to prove $(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. It is thus either the case that $t_1 \rightsquigarrow^0 e_1'$ or that there exists t_1' such that $t_1 \rightsquigarrow^1 t_1' \rightsquigarrow^* e_1'$. In the former case, the result follows by Proposition 4.10 and the assumption $(e_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. In the latter case we have, by the LÖB-inductive hypothesis (*i.e.,* the ▷-ed version of our original goal) together with the distributivity of ▷ over $\forall$ and $\Rightarrow$, that $\triangleright(t_1' : \rho_1\tau \wedge t_1' \rightsquigarrow^* e_1') \Rightarrow \triangleright(t_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. We already know that $t_1' \rightsquigarrow^* e_1'$, and $t_1' : \rho_1\tau$ follows by type preservation, so by Rule MONO, we have that $\triangleright(t_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. The result then follows from $t_1 \rightsquigarrow^1 t_1'$ and Rule EXP-▷. □

Proposition 4.13. *Rule* CIU *is derivable.*

Proof. As for Rule EXP, the proof here is by LÖB induction. Given the premises of Rule CIU, we prove the following and then instantiate t_1 to e_1:

$$\forall t_1.\ (t_1, e_2') \in \mathcal{E}\,[\![\tau]\!]\,\rho \Rightarrow (t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$$

Assume this is true later, and we proceed to prove it now. So assume $(t_1, e_2') \in \mathcal{E}\,[\![\tau]\!]\,\rho$, and we want to prove $(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$.

First, suppose $t_1 \Downarrow^0 x_1$. Then, there exists x_2 such that $(x_1, x_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho$ and $x_2 \preceq e_2'$. Since by assumption $e_2' \preceq e_2$ and $\preceq$ is transitive, we have that $x_2 \preceq e_2$, so we are done.

Second, suppose $t_1 \rightsquigarrow^1 t_1'$. Then, $\triangleright(t_1', e_2') \in \mathcal{E}\,[\![\tau]\!]\,\rho$, so by the LÖB-inductive hypothesis, $\triangleright(t_1', e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. □

Proposition 4.14. *Rule* BIND *is derivable.*

Proof. Define $P(t_1)$ to be the proposition:

$$\forall x_1, x_2.\ ((x_1, x_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho \wedge t_1 \rightsquigarrow^* x_1 \wedge x_2 \preceq e_2) \Rightarrow (E[x_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$$

We want to prove that

$$\forall t_1.\ ((t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho \wedge P(t_1)) \Rightarrow (E[t_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$$

By the LÖB rule, we assume this proposition is true later and proceed to prove it now. So assume that $(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$ and $P(t_1)$, and we want to prove $(E[t_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$.

First, suppose that $E[t_1] \Downarrow^0 x_1$ for some x_1. Then, it must be the case that $t_1 \Downarrow^0 y_1$ for some y_1, and also that $E[t_1] \rightsquigarrow^0 E[y_1] \Downarrow^0 x_1$. Since $(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$, we know there exists some y_2 such that $y_2 \preceq e_2$ and $(y_1, y_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho$. Thus, by $P(t_1)$, we know that $(E[y_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$. Then, by Rule EXP, $(E[t_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$.

Second, suppose that $E[t_1] \rightsquigarrow^1 t_1'$. There are two cases:

Case 1:

There exists y_1 such that $t_1 \Downarrow^0 y_1$, and also that $E[t_1] \rightsquigarrow^0 E[y_1] \rightsquigarrow^1 t_1'$. The proof is identical to the previous case shown above.

Case 2:

There exists u_1 such that $t_1 \rightsquigarrow^1 u_1$, and also that $E[t_1] \rightsquigarrow^1 E[u_1] \rightsquigarrow^0 t_1'$. Since $(t_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$, we know that $\triangleright(u_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$. Also, it is easy to show that $P(t_1)$ implies $P(u_1)$. Thus, by appealing to our LÖB-inductive hypothesis, we have that $\triangleright(E[u_1], f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$. Finally, by Rule RED, $\triangleright(t_1', f) \in \mathcal{E}\,[\![\tau']\!]\,\rho'$. □

Proposition 4.15. *Rule* BIND2 *is derivable.*

Proof. By Rules BIND and CIU, together with the fact that $x_2 \preceq e_2$ implies $E_2[x_2] \preceq E_2[e_2]$, by part (4) of Proposition 4.4. □

Proposition 4.16. *Rule* APP *is derivable.*

Proof. By Rule BIND2, using evaluation contexts $[\cdot]\,e_1$ and $[\cdot]\,e_2$, the goal reduces to showing that $(x_1\,e_1, x_2\,e_2) \in \mathcal{E}\,[\![\tau'']\!]\,\rho$ under the assumption that $(x_1, x_2) \in \mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho$. By Rule BIND2 again, this time using evaluation contexts $x_1\,[\cdot]$ and $x_2\,[\cdot]$, the goal reduces to showing that $(x_1\,y_1, x_2\,y_2) \in \mathcal{E}\,[\![\tau'']\!]\,\rho$ under the assumption that $(y_1, y_2) \in \mathcal{V}\,[\![\tau']\!]\,\rho$. The result then follows by unrolling the definition of $\mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho$. □

Proposition 4.17. *Rule* UNROLL *is derivable.*

Proof. By Rule BIND2, using the evaluation context $\texttt{unroll}\,[\cdot]$ on both sides, the goal reduces to showing that $(\texttt{unroll}\,x_1, \texttt{unroll}\,x_2) \in \mathcal{E}\,[\![\tau[\mu\alpha.\tau/\alpha]]\!]\,\rho$ under the assumption that $(x_1, x_2) \in \mathcal{V}\,[\![\mu\alpha.\tau]\!]\,\rho$. Unrolling the definition of $\mathcal{V}\,[\![\mu\alpha.\tau]\!]\,\rho$, we have that $x_1 = \texttt{roll}\,y_1$, $x_2 = \texttt{roll}\,y_2$, and $\triangleright(y_1, y_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho, \alpha \mapsto \mathcal{V}\,[\![\mu\alpha.\tau]\!]\,\rho$ for some y_1 and y_2. By Proposition 4.6, $\triangleright(y_1, y_2) \in \mathcal{V}\,[\![\tau[\mu\alpha.\tau/\alpha]]\!]\,\rho$. Also, we have that $\texttt{unroll}\,x_i = \texttt{unroll}\,(\texttt{roll}\,y_i) \rightsquigarrow^1 y_i$ (and thus $y_i \preceq \texttt{unroll}\,x_i$ as well). Thus, the desired result follows directly by Rule EXP-▷ and Rule CIU. □

Proposition 4.18. *Rule* FUNEXT *is derivable.*

Proof. Immediate, by unfolding the definition of $\mathcal{V}\,[\![\tau' \to \tau'']\!]\,\rho$. □

Proposition 4.19. *Rule* FIX *is derivable.*

Proof. By straightforward combination of Rules LÖB, FUNEXT, WEAK-▷, EXP-▷, and CIU, given the fact that $F_i\, x_i \rightsquigarrow^1 e_i[F_i/f]$. □

4.6. **Soundness and Completeness of the Logical Relation.** We now state some key theorems concerning the logical relation, the primary ones being that it is sound and complete w.r.t. contextual approximation.

Definition 4.20 (Logical Approximation)**.** Let $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$.
Suppose $\Gamma = \alpha_1, \ldots, \alpha_n, x_1 : \tau_1, \ldots, x_m : \tau_m$. Let

$$\begin{aligned}
\mathcal{X} &= \alpha_1^1, \alpha_1^2, \ldots, \alpha_n^1, \alpha_n^2, x_1^1, x_1^2, \ldots, x_m^1, x_m^2 \\
\mathcal{R} &= r_1, \ldots, r_n \\
\rho &= \alpha_1 \mapsto (\alpha_1^1, \alpha_1^2, r_1), \ldots, \alpha_n \mapsto (\alpha_n^1, \alpha_n^2, r_n) \\
\mathcal{P} &= r_1 : \mathrm{VRel}(\alpha_1^1, \alpha_1^2), \ldots, r_n : \mathrm{VRel}(\alpha_n^1, \alpha_n^2), \\
&\quad\ (x_1^1, x_1^2) \in \mathcal{V}\,[\![\tau_1]\!]\,\rho, \ldots, (x_m^1, x_m^2) \in \mathcal{V}\,[\![\tau_m]\!]\,\rho \\
\gamma_j &= x_1 \mapsto x_1^j, \ldots, x_m \mapsto x_m^j \text{ (where } j \in \{1, 2\})
\end{aligned}$$

Then

$$\Gamma \vdash e_1 \preceq^{log} e_2 : \tau \ \stackrel{\text{def}}{=}\ \mathcal{X}; \mathcal{R}; \mathcal{P} \vdash (\rho_1\gamma_1 e_1, \rho_2\gamma_2 e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho$$

Theorem 4.21 (Fundamental Theorem of Logical Relations)**.**
If $\Gamma \vdash e : \tau$ *then* $\Gamma \vdash e \preceq^{log} e : \tau$.

Proof. By induction on typing derivations. In the case when e is a variable, the goal follows directly from the hypotheses $\mathcal{P}$ in Definition 4.20. All of the other cases follow immediately from the compatibility rules, which are all completely straightforward to prove (in the style of Rule APP). The only slightly interesting compatibility rule is Rule UNROLL, which we proved in Section 4.5. □

Theorem 4.22 (Adequacy)**.**
If $\vdash (e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]$ *and* $e_1 \Downarrow$, *then* $e_2 \Downarrow$.

Proof. Suppose $e_1 \Downarrow v_1$. Let n be the number of `unroll-roll` reductions that occur in the evaluation of e_1 to v_1. It is easy to show by induction on n, and by unfolding the definition of $\mathcal{E}\,[\![\tau]\!]$, that $\vdash \triangleright^n(v_1, e_2) \in \mathcal{E}\,[\![\tau]\!]$ (where $\triangleright^n$ denotes n applications of the $\triangleright$ modality). Thus, $\vdash \triangleright^n(\exists x_2 \downarrow \tau.\ x_2 \preceq e_2)$.

Appealing to the model, we have that $\forall k \geq 0.\ [\![\triangleright^n(\exists x_2 \downarrow \tau.\ x_2 \preceq e_2)]\!]k$. Choosing $k > n$, this means that there exists a value $v_2 : \tau$ such that $v_2 \preceq^{ciu} e_2$. Hence, $e_2 \Downarrow$. □

Theorem 4.23 (Logical Approximation ⇒ Contextual Approximation)**.**
If $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$, *then* $\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$.

Proof. Given a context $C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')$, we show that $\Gamma' \vdash C[e_1] \preceq^{log} C[e_2] : \tau'$. The proof of this part is by induction on the context C, and as in the proof of the Fundamental Theorem, all of the cases follow immediately from the compatibility rules. Thus, if Γ' is empty, we know that $\vdash (C[e_1], C[e_2]) \in \mathcal{E}\,[\![\tau']\!]$. Consequently, by Adequacy, we know that $C[e_1] \Downarrow$ implies $C[e_2] \Downarrow$. □

Theorem 4.24 (Ciu-Transitivity of the Logical Relation)**.**
If $\Gamma \vdash e_1 \preceq^{log} e_2' : \tau$ *and* $\Gamma \vdash e_2' \preceq^{ciu} e_2 : \tau$*, then* $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$*.*

Proof. Let $\mathcal{X}$, $\mathcal{R}$, $\mathcal{P}$, ρ, and γ_j be as defined in Definition 4.20. From the second assumption, it is easy to show by appeal to the model that $\mathcal{X}; \mathcal{R}; \mathcal{P} \vdash \rho_2\gamma_2 e_2' \preceq \rho_2\gamma_2 e_2$. Thus, the result follows immediately by Rule CIU. □

Theorem 4.25 (Ciu Approximation ⇒ Logical Approximation)**.**
If $\Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau$*, then* $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$*.*

Proof. By the Fundamental Theorem of Logical Relations, $\Gamma \vdash e_1 \preceq^{log} e_1 : \tau$. The result then follows directly by Theorem 4.24. □

Corollary 4.26 ($\preceq^{ctx} \equiv \preceq^{ciu} \equiv \preceq^{log}$)**.**
$\Gamma \vdash e_1 \preceq^{ctx} e_2 : \tau$ *iff* $\Gamma \vdash e_1 \preceq^{ciu} e_2 : \tau$ *iff* $\Gamma \vdash e_1 \preceq^{log} e_2 : \tau$*.*

Proof. By Theorems 4.5, 4.23 and 4.25. □

4.7. **Symmetric Version of the Logical Relation.** We have shown that our logical relation supports sound *inequational* reasoning about contextual approximation, 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 binary term relation R, let R^{op} denote $(t_2, t_1).(t_1, t_2) \in R$. 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 $\mathcal{E}\,[\![\tau]\!]$) 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 are equivalent themselves; to show equivalence, we would need to establish relatedness of f_1 and f_2 in both directions, which would at a minimum require that they map $\mathcal{V}\,[\![\tau']\!]$-related arguments (which are not necessarily equivalent) to $\mathcal{E}\,[\![\tau'']\!]$-related results. Merely knowing how f_1 and f_2 behave on *equivalent* arguments is not enough to establish that.

Thus, instead, we define the symmetric relation as shown in Figure 8. Here, d is a value variable of type `bool` that we assume is bound in the context in which 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 6, replace all occurrences of $\mathcal{V}\,[\![\tau]\!]\,\rho$

$$
\begin{array}{lcl}
\mathcal{V}^{\approx}[\![\tau]\!]\,\rho & \stackrel{\text{def}}{=} & (t_1 : \rho_1\tau, t_2 : \rho_2\tau).\\
& & \quad (d = \texttt{true} \Rightarrow (t_1, t_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho) \wedge\\
& & \quad (d = \texttt{false} \Rightarrow (t_2, t_1) \in \mathcal{V}\,[\![\tau]\!]\,\rho^{\mathrm{op}})\\
\mathcal{E}^{\approx}[\![\tau]\!]\,\rho & \stackrel{\text{def}}{=} & (t_1 : \rho_1\tau, t_2 : \rho_2\tau).\\
& & \quad (d = \texttt{true} \Rightarrow (t_1, t_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho) \wedge\\
& & \quad (d = \texttt{false} \Rightarrow (t_2, t_1) \in \mathcal{E}\,[\![\tau]\!]\,\rho^{\mathrm{op}})\\
e_1 \preceq_1 e_2 & \stackrel{\text{def}}{=} & (d = \texttt{true} \Rightarrow e_2 \rightsquigarrow^* e_1) \wedge\\
& & (d = \texttt{false} \Rightarrow e_1 \preceq e_2)\\
e_1 \preceq_2 e_2 & \stackrel{\text{def}}{=} & (d = \texttt{true} \Rightarrow e_1 \preceq e_2) \wedge\\
& & (d = \texttt{false} \Rightarrow e_2 \rightsquigarrow^* e_1)
\end{array}
$$

Figure 8: Symmetric Version of the F^{μ} Logical Relation and Related Definitions

and $\mathcal{E}\,[\![\tau]\!]\,\rho$ with their symmetric versions, and substitute $\equiv$ for $\stackrel{\text{def}}{=}$, we have a set of valid relational equivalences. The same goes for the relational equivalences in Proposition 4.6. (The same is not true, however, for the definition of $\mathcal{E}\,[\![\tau]\!]\,\rho$, because it is inherently asymmetric.)

The proofs of these symmetric relational equivalences are all quite easy—each one splits into two cases, one for $d = \texttt{true}$ and one for $d = \texttt{false}$. Here, we sketch the proof for the recursive type case, which is the most interesting since it uses the LÖB rule.

Proposition 4.27. $\mathcal{V}^{\approx}[\![\mu\alpha.\,\tau]\!]\,\rho \equiv \mu r.(x_1 \downarrow \rho_1(\mu\alpha.\,\tau), x_2 \downarrow \rho_2(\mu\alpha.\,\tau)).$
$\exists y_1, y_2.\ x_1 = \texttt{roll}\, y_1\ \wedge\ x_2 = \texttt{roll}\, y_2\ \wedge$
$\quad \triangleright(y_1, y_2) \in \mathcal{V}^{\approx}[\![\tau]\!]\,\rho, \alpha \mapsto (\rho_1(\mu\alpha.\,\tau), \rho_2(\mu\alpha.\,\tau), r)$

Proof. Let R_1 and R_2 denote the relations on the left and right sides of the equivalence, respectively. By the LÖB rule, we can assume that $\triangleright(R_1 \equiv R_2)$. By Canonical Forms, either $d = \texttt{true}$ or $d = \texttt{false}$:

Case $d = \texttt{true}$**:**
Unrolling definitions, the proof reduces to showing that $\triangleright(y_1, y_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho, \alpha \mapsto (\ldots, R_1)$ iff $\triangleright(y_1, y_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho, \alpha \mapsto (\ldots, R_2)$. This follows from the basic axioms together with the LÖB-inductive hypothesis $\triangleright(R_1 \equiv R_2)$.

Case $d = \texttt{false}$**:**
Similarly, the proof reduces to showing that $\triangleright(y_2, y_1) \in \mathcal{V}\,[\![\tau]\!]\,\rho^{\mathrm{op}}, \alpha \mapsto (\ldots, {R_1}^{\mathrm{op}})$ iff $\triangleright(y_2, y_1) \in \mathcal{V}\,[\![\tau]\!]\,\rho^{\mathrm{op}}, \alpha \mapsto (\ldots, {R_2}^{\mathrm{op}})$. Again, this follows from the basic axioms together with the LÖB-inductive hypothesis $\triangleright(R_1 \equiv R_2)$. □

Furthermore, we can easily derive symmetric versions of most of our derived rules. In most cases, including all the compatibility properties, 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 the rules from EXP to BIND2 in Figure 7. In Figure 9, we give symmetric versions of several of these, the last two of which employ the relations $e_1 \preceq_1 e_2$ and $e_1 \preceq_2 e_2$ defined in Figure 8. These relations are merely a technical device to enable a symmetric presentation of certain premises that have the form $e_2 \rightsquigarrow^* e_1$ for one direction of approximation and $e_1 \preceq e_2$ for the other direction. The proofs of these rules are all completely straightforward, relying heavily on the fact (from Proposition 4.4) that $e_1 \rightsquigarrow^* e_2$ implies $e_1 \approx^{ciu} e_2$. (Note

$$\frac{\mathcal{C} \vdash e_1 : \rho_1\tau \quad \mathcal{C} \vdash e_2 : \rho_2\tau \quad \mathcal{C} \vdash e_1 \rightsquigarrow^* e_1' \quad \mathcal{C} \vdash e_2 \rightsquigarrow^* e_2' \quad \mathcal{C} \vdash (e_1', e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho} \text{ (SYM-EXP)}$$

$$\frac{\mathcal{C} \vdash e_1 : \rho_1\tau \quad \mathcal{C} \vdash e_2 : \rho_2\tau \quad \mathcal{C} \vdash e_1 \rightsquigarrow^1 e_1' \quad \mathcal{C} \vdash e_2 \rightsquigarrow^1 e_2' \quad \mathcal{C} \vdash \triangleright(e_1', e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho} \text{ (SYM-EXP-}\triangleright\text{)}$$

$$\frac{\mathcal{C} \vdash e_1' \rightsquigarrow^0 e_1 \quad \mathcal{C} \vdash e_2' \rightsquigarrow^0 e_2 \quad \mathcal{C} \vdash (e_1', e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho} \text{ (SYM-RED)}$$

$$\frac{\mathcal{C} \vdash e_1 : \rho_1\tau \quad \mathcal{C} \vdash e_2 : \rho_2\tau \quad \mathcal{C} \vdash (e_1', e_2') \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho \quad \mathcal{C} \vdash e_1' \preceq_1 e_1 \quad \mathcal{C} \vdash e_2' \preceq_2 e_2}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho} \text{ (SYM-CIU)}$$

$$\frac{\mathcal{C} \vdash E_1 : \rho_1\tau \rightsquigarrow \rho_1'\tau' \quad \mathcal{C} \vdash E_2 : \rho_2\tau \rightsquigarrow \rho_2'\tau' \quad \mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,\rho \quad \mathcal{C}, x_1, x_2, (x_1, x_2) \in \mathcal{V}^{\approx}[\![\tau]\!]\,\rho, x_1 \preceq_1 e_1, x_2 \preceq_2 e_2 \vdash (E_1[x_1], E_2[x_2]) \in \mathcal{E}^{\approx}[\![\tau']\!]\,\rho'}{\mathcal{C} \vdash (E_1[e_1], E_2[e_2]) \in \mathcal{E}^{\approx}[\![\tau']\!]\,\rho'} \text{ (SYM-BIND)}$$

Figure 9: Symmetric Versions of Several Derivable Rules

that the context $\mathcal{C}$ appearing in all these rules is assumed to bind d in its variable context and contain $d \downarrow \mathsf{bool}$ in its proposition context.)

To give the reader a concrete sense of how these rules work, we present in the next section three detailed examples of how to use them to prove contextual equivalences.

Finally, since LSLR is inspired by Plotkin and Abadi's logic for parametricity, one might expect to see some rule corresponding to "identity extension." Denoting contextual equivalence at type σ by $\approx_\sigma^{ctx}$, identity extension would mean that, for any open type $\alpha \vdash \tau$, we would have that $\mathcal{E}^{\approx}[\![\tau]\!]\,(\alpha \mapsto \approx_\sigma^{ctx})$ equals $\approx_{\tau[\sigma/\alpha]}^{ctx}$. In fact, we do not have such a rule since, as we 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 any 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 $\{(e_1, e_2) \mid \forall n.\, [\![(e_1, e_2) \in \mathcal{E}^{\approx}[\![\tau]\!]\,]\!]n\}$. The identity extension lemma has traditionally been used to prove representation independence results, aka *free theorems* [35], and, for pure calculi, definability results for types [30]. In spite of the lack of identity extension we are still able to prove some free theorems, as we demonstrate in Section 5.3.

5. Examples

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

The first example is from Crary and Harper [13] (who adapted it from one in Sumii and Pierce [34]) and concerns representation independence of "objects" with existential recursive type. The second, from Sumii and Pierce [34], is concerned with proving the syntactic minimal invariant property associated with a general recursive type [27, 10, 13]. The third is a canonical example of a Wadler-style "free theorem" [35].

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 mention of step indices!

5.1. **Flag Objects.** 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.

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

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

$$\begin{array}{rcl} \mathtt{bflag} & = & \mathtt{pack}\,\mathsf{bool}, (\mathtt{roll}\,\langle \mathtt{true}, \langle \mathtt{bflip}, \mathtt{bret}\rangle\rangle)\,\mathtt{as}\,\mathsf{flag} \\ \mathtt{bflip} & = & \lambda x : \mathsf{fld}_{\mathsf{bool}}.\,\mathtt{roll}\,\langle \mathtt{not}\,(\mathtt{fst}\,(\mathtt{unroll}\,x)), \mathtt{snd}\,(\mathtt{unroll}\,x)\rangle \\ \mathtt{bret} & = & \lambda x : \mathsf{fld}_{\mathsf{bool}}.\,\mathtt{fst}\,(\mathtt{unroll}\,x) \end{array}$$

$$\begin{array}{rcl} \mathtt{iflag} & = & \mathtt{pack}\,\mathsf{int}, (\mathtt{roll}\,\langle 0, \langle \mathtt{iflip}, \mathtt{iret}\rangle\rangle)\,\mathtt{as}\,\mathsf{flag} \\ \mathtt{iflip} & = & \lambda x : \mathsf{fld}_{\mathsf{int}}.\,\mathtt{roll}\,\langle 1 + (\mathtt{fst}\,(\mathtt{unroll}\,x)), \mathtt{snd}\,(\mathtt{unroll}\,x)\rangle \\ \mathtt{iret} & = & \lambda x : \mathsf{fld}_{\mathsf{int}}.\,\mathtt{even}\,(\mathtt{fst}\,(\mathtt{unroll}\,x)) \end{array}$$

To prove equivalence of $\mathtt{bflag}$ and $\mathtt{iflag}$, it suffices to show $d, d \downarrow \mathsf{bool} \vdash (\mathtt{bflag}, \mathtt{iflag}) \in \mathcal{E}^{\approx}[\![\mathsf{flag}]\!]$. Equivalently, by Rule VAL, since both terms are values, it is enough to show that $d, d \downarrow \mathsf{bool} \vdash (\mathtt{bflag}, \mathtt{iflag}) \in \mathcal{V}^{\approx}[\![\mathsf{flag}]\!]$. Unfolding the definition of $\mathcal{V}^{\approx}[\![\exists\alpha.\,\mathsf{fld}_\alpha]\!]$, we choose $\alpha_1 \mapsto \mathsf{bool}$, $\alpha_2 \mapsto \mathsf{int}$, $y_1 \mapsto v_1$, $y_2 \mapsto v_2$, and $r \mapsto R$ as the substitution for its existentially-bound variables, where $v_1 = \mathtt{roll}\,\langle \mathtt{true}, \langle \mathtt{bflip}, \mathtt{bret}\rangle\rangle$, $v_2 = \mathtt{roll}\,\langle 0, \langle \mathtt{iflip}, \mathtt{iret}\rangle\rangle$, and

$$R = (x_1 \downarrow \mathsf{bool}, x_2 \downarrow \mathsf{int}).\,\exists y \downarrow \mathsf{int}.\,(x_1 = \mathtt{true} \wedge 2y \Downarrow x_2) \vee (x_1 = \mathtt{false} \wedge 2y{+}1 \Downarrow x_2)$$

Let $\rho = \alpha \mapsto (\mathsf{bool}, \mathsf{int}, R)$. 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 VAL):

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

(2) Show $(\mathtt{bflip}, \mathtt{iflip}) \in \mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha \to \mathsf{fld}_\alpha]\!]\,\rho$. By the compatibility rule for functions, we assume that $(x_1, x_2) \in \mathcal{V}^{\approx}[\![\mathsf{fld}_\alpha]\!]\,\rho$, and are required to show:

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

By compatibility, we have that $(\mathtt{fst}\,(\mathtt{unroll}\,x_1), \mathtt{fst}\,(\mathtt{unroll}\,x_2)) \in \mathcal{E}^{\approx}[\![\alpha]\!]\rho$. Thus, by Rule SYM-BIND, we can assume that $(z_1, z_2) \in \mathcal{V}^{\approx}[\![\alpha]\!]\,\rho \equiv R$ for some z_1 and z_2, and the proof reduces to showing

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

By compatibility again, this reduces to showing that $(\mathtt{not}\,z_1, 1 + z_2) \in \mathcal{E}^{\approx}[\![\alpha]\!]\,\rho$. By Rule SYM-EXP, it simply remains to show that $\mathtt{not}\,z_1$ and $1 + z_2$ evaluate to values that are related by R. The following lemma suffices:

$$\forall z_1, z_2.\,(z_1, z_2) \in R \Rightarrow \exists z_1', z_2'.\,\mathtt{not}\,z_1 \Downarrow z_1' \wedge 1 + z_2 \Downarrow z_2' \wedge (z_1', z_2') \in R$$

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.

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

$$\forall z_1, z_2.\, (z_1, z_2) \in R \Rightarrow \texttt{even}\, z_2 \Downarrow z_1$$

□

5.2. **Syntactic Minimal Invariance.** The proof of our next example relies on Canonical Forms, a first-order lemma about F^μ that we assume is proven outside LSLR by traditional means. This standard lemma, which characterizes the shape of well-typed values, is only available to us because (following Pitts [26]) we have constructed the logical relation from syntactically well-typed terms. For further discussion of this point, see Section 7.

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

$$\begin{array}{l} v = \texttt{fix}\, f(x\!:\!\tau).\, \texttt{case}\,(\texttt{unroll}\, x)\,\texttt{of}\,\texttt{inl}\,\_ \Rightarrow \texttt{roll}\,(\texttt{inl}\,\langle\rangle) \\ \qquad\qquad\qquad\qquad \mid \texttt{inr}\, g \Rightarrow \texttt{roll}\,(\texttt{inr}\,(\lambda y\!:\!\tau.\, f(g(f\, y)))) \end{array}$$

This corresponds to the *minimal invariant* property in the domain-theoretic work of Pitts [27], which Birkedal and Harper subsequently proved in an operational setting [10].

To prove contextual equivalence of $\texttt{id}$ and v, we can show $d, d \downarrow \mathsf{bool} \vdash (\texttt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$. Our proof will be parametric in d. By the LÖB rule, we assume $\triangleright(\texttt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$ and proceed to prove $(\texttt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$. Now, by (the symmetric version of) Rule FUNEXT and SYM-EXP, we assume $(x_1, x_2) \in \mathcal{V}^{\approx}[\![\tau]\!]$, and it suffices to show

$$\begin{array}{l} (x_1, \texttt{case}\,(\texttt{unroll}\, x_2)\,\texttt{of}\,\texttt{inl}\,\_ \Rightarrow \texttt{roll}\,(\texttt{inl}\,\langle\rangle) \\ \qquad\qquad \mid \texttt{inr}\, g \Rightarrow \texttt{roll}\,(\texttt{inr}\,(\lambda y\!:\!\tau.\, v(g(v\, y))))) \in \mathcal{E}^{\approx}[\![\tau]\!] \end{array}$$

By relatedness of x_1 and x_2, we know that there exist y_1 and y_2 such that $x_1 = \texttt{roll}\, y_1$, $x_2 = \texttt{roll}\, y_2$, and $\triangleright(y_1, y_2) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$. By Canonical Forms, since $y_2 \downarrow \mathsf{unit} + (\tau \to \tau)$, we know that either $y_2 = \texttt{inl}\,\langle\rangle$ or there exists y_2' such that $y_2 = \texttt{inr}\, y_2'$. In either case, there exists $z \downarrow \mathsf{unit} + (\tau \to \tau)$ such that the $\texttt{case}$ expression above evaluates to $\texttt{roll}\, z$. Consequently, by Rule SYM-EXP, the goal reduces to showing

$$(\texttt{roll}\, y_1, \texttt{roll}\, z) \in \mathcal{V}^{\approx}[\![\mu\alpha.\, \mathsf{unit} + (\alpha \to \alpha)]\!]$$

Unfolding the definition of $\mathcal{V}^{\approx}[\![\mu\alpha.\, \mathsf{unit} + (\alpha \to \alpha)]\!]$, it suffices by Rule WEAK-$\triangleright$ to show

$$(y_1, z) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$$

under a strengthened (*i.e.,* $\triangleleft$'d) context in which the $\triangleright$ has been removed from any of our previous assumptions. In particular, we may now assume our LÖB-inductive hypothesis $(\texttt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, as well as $(y_1, y_2) \in \mathcal{V}^{\approx}[\![\mathsf{unit} + (\tau \to \tau)]\!]$, to hold "now" as opposed to "later". The latter assumption yields two cases:

Case $\texttt{inl}$**:**

$y_1 = y_2 = z = \texttt{inl}\,\langle\rangle$. Trivial.

Case $\texttt{inr}$**:**

$y_1 = \texttt{inr}\, y_1'$, $y_2 = \texttt{inr}\, y_2'$, $(y_1', y_2') \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, and $z = \texttt{inr}\,(\lambda y\!:\!\tau.\, v\,(y_2'\,(v\, y)))$. Thus, 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 FUNEXT (in its symmetric form) and Rule SYM-EXP, we assume $(z_1, z_2) \in \mathcal{V}^{\approx}[\![\tau]\!]$, 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 APP and SYM-RED that $(z_1, v\, z_2) \in \mathcal{E}^{\approx}[\![\tau]\!]$. By relatedness of y_1' and y_2' and Rule APP, we have that $(y_1'\, z_1, (y_2'\,(v\, z_2))) \in \mathcal{E}^{\approx}[\![\tau]\!]$. Thus, by Rule SYM-BIND, choosing as the evaluation contexts of interest $[\cdot]$ and $v\,[\cdot]$, our goal reduces to showing that for any z_1', z_2', if $(z_1', z_2') \in \mathcal{V}^{\approx}[\![\tau]\!]$, then $(z_1', v\, z_2') \in \mathcal{E}^{\approx}[\![\tau]\!]$. As before, this follows from $(\mathtt{id}, v) \in \mathcal{V}^{\approx}[\![\tau \to \tau]\!]$, together with Rules APP and SYM-RED. □

5.3. **A "Free Theorem".** Suppose that τ and σ are closed types, that h and f are values such that $h : \forall\alpha.\, \alpha \to \alpha \to \alpha$ and $f : \tau \to \sigma$, and that v and w are values of type τ. We will prove that $h\,\sigma\,(f\, v)\,(f\, w)$ contextually approximates $f\,(h\,\tau\, v\, w)$ unconditionally, and that the reverse approximation also holds if f is total (a sufficient, but not necessary, condition), defined as $\mathsf{total}(f) \stackrel{\text{def}}{=} \forall x.\ x \downarrow \tau \Rightarrow \exists y.\ f\, x \Downarrow y$.

The proof is interesting in that it is mostly done in a symmetric fashion, except for one inner lemma, which requires us to split into cases, one for each asymmetric direction of approximation. Since one of the two directions includes an extra assumption concerning the totality of f, we will actually prove the theorem

$$\mathcal{C} \vdash (h\,\sigma\,(f\, v)\,(f\, w),\ f\,(h\,\tau\, v\, w)) \in \mathcal{E}^{\approx}[\![\sigma]\!]$$

where $\mathcal{C} = d, d \downarrow \mathsf{bool}, d = \mathtt{false} \Rightarrow \mathsf{total}(f)$. To prove the theorem, we use Rule SYM-BIND with the evaluation contexts $[\cdot]$ and $f\,[\cdot]$, respectively. The proof is in two parts.

Part 1 First, we prove that

$$(h\,\sigma\,(f\, v)\,(f\, w), h\,\tau\, v\, w) \in \mathcal{E}^{\approx}[\![\alpha]\!]\,\rho$$

where $\rho = \alpha \mapsto (\sigma, \tau, R)$ and

$$R = (y_1 \downarrow \sigma, y_2 \downarrow \tau).\ (y_1, f\, y_2) \in \mathcal{E}^{\approx}[\![\sigma]\!]$$

By Theorem 4.21, $(h, h) \in \mathcal{E}^{\approx}[\![\forall\alpha.\, \alpha \to \alpha \to \alpha]\!]$. Thus, $(h\,\sigma, h\,\tau) \in \mathcal{E}^{\approx}[\![\alpha \to \alpha \to \alpha]\!]\,\rho$. To prove our desired result (by Rule APP), it remains to show that $(f\, v, v) \in \mathcal{E}^{\approx}[\![\alpha]\!]\,\rho$ and $(f\, w, w) \in \mathcal{E}^{\approx}[\![\alpha]\!]\,\rho$. We show the proof for the former; the latter is exactly the same.

This is the place where we need to split into cases depending on the direction of the proof. Both cases use the fact, due to the Fundamental Theorem, that $(f\, v, f\, v) \in \mathcal{E}^{\approx}[\![\sigma]\!]$.

Case $d = \mathtt{true}$**:**

We need to show $(f\, v, v) \in \mathcal{E}\,[\![\alpha]\!]\,\rho$. Since $(f\, v, f\, v) \in \mathcal{E}\,[\![\sigma]\!]$, by Rule BIND (using evaluation context $[\cdot]$) and Rule VAL we may assume that there exist x_1, x_2 such that $(x_1, x_2) \in \mathcal{V}\,[\![\sigma]\!]$ and $x_2 \preceq f\, v$, and it remains to show $(x_1, v) \in \mathcal{V}\,[\![\alpha]\!]\,\rho = R$. The latter is equivalent to $(x_1, f\, v) \in \mathcal{E}\,[\![\sigma]\!]$, which follows directly from the assumptions by Rule CIU.

Case $d = \mathtt{false}$**:**

We need to show $(v, f\, v) \in \mathcal{E}\,[\![\alpha]\!]\,\rho^{\mathrm{op}}$. Using the assumption $\mathsf{total}(f)$, which is available since $d = \mathtt{false}$, we know that there exists $x \downarrow \sigma$ such that $f\, v \Downarrow x$. Thus, $f\, v \preceq x$ and $x \preceq f\, v$. By Rules CIU and VAL, it suffices to show $(v, x) \in \mathcal{V}\,[\![\alpha]\!]\,\rho^{\mathrm{op}} = R^{\mathrm{op}}$. Unrolling

the definition of R, we see that the goal is equivalent to $(f\,v, x) \in \mathcal{E}\,[\![\sigma]\!]$, which follows from $(f\,v, f\,v) \in \mathcal{E}\,[\![\sigma]\!]$ and $f\,v \preceq x$ by Rule CIU.

Part 2 Next, we assume that $(z_1, z_2) \in \mathcal{V}^{\approx}[\![\alpha]\!]\,\rho \equiv R$ and we need to show that

$$(z_1, f\,z_2) \in \mathcal{E}^{\approx}[\![\sigma]\!]$$

But this falls out directly from the definition of R, so we are done. □

6. The Merits of Our Approach

By way of comparison with previous work, we now informally present an alternative proof of the "flag objects" example (from Section 5.1) in the style of Ahmed [4]. Following that, we discuss how our LSLR proof relates to and improves on this alternative proof.

6.1. **Flag Objects Proof With Explicit Step Manipulation.** We now sketch a proof for the "flag objects" example from Section 5.1 using Ahmed's logical relation [4]. Since the latter is asymmetric, to prove equivalence of `bflag` and `iflag` at type flag, we must show that for all $n \geq 0$, $(n, \mathtt{bflag}, \mathtt{iflag}) \in \mathcal{E}\,[\![\mathsf{flag}]\!]$ and $(n, \mathtt{iflag}, \mathtt{bflag}) \in \mathcal{E}\,[\![\mathsf{flag}]\!]$, where $\mathcal{E}\,[\![\cdot]\!]$ is the asymmetric logical relation for closed terms from Ahmed's paper. Here, writing $(n, e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]$ means that e_1 and e_2 are related for n steps—or more specifically, that if e_1 terminates in less than n steps then e_2 will terminate (in any number of steps) and the resulting values will be related for the remaining number of steps. We discuss only one direction of the proof; the other direction is similar.

To prove that $(n, \mathtt{bflag}, \mathtt{iflag}) \in \mathcal{E}\,[\![\mathsf{flag}]\!]$ for arbitrary $n \geq 0$, it suffices to show $(n, \mathtt{bflag}, \mathtt{iflag}) \in \mathcal{V}\,[\![\mathsf{flag}]\!]$, since `bflag` and `iflag` are values. We take $\tau_1 = \mathsf{bool}$, $\tau_2 = \mathsf{int}$, and

$$\begin{aligned} R = \{\, (n', v_1, v_2) \mid\ & \vdash v_1 : \mathsf{bool} \wedge\ \vdash v_2 : \mathsf{int} \wedge \\ & \exists y : \mathsf{int}.\, (v_1 = \mathtt{true} \wedge 2y \Downarrow v_2) \vee (v_1 = \mathtt{false} \wedge 2y+1 \Downarrow v_2)\} \end{aligned}$$

Let $\rho = \alpha \mapsto (\mathsf{bool}, \mathsf{int}, R)$. It then suffices to show, for all $m < n$, that

$$(m, \mathtt{roll}\,\langle \mathtt{true}, \langle \mathtt{bflip}, \mathtt{bret}\rangle\rangle, \mathtt{roll}\,\langle 0, \langle \mathtt{iflip}, \mathtt{iret}\rangle\rangle) \in \mathcal{V}\,[\![\mathsf{fld}_\alpha]\!]\,\rho$$

Unwinding the definitions of $\mathcal{V}\,[\![\mu\beta.\,\tau]\!]$ and $\mathcal{V}\,[\![\tau_1 \times \tau_2]\!]$, it now suffices to show the following for all $k < m$:

(1) Show $(k, \mathtt{true}, 0) \in \mathcal{V}\,[\![\alpha]\!]\,\rho$. This is immediate from the definition of R, choosing $y = 0$ as before.

(2) Show $(k, \mathtt{bflip}, \mathtt{iflip}) \in \mathcal{V}\,[\![\mathsf{fld}_\alpha \to \mathsf{fld}_\alpha]\!]\,\rho$. For arbitrary $j < k$, assuming we are given $(j, v_{a1}, v_{a2}) \in \mathcal{V}\,[\![\mathsf{fld}_\alpha]\!]\,\rho$, we are required to show:

$$\begin{aligned} (j, & \mathtt{roll}\,\langle \mathtt{not}\,(\mathtt{fst}\,(\mathtt{unroll}\,v_{a1})), \mathtt{snd}\,(\mathtt{unroll}\,v_{a1})\rangle, \\ & \mathtt{roll}\,\langle 1 + (\mathtt{fst}\,(\mathtt{unroll}\,v_{a2})), \mathtt{snd}\,(\mathtt{unroll}\,v_{a2})\rangle) \in \mathcal{E}\,[\![\mathsf{fld}_\alpha]\!]\,\rho \end{aligned}$$

We assume that $\mathtt{roll}\,\langle \mathtt{not}\,(\mathtt{fst}\,(\mathtt{unroll}\,v_{a1})), \mathtt{snd}\,(\mathtt{unroll}\,v_{a1})\rangle$ evaluates to a value v_{f1} in $i < j$ steps. We are required to show that there exists a value v_{f2} such that $\mathtt{roll}\,\langle 1 + (\mathtt{fst}\,(\mathtt{unroll}\,v_{a2})), \mathtt{snd}\,(\mathtt{unroll}\,v_{a2})\rangle$ evaluates to v_{f2} and $(j - i, v_{f1}, v_{f2}) \in \mathcal{V}\,[\![\mathsf{fld}_\alpha]\!]\,\rho$. Since these expressions clearly require more than one step of evaluation, we know that $j > 2$ (which is relevant here when we talk about $j - 1$).

From $(j, v_{a1}, v_{a2}) \in \mathcal{V}[\![\mathsf{fld}_\alpha]\!]\,\rho$, it follows that $v_{a1} = \mathtt{roll}\, v_{10}$ and $v_{a2} = \mathtt{roll}\, v_{20}$, and furthermore that $v_{10} = \langle v_{11}, v_{12}\rangle$ and $v_{20} = \langle v_{21}, v_{22}\rangle$, where $(j-1, v_{11}, v_{21}) \in \mathcal{V}[\![\alpha]\!]\,\rho$ and $(j-1, v_{12}, v_{22}) \in \mathcal{V}[\![(\mathsf{fld}_\alpha \to \mathsf{fld}_\alpha) \times (\mathsf{fld}_\alpha \to \mathsf{bool})]\!]\,\rho$.

Hence, by the operational semantics, we have that:

$$\begin{array}{l}
\mathtt{roll}\,\langle \mathtt{not}\,(\mathtt{fst}\,(\mathtt{unroll}\, v_{a1})), \mathtt{snd}\,(\mathtt{unroll}\, v_{a1})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle \mathtt{not}\,(\mathtt{fst}\, v_{10}), \mathtt{snd}\,(\mathtt{unroll}\, v_{a1})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle \mathtt{not}\, v_{11}, \mathtt{snd}\,(\mathtt{unroll}\, v_{a1})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle \neg v_{11}, \mathtt{snd}\,(\mathtt{unroll}\, v_{a1})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle \neg v_{11}, \mathtt{snd}\, v_{10}\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle \neg v_{11}, v_{12}\rangle \\
= v_{f1}
\end{array}$$

where $\neg v_{11}$ is a value denoting the negation of v_{11}.

Also, by the operational semantics:

$$\begin{array}{l}
\mathtt{roll}\,\langle 1 + (\mathtt{fst}\,(\mathtt{unroll}\, v_{a2})), \mathtt{snd}\,(\mathtt{unroll}\, v_{a2})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle 1 + (\mathtt{fst}\, v_{20}), \mathtt{snd}\,(\mathtt{unroll}\, v_{a2})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle 1 + v_{21}, \mathtt{snd}\,(\mathtt{unroll}\, v_{a2})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle 1 \hat{+} v_{21}, \mathtt{snd}\,(\mathtt{unroll}\, v_{a2})\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle 1 \hat{+} v_{21}, \mathtt{snd}\, v_{20}\rangle \rightsquigarrow \\
\mathtt{roll}\,\langle 1 \hat{+} v_{21}, v_{22}\rangle \\
= v_{f2}
\end{array}$$

where $1 \hat{+} v_{21}$ is a value denoting the sum of 1 and v_{21}.

It remains for us to show that $(j-i, v_{f1}, v_{f2}) \in \mathcal{V}[\![\mathsf{fld}_\alpha]\!]\,\rho$. By the definition of $\mathcal{V}[\![\mu\beta.\,\tau]\!]$ and $\mathcal{V}[\![\tau_1 \times \tau_2]\!]$, it suffices to show that, assuming $j - i > 0$:

- $(j-i-1, \neg v_{11}, 1 \hat{+} v_{21}) \in \mathcal{V}[\![\alpha]\!]\,\rho$, which follows from $(j-1, v_{11}, v_{21}) \in \mathcal{V}[\![\alpha]\!]\,\rho$ and the definition of R.
- $(j-i-1, v_{12}, v_{22}) \in \mathcal{V}[\![(\mathsf{fld}_\alpha \to \mathsf{fld}_\alpha) \times (\mathsf{fld}_\alpha \to \mathsf{bool})]\!]\,\rho$, which follows from the fact above that v_{12} and v_{22} are related for $j-1$ steps, which means that they must be related for fewer steps.

(3) Show $(k, \mathtt{bret}, \mathtt{iret}) \in \mathcal{V}[\![\mathsf{fld}_\alpha \to \mathsf{bool}]\!]\,\rho$. This is similar to the proof of part (2). □

6.2. **What Have We Achieved?** One can see that the above proof requires quite a bit of pedantic step manipulation that is entirely unimportant in terms of the overall proof. The proof using LSLR allows us to ignore steps and focus on the interesting parts of the proof.

Perhaps more importantly, the above proof is almost "mindless" in the sense that it proceeds by simply unrolling definitions. For instance, step (2) of the proof proceeds to prove relatedness of two terms for j steps in $\mathcal{E}[\![\mathsf{fld}_\alpha]\!]\,\rho$ by symbolically evaluating them to values and then showing that the resulting values are related for $j-i$ steps, where i is the number of steps it takes to evaluate the first term. This is exactly how one would attempt to prove the subgoal if one were just to expand the definition of $\mathcal{E}[\![\mathsf{fld}_\alpha]\!]\,\rho$. But as a result, one is forced to talk about the particular number of steps the first term takes to evaluate, and moreover, the *idea* of the proof is obscured.

In contrast, the LSLR proof of this example has a much clearer structure because it is constructed using higher-level proof rules. In the aforementioned step (2), the LSLR proof does not need to symbolically execute the terms because it is possible to use compatibility rules, together with the SYM-BIND rule, instead. This combination is applicable precisely

because the two terms being related have a very similar structure and only differ in one place. Thus, the ability to prove the relatedness of the terms using those rules sheds light on *why* they are equivalent.

That said, the reader may wonder: is the logic LSLR really *necessary*? Can we take the proof rules that we have derived in LSLR and interpret them back into the step-indexed model, thus resulting in proof principles for the step-indexed model that *do* mention steps but nonetheless help one to write proofs in a more structured way? We believe that to some extent this should be possible. For example, here is a variant of the BIND2 rule that holds (ignoring syntactic typing side conditions) for Ahmed's model:

$$\frac{(j, e_1, e_2) \in \mathcal{E}\,[\![\tau]\!]\,\rho \qquad \forall i \leq j.\ \forall v_1, v_2.\ (i, v_1, v_2) \in \mathcal{V}\,[\![\tau]\!]\,\rho \Longrightarrow (i, E_1[v_1], E_2[v_2]) \in \mathcal{E}\,[\![\tau']\!]\,\rho'}{(j, E_1[e_1], E_2[e_2]) \in \mathcal{E}\,[\![\tau']\!]\,\rho'}$$

This proof principle is almost as clean as the BIND2 rule, the only difference being that this step-indexed version requires an explicit quantification over future worlds i, whereas in LSLR that quantification is baked into the interpretation of the logical judgment. While this explicit quantification is annoying, the above rule should still (we believe) be useful in improving the structure of "direct" step-indexed proofs. (It is less clear how to interpret the symmetric rules from Figure 9 into useful step-indexed rules.)

Thus, what we view as the major contribution of this work is the development of a set of proof principles to enable better structuring of step-indexed proofs. By working at the logical level, instead of directly in the step-indexed model, we have been forced to come up with clean high-level rules that do not mention steps, but at least some of these rules should in retrospect also be useful for improving the structure of direct step-indexed proofs.

7. Comparison With an Earlier Version of LSLR

In this section, we explain the four main differences between the present version of LSLR and the earlier version that we described in our LICS 2009 paper [14].

Atomic Typing and Value Predicates. In the earlier version of LSLR, we built in the atomic predicates of syntactic typing ($e : \tau$) and value-hood (Val) as primitive notions in the logic, instead of treating them as ordinary atomic relations as we do presently. Specifically, we imposed a distinction in the variable context $\mathcal{X}$ between value variables x and term variables t and required typing annotations on their context bindings. We also required relation variables to be bound with explicit relation types $\mathrm{TRel}(\tau_1, \tau_2)$ and $\mathrm{VRel}(\tau_1, \tau_2)$ (relations were restricted to be binary). In the present version, we also make use of relation types, but these are definable in the logic and need not be made primitive.

There was in retrospect no particularly good reason for giving these predicates special treatment, nor for restricting the arity of relations to 2. We feel our present treatment is simpler, cleaner, and more general.

Distinction Between Logic and Model. In the earlier version of LSLR, we made a distinction between our main logical judgment, $\mathcal{C} \vdash P$, defined by a set of core inference rules, and its interpretation into the model, which we wrote as $\mathcal{C} \models P$. This enabled a more precise characterization of what it means for a rule (like those in Figure 7) to be "derivable".

In the present paper, we conflate $\vdash$ and $\models$, thus allowing arbitrary new inference rules to be added to the logic at a later time as long as they can be proven sound. We have

made this change because ultimately it is not clear to us why (or that) the core set of inference rules we gave in Figure 5 are the "right" (or "canonical") ones. They are simply a set of sound rules that we have found to be useful for doing nearly all of our proofs about logical relations in LSLR. However, as in the LICS paper, those core rules are not "complete"—occasionally, as in the proof of Adequacy of our logical relation, one needs to reason directly in the model. We therefore feel there is no particular need to grant those core rules "definitional" status.

Completeness of the Logical Relation. In the LICS paper, we defined a logical relation for F^μ that—like Ahmed's original logical relation for F^μ [4]—was sound, but not complete, with respect to contextual approximation. (The incompleteness is related to the treatment of existential types, cf. Example 7.7.4 in Pitts [26].) The only substantive difference between that logical relation and our present version is in the definition of $\mathcal{E}[\![\tau]\!]\rho$. If $(e_1, e_2) \in \mathcal{E}[\![\tau]\!]\rho$, then in the case when $e_1 \Downarrow^0 v_1$, the LICS logical relation would insist that e_2 *evaluate to* some value v_2 such that $(v_1, v_2) \in \mathcal{V}[\![\tau]\!]\rho$. In our present logical relation, we only insist that e_2 *be ciu-approximated by* such a value v_2. This added flexibility is important in proving the Ciu-Transitivity property (Theorem 4.24), which is the key to showing completeness of our present version of the logical relation.

This change to the logical relation has resulted in changes to some of the derivable rules in Figures 7 and 9 as well. Rule CIU, for instance, is more flexible than the corresponding Rule 3 in the LICS paper, whereas Rule BIND is more restrictive than the corresponding Rule 6 in the LICS paper. Practically speaking, though, these differences seem to be very minor, and they have not induced any serious changes to our proofs of the examples in Section 5.

Fixing a Technical Flaw. Our present account of LSLR fixes a technical flaw in the LICS version, namely that three inference rules in that paper are unsound (and all three for similar reasons). Luckily, none of the rules was of critical importance. The common error we made in our proofs for all three rules was in forgetting that, when reasoning about the $\triangleright$ operator, the interesting "base case" is often not world 0 but world 1.

The first unsound rule is Rule $\triangleright\exists 1$ from Figure 5, in the case where $\mathcal{X}$ is of the form $x : \tau$. (Note: our present version of LSLR does not run afoul of this bug precisely because we no longer bake typing or value predicates into the $\mathcal{X}$.) The problem arises when τ is an uninhabited type, such as $\forall\alpha.\alpha$. The $\triangleright\exists 1$ rule says that $\triangleright\exists x : \tau.P$ implies $\exists x : \tau.\triangleright P$. In order for this to be sound it must at least be the case that $[\![\triangleright\exists x : \tau.P]\!]\,1$ implies $[\![\exists x : \tau.\triangleright P]\!]\,1$. However, the former is trivially true, and the latter is false because there is no value of type τ. The rule is easy to show sound under the side condition that τ is inhabited.

The second and third unsound rules are those numbered Rule 10 and Rule 8 (the backwards direction) in the LICS paper, which are as follows:

$$\frac{\mathcal{C} \vdash \triangleright(e_1, e_2) \in \mathcal{E}[\![\tau[\mu\alpha.\,\tau/\alpha]]\!]\rho}{\mathcal{C} \vdash (\mathtt{roll}\, e_1, \mathtt{roll}\, e_2) \in \mathcal{E}[\![\mu\alpha.\,\tau]\!]\rho} \qquad \frac{\mathcal{C} \vdash (\mathtt{unroll}\, e_1, \mathtt{unroll}\, e_2) \in \mathcal{E}[\![\tau[\mu\alpha.\,\tau/\alpha]]\!]\rho}{\mathcal{C} \vdash (e_1, e_2) \in \mathcal{E}[\![\mu\alpha.\,\tau]\!]\rho}$$

The problem with these rules, again, is that the implications do not hold when the propositions are interpreted at world 1. In our buggy proofs of derivability for these rules, the error manifested itself as a need to derive $e_2 \Downarrow v_2$ in a context where we only knew $\triangleright(e_2 \Downarrow v_2)$. Interestingly, $[\![\triangleright(e_2 \Downarrow v_2)]\!]\,n$ *does* imply $[\![e_2 \Downarrow v_2]\!]\,n$ for all n except $n = 1$.

Fortunately, the only one of these rules that we actually made any use of was the last one. We used it in the proof of the syntactic minimal invariance example, and thus our

present proof of that example is somewhat different than the one given in the LICS paper. In particular, in proving that example, we now make critical use of the standard Canonical Forms property for well-typed values, which we did not in the LICS paper.

8. 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, Melliès, Richards, and Vouillon'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 [29] 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 [11]), whereas our logic LSLR does not. Bierman, Pitts and Russo [9] 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.

The main difference between our work and AMRV's very modal model is the application: whereas AMRV 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). Moreover, a significant component of our contribution is the derivation of a set of useful, language-specific inference rules and the application of those rules to several representative examples from the literature.

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 [34], or Lassen and Levy's [20]. Sumii and Pierce give several example applications of their method in a language setting very similar to the one we consider here. In Section 5, we already showed how to use LSLR to prove two examples adapted from their paper, and our approach is capable of straightforwardly handling the other examples that their method can prove as well.

That said, Sumii and Pierce do present one equivalence, the "IntSet" example at the beginning of Section 7 of their paper, which does not seem to be provable *directly* within our logic, although it is provable through a transitive combination of equivalence proofs. They use this example to exhibit a limitation of their method with respect to reasoning about higher-order functions, and hence to motivate an "up-to-context" extension of their bisimulation that alleviates the problem. However, they do not actually offer a proof of the IntSet example (using the up-to-context extension or otherwise), and we believe the proof to be considerably more involved than for the other up-to-context examples in their paper. Ahmed [4] has given a proof of this example using step-indexed logical relations (see her

technical report), but her proof is closely tailored to the specific example and seems difficult to adapt, *e.g.,* if the ADT in the example is extended with a "remove" operation.

The IntSet example is challenging because it involves an equivalence between two recursive functions that are structurally quite dissimilar in their recursive calling patterns, and the hard work in the proof involves demonstrating that both functions ultimately call a certain (unknown) function on the same multiset of arguments (albeit in a different order). The clearest way to establish this fact is using *inductive* reasoning about computations on lists and trees, which can be accomplished using standard proof techniques and is orthogonal to the coinductive, relational style of reasoning that LSLR (and in particular the $\triangleright$ operator) provides. While for this example the inductive and coinductive bits of the proof can be easily combined using a transitive combination of equivalences, it would be interesting to explore in future work how to better integrate inductive reasoning into our logic.

Bisimulations have also been developed for relational reasoning in languages with general references and/or control operators [19, 32, 31, 33]. 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 [12], 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, besides step-indexed logical relations, a number of other logical relations methods have been proposed for languages with parametric polymorphism, recursion, and/or recursive types, *e.g.,* [25, 26, 18, 22, 10, 13]. One of the most important advances in this domain is the idea of $\top\top$-closure (aka *biorthogonality*). In developing a logical relation for a language with impredicative polymorphism, existential types, and general recursion, Pitts [25, 26] proposed $\top\top$-closure as a useful operational technique for guaranteeing admissibility of relations (in the denotational sense). In the step-indexed model, the whole issue of admissibility is sidestepped. Intuitively, there is no need to worry about a fixed-point behaving like the limit of its finite approximations if we restrict attention to how programs behave in a finite amount of time (as the step-indexed model does).

For non-step-indexed logical relations it is well-known that $\top\top$-closure also has the pleasing side effect of rendering the relations complete w.r.t. contextual equivalence. This is also the case for step-indexed logical relations, as shown in recent work of Dreyer *et al.* [15]. We have presented in this paper an alternative technique for ensuring completeness, namely closure w.r.t. *ciu*-approximation (in the definition of $\mathcal{E}[\![\tau]\!]\rho$). We believe our approach is simpler and more direct than $\top\top$-closure, but neither approach subsumes the either. On the one hand, $\top\top$-closure is applicable in more general settings, such as lower-level languages [8, 17] or languages with control operators [15], where the behavior of a term depends on its evaluation context. On the other hand, this added generality means that a $\top\top$-closed relation is incapable of validating some of the inference rules that hold in our more restricted setting. For example, the SYM-BIND rule (Figure 9) would not hold in a $\top\top$-closed model unless we were to remove the assumptions in the last premise connecting the x_i's and the e_i's, thus weakening the rule somewhat. We do believe, however, that it should be possible to formalize a variant of our LSLR logical relation that uses $\top\top$-closure instead of ciu-closure. Understanding the tradeoffs between the two closure techniques remains an interesting problem for future work.

Non-step-indexed logical relations for languages with *recursive types* are notoriously tricky to construct; the construction of such relations relies on the use of *syntactic minimal invariance*, mimicking the construction used in domain theory [27, 10, 13]. An advantage of this more elaborate construction over step-indexed logical relations is that the resulting proof method is more abstract and does not involve steps. In this paper, we have shown how to devise a more abstract proof method for step-indexed logical relations. Our resulting proof method is at roughly the same level of abstraction as that of non-step-indexed logical relations. This point was illustrated explicitly with the various examples in Section 5. For yet another example, just involving recursive types, the reader might want to consider Birkedal and Harper's example of stream operations [10]. Their proof uses a coinduction proof principle that is derived as a corollary of the elaborate construction of the logical relation. This example can also be proved in LSLR in a very similar manner, except that we use a combination of the LÖB and SYM-EXP-$\triangleright$ rules instead of actual coinduction.

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, 24]. We believe that the work presented here makes an important first step toward *logical* step-indexed logical relations for effectful programs. Indeed, since publication of our original LICS paper [14], a promising variant/extension of LSLR (called LADR) has been developed [16], which enables abstract relational reasoning about a step-indexed model of $\mathsf{F}^{\mu!}$ (an extension of F^{μ} with general references).

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] Nick Benton and Chung-Kil Hur. Biorthogonality, step-indexing and compiler correctness. In *ICFP*, 2009.

[9] 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.

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

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

[12] Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticae*, 33(4):309–338, 1998.

[13] 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.

[14] Derek Dreyer, Amal Ahmed, and Lars Birkedal. Logical step-indexed logical relations. In *LICS*, 2009.

[15] Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP*, 2010.
[16] Derek Dreyer, Georg Neis, Andreas Rossberg, and Lars Birkedal. A relational modal logic for higher-order stateful ADTs. In *POPL*, 2010.
[17] Chung-Kil Hur and Derek Dreyer. A Kripke logical relation between ML and assembly. In *POPL*, 2011.
[18] Patricia Johann and Janis Voigtländer. The impact of seq on free theorems-based program transformations. *Fundamenta Informaticae*, 69(1–2):63–102, 2006.
[19] Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL*, 2006.
[20] Soren B. Lassen and Paul B. Levy. Normal form bisimulation for parametric polymorphism. In *LICS*, 2008.
[21] Ian Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.
[22] Paul-André Melliès and Jérôme Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS*, 2005.
[23] Hiroshi Nakano. A modality for recursion. In *LICS*, 2000.
[24] Georg Neis, Derek Dreyer, and Andreas Rossberg. Non-parametric parametricity. In *ICFP*, 2009.
[25] A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.
[26] Andrew Pitts. Typed operational reasoning. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7. MIT Press, 2005.
[27] Andrew M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
[28] Gordon D. Plotkin. Denotational semantics with partial functions. Lecture at C.S.L.I. Summer School, 1985.
[29] Gordon D. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, February 1993.
[30] Gordon D. Plotkin and Martín Abadi. A logic for parametric polymorphism. In *TLCA*, 1993.
[31] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *LICS*, 2007.
[32] Kristian Støvring and Soren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *POPL*, 2007.
[33] Eijiro Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In *CSL*, 2009.
[34] Eijiro Sumii and Benjamin Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5):1–43, 2007.
[35] Philip Wadler. Theorems for free! In *FPCA*, 1989.

Appendix A. Additional Details of F^{μ}

Typing Contexts $\Gamma ::= \cdot \mid \Gamma, \alpha \mid \Gamma, x : \tau$

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

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \langle\rangle : \mathsf{unit}} \qquad \frac{}{\Gamma \vdash \pm n : \mathsf{int}}$$

$$\frac{}{\Gamma \vdash \mathtt{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathtt{false} : \mathsf{bool}} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathtt{if}\, e\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{fst}\, e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{snd}\, e : \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathtt{inl}_{\tau_1+\tau_2}\, e : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathtt{inr}_{\tau_1+\tau_2}\, e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \qquad \Gamma, x_2 : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathtt{case}\, e\, \mathtt{of}\, \mathtt{inl}\, x_1 \Rightarrow e_1 \mid \mathtt{inr}\, x_2 \Rightarrow e_2 : \tau}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\, e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau}$$

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.\, e : \forall\alpha.\, \tau} \qquad \frac{\Gamma \vdash e : \forall\alpha.\, \tau \qquad \mathrm{FV}(\tau_1) \subseteq \Gamma}{\Gamma \vdash e\, \tau_1 : \tau[\tau_1/\alpha]}$$

$$\frac{\mathrm{FV}(\tau_1) \subseteq \Gamma \qquad \Gamma \vdash e : \tau[\tau_1/\alpha]}{\Gamma \vdash \mathtt{pack}\, \tau_1, e\, \mathtt{as}\, \exists\alpha.\, \tau : \exists\alpha.\, \tau} \qquad \frac{\Gamma \vdash e_1 : \exists\alpha.\, \tau_1 \qquad \mathrm{FV}(\tau) \subseteq \Gamma \qquad \Gamma, \alpha, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \mathtt{unpack}\, e_1\, \mathtt{as}\, \alpha, x\, \mathtt{in}\, e_2 : \tau}$$

$$\frac{\Gamma \vdash e : \tau[\mu\alpha.\, \tau/\alpha] \qquad \mathrm{FV}(\mu\alpha.\, \tau) \subseteq \Gamma}{\Gamma \vdash \mathtt{roll}_{\mu\alpha.\, \tau}\, e : \mu\alpha.\, \tau} \qquad \frac{\Gamma \vdash e : \mu\alpha.\, \tau}{\Gamma \vdash \mathtt{unroll}\, e : \tau[\mu\alpha.\, \tau/\alpha]}$$

Figure 10: F^{μ} Static Semantics

$$
\begin{array}{rrl}
\textit{Contexts} & C & ::= \; [\cdot] \mid o(e_1, \ldots, e_{i-1}, C, e_{i+1}, \ldots, e_n) \mid \\
& & \texttt{if}\, C\, \texttt{then}\, e_1\, \texttt{else}\, e_2 \mid \texttt{if}\, e\, \texttt{then}\, C\, \texttt{else}\, e_2 \mid \texttt{if}\, e\, \texttt{then}\, e_1\, \texttt{else}\, C \mid \\
& & \langle C, e_2 \rangle \mid \langle e_1, C \rangle \mid \texttt{fst}\, C \mid \texttt{snd}\, C \mid \\
& & \texttt{inl}_\tau\, C \mid \texttt{inr}_\tau\, C \mid \texttt{case}\, C\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow e_1 \mid \texttt{inr}\, x_2 \Rightarrow e_2 \mid \\
& & \texttt{case}\, e\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow C \mid \texttt{inr}\, x_2 \Rightarrow e_2 \mid \texttt{case}\, e\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow e_1 \mid \texttt{inr}\, x_2 \Rightarrow C \mid \\
& & \lambda x\!:\!\tau.\, C \mid C\, e \mid e\, C \mid \Lambda\alpha.\, C \mid C\, \tau \mid \\
& & \texttt{pack}\, \tau_1, C\, \texttt{as}\, \exists\alpha.\, \tau \mid \texttt{unpack}\, C\, \texttt{as}\, \alpha, x\, \texttt{in}\, e_2 \mid \texttt{unpack}\, e_1\, \texttt{as}\, \alpha, x\, \texttt{in}\, C \mid \\
& & \texttt{roll}_\tau\, C \mid \texttt{unroll}\, C \mid
\end{array}
$$

$\boxed{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$

$$\frac{\Gamma \subseteq \Gamma'}{\vdash [\cdot] : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau)} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \mathsf{bool}) \qquad \Gamma' \vdash e_1 : \tau' \qquad \Gamma' \vdash e_2 : \tau'}{\vdash \texttt{if}\, C\, \texttt{then}\, e_1\, \texttt{else}\, e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e : \mathsf{bool} \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau') \qquad \Gamma' \vdash e_2 : \tau'}{\vdash \texttt{if}\, e\, \texttt{then}\, C\, \texttt{else}\, e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e : \mathsf{bool} \qquad \Gamma' \vdash e_1 : \tau' \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}{\vdash \texttt{if}\, e\, \texttt{then}\, e_1\, \texttt{else}\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1) \qquad \Gamma' \vdash e_2 : \tau_2}{\vdash \langle C, e_2 \rangle : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 \times \tau_2)} \qquad \frac{\Gamma' \vdash e_1 : \tau_1 \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_2)}{\vdash \langle e_1, C \rangle : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 \times \tau_2)}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 \times \tau_2)}{\vdash \texttt{fst}\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1)} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 \times \tau_2)}{\vdash \texttt{snd}\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_2)}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1)}{\vdash \texttt{inl}_{\tau_1+\tau_2}\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 + \tau_2)} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_2)}{\vdash \texttt{inr}_{\tau_1+\tau_2}\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 + \tau_2)}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 + \tau_2) \qquad \Gamma', x_1 : \tau_1 \vdash e_1 : \tau' \qquad \Gamma', x_2 : \tau_2 \vdash e_2 : \tau'}{\vdash \texttt{case}\, C\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow e_1 \mid \texttt{inr}\, x_2 \Rightarrow e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e : \tau_1 + \tau_2 \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma', x_1 : \tau_1 \vdash \tau') \qquad \Gamma', x_2 : \tau_2 \vdash e_2 : \tau'}{\vdash \texttt{case}\, e\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow C \mid \texttt{inr}\, x_2 \Rightarrow e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e : \tau_1 + \tau_2 \qquad \Gamma', x_1 : \tau_1 \vdash e_1 : \tau' \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma', x_2 : \tau_2 \vdash \tau')}{\vdash \texttt{case}\, e\, \texttt{of}\, \texttt{inl}\, x_1 \Rightarrow e_1 \mid \texttt{inr}\, x_2 \Rightarrow C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma', x : \tau_1 \vdash \tau_2)}{\vdash \lambda x\!:\!\tau_1.\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_1 \to \tau_2)} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_2 \to \tau') \qquad \Gamma' \vdash e_2 : \tau_2}{\vdash C\, e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e_1 : \tau_2 \to \tau' \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau_2)}{\vdash e_1\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

Figure 11: F^μ Program Contexts: Syntax and Static Semantics I

$\boxed{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$ (contd. from Figure 11)

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma', \alpha \vdash \tau')}{\vdash \Lambda\alpha.\, C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \forall\alpha.\, \tau')} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \forall\alpha.\, \tau') \qquad \mathrm{FV}(\tau_1) \subseteq \Gamma'}{\vdash C\,\tau_1 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau'[\tau_1/\alpha])}$$

$$\frac{\mathrm{FV}(\tau_1) \subseteq \Gamma' \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau'[\tau_1/\alpha])}{\vdash \mathtt{pack}\,\tau_1, C\,\mathtt{as}\,\exists\alpha.\,\tau' : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \exists\alpha.\,\tau')}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \exists\alpha.\,\tau_1) \qquad \mathrm{FV}(\tau') \subseteq \Gamma' \qquad \Gamma', \alpha, x : \tau_1 \vdash e_2 : \tau'}{\vdash \mathtt{unpack}\,C\,\mathtt{as}\,\alpha, x\,\mathtt{in}\,e_2 : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\Gamma' \vdash e_1 : \exists\alpha.\,\tau_1 \qquad \mathrm{FV}(\tau') \subseteq \Gamma' \qquad \vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma', \alpha, x : \tau_1 \vdash \tau')}{\vdash \mathtt{unpack}\,e_1\,\mathtt{as}\,\alpha, x\,\mathtt{in}\,C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau')}$$

$$\frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau'[\mu\alpha.\,\tau'/\alpha]) \quad \mathrm{FV}(\mu\alpha.\,\tau') \subseteq \Gamma'}{\vdash \mathtt{roll}_{\mu\alpha.\,\tau'}\,C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \mu\alpha.\,\tau')} \qquad \frac{\vdash C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \mu\alpha.\,\tau')}{\vdash \mathtt{unroll}\,C : (\Gamma \vdash \tau) \rightsquigarrow (\Gamma' \vdash \tau'[\mu\alpha.\,\tau'/\alpha])}$$

Figure 12: F^{μ} Program Contexts: Static Semantics II

Appendix B. Remaining Inference Rules for LSLR

Here, we present the LSLR judgments of relation and substitution well-formedness, as well as additional inference rules that are entirely standard. Prop is synonymous with Rel(0).

$\boxed{\mathcal{X}; \mathcal{R} \vdash R :: \mathrm{Rel}(n)}$

$$\frac{r \in \mathcal{R} \qquad \mathrm{arity}(r) = n}{\mathcal{X}; \mathcal{R} \vdash r :: \mathrm{Rel}(n)} \qquad \frac{\mathrm{FV}(e_1, e_2) \subseteq \mathcal{X}}{\mathcal{X}; \mathcal{R} \vdash e_1 = e_2 :: \mathrm{Prop}} \qquad \frac{}{\mathcal{X}; \mathcal{R} \vdash \mathrm{Val} :: \mathrm{Rel}(1)} \qquad \frac{\mathrm{FV}(e, \tau) \subseteq \mathcal{X}}{\mathcal{X}; \mathcal{R} \vdash e : \tau :: \mathrm{Prop}}$$

$$\frac{\mathrm{FV}(C, \tau, \tau') \subseteq \mathcal{X}}{\mathcal{X}; \mathcal{R} \vdash C : \tau \rightsquigarrow \tau' :: \mathrm{Prop}} \qquad \frac{\mathrm{FV}(e_1, e_2) \subseteq \mathcal{X}}{\mathcal{X}; \mathcal{R} \vdash e_1 \rightsquigarrow^{\{*,0,1\}} e_2 :: \mathrm{Prop}} \qquad \frac{\mathrm{FV}(e_1, e_2) \subseteq \mathcal{X}}{\mathcal{X}; \mathcal{R} \vdash e_1 \preceq e_2 :: \mathrm{Prop}}$$

$$\frac{}{\mathcal{X}; \mathcal{R} \vdash \top :: \mathrm{Prop}} \qquad \frac{}{\mathcal{X}; \mathcal{R} \vdash \bot :: \mathrm{Prop}} \qquad \frac{\mathcal{X}; \mathcal{R} \vdash P :: \mathrm{Prop} \qquad \mathcal{X}; \mathcal{R} \vdash Q :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash P \wedge Q :: \mathrm{Prop}}$$

$$\frac{\mathcal{X}; \mathcal{R} \vdash P :: \mathrm{Prop} \qquad \mathcal{X}; \mathcal{R} \vdash Q :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash P \vee Q :: \mathrm{Prop}} \qquad \frac{\mathcal{X}; \mathcal{R} \vdash P :: \mathrm{Prop} \qquad \mathcal{X}; \mathcal{R} \vdash Q :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash P \Rightarrow Q :: \mathrm{Prop}}$$

$$\frac{\mathcal{X}, \mathcal{X}'; \mathcal{R} \vdash P :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash \forall\mathcal{X}'.P :: \mathrm{Prop}} \qquad \frac{\mathcal{X}; \mathcal{R}, \mathcal{R}' \vdash P :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash \forall\mathcal{R}'.P :: \mathrm{Prop}} \qquad \frac{\mathcal{X}, \mathcal{X}'; \mathcal{R} \vdash P :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash \exists\mathcal{X}'.P :: \mathrm{Prop}} \qquad \frac{\mathcal{X}; \mathcal{R}, \mathcal{R}' \vdash P :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash \exists\mathcal{R}'.P :: \mathrm{Prop}}$$

$$\frac{\mathcal{X}, \overline{x}; \mathcal{R} \vdash P :: \mathrm{Prop} \qquad \overline{x} = x_1, \ldots, x_n}{\mathcal{X}; \mathcal{R} \vdash \overline{x}.P :: \mathrm{Rel}(n)} \qquad \frac{\mathrm{FV}(\overline{e}) \subseteq \mathcal{X} \qquad \overline{e} = e_1, \ldots, e_n \qquad \mathcal{X}; \mathcal{R} \vdash R :: \mathrm{Rel}(n)}{\mathcal{X}; \mathcal{R} \vdash \overline{e} \in R :: \mathrm{Prop}}$$

$$\frac{\mathcal{X}; \mathcal{R}, r \vdash R :: \mathrm{Rel}(n) \qquad \mathrm{arity}(r) = n \qquad R \text{ contractive in } r}{\mathcal{X}; \mathcal{R} \vdash \mu r.R :: \mathrm{Rel}(n)} \qquad \frac{\mathcal{X}; \mathcal{R} \vdash P :: \mathrm{Prop}}{\mathcal{X}; \mathcal{R} \vdash \triangleright P :: \mathrm{Prop}}$$

$\boxed{\mathcal{X} \vdash \gamma :: \mathcal{X}'}$

$$\frac{\mathrm{dom}(\gamma) = \mathcal{X}' \qquad \forall \alpha \in \mathcal{X}'.\ \mathrm{FV}(\gamma\alpha) \subseteq \mathcal{X} \qquad \forall x \in \mathcal{X}'.\ \mathrm{FV}(\gamma x) \subseteq \mathcal{X}}{\mathcal{X} \vdash \gamma :: \mathcal{X}'}$$

$\boxed{\mathcal{X}; \mathcal{R} \vdash \varphi :: \mathcal{R}'}$

$$\frac{\mathrm{dom}(\varphi) = \mathcal{R}' \qquad \forall r \in \mathcal{R}'.\ \mathrm{arity}(r) = n \Rightarrow \mathcal{X}; \mathcal{R} \vdash \varphi r :: \mathrm{Rel}(n)}{\mathcal{X}; \mathcal{R} \vdash \varphi :: \mathcal{R}'}$$

$\boxed{\mathcal{C} \vdash P}$

$$\frac{\mathcal{C} \vdash P}{\mathcal{C}, \mathcal{C}' \vdash P} \qquad \frac{}{\mathcal{C}, P \vdash P} \qquad \frac{}{\mathcal{C} \vdash \top} \qquad \frac{\mathcal{C} \vdash \bot}{\mathcal{C} \vdash P} \qquad \frac{\mathcal{C} \vdash P \quad \mathcal{C} \vdash Q}{\mathcal{C} \vdash P \wedge Q} \qquad \frac{\mathcal{C} \vdash P \wedge Q}{\mathcal{C} \vdash P} \qquad \frac{\mathcal{C} \vdash P \wedge Q}{\mathcal{C} \vdash Q}$$

$$\frac{\mathcal{C} \vdash P}{\mathcal{C} \vdash P \vee Q} \qquad \frac{\mathcal{C} \vdash Q}{\mathcal{C} \vdash P \vee Q} \qquad \frac{\mathcal{C} \vdash P \vee Q \qquad \mathcal{C}, P \vdash C \qquad \mathcal{C}, Q \vdash C}{\mathcal{C} \vdash C}$$

$$\frac{\mathcal{C}, P \vdash Q}{\mathcal{C} \vdash P \Rightarrow Q} \qquad \frac{\mathcal{C} \vdash P \Rightarrow Q \qquad \mathcal{C} \vdash P}{\mathcal{C} \vdash Q}$$

$$\frac{\mathcal{C}, \mathcal{X}' \vdash P}{\mathcal{C} \vdash \forall \mathcal{X}'.P} \qquad \frac{\mathcal{C} \vdash \forall \mathcal{X}'.P \qquad \mathcal{C} \vdash \gamma :: \mathcal{X}'}{\mathcal{C} \vdash \gamma P} \qquad \frac{\mathcal{C}, \mathcal{R}' \vdash P}{\mathcal{C} \vdash \forall \mathcal{R}'.P} \qquad \frac{\mathcal{C} \vdash \forall \mathcal{R}'.P \qquad \mathcal{C} \vdash \varphi :: \mathcal{R}'}{\mathcal{C} \vdash \varphi P}$$

$$\frac{\mathcal{C} \vdash \gamma :: \mathcal{X}' \qquad \mathcal{C} \vdash \gamma P}{\mathcal{C} \vdash \exists \mathcal{X}'.P} \qquad \frac{\mathcal{C} \vdash \exists \mathcal{X}'.P \qquad \mathcal{C}, \mathcal{X}', P \vdash Q}{\mathcal{C} \vdash Q}$$

$$\frac{\mathcal{C} \vdash \varphi :: \mathcal{R}' \qquad \mathcal{C} \vdash \varphi P}{\mathcal{C} \vdash \exists \mathcal{R}'.P} \qquad \frac{\mathcal{C} \vdash \exists \mathcal{R}'.P \qquad \mathcal{C}, \mathcal{R}', P \vdash Q}{\mathcal{C} \vdash Q}$$