A Relational Modal Logic for Higher-Order Stateful ADTs

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

Georg Neis
MPI-SWS
neis@mpi-sws.org

Andreas Rossberg
MPI-SWS
rossberg@mpi-sws.org

Lars Birkedal
ITU-Copenhagen
birkedal@itu.dk

Abstract

The method of *logical relations* is a classic technique for proving the equivalence of higher-order programs that implement the same observable behavior but employ different internal data representations. Although it was originally studied for pure, strongly normalizing languages like System F, it has been extended over the past two decades to reason about increasingly realistic languages. In particular, Appel and McAllester's idea of *step-indexing* has been used recently to develop syntactic Kripke logical relations for ML-like languages that mix functional and imperative forms of data abstraction. However, while step-indexed models are powerful tools, reasoning with them directly is quite painful, as one is forced to engage in tedious step-index arithmetic to derive even simple results.

In this paper, we propose a logic LADR for equational reasoning about higher-order programs in the presence of existential type abstraction, general recursive types, and higher-order mutable state. LADR exhibits a novel synthesis of features from Plotkin-Abadi logic, Gödel-Löb logic, S4 modal logic, and relational separation logic. Our model of LADR is based on Ahmed, Dreyer, and Rossberg's state-of-the-art step-indexed Kripke logical relation, which was designed to facilitate proofs of representation independence for "state-dependent" ADTs. LADR enables one to express such proofs at a much higher level, without counting steps or reasoning about the subtle, step-stratified construction of possible worlds.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; D.3.3 [*Programming Languages*]: Language Constructs and Features—Abstract data types; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords Abstract data types, step-indexed logical relations, modal logic, separation logic, Plotkin-Abadi logic, local state

1. Introduction

The method of *logical relations* is a classic technique for proving the equivalence of higher-order programs that implement the same observable behavior but employ different internal data representations. The basic idea is to lift the notion of observable equivalence at base type to one at higher type by defining a notion of "logical equivalence" inductively on the type structure of the language. The Curry-Howard isomorphism dictates which logical connective to use in assigning a relational operation to each type constructor.

Since Reynolds' seminal paper on *relational parametricity* [37], which presented logical relations for reasoning about the pure, strongly normalizing System F, there has been a lot of work on generalizing and extending the method to handle increasingly realistic languages [1, 8, 10, 23, 30, 31]. In particular, Ahmed, Dreyer, and Rossberg (hereafter, ADR) have recently developed a powerful logical relation for an ML-like language with universal and existential types, general recursive types, and higher-order mutable state [2].

The ADR logical relation is notable in two ways. First, it is a *Kripke logical relation*, which means that the relation is parameterized by *possible worlds*. These in turn allow for the encoding of invariants on *local state*, that is, mutable state that is accessible to the terms being logically related but not publicly accessible to "the rest of the program". For instance, two functions might be distinguishable in an arbitrary program context, but equivalent under the assumption that (*i.e.,* in a possible world that demands that) a particular local reference cell always stores, say, an even number.

Compared to previous Kripke logical relations for reasoning about state, the most interesting and novel aspect of ADR's possible worlds is that they allow one to establish not only invariants, but also properties about local state that *evolve* over time. ADR achieve this by instrumenting possible worlds with *populations*, which may be used to effectively encode an abstract trace of the program. As we will review in Section 3, populations are very useful when proving representation independence for "generative" or "state-dependent" ADTs, *i.e.,* ADTs whose inhabitants grow over time in correspondence with changes to some local state.

A second key aspect of the ADR logical relation is its use of Appel and McAllester's *step-indexing* technique [3]. Step-indexing is useful as a way of modeling language features like recursive types and higher-order state, whose relational interpretations are not inductive on the structure of types. The idea is to index the logical relation by a second induction metric, namely a natural number representing (roughly) the number of steps of computation for which the terms in question are indistinguishable. (Thus, if two terms are related for an *arbitrary* "step-index", we know they are really indistinguishable.) In addition to stratifying the logical relation, step-indexing is also helpful in stratifying possible worlds, which in the presence of higher-order state may be self-referential.

For the purpose of constructing logical relations for recursive types and higher-order state, step-indexing is not the only technique available. A variety of denotational techniques—minimal invariance [32], FM-cpos [8], ultrametric spaces [10]—have been and are being developed for this purpose as well. But the promise of step-indexing is that it is readily applicable in a variety of settings—*e.g.,* both in ML-like languages and low-level code [4]—and, moreover, that it is conceptually and mathematically elementary.

Unfortunately, step-indexed models are so elementary that they can be quite painful to use directly. Unlike the abovementioned denotational models, which involve some heavy mathematical con-

POPL'10, January 17–23, 2010, Madrid, Spain.

structions but in the end produce fairly clean equational reasoning principles, step-indexed logical relations force their users to engage in tedious step-index arithmetic to derive even simple results. The crux of the problem is that *you get what you pay for*: step-indexed models are fairly straightforward to construct because one doesn't have to prove anything about limits of chains of finite approximations of a relation—the model consists *only* of the finite approximations—but, as a result, the user of the model has to reason directly about the finite approximations as well.

For instance, one might hope to prove that two functions f_1 and f_2 are logically equivalent by showing that they map logically equivalent arguments to logically equivalent results. However, in a step-indexed logical relation, this proof principle does not hold.

Why? First, it is difficult to prove anything about logical *equivalence* directly because step-indexed logical relations are (for various technical reasons [1]) typically asymmetric, *i.e.,* they define *approximation* relations between terms. One can of course define logical equivalence to mean mutual approximation, but unfortunately that notion of equivalence does not enjoy the extensionality proof principle described above. In particular, showing that f_1 and f_2 map mutually-approximate arguments to mutually-approximate results does not imply that f_1 and f_2 are themselves mutually-approximate. Indeed, to show the latter, one must demonstrate that if v_1 approximates v_2, then $f_1 v_1$ approximates $f_2 v_2$, and if v_2 approximates v_1, then $f_2 v_2$ approximates $f_1 v_1$. Thus, one must effectively divide the proof of f_1 and f_2's equivalence into two, often very similar, proofs—one for each direction of approximation.

Second, when showing that f_1 and f_2 are logically related (for either direction of approximation), it does not suffice to show that f_1 and f_2 map arguments that are logically related for all steps to results that are logically related for all steps. One must show the stronger condition that, for any $n \in \mathbb{N}$, if v_1 and v_2 are logically related for n steps, then $f_1(v_1)$ and $f_2(v_2)$ are logically related for n steps as well. The step-stratified possible worlds that arise in a model like ADR only make matters worse by requiring additional quantification over future worlds throughout the proof.

Hopefully, this example suggests the pressing need to develop more abstract, high-level, step-free, equational proof principles for step-indexed Kripke logical relations.

1.1 Existing Work on Reasoning About Step-Indexed Models

Aside from the asymmetry of step-indexed binary relations, most of the issues discussed above also arise in *unary* step-indexed models.

To address them in that setting, Appel, Melliès, Richards, and Vouillon [4] have proposed the use of a modal logic with an *approximation modality* $\triangleright$ (pronounced "later") [25]. Essentially, one can view step-indices as a form of possible worlds, with smaller indices representing future worlds. Proposition P holds in world k if it holds (intuitively) for k steps of computation, and $\triangleright P$ holds in world k if P holds, "one step later", in world $k-1$. This interpretation validates the *Löb rule* derived from Gödel-Löb logic, $(\triangleright P \Rightarrow P) \Rightarrow P$, which provides a clean induction principle over step indices. In subsequent work, Hobor, Appel, *et al.* [15, 18] extend the $\triangleright$ logic with other modalities, as well as the separating conjunction $P * Q$ from *separation logic* [38], which they use for proving the semantic soundness of Concurrent Cminor in Coq. They describe multimodal separation logic as transforming their step-indexed proof effort "from infeasible to feasible".

More recently, Appel *et al.*'s approximation modality $\triangleright$ has been used to reason about *relational* step-indexed models.

Benton and Tabareau [9] prove correctness of a compiler for a simply-typed functional language, using a step-indexed model that interprets source-language types in terms of relations on low-level programs. Their proof exploits both the $\triangleright$ operator and separating conjunction [44] in order to modularize the relational constructions.

Dreyer, Ahmed, and Birkedal [16] propose LSLR, a logic for reasoning abstractly about Ahmed's step-indexed logical relations for pure call-by-value System F with recursive types [1]. The basic idea of LSLR is to incorporate the approximation modality $\triangleright$ into a variant of Plotkin and Abadi's relational logic for parametric polymorphism [33]. Plotkin-Abadi logic provides a clean way of encoding logical relations for second-order polymorphism in terms of quantification over second-order relation variables (which are primitive in the logic). Extending it with the $\triangleright$ modality enables a simple well-founded mechanism for *recursively defined relations* $\mu r.R$, with which one can then define a logical relation for general recursive types quite elegantly. Furthermore, Dreyer *et al.* derive a set of proof principles for *direct equational reasoning* by treating the direction of the step-indexed logical approximation relation as a "hidden" parameter of the LSLR judgment.

1.2 A Logic for Step-Indexed Kripke Logical Relations

In this paper, we develop LADR, a modal logic for reasoning about (a slight variant of) the ADR logical relation. Using LADR, we can express ADR-style contextual equivalence proofs at a much higher level of abstraction, avoiding low-level details about steps and possible worlds. LADR extends LSLR with a variety of features, most notably: (1) an abstract, logical characterization of ADR's possible worlds and populations, (2) the $\Box$ modality from S4 modal logic, and (3) a simple fragment of relational separation logic [44]. In the remainder of this section, we motivate these extensions.

Local Reasoning About Islands Kripke logical relations in the tradition of Pitts and Stark [31], of which ADR is one, employ possible worlds that are essentially sets of *islands*—relational properties concerning *disjoint* pieces of the heap. When we prove two terms related under a particular world, we typically only know about some small (often singleton) set of islands that exist in that world, which in turn concern some piece(s) of local state that the terms have references to. However, due to the quantifications over future worlds that arise in the proof of logical relatedness, the world in question may contain arbitrary other islands. Fortunately, we can safely ignore those other islands in the proof—because they are guaranteed to only place restrictions on other separate pieces of the heap that our terms do not directly touch or care about—but, when working directly with the model, we still have to mindlessly push the other islands around.

In LADR, to avoid such tedium, we instrument our judgments with a context describing what islands *must* exist in the current world, without placing any restrictions on what other islands *may* exist. This allows us to avoid reasoning explicitly about possible worlds. Rather, it enables us to reason *locally* about only the islands we care about, yet be assured that our results hold in the presence of arbitrary invariants concerning other pieces of the heap.

Modeling Populations As mentioned above, one of the key advances of ADR is that islands contain *populations*, which may grow in future worlds. These populations may then be used to define relational interpretations of abstract types that are *dynamic*, in the sense that they absorb more and more inhabitants over the execution of the program. In order to support the encoding of such dynamic relations in LADR, we equip each island in the island context with a *population variable* p. This p is a primitive dynamic relation representing the island's ever-growing population, which may be used as a basic building block in the definition of other dynamic relations. Population variables are sufficient to account for LADR's lack of explicit worlds, since the population of an island is the central aspect of the island that is dynamic, *i.e.,* changes in future worlds. (The heap property of an island changes as well, but the way it changes is completely determined by how the population changes.)

$$
\begin{array}{lrcl}
\text{Term \& Type Environments} & \Gamma & ::= & \cdot \mid \Gamma, \alpha \mid \Gamma, x{:}\tau \\
\text{Heap Environments} & \Sigma & ::= & \cdot \mid \Sigma, l{:}\tau \\
\text{Base Types} & & & \\
\tau_b & & ::= & \mathsf{unit} \mid \mathsf{int} \mid \mathsf{bool} \\
\text{Types} & & & \\
\tau & & ::= & \alpha \mid \tau_b \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \mid \\
 & & & \forall\alpha.\,\tau \mid \exists\alpha.\,\tau \mid \mu\alpha.\,\tau \mid \mathsf{ref}\,\tau \\
\text{Expressions} & & & \\
e & & ::= & x \mid \langle\rangle \mid l \mid n \mid e_1 = e_2 \mid e_1 \le e_2 \mid e_1 + e_2 \mid \ldots \mid \\
 & & & \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\,e\,\mathsf{then}\,e_1\,\mathsf{else}\,e_2 \mid \\
 & & & \langle e_1, e_2\rangle \mid \mathsf{fst}\,e \mid \mathsf{snd}\,e \mid \\
 & & & \mathsf{inl}\,e \mid \mathsf{inr}\,e \mid \mathsf{case}\,e\,\mathsf{of}\,\mathsf{inl}\,x_1 \Rightarrow e_1 \mid \mathsf{inr}\,x_2 \Rightarrow e_2 \mid \\
 & & & \lambda x{:}\tau.\,e \mid e_1\,e_2 \mid \Lambda\alpha.\,e \mid e\,\tau \mid \\
 & & & \mathsf{pack}\,\tau_1, e\,\mathsf{as}\,\exists\alpha.\,\tau \mid \mathsf{unpack}\,e_1\,\mathsf{as}\,\alpha, x\,\mathsf{in}\,e_2 \mid \\
 & & & \mathsf{roll}\,e \mid \mathsf{unroll}\,e \mid \mathsf{ref}\,e \mid\, !e \mid e_1 := e_2 \mid e_1 == e_2 \\
\text{Values} & & & \\
v & & ::= & \langle\rangle \mid l \mid n \mid \mathsf{true} \mid \mathsf{false} \mid \langle v_1, v_2\rangle \mid \mathsf{inl}\,v \mid \mathsf{inr}\,v \mid \\
 & & & \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,e \mid \mathsf{pack}\,\tau_1, v\,\mathsf{as}\,\exists\alpha.\,\tau \mid \mathsf{roll}\,v
\end{array}
$$

Figure 1. Syntax of $\mathsf{F}^{\mu!}$

Monotonicity and the $\Box$ Modality A key notion in Kripke models is *world extension*, aka the accessibility relation between worlds. In ADR, there are three ways in which a future world W_2 can extend a starting world W_1: time extension, width extension, and depth extension. Time extension means that W_2's step-index may be smaller than W_1's. Width extension means that W_2 may contain more islands than W_1. Depth extension means that the population of an island in W_2 may be larger than the population of the same island (if it exists) in W_1.

When reasoning about logical relations, it seems that all the propositions we are interested in are *monotone* with respect to time and width extension, so we build those forms of monotonicity into our model of LADR. However, some propositions of interest are not monotone with respect to depth extension. In particular, in order to reason about extending the population of an island, we need to have a way of characterizing precisely what is in the population *currently*, even though the population may grow to contain more elements in future worlds. Thus, we do not build depth monotonicity into our model, but rather represent it explicitly using the necessity operator $\Box$ from S4 modal logic. That is, if a proposition P holds in world W, it must hold in all future worlds that extend W by time and width, but not depth. If $\Box P$ holds in world W, then it must hold in all future worlds of W, period.

Encoding Heap Relations in Relational Separation Logic Ultimately, the main goal of islands is to encode relations between the heaps of the two programs being logically related. In LADR, we express these heap relations using an *intuitionistic* fragment of Yang's *relational separation logic* [44]. Separation logic was developed by Reynolds, O'Hearn, and others as a way of generalizing Hoare logic to account for local reasoning about shared mutable data structures [38]. Yang's relational version, which is closely related to Benton's relational Hoare logic [7], allows for Hoare-style reasoning about the *equivalence* of two pointer programs using Hoare *quadruples*. For our purposes, when proving that two terms are logically related, relational separation logic is useful as a way of reasoning cleanly about their execution in related heaps. Furthermore, the separating conjunction $*$ is useful for joining the heap relations of multiple islands together into a single relation.

2. The Programming Language

The programming language we will be reasoning about is $\mathsf{F}^{\mu!}$, whose syntax is given in Figure 1.

$\mathsf{F}^{\mu!}$ is a completely standard polymorphic λ-calculus, extended with primitive support for product, sum, existential, recursive, and ML-style reference types. Equality testing for base types τ_b is written $e_1 = e_2$, whereas pointer equality is written $e_1{=}{=}e_2$. $\mathsf{F}^{\mu!}$ has a standard left-to-right call-by-value operational semantics (not shown here), specified in the style of Felleisen and Hieb, with E denoting an evaluation context. It is easy to define a notion of contextual equivalence for $\mathsf{F}^{\mu!}$, written $\Gamma; \Sigma \vdash e_1 \approx^{ctx} e_2 : \tau$, which asserts that e_1 and e_2 co-terminate when placed in any closing $\mathsf{F}^{\mu!}$ context of the appropriate type. For space reasons, we refer the reader to the online appendix for further details [17].

3. Key Features of the ADR Logical Relation

In this section, we review the key features of the ADR logical relation, and briefly sketch how we support them in LADR.

3.1 The Method of Populations

Consider the "twin abstraction" example (from ADR), in which we want to prove the programs e_1 and e_2 contextually equivalent:

$$
\begin{array}{rcl}
\tau & = & \exists\alpha,\beta.\,(\mathsf{unit} \to \alpha) \times (\mathsf{unit} \to \beta) \times (\alpha \times \beta \to \mathsf{bool}) \\
C & = & \mathsf{let}\,x = \mathsf{ref}\ 0\,\mathsf{in} \\
 & & \mathsf{pack}\,\mathsf{int}, \mathsf{int}, \langle \lambda\_.{+}{+}x, \lambda\_.{+}{+}x, \lambda y.[\bullet]\rangle\,\mathsf{as}\,\tau \\
e_1 & = & C[\mathsf{fst}\,y = \mathsf{snd}\,y] \\
e_2 & = & C[\mathsf{false}]
\end{array}
$$

Both e_1 and e_2 implement an ADT for a simple symbol generator. Internally, they both represent symbols as integers, and create new symbols via a local integer pointer x. The first two functions they export generate symbols of abstract type α and β, respectively. The two functions are implemented in the same way, by bumping up the counter ($+{+}x$ is short for $(x := x{+}1; !x)$) and returning the current value as a "fresh" symbol. The third function tests whether a symbol of type α and a symbol of type β are equal. In e_1's implementation, there is an integer equality test, but e_2's implementation always returns false.

Intuitively, the reason e_1 and e_2 are equivalent is that, whenever a new symbol is generated, it either becomes a value of type α or of type β, but not both. Thus, an equality test between values of type α and β must always fail. However, formalizing this intuition is tricky: when proving the implementations equivalent, what are the right relational interpretations to choose for α and β? Since we don't know ahead of time which integers will become α's and which will become β's, how can we characterize up front what it means to inhabit α vs. β?

The ADR logical relation solves this problem by introducing *populations* into its model of possible worlds. The basic idea is as follows: When we allocate a fresh piece of local state, we get to extend the current world with a new island governing how that state is maintained. In ADR, the island may contain a population (formalized as a value relation), which can grow in future worlds and be used to track some knowledge about the *history* of the local state. The island also contains a *law*, specifying (1) what populations are legally valid, and (2) what relational properties the local state must satisfy depending on what the population is.

For the twin abstraction example, the island governing the local counter x would include an (initially empty) population tracking (1) how many symbols have been generated so far and (2) which of those symbols are α's vs. β's. Formally, this can be encoded as the relational map of some total function from $\{1, \ldots, n\}$ to $\{1, 2\}$ (for some n). The law of the island would demand that, whenever the size of the population relation is n, the island's heap relation asserts that x points to n in both programs.

Finally, we can now define relational interpretations for α and β that are dependent on the population of the island governing x.

Specifically, we can say that v_1 and v_2 are related at α under world W if there exists a positive integer n such that $v_1 = v_2 = n$ and $(n, 1)$ is a member of the population of the island governing x in W. Similarly, they are related at β if $(n, 2)$ is in the population. Note that, by this definition, there is no way v_1 and v_2 can be related both at α and β (in the same world W), which is what we needed to establish in order to prove the equivalence of e_1 and e_2. Note also that the interpretations of α and β are examples of what we described in the Introduction as *dynamic* relations, *i.e.,* they grow in future worlds in accordance with changes to some local state.

In LADR, to express relations (like those for α and β) that depend on the current population of an island, we employ *population variables*. As described in the Introduction, we attach a population variable p to every island bound in the island context of our logical judgment. Terms e_1 and e_2 are related by p in world W iff (e_1, e_2) is in the population of the island corresponding to p in W. Returning to the twin abstraction example, assuming p is the variable corresponding to the island governing the counter x, we can define the relation for α in LADR as $(x_1, x_2).\ x_1 = x_2 \wedge (x_1, 1) \in p$, and the relation for β as $(x_1, x_2).\ x_1 = x_2 \wedge (x_1, 2) \in p$. Thus, we avoid ever talking explicitly about the world W.

3.2 Step-Stratified Worlds for Higher-Order State

When are two values v_1 and v_2 logically related at the type $\mathsf{ref}\ \tau$? Intuitively, the answer is: when they are memory locations l_1 and l_2 whose contents are currently logically related at type τ and, moreover, will *continue* to be related at type τ in the future. This strong condition on l_1 and l_2's contents is necessary in order to ensure that dereferencing and assignment preserve logical relatedness. The condition is also easy to express using an island. In particular, ADR define l_1 and l_2 to be logically related at $\mathsf{ref}\ \tau$ in world W if W contains an island with a heap invariant ensuring that the contents of those locations are logically related at type τ.

Now, in order for a heap invariant to specify that the contents of l_1 and l_2 are logically related at type τ, it must also specify the world in which that relation is considered. (If our language only supported first-order state, and τ were restricted to be a base type like int, the world would be irrelevant, but in the presence of higher-order state, the world matters.) Ideally, we would choose it to be whatever is the "current" world. However, since the "current" world changes all the time (as the world is extended), ADR instead *parameterize* heap relations over the world in which they are considered. Then, when checking whether two heaps satisfy the demands of a world W, the heap relation in each island of W is instantiated to the current world (namely, W).

Unfortunately, it's not quite that "simple". It is easy to see by a simple cardinality argument that we cannot construct worlds that contain heap relations parameterized by worlds. This is where step-indexing comes in. ADR stratify worlds by a step-index, so that the heap relations in k-indexed worlds are parameterized by $(k-1)$-indexed worlds. This is sufficient for encoding the logical relation at type $\mathsf{ref}\ \tau$ (as described above) because it takes a step of computation to observe the contents of l_1 and l_2, and so we only need to know their contents are logically related "one step later".

In LADR, much of this technical detail is kept hidden away from the user of the logic and buried in the model. The heap relations that appear in LADR islands are not explicitly parameterized over worlds because the logic does not talk about worlds explicitly at all. To express logically the restriction that an island heap relation H can only talk about logical relatedness of heap contents "one step later", we require syntactically that H be what we call *delayed*. This means essentially that any world-dependent propositions contained within H (such as logical relatedness at an arbitrary higher type τ) must appear under the "later" modality ($\triangleright$).

Absolute Relation Variables	a, b	$\in$	$\mathrm{AbsRelVar}$
Normal Relation Variables	p, q, r, s	$\in$	RelVar
Variable Contexts	$\mathcal{X}$	$::=$	$\cdot \mid \mathcal{X}, \alpha \mid \mathcal{X}, x$
Relation Contexts	$\mathcal{R}$	$::=$	$\cdot \mid \mathcal{R}, a \mid \mathcal{R}, p$
Island Contexts	$\mathcal{L}$	$::=$	$\cdot \mid \mathcal{L}, p \propto a.(B, H)$
Hypothesis Contexts	$\mathcal{P}$	$::=$	$\cdot \mid \mathcal{P}, P$
Joint Contexts	$\mathcal{C}$	$::=$	$\mathcal{X}; \mathcal{R}; \mathcal{L}; \mathcal{P}$

Absolute Relations

$$A, B \ ::= \ e_1 = e_2 \mid e_1 \rightsquigarrow^0 e_2 \mid e_1 \rightsquigarrow^1 e_2 \mid e_1 \rightsquigarrow^* e_2 \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid \forall \mathcal{X}.A \mid \exists \mathcal{X}.A \mid \forall \mathcal{R}.A \mid \exists \mathcal{R}.A \mid \overline{e} \in A \mid a \mid \overline{x}.A \mid \mathrm{Val} \mid \mathrm{Const}_{\tau_b} \mid \mathrm{Loc}$$

Normal Relations

$$P, Q, R, S \ ::= \ A \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid \forall \mathcal{X}.P \mid \exists \mathcal{X}.P \mid \forall \mathcal{R}.P \mid \exists \mathcal{R}.P \mid \propto a.(B, H) \mid \triangleright P \mid \Box P \mid \overline{e} \in R \mid r \mid \overline{x}.P \mid \mu r.R \mid \uparrow R \mid \mathrm{Term}_i \mid H \Rrightarrow J$$

Heap Relations

$$H, J \ ::= \ e_1 \hookrightarrow_i e_2 \mid H * J \mid H \vee J \mid \exists \mathcal{X}.H \mid \Box P$$

Figure 2. Syntax of LADR

$$\mathrm{Val}_i \stackrel{\mathrm{def}}{=} x.\ x \in \mathrm{Term}_i \wedge x \in \mathrm{Val}$$

$$R : \mathrm{VRel} \stackrel{\mathrm{def}}{=} \Box(\forall x_1, x_2.\ (x_1, x_2) \in R \Rightarrow x_1 \in \mathrm{Val}_1 \wedge x_2 \in \mathrm{Val}_2)$$

$$R : \mathrm{Type} \stackrel{\mathrm{def}}{=} R : \mathrm{VRel} \wedge \Box(\forall x_1, x_2.\ (x_1, x_2) \in R \Rightarrow \Box(x_1, x_2) \in R)$$

$$(x_1 \in R_1, x_2 \in R_2).P \stackrel{\mathrm{def}}{=} (x_1, x_2).\ x_1 \in R_1 \wedge x_2 \in R_2 \wedge P$$

$$\dagger(\mathcal{X}; \mathcal{R}; \mathcal{L}; \overline{P}) \stackrel{\mathrm{def}}{=} \mathcal{X}; \mathcal{R}; \mathcal{L}; \overline{\dagger P}$$
$$\dagger(\Box P) \stackrel{\mathrm{def}}{=} \Box P \qquad \dagger P \stackrel{\mathrm{def}}{=} \top \ \ (\text{if } P \neq \Box P')$$

$$\ast_\epsilon \stackrel{\mathrm{def}}{=} \Box\top \qquad \ast H, \overline{H} \stackrel{\mathrm{def}}{=} H * (\ast \overline{H})$$

$$\triangleleft(\mathcal{X}; \mathcal{R}; \mathcal{L}; \overline{P}) \stackrel{\mathrm{def}}{=} \mathcal{X}; \mathcal{R}; \mathcal{L}; \overline{\triangleleft P}$$
$$\triangleleft(\triangleright P) \stackrel{\mathrm{def}}{=} P \qquad \triangleleft P \stackrel{\mathrm{def}}{=} P \ \ (\text{if } P \neq \triangleright P')$$

$$\bigwedge_\epsilon \stackrel{\mathrm{def}}{=} \top \qquad \bigwedge P, \overline{P} \stackrel{\mathrm{def}}{=} P \wedge (\bigwedge \overline{P})$$

Figure 3. Auxiliary Notation for the Logic

4. LADR: Syntax

The syntax of our logic LADR is given in Figure 2.

LADR is a relational second-order intuitionistic modal logic, supporting relations of arbitrary arity. *Propositions* are just nullary relations. While P, Q, R, and S may denote any kind of relation, we will typically use P and Q to represent propositions, and R and S to represent binary relations. *Absolute* relations A and B are a useful syntactic subcategory of relations; such relations are "absolute" in the sense that their meaning essentially does not depend on the world in which they are considered. We say "essentially" because we consider everything to be true in a trivial world (*i.e.,* world with step index 0). Thus, an absolute proposition A is true in any *non-trivial* world iff A is true in all non-trivial worlds.

Most of the atomic relations in LADR are absolute. $e_1 = e_2$ denotes syntactic equality. $e_1 \rightsquigarrow^* e_2$ says that e_1 reduces to e_2 in an arbitrary number of pure (non-heap-dependent) reduction steps. $e_1 \rightsquigarrow^0 e_2$ says that $e_1 \rightsquigarrow^* e_2$ without making any $\mathsf{unroll\text{-}roll}$ reductions. $e_1 \rightsquigarrow^1 e_2$ says that $e_1 \rightsquigarrow^* e_2$, making *exactly one* $\mathsf{unroll\text{-}roll}$ reduction. These latter two propositions are borrowed from LSLR [16]; the distinction they make between $\mathsf{unroll\text{-}roll}$ and other reductions is useful because our logical relation for terms (defined below in Section 5) is closed under β-reduction and other forms of pure reduction, but not under $\mathsf{unroll\text{-}roll}$ reduction.

A term e is in Val if e is a value, in Const_{τ_b} if e is a constant of base type τ_b, and in Loc if e is a memory location l. The predicate Term_i describes terms whose free locations are bound in the heap

$$
\begin{array}{rcl}
\mathcal{V}[\![\alpha]\!]\rho & \stackrel{\text{def}}{=} & \rho(\alpha) \\
\mathcal{V}[\![\tau_b]\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Const}_{\tau_b}, x_2 \in \text{Const}_{\tau_b}).\ x_1 = x_2 \\
\mathcal{V}[\![\tau' \times \tau'']\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \exists x_1', x_1'', x_2', x_2''.\ x_1 = \langle x_1', x_1'' \rangle \wedge x_2 = \langle x_2', x_2'' \rangle \wedge (x_1', x_2') \in \mathcal{V}[\![\tau']\!]\rho \wedge (x_1'', x_2'') \in \mathcal{V}[\![\tau'']\!]\rho \\
\mathcal{V}[\![\tau' + \tau'']\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ (\exists x_1', x_2'.\ x_1 = \mathsf{inl}\, x_1' \wedge x_2 = \mathsf{inl}\, x_2' \wedge (x_1', x_2') \in \mathcal{V}[\![\tau']\!]\rho) \vee \\
& & \qquad (\exists x_1'', x_2''.\ x_1 = \mathsf{inr}\, x_1'' \wedge x_2 = \mathsf{inr}\, x_2'' \wedge (x_1'', x_2'') \in \mathcal{V}[\![\tau'']\!]\rho) \\
\mathcal{V}[\![\tau' \to \tau'']\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \Box(\forall y_1, y_2.\ (y_1, y_2) \in \mathcal{V}[\![\tau']\!]\rho \Rightarrow (x_1\, y_1, x_2\, y_2) \in \mathcal{E}[\![\tau'']\!]\rho) \\
\mathcal{V}[\![\forall\alpha.\,\tau]\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \Box(\forall\alpha_1, \alpha_2.\ \forall r.\ r : \text{Type} \Rightarrow (x_1\, \alpha_1, x_2\, \alpha_2) \in \mathcal{E}[\![\tau]\!]\rho, \alpha{\mapsto}r) \\
\mathcal{V}[\![\exists\alpha.\,\tau]\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \exists\alpha_1, \alpha_2, \alpha_1', \alpha_2'.\ \exists y_1, y_2.\ \exists r.\ r : \text{Type} \wedge \\
& & \qquad x_1 = \mathsf{pack}\,\alpha_1, y_1\,\mathsf{as}\,\alpha_1' \wedge x_2 = \mathsf{pack}\,\alpha_2, y_2\,\mathsf{as}\,\alpha_2' \wedge (y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho, \alpha{\mapsto}r \\
\mathcal{V}[\![\mu\alpha.\,\tau]\!]\rho & \stackrel{\text{def}}{=} & \mu r.\ (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \exists y_1, y_2.\ x_1 = \mathsf{roll}\, y_1 \wedge x_2 = \mathsf{roll}\, y_2 \wedge \triangleright(y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho, \alpha{\mapsto}r \\
\mathcal{V}[\![\mathsf{ref}\,\tau]\!]\rho & \stackrel{\text{def}}{=} & (x_1 \in \text{Val}_1, x_2 \in \text{Val}_2).\ \propto a.(a \equiv \{(x_1, x_2)\}, \exists y_1, y_2.\ x_1 \hookrightarrow_1 y_1 * x_2 \hookrightarrow_2 y_2 * \Box\triangleright(y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho) \\
\mathcal{E}[\![\tau]\!]\rho & \stackrel{\text{def}}{=} & \uparrow\mathcal{V}[\![\tau]\!]\rho
\end{array}
$$

Figure 4. Syntactic Logical Relation for $\mathsf{F}^{\mu!}$

for program i (where i ranges over $\{1, 2\}$ with 1 representing the program on the "left" and 2 representing the program on the "right" of the logical relation). As new locations can get allocated during the execution of the programs (and thus in future worlds), Term_i is not an absolute relation. We write Val_i to denote $\text{Term}_i \cap \text{Val}$.

Truth, falsehood, conjunction, disjunction, and implication are all standard. The first-order $\forall\mathcal{X}.P$ and $\exists\mathcal{X}.P$ quantify over variable contexts $\mathcal{X}$, which bind type variables α, β and term variables x, y (which may or may not be values). The second-order $\forall\mathcal{R}.P$ and $\exists\mathcal{R}.P$ quantify over relation contexts $\mathcal{R}$, which bind normal and absolute relation variables (p, q, r, s and a, b, respectively).

In the sequel, we use $\equiv$ and $\subseteq$ to denote relational equivalence and inclusion, defined logically in the obvious way. We also use set notation (*e.g.,* $\{\}$, $\cup$, $\cap$) as shorthand for the corresponding LADR relations and relational operations, all defined logically in the obvious way.

The proposition $\propto a.(B, H)$ asserts that in the current world there is an island with *population law* B and *heap law* H, where the absolute relation variable a is bound in both B and H. Given a population (represented by a), the population law says whether a is a valid population for the island (*i.e., A* is valid if $B[A/a]$ is true). Note that this law is absolute and does not change in future worlds. The heap law specifies the heap relation of the island as a function of the current population a. For the $\propto$ proposition to be well-formed, the heap law H must be *delayed*, a notion which we discuss in conjunction with heap relations below.

Closely related to the $\propto$ proposition is the notion of an *island context* $\mathcal{L}$. We use island contexts to keep track of the islands in the world that we know and care about. In addition to specifying the law for each island, $\mathcal{L}$ associates with each island a population variable p, as we discussed in Section 3.1.

The modality $\triangleright P$ says that P holds "one step later" (but not necessarily now), a notion we make formal in Section 6. The modality $\Box P$ says that P holds now and in all future worlds.

$\overline{x}.P$ denotes the relation $\{(\overline{x}) \mid P\}$, and $\overline{e} \in R$ holds if $\overline{e}$ belongs to the relation R. (Here, and throughout, we write $\overline{\text{foo}}$ to denote a sequence of zero or more foo's, separated by commas.)

The recursive relation $\mu r.R$ is useful in defining the logical relation for recursive types. To be semantically well-founded, we insist that R must be *contractive* in r, meaning that it may only mention r under a $\triangleright$ modality. Thus, the apparently circular meaning of $\mu r.R$ is really inductive in the step-indices that stratify worlds.

The relation $\uparrow R$ denotes the *lifting* of a binary value relation R to a term relation. Roughly, $(e_1, e_2) \in \uparrow R$ if e_1 and e_2, when evaluated under any heaps h_1 and h_2 (respectively) that satisfy the demands of the current world, either both terminate or both diverge. In the case that they terminate, there must exist some future world such that the resulting heaps satisfy that future world and the resulting values are related by R in that future world. This intuitive description of $\uparrow R$ leaves out the details of how step-indices play into the picture, which will become clear when we present the model in Section 6.

Last among the propositions is $H \Rrightarrow J$, which denotes heap relation *entailment*. The *heap relations* denoted by H and J express relations between the heaps of the two programs we are reasoning about. Instead of incorporating explicit heap objects into LADR and reasoning about them directly, we rely on primitives of intuitionistic separation logic to express our heap relations. Thus, the entailment $H \Rrightarrow J$ has the usual separation logic interpretation—*i.e.,* any pair of heaps that satisfy H must also satisfy J.

The *points-to* relation $e \hookrightarrow_i e'$ relates two heaps h_1 and h_2 if e is a location in the domain of heap h_i, e' is in Val_i, and $h_i(e) = e'$. This relation is intuitionistic in that it makes no restrictions on heap $h_{\neg i}$ (where $\neg i = 3 - i$), and does not require that e be the only location in the domain of h_i. The *separating conjunction* $H_1 * H_2$ relates h_1 and h_2 if h_1 and h_2 can be split into disjoint pieces such that H_1 relates one piece and H_2 relates the other.

Disjunction and existential quantification are standard. Finally, we come to $\Box P$, which embeds the proposition language of our logic in the heap relation language. $\Box P$ is like a pure assertion in traditional separation logic in that it ignores the heaps entirely. It is explicitly $\Box$'d to ensure that all heap relations are (by construction) monotone w.r.t. all forms of world extension. This requirement, carried over from ADR, ensures that when we grow the population on one island of a world—a form of *depth extension* (see the Introduction)—we do not violate heap properties on other islands.

For general heap relations, there is no restriction on subformulas of the form $\Box P$. However, for the heap relations appearing in island heap laws (either in a $\propto$ proposition or an island context $\mathcal{L}$), we insist that P be *delayed*. Semantically, this means that the meaning of P only depends on what the world looks like "one step later". As a syntactic approximation of this semantic criterion, we define P to be delayed if all non-absolute constructs in it (with the exception of Term_i) appear under a $\triangleright$ modality. When we give our model of LADR, the delayed restriction will enable us to interpret an island heap law by a semantic heap relation that is indexed by worlds of a lower step-index. This is critical for the step-stratified construction of worlds, as we explained in Section 3.2.

The full definition of syntactic well-formedness for all relations and contexts of LADR is formalized in the online appendix [17].

$$
\begin{array}{rcl}
\mathit{HeapAtom}_n & \stackrel{\text{def}}{=} & \{(W, h_1, h_2) \mid W \in \mathit{World}_n \wedge h_1, h_2 \in \mathit{Heap}\} \\
\mathit{HeapRel}_n & \stackrel{\text{def}}{=} & \{\psi \subseteq \mathit{HeapAtom}_n \mid \forall (W, h_1, h_2) \in \psi.\, \forall W' \sqsupseteq W.\, (W', h_1, h_2) \in \psi\} \\
\mathit{Island}_n & \stackrel{\text{def}}{=} & \{(CP, PL, HL) \mid CP \in PL \subseteq \mathcal{P}(\mathrm{Term}^k) \wedge HL \in \mathcal{P}(\mathrm{Term}^k) \to \mathit{HeapRel}_n\} \\
\mathit{World}_n & \stackrel{\text{def}}{=} & \{(k, d, \varsigma_1, \varsigma_2, \mathcal{I}) \mid k < n \wedge d \in \{\to, \leftarrow\} \wedge \varsigma_1, \varsigma_2 \subseteq \mathrm{Loc} \wedge \mathcal{I} \in \mathit{IslandName} \xrightarrow{\text{fin}} \mathit{Island}_k\} \\
\mathit{SemRel}^k & \stackrel{\text{def}}{=} & \{\Psi \in \mathit{World} \to \mathcal{P}(\mathrm{Term}^k) \mid \forall W, W' \in \mathit{World}.\, W' \supseteq W \Rightarrow \Psi(W') \supseteq \Psi(W)\}
\end{array}
$$

$$
\begin{array}{rcl}
W' \supseteq W & \stackrel{\text{def}}{=} & W'.k \le W.k \;\wedge\; W'.d = W.d \;\wedge\; W'.\varsigma_1 \supseteq W.\varsigma_1 \;\wedge\; W'.\varsigma_2 \supseteq W.\varsigma_2 \;\wedge\; W'.\mathcal{I} \supseteq \lfloor W.\mathcal{I} \rfloor_{W'.k} \\
W' \sqsupseteq W & \stackrel{\text{def}}{=} & W'.k \le W.k \;\wedge\; W'.d = W.d \;\wedge\; W'.\varsigma_1 \supseteq W.\varsigma_1 \;\wedge\; W'.\varsigma_2 \supseteq W.\varsigma_2 \;\wedge\; W'.\mathcal{I} \sqsupseteq \lfloor W.\mathcal{I} \rfloor_{W'.k} \\
\mathcal{I}' \sqsupseteq \mathcal{I} & \stackrel{\text{def}}{=} & \forall \iota \in \mathrm{dom}(\mathcal{I}).\; \mathcal{I}'(\iota).CP \supseteq \mathcal{I}(\iota).CP \;\wedge\; \mathcal{I}'(\iota).PL = \mathcal{I}(\iota).PL \;\wedge\; \mathcal{I}'(\iota).HL = \mathcal{I}(\iota).HL
\end{array}
$$

$$
\begin{array}{rcl}
\triangleright W & \stackrel{\text{def}}{=} & (W.k - 1, W.d, W.\varsigma_1, W.\varsigma_2, \lfloor W.\mathcal{I} \rfloor_{W.k-1}) \\
\triangleleft W & \stackrel{\text{def}}{=} & (W.k + 1, W.d, W.\varsigma_1, W.\varsigma_2, W.\mathcal{I}) \\
\mathrm{pop}(\iota) & \stackrel{\text{def}}{=} & \lambda W.\, W.\mathcal{I}(\iota).CP
\end{array}
\qquad
\begin{array}{rcl}
\lfloor \mathcal{I} \rfloor_k & \stackrel{\text{def}}{=} & \lambda \iota. \lfloor \mathcal{I}(\iota) \rfloor_k \\
\lfloor (CP, PL, HL) \rfloor_k & \stackrel{\text{def}}{=} & (CP, PL, \lfloor HL \rfloor_k) \\
\lfloor HL \rfloor_k & \stackrel{\text{def}}{=} & \lambda CP. \lfloor HL(CP) \rfloor_k \\
\lfloor \psi \rfloor_k & \stackrel{\text{def}}{=} & \{(W, h_1, h_2) \in \psi \mid W.k < k\}
\end{array}
$$

$$
h_1, h_2 : W \;\stackrel{\text{def}}{=}\; \vdash h_1 \;\wedge\; \vdash h_2 \;\wedge\; \mathrm{dom}(h_1) \supseteq W.\varsigma_1 \;\wedge\; \mathrm{dom}(h_2) \supseteq W.\varsigma_2 \;\wedge\; \exists h'_1, h'_2.\; h'_1 \subseteq h_1 \;\wedge\; h'_2 \subseteq h_2 \;\wedge\; h'_1, h'_2 :_{\mathrm{dom}(W.\mathcal{I})} W
$$

$$
\begin{array}{rcl}
h_1, h_2 :_\omega W & \stackrel{\text{def}}{=} & W.k > 0 \;\Rightarrow\; \exists h^1_1, \ldots, h^n_1, h^1_2, \ldots, h^n_2.\; h_1 = h^1_1 \uplus \cdots \uplus h^n_1 \;\wedge\; h_2 = h^1_2 \uplus \cdots \uplus h^n_2 \;\wedge \\
& & \qquad \omega = \{\iota_1, \ldots, \iota_n\} \;\wedge\; \forall k \in \{1, \ldots, n\}.\; (\triangleright W, h^k_1, h^k_2) \in W.\mathcal{I}(\iota_k).HL(W.\mathcal{I}(\iota_k).CP)
\end{array}
$$

$$
\begin{array}{rcl}
W \vdash (h_1; e_1) \approx (h_2; e_2) : \Psi & \stackrel{\text{def}}{=} & \mathrm{FL}(e_1) \subseteq W.\varsigma_1 \;\wedge\; \mathrm{FL}(e_2) \subseteq W.\varsigma_2 \;\wedge \\
& & (W.d = \to \;\wedge\; \forall j < W.k.\, \forall h'_1, e'_1.\, (h_1; e_1) \Downarrow^j (h'_1; e'_1) \Rightarrow \\
& & \qquad \exists h'_2, e'_2, W'.\, W'.k = W.k - j \wedge W' \sqsupseteq W \wedge (h_2; e_2) \Downarrow (h'_2; e'_2) \wedge \Psi W'(e'_1, e'_2) \wedge h'_1, h'_2 : W') \;\vee \\
& & (W.d = \leftarrow \;\wedge\; \forall j < W.k.\, \forall h'_2, e'_2.\, (h_2; e_2) \Downarrow^j (h'_2; e'_2) \Rightarrow \\
& & \qquad \exists h'_1, e'_1, W'.\, W'.k = W.k - j \wedge W' \sqsupseteq W \wedge (h_1; e_1) \Downarrow (h'_1; e'_1) \wedge \Psi W'(e'_1, e'_2) \wedge h'_1, h'_2 : W')
\end{array}
$$

Figure 5. Worlds

5. LADR: A Logical Relation for $\mathsf{F}^{\mu!}$

Following Plotkin and Abadi [33], we do not bake in logical relations for $\mathsf{F}^{\mu!}$ as primitive notions in LADR, for they are already expressible directly in terms of existing constructs. Figure 4 defines a logical relation for $\mathsf{F}^{\mu!}$ syntactically in terms of LADR relations.

$\mathcal{V}[\![\tau]\!]\rho$ defines the logical relation on values of type τ, where ρ maps the free variables of τ to their interpretations as LADR relations. $\mathcal{E}[\![\tau]\!]\rho$ defines the logical relation for terms, which is just the lifting (via the primitive $\uparrow$ operation) of the one for values. The definition is inductive on the structure of τ.

For the most part, the definition is straightforward, interpreting each type using the appropriate logical connective according to the Curry-Howard correspondence. A key property of $\mathcal{V}[\![\tau]\!]\rho$ is that it is monotone w.r.t. all forms of world extension (formally, that $\mathcal{V}[\![\tau]\!]\rho$: Type as defined in Figure 3, under the assumption that $\forall \alpha \in \mathrm{dom}(\rho).\ \rho(\alpha)$: Type). Thus, in the cases of function and universal types, the relations are explicitly $\Box$'d to ensure monotonicity, and in the cases of universal and existential types, we require the universally or existentially quantified relation variable r to satisfy r : Type.

Recursive types are interpreted, quite naturally, as recursive relations. Moreover, by using the recursive relation primitive, we can define $\mathcal{V}[\![\mu\alpha.\tau]\!]\rho$ inductively on the type. Due to the syntactic restriction that the body of a recursive relation be contractive in the recursive relation variable r, we define $\mathsf{roll}\, v_1$ and $\mathsf{roll}\, v_2$ to be logically related (at $\mu\alpha.\tau$) if v_1 and v_2 are related (at $\tau[\mu\alpha.\tau/\alpha]$) *later*. This is typical of a step-indexed model. Intuitively, it makes sense because it takes one unroll-roll reduction step to reduce the equivalence of $\mathsf{roll}\, v_1$ and $\mathsf{roll}\, v_2$ to the equivalence of v_1 and v_2, so if the latter are related for n steps, the former should be related for $n + 1$ steps.

Reference types are interpreted in essentially the way we described in Section 3.2. Namely, two locations are related at $\mathsf{ref}\ \tau$ if there exists an island in the world whose heap law asserts that their contents are related at type τ one step later. (Note: the $\Box$ in front of the $\triangleright$ is only there because $\triangleright(y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho$ is a "normal" proposition, which must therefore be explicitly $\Box$'d in order to appear inside the heap law.) The population law $a \equiv \{(x_1, x_2)\}$ asserts that the population relation is fixed and equals the singleton relation relating the two locations. While this is not a very interesting use of populations, we find it useful as a way of distinguishing the island for one location from another (see the proof of compatibility for $==$ in the online appendix [17]).

6. LADR: Model

Our model for LADR is based closely on the ADR model. We interpret normal LADR relations into *semantic relations*, which are simply relations on closed terms (possibly with free locations) indexed by possible worlds. (Absolute relations may be interpreted straightforwardly as ordinary relations with no world index, and we give such an interpretation in the online appendix [17].)

6.1 Worlds

Possible worlds are defined in Figure 5, along with some auxiliary notation. A world is a 5-tuple $(k, d, \varsigma_1, \varsigma_2, \mathcal{I})$ consisting of a step level k, a direction parameter d, sets of valid locations ς_1 and ς_2 (for the left and right terms, respectively), and an island map $\mathcal{I}$. The direction parameter d, ranging over $\{\to, \leftarrow\}$, describes which direction of logical approximation we are proving (*i.e.,* that the term on the left approximates the one on the right or vice versa). By building the direction in as a parameter of the world, we ensure that proofs of logical relatedness establish logical *equivalence*.

An island map associates island names with islands of the form (CP, PL, HL), where CP is the current population of the island, PL is the population law, and HL is the heap law. Formally, the current population is just a term relation (of arbitrary arity), PL is a set of such populations, and HL is a function from populations to heap relations. Heap relations, in turn are indexed by worlds, but

$\boxed{[\![P]\!]\delta W\overline{e}}$

$$[\![P]\!]\delta W\overline{e} \stackrel{\text{def}}{=} \top \quad \text{if } W.k = 0. \text{ Otherwise:}$$

$$\begin{aligned}
[\![\top]\!]\delta W &\stackrel{\text{def}}{=} \top \\
[\![\bot]\!]\delta W &\stackrel{\text{def}}{=} \bot \\
[\![P \wedge Q]\!]\delta W &\stackrel{\text{def}}{=} [\![P]\!]\delta W \wedge [\![Q]\!]\delta W \\
[\![P \vee Q]\!]\delta W &\stackrel{\text{def}}{=} [\![P]\!]\delta W \vee [\![Q]\!]\delta W \\
[\![P \Rightarrow Q]\!]\delta W &\stackrel{\text{def}}{=} \forall W' \sqsupseteq W.\ [\![P]\!]\delta W' \Rightarrow [\![Q]\!]\delta W' \\
[\![\forall \mathcal{X}.P]\!]\delta W &\stackrel{\text{def}}{=} \forall \gamma \in [\![\mathcal{X}]\!].\ [\![\gamma P]\!]\delta W \\
[\![\exists \mathcal{X}.P]\!]\delta W &\stackrel{\text{def}}{=} \exists \gamma \in [\![\mathcal{X}]\!].\ [\![\gamma P]\!]\delta W \\
[\![\forall \mathcal{R}.P]\!]\delta W &\stackrel{\text{def}}{=} \forall \delta' \in [\![\mathcal{R}]\!].\ [\![P]\!](\delta, \delta')W \\
[\![\exists \mathcal{R}.P]\!]\delta W &\stackrel{\text{def}}{=} \exists \delta' \in [\![\mathcal{R}]\!].\ [\![P]\!](\delta, \delta')W \\
[\![\propto a.(B,H)]\!]\delta W &\stackrel{\text{def}}{=} \exists \iota.\ [\![p \propto a.(B,H)]\!]\delta' W\{\iota\} \\
&\qquad \text{where } p \notin \mathrm{dom}(\delta) \wedge \delta' = \delta, p{\mapsto}\mathrm{pop}(\iota) \\
[\![\Box P]\!]\delta W &\stackrel{\text{def}}{=} \forall W' \sqsupseteq W.\ [\![P]\!]\delta W' \\
[\![\triangleright P]\!]\delta W &\stackrel{\text{def}}{=} [\![P]\!]\delta(\triangleright W) \\
[\![\overline{e} \in R]\!]\delta W &\stackrel{\text{def}}{=} [\![R]\!]\delta W\overline{e} \\
[\![a]\!]\delta W\overline{e} &\stackrel{\text{def}}{=} \delta(a)\overline{e} \\
[\![r]\!]\delta W\overline{e} &\stackrel{\text{def}}{=} \delta(r)W\overline{e} \\
[\![\overline{x}.P]\!]\delta W\overline{e} &\stackrel{\text{def}}{=} [\![P[\overline{e}/\overline{x}]]\!]\delta W \\
[\![\mu r.R]\!]\delta W\overline{e} &\stackrel{\text{def}}{=} [\![R[\mu r.R/r]]\!]\delta W\overline{e} \\
[\![\uparrow R]\!]\delta W(e_1, e_2) &\stackrel{\text{def}}{=} \forall W' \sqsupseteq W.\ \forall h_1, h_2 : W'. \\
&\qquad W' \vdash (h_1; e_1) \approx (h_2; e_2) : [\![R]\!]\delta \\
[\![\mathrm{Term}_i]\!]\delta W e &\stackrel{\text{def}}{=} \mathrm{FL}(e) \subseteq W.\varsigma_i \\
[\![H \Rrightarrow J]\!]\delta W e &\stackrel{\text{def}}{=} \forall W' \sqsupseteq W.\ \forall h_1, h_2. \\
&\qquad [\![H]\!]\delta W'(h_1, h_2) \Rightarrow [\![J]\!]\delta W'(h_1, h_2) \\
[\![P_1, \ldots, P_n]\!]\delta W &\stackrel{\text{def}}{=} \textstyle\bigwedge_{i=1}^{n} [\![P_i]\!]\delta W
\end{aligned}$$

$\boxed{[\![H]\!]\delta W(h_1, h_2)}$

$$[\![H]\!]\delta W(h_1, h_2) \stackrel{\text{def}}{=} \top \quad \text{if } W.k = 0. \text{ Otherwise:}$$

$$\begin{aligned}
[\![e_1 \hookrightarrow_i e_2]\!]\delta W(h_1, h_2) &\stackrel{\text{def}}{=} [\![e_1 \in \mathrm{Val}_i]\!]\delta W \wedge [\![e_2 \in \mathrm{Val}_i]\!]\delta W \wedge \\
&\qquad h_i(e_1) = e_2 \\
[\![H_1 * H_2]\!]\delta W(h_1, h_2) &\stackrel{\text{def}}{=} \exists h_1^1, h_1^2, h_2^1, h_2^2.\ \forall i. \\
&\qquad h_i = h_i^1 \uplus h_i^2 \wedge [\![H_i]\!]\delta W(h_1^i, h_2^i) \\
[\![H_1 \vee H_2]\!]\delta W(h_1, h_2) &\stackrel{\text{def}}{=} [\![H_1]\!]\delta W(h_1, h_2) \vee [\![H_2]\!]\delta W(h_1, h_2) \\
[\![\exists \mathcal{X}.H]\!]\delta W(h_1, h_2) &\stackrel{\text{def}}{=} \exists \gamma \in [\![\mathcal{X}]\!].\ [\![\gamma H]\!]\delta W(h_1, h_2) \\
[\![\Box P]\!]\delta W(h_1, h_2) &\stackrel{\text{def}}{=} [\![\Box P]\!]\delta W
\end{aligned}$$

$\boxed{[\![\mathcal{L}]\!]\delta W\omega}$

$$[\![\mathcal{L}]\!]\delta W\omega \stackrel{\text{def}}{=} \top \quad \text{if } W.k = 0. \text{ Otherwise:}$$

$$\begin{aligned}
[\![\cdot]\!]\delta W\omega &\stackrel{\text{def}}{=} \omega = \emptyset \\
[\![\mathcal{L}, p \propto a.(B,H)]\!]\delta W\omega &\stackrel{\text{def}}{=} \exists \iota \in \omega.\ \delta(p) = \mathrm{pop}(\iota) \wedge \\
&\quad [\![\mathcal{L}]\!]\delta W(\omega - \{\iota\}) \wedge \\
&\quad W.\mathcal{I}(\iota).PL = \{CP \mid [\![B]\!](\delta, a{\mapsto}CP)\} \wedge \\
&\quad \forall CP \in W.\mathcal{I}(\iota).PL.\ \forall W' \sqsupseteq \triangleright W. \\
&\qquad (W', h_1, h_2) \in W.\mathcal{I}(\iota).HL(CP) \iff \\
&\qquad [\![H]\!](\delta, a{\mapsto}CP)(\triangleleft W')(h_1, h_2)
\end{aligned}$$

Figure 6. Model (Part 1): Relations and Islands

at one step level lower than the world containing them—thus, the construction of worlds can be stratified by the step level n. When we write W, we draw it from $\bigcup_{n\in\mathbb{N}} World_n$, and similarly for the other metavariables. Note also that we require heap relations to be monotone by construction, for the reasons explained in Section 4.

For convenience, we use dot notation (*e.g.,* $W.\mathcal{I}(\iota).HL$) to project components out of worlds in the obvious way.

Unlike in ADR, worlds as we have defined them here do not make disjointness of islands manifest. Instead of explicitly including in each island the set of locations "owned" by the island, as the ADR model does, our worlds just record the global sets of locations in ς_1 and ς_2. However, heap separation is enforced in the definition of heap satisfaction: a pair of heaps *satisfy* a world, written $h_1, h_2 : W$, if they contain sub-heaps that can be split up into pairs of disjoint parts, each of which is related by the heap relation of one particular island.

Two *configurations* are related in a given world W, written $W \vdash (h_1; e_1) \approx (h_2; e_2) : \Psi$, if the termination of one in $j < W.k$ *"serious" steps* implies the termination of the other, with the resulting heaps and values related by the semantic relation Ψ in some $(W.k - j)$-level future world of W. Which configuration's termination implies the other's is determined by the direction parameter d. By "serious" steps, we mean either unroll-roll reductions or "impure" steps that inspect or modify the heap. Also, note that $(h_1; e_1) \Downarrow^j (h_1'; e_1')$ implies that e_1' cannot reduce further, but not necessarily that e_1' is a value—it could be a stuck term.

6.2 Interpretation of Relations

Figure 6 shows the interpretation of LADR relations. As a relation R may have free relation variables $\mathcal{R}$, its interpretation $[\![R]\!]$ is parameterized by a relational interpretation δ, which maps the variables in $\mathcal{R}$ to semantic relations of the appropriate kind.

Following LSLR, we define $[\![R]\!]\delta W$ by a two-level induction, first on the step level $W.k$, and second on the size of R (where, crucially, $\triangleright P$ is considered to have constant size). At step level 0, all propositions are trivially true because "Time's up!" Otherwise, in terms of ensuring that the induction metric is obeyed, the only interesting cases are for $\triangleright P$ and $\mu r.R$. The $\triangleright$ modality is interpreted by "going down one step", *i.e.,* moving to the world one step later (abbreviated $\triangleright W$, as defined in Figure 5). Thus, even though $\triangleright P$ has constant size and thus P is potentially *larger*, the world in which we interpret P has a lower step-index, so the induction metric gets smaller. A recursive relation $\mu r.R$ is defined to be equivalent to its expansion $R[\mu r.R/r]$; assuming (as we do) that $\mu r.R$ is well-formed, that means R must be contractive in r, and thus the expansion is actually smaller in size than $\mu r.R$.

As explained in the Introduction, all propositions are required to be monotone with respect to time and width extension (written $\sqsupseteq$, which is defined in Figure 5). This property is stipulated formally in the definition of semantic relations (see $SemRel^k$, also defined in Figure 5). Consequently, the interpretations of both implication ($\Rightarrow$) and entailment ($\Rrightarrow$) quantify over worlds W' that extend the current world W in time and/or width. In contrast, the truth of $\Box$'d propositions must be preserved under arbitrary world extension (written $\sqsupseteq$).

The interpretation of $\uparrow R$ relates e_1 and e_2 if, for any heaps h_1 and h_2 satisfying the current world W, $(h_1; e_1)$ and $(h_2; e_2)$ are related configurations under W.

Besides $\uparrow R$, the other interesting world-dependent proposition is $\propto a.(B, H)$. It holds if the world contains an island named ι that correctly models the singleton island context $p \propto a.(B, H)$. (For more details on the model of island contexts, see below.)

Heap relations H are interpreted in mostly the standard separation logic way, except that we require values occurring in a reference assignment $e_1 \hookrightarrow_i e_2$ to be well-formed in the current world, *i.e.,* contain only locations from the respective heap $W.\varsigma_i$.

Last but not least, Figure 6 gives the semantic interpretation of island contexts $\mathcal{L}$. We say that the islands ω in world W correctly model island context $\mathcal{L}$ (written $[\![\mathcal{L}]\!]\delta W\omega$) if there is a bijection between ω and $\mathcal{L}$ such that the *semantic* population and heap laws of each island in ω faithfully model the *syntactic* population and heap laws of the corresponding entry in $\mathcal{L}$.

There are two particularly interesting points here. First, note that, for each island described by $\mathcal{L}$, we only require its heap law H to match the corresponding heap law of W when attention is restricted to the relatedness of heaps at *strictly future* worlds of W (*i.e.,* at worlds $W' \sqsupseteq \triangleright W$). Intuitively, this makes sense because (1) given the step-stratified construction of worlds, the actual heap law of W can only possibly contain heap atoms (W', h_1, h_2), where $W'.k < W.k$, and (2) since W is our "starting" world, we can safely ignore worlds that are not future worlds of it. (Putting these observations together, $W'.k < W.k \wedge W' \sqsupseteq W$ is equivalent to $W' \sqsupseteq \triangleright W$.) Moreover, restricting attention to strictly future worlds seems to be critical in proving the soundness of some of the proof rules that we present in Section 7, such as the $\propto$-INTRO rule.

Second, note the mysterious appearance of $\triangleleft W'$ in the modeling of the heap law H in $[\![\mathcal{L}]\!]$. The $\triangleleft$ operator is defined formally in Figure 5, and we pronounce it as "*earlier*" because it lifts the world W' *up* one level instead of pushing it down one level (as $\triangleright$ does). What is going on here? Basically, as we explained in Section 3.2, the issue is that we model the *syntactic* heap law H—which is presumed to hold true at the "current" step level, *i.e., W.k*—using a *semantic* heap law ($W.\mathcal{I}(\iota).HL$) that is indexed by worlds with step levels *strictly less* than $W.k$. This results in an unavoidable "off-by-one" error, for which the $\triangleleft$ operator then serves as a patch.

To understand how our use of $\triangleleft$ works, consider the following situation, variations of which arise in proving the soundness of several key LADR proof rules (such as $\uparrow$-IMPURE and ISL-UPD, described in Section 7). Suppose the "current" world under consideration is W, and suppose we are given the assumption that heaps h_1 and h_2 satisfy the heap law of some island ι in W (*i.e.,* $h_1, h_2 :_{\{\iota\}} W$). Suppose further that the corresponding island in the island context $\mathcal{L}$ has syntactic heap law H.

Given these assumptions, we *expect* it to be the case that $[\![H]\!]\delta W(h_1, h_2)$ for some appropriate δ—*i.e.,* that h_1 and h_2 satisfy H under world W—and we rely on this fact at various points, but it is something we need to show. Our assumption $h_1, h_2 :_{\{\iota\}} W$ tells us that $(\triangleright W, h_1, h_2) \in W.\mathcal{I}(\iota).HL(W.\mathcal{I}(\iota).CP)$, *i.e.,* that h_1 and h_2 satisfy W's *actual* heap law under world $\triangleright W$. According to the model of $[\![\mathcal{L}]\!]$ (instantiating W' with $\triangleright W$), we therefore know that h_1 and h_2 satisfy H under world $\triangleleft\triangleright W$, and from this we must derive that they satisfy H under W.

If H were an arbitrary heap relation, $[\![H]\!]\delta W$ and $[\![H]\!]\delta(\triangleleft\triangleright W)$ would not necessarily coincide. However, since H came from the island context $\mathcal{L}$, we know that it must be *delayed* (cf. Section 4), which ensures that its meaning $[\![H]\!]\delta W$ can only depend on what W looks like at one lower step level, *i.e.,* on what $\triangleright W$ looks like. Thus, since $\triangleright W = \triangleright\triangleleft\triangleright W$, the proof is complete. Note that it is *not* the case that $[\![H]\!]\delta W$ coincides with $[\![H]\!]\delta(\triangleright W)$—more things are related at $\triangleright W$ because its step level is lower than W's—so the use of $\triangleleft$ here is critical in inflating $\triangleright W$ to the same step level as W.

6.3 Judgments

The LADR inference rules concern three judgments. They all use a context $\mathcal{C}$ of the form $(\mathcal{X}; \mathcal{R}; \mathcal{L}; \mathcal{P})$, where $\mathcal{X}$ binds type and term variables, $\mathcal{R}$ binds relation variables, $\mathcal{L}$ associates (but does *not* bind) population variables with their related island definitions, and $\mathcal{P}$ specifies a set of hypotheses. As with worlds, we use dot notation to project out the component contexts of a $\mathcal{C}$.

The interpretation of contexts is shown in Figure 7. A combined context $(\mathcal{X}; \mathcal{R}; \mathcal{L}; \mathcal{P})$ is interpreted by all tuples $(\gamma, \delta, W, \omega_1, \omega_2)$, where: (1) γ and δ are closing instantiations for $\mathcal{X}$ and $\mathcal{R}$, (2) the names of W's islands can be split into two sets, ω_1 and ω_2, such that ω_1 corresponds to the islands specified in $\mathcal{L}$, and (3) the hypotheses in $\mathcal{P}$ hold under δ and W. Note that the interpretation of island contexts enforces that δ will map each population variable in $\mathcal{L}$ to $\mathrm{pop}(\iota)$, the population relation for some island ι in ω_1.

$$
\begin{array}{l}
[\![\mathcal{X}]\!] \stackrel{\text{def}}{=} \{\gamma \mid \mathrm{dom}(\gamma) = \mathcal{X} \ \wedge\ \mathrm{FV}(\mathrm{rng}(\gamma)) = \emptyset\} \\
[\![\mathcal{R}]\!] \stackrel{\text{def}}{=} \{\delta \mid \mathrm{dom}(\delta) = \mathcal{R} \wedge \forall a \in \mathcal{R}.\, \delta a \in \mathcal{P}(\mathrm{Term}^{\mathrm{arity}(a)}) \wedge \\
\qquad\qquad \forall r \in \mathcal{R}.\, \delta r \in \mathit{SemRel}^{\mathrm{arity}(r)}\} \\
[\![\mathcal{X}; \mathcal{R}; \mathcal{L}; \mathcal{P}]\!] \stackrel{\text{def}}{=} \{(\gamma, \delta, W, \omega_1, \omega_2) \mid \gamma \in [\![\mathcal{X}]\!] \wedge \delta \in [\![\mathcal{R}]\!] \wedge \\
\qquad\qquad [\![\gamma\mathcal{L}]\!]\delta W\omega_1 \wedge [\![\gamma\mathcal{P}]\!]\delta W \wedge \\
\qquad\qquad \mathrm{dom}(W.\mathcal{I}) = \omega_1 \uplus \omega_2\} \\
\boxed{\mathcal{C} \vdash P} \stackrel{\text{def}}{=} \forall(\gamma, \delta, W, \omega_1, \omega_2) \in [\![\mathcal{C}]\!]\,.\, [\![\gamma P]\!]\delta W \\
\boxed{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}} \stackrel{\text{def}}{=} \\
\qquad \forall(\gamma, \delta, W, \omega_1, \omega_2) \in [\![\mathcal{C}]\!]\,.\, W.k > 0 \ \Rightarrow\ \forall h_1, h_2, h_1^1, h_1^2, h_2^1, h_2^2. \\
\qquad \vdash h_i \ \wedge\ \mathrm{dom}(h_i) \supseteq W.\varsigma_i \ \wedge\ h_i = h_i^1 \uplus h_i^2 \ \wedge \\
\qquad [\![\gamma H]\!]\delta W(h_1^1, h_2^1) \ \wedge\ h_1^2, h_2^2 :_{\omega_2} W \\
\qquad\qquad \Rightarrow W \vdash (h_1; \gamma e_1) \approx (h_2; \gamma e_2) : [\![\gamma R]\!]\delta \\
\boxed{\mathcal{C} \vdash \{H\}\ e \stackrel{\mathcal{C}'}{\longmapsto}_i e'\ \{H'\}} \stackrel{\text{def}}{=} \\
\qquad \mathcal{C}' = \overline{x, x \in \mathrm{Val}_i} \ \wedge\ \forall(\gamma, \delta, W, \omega_1, \omega_2) \in [\![\mathcal{C}]\!]\,.\, \forall h_1, h_2. \\
\qquad [\![\gamma H]\!]\delta W(h_1, h_2) \ \wedge\ W.k > 0 \ \Rightarrow\ \forall \bar{l} \notin \mathrm{dom}(h_i) \cup W.\varsigma_i. \\
\qquad \exists \hat{h}_1, \hat{h}_2.\, \hat{h}_{\neg i} = h_{\neg i} \ \wedge\ (h_i; \gamma e) \stackrel{\bar{l}}{\longmapsto}^1 (\hat{h}_i; \gamma e'[\bar{l}/\overline{x}]) \ \wedge \\
\qquad [\![\gamma H'[\bar{l}/\overline{x}]]\!]\delta(W[\varsigma_i := W.\varsigma_i \uplus \{\bar{l}\}])(\hat{h}_1, \hat{h}_2)
\end{array}
$$

Figure 7. Model (Part 2): Contexts and Judgments

The *main judgment* of LADR is of the form $\mathcal{C} \vdash P$, which states that P is true for any interpretation of $\mathcal{C}$. If one is only reasoning about pure terms, one never needs to leave this judgment.

Equivalence of impure terms can be derived by going through the *separation judgment* $\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}$. Essentially, it states that, for any heaps h_1 and h_2 satisfying the precondition H, the configurations $(h_1; e_1)$ and $(h_2; e_2)$ are related in the current world, producing values related by R. But it is actually a bit more complicated than that, as the judgment also bakes in a *framing* condition. The condition says that h_1 and h_2 can be split into disjoint pieces, such that the first piece satisfies H, and the second piece satisfies the heap laws of the islands ω_2. This rather subtle framing condition is necessary in order to ensure that all the "action" that takes place in the separation judgment only affects the piece of the heap governed by the laws in $\mathcal{C}.\mathcal{L}$, which in turn is important in proving sound the critical ISL-UPD rule (Section 7).

The auxiliary *small-step judgment* $\mathcal{C} \vdash \{H\}\ e \stackrel{\mathcal{C}'}{\longmapsto}_i e'\ \{H'\}$ states that under precondition H, e reduces to e' in one "serious" step (cf. Section 6.1), implying postcondition H'. If the step is an allocation, then $\mathcal{C}'$ introduces a new variable x (which is a bound variable of the judgment) that stands for the resulting fresh location in e' and H'; otherwise $\mathcal{C}'$ is empty. The i specifies which program is being executed, the one on the left ($i = 1$) or the one on the right ($i = 2$). For the other program ($\neg i = 3 - i$), the heap is guaranteed not to change. Note that, in the model of the judgment, we quantify over all possible choices of a fresh location, and then extend the world W with that location.

7. LADR: Proof Rules

Figures 8 and 9 present the most important proof rules of LADR. Soundness proofs for all these rules are given in the appendix [17]. We omit introduction and elimination rules for the standard logical connectives here in the interest of space. We also omit standard axioms about atomic propositions (like Val, Const_{τ_b}, $e_1 \rightsquigarrow^k e_2$, etc.), which can be proven easily in the model. Throughout, we make the implicit assumption that the terms appearing in rules are in Term_1 or Term_2, as appropriate, under the given context $\mathcal{C}$. These assumptions can be built into the rules straightforwardly, at the expense of cluttering the presentation.

$$\frac{\mathcal{C} \vdash e_1 = e_2 \quad \mathcal{C} \vdash \mathcal{J}[e_1/x]}{\mathcal{C} \vdash \mathcal{J}[e_2/x]}\ \text{REPLACE} \qquad \frac{\mathcal{C} \vdash \mathcal{J}}{\mathcal{C}, \mathcal{X}, \mathcal{R}, \mathcal{P} \vdash \mathcal{J}}\ \text{WEAKEN} \qquad \frac{\mathcal{C} \vdash P \quad \mathcal{C}, P \vdash \mathcal{J}}{\mathcal{C} \vdash \mathcal{J}}\ \text{CUT}$$

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

$$\frac{\mathcal{C} \vdash P}{\mathcal{C}, \mathcal{L} \vdash P}\ \mathcal{L}\text{-WEAKEN} \qquad \frac{\mathcal{C} \vdash P}{\mathcal{C} \vdash \triangleright P}\ \triangleright\text{-MONO} \qquad \frac{\triangleleft \mathcal{C} \vdash P}{\mathcal{C} \vdash \triangleright P}\ \triangleright\text{-WEAKEN} \qquad \frac{\mathcal{C}, \triangleright P \vdash P}{\mathcal{C} \vdash P}\ \text{LÖB}$$

$$\frac{\mathcal{C} \vdash \triangleright(P \Rightarrow Q)}{\mathcal{C} \vdash \triangleright P \Rightarrow \triangleright Q} \qquad \frac{\mathcal{C} \vdash \triangleright(P \wedge Q)}{\mathcal{C} \vdash \triangleright P \wedge \triangleright Q} \qquad \frac{\mathcal{C} \vdash \triangleright(P \vee Q)}{\mathcal{C} \vdash \triangleright P \vee \triangleright Q} \qquad \frac{\mathcal{C} \vdash \triangleright \forall \mathcal{X}.P}{\mathcal{C} \vdash \forall \mathcal{X}.\triangleright P} \qquad \frac{\mathcal{C} \vdash \triangleright \forall \mathcal{R}.P}{\mathcal{C} \vdash \forall \mathcal{R}.\triangleright P} \qquad \frac{\mathcal{C} \vdash \triangleright \exists \mathcal{X}.P}{\mathcal{C} \vdash \exists \mathcal{X}.\triangleright P} \qquad \frac{\mathcal{C} \vdash \triangleright \exists \mathcal{R}.P}{\mathcal{C} \vdash \exists \mathcal{R}.\triangleright P}$$

$$\frac{\dagger \mathcal{C} \vdash P}{\mathcal{C} \vdash \square P}\ \square\text{-INTRO} \qquad \frac{\mathcal{C} \vdash A}{\mathcal{C} \vdash \square A}\ \square\text{-INTRO-ABS} \qquad \frac{\mathcal{C} \vdash e \in \mathrm{Term}_i}{\mathcal{C} \vdash \square(e \in \mathrm{Term}_i)}\ \square\text{-INTRO-TERM} \qquad \frac{\mathcal{C} \vdash \square P}{\mathcal{C} \vdash P}\ \square\text{-ELIM} \qquad \frac{\mathcal{C} \vdash \triangleright \square P}{\mathcal{C} \vdash \square \triangleright P}\ \triangleright\square\text{-SWAP}$$

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

$$\frac{p \propto a.(B, H) \in \mathcal{C}.\mathcal{L} \quad \mathcal{C} \vdash \overline{e} \in p}{\mathcal{C} \vdash \square(\overline{e} \in p)}\ \text{POP-MONO} \qquad \frac{p \propto a.(B, H) \in \mathcal{C}.\mathcal{L} \quad \mathcal{C} \vdash A \equiv p}{\mathcal{C} \vdash B[A/a]}\ \text{POP-LAW} \qquad \frac{p \propto a.(B, H) \in \mathcal{C}.\mathcal{L}}{\mathcal{C} \vdash \exists a.a \equiv p}\ \text{POP-SNAP}$$

$$\frac{\begin{array}{c} p \propto a.(B', H') \in \mathcal{C}.\mathcal{L} \\ \mathcal{C} \vdash \forall a.B \equiv B' \\ \mathcal{C} \vdash \square(\forall a.B \Rightarrow (H \Leftrightarrow H')) \end{array}}{\mathcal{C} \vdash \propto a.(B, H)}\ \propto\text{-INTRO} \qquad \frac{\begin{array}{c} \mathcal{C} \vdash \propto a.(B, H) \quad \mathcal{C}, p, p \propto a.(B, H) \vdash P \\ \forall p' \propto a.(B', H') \in \mathcal{C}.\mathcal{L}\colon\ \mathcal{C}, \forall a.B \equiv B', \square(\forall a.B \Rightarrow (H \Leftrightarrow H')) \vdash P \end{array}}{\mathcal{C} \vdash P}\ \propto\text{-ELIM}$$

$$\frac{\mathcal{C} \vdash \square P}{\mathcal{C} \vdash H \Rrightarrow \square P} \qquad \frac{\mathcal{C} \vdash H \Rrightarrow \square P}{\mathcal{C} \vdash H \Rrightarrow H * \square P} \qquad \frac{\mathcal{C}, \square P \vdash H \Rrightarrow H'}{\mathcal{C} \vdash \square P * H \Rrightarrow H'} \qquad \frac{}{\mathcal{C} \vdash e_1 \hookrightarrow_i e_2 \Rrightarrow \square(e_1 \in \mathrm{Loc} \wedge e_1 \in \mathrm{Val}_i \wedge e_2 \in \mathrm{Val}_i)}$$

$$\frac{\mathcal{C} \vdash (e_1', e_2') \in \uparrow R \quad \mathcal{C} \vdash e_1 \rightsquigarrow^* e_1' \quad \mathcal{C} \vdash e_2 \rightsquigarrow^* e_2'}{\mathcal{C} \vdash (e_1, e_2) \in \uparrow R}\ \uparrow\text{-EXPAND} \qquad \frac{\mathcal{C} \vdash (e_1', e_2') \in \uparrow R \quad \mathcal{C} \vdash e_1' \rightsquigarrow^0 e_1 \quad \mathcal{C} \vdash e_2' \rightsquigarrow^0 e_2}{\mathcal{C} \vdash (e_1, e_2) \in \uparrow R}\ \uparrow\text{-REDUCE} \qquad \frac{\triangleleft \mathcal{C} \vdash (e_1', e_2') \in \uparrow R \quad \mathcal{C} \vdash e_1 \rightsquigarrow^1 e_1' \quad \mathcal{C} \vdash e_2 \rightsquigarrow^1 e_2'}{\mathcal{C} \vdash (e_1, e_2) \in \uparrow R}\ \uparrow\text{-UNROLL}$$

$$\frac{\mathcal{C} \vdash (e_1, e_2) \in R \quad \mathcal{C} \vdash R : \mathrm{VRel}}{\mathcal{C} \vdash (e_1, e_2) \in \uparrow R}\ \uparrow\text{-RETURN} \qquad \frac{\mathcal{C} \vdash (e_1, e_2) \in \uparrow S \quad \dagger \mathcal{C}, x_1, x_2, (x_1, x_2) \in S \vdash (E_1[x_1], E_2[x_2]) \in \uparrow R}{\mathcal{C} \vdash (E_1[e_1], E_2[e_2]) \in \uparrow R}\ \uparrow\text{-BIND}$$

$$\frac{\mathcal{C}.\mathcal{L} = \overline{p \propto a.(B, H)} \quad \mathcal{C}, \overline{a}, \overline{p \equiv a} \vdash \{* \overline{H}\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash (e_1, e_2) \in \uparrow R}\ \uparrow\text{-IMPURE}$$

Figure 8. Key Inference Rules of LADR (Part 1): Structural Rules and Main Judgment

The first three rules are standard laws for replacement of equal terms, weakening, and cut. We write $\mathcal{J}$ to range over all three judgments introduced in Section 6.3. Note that the general weakening rule does not allow the addition of islands. Weakening of the island context is allowed for the main judgment (rule $\mathcal{L}$-WEAKEN), because we always allow the world to contain additional islands not mentioned in $\mathcal{C}.\mathcal{L}$. Yet weakening does *not* hold in the separation judgment, where the island laws also represent implicit *postconditions*, which induce proof obligations in rule ISL-UPD (see below).

The rules $\triangleright$-MONO to LÖB are taken directly out of LSLR. Rule $\triangleright$-MONO says that, due to time monotonicity, any proposition true now is still true later. Rule $\triangleright$-WEAKEN allows proving a proposition true one step later by proving it in a context where all later assumptions have been "moved to the present" (notation $\triangleleft\mathcal{C}$, defined in Figure 3). This rule is actually derivable from $\triangleright$-MONO and the law of distributivity of $\triangleright$ over $\Rightarrow$ (given in the next line).

Like in LSLR, the LÖB rule provides a simple induction principle over step levels: if, under the assumption that P is true later, we can prove P now, then by induction P is true now. See Section 9.3 for an example of its use.

The next couple of rules allow introduction of the $\square$-modality for different kinds of monotone propositions. According to the $\square$-INTRO rule, P is monotone if we can prove it without any potentially non-monotone assumption ($\dagger\mathcal{C}$ removes all but $\square$'d assumptions from the context, cf. Figure 3). Absolute propositions are always monotone, and so is well-formedness of terms (rules $\square$-INTRO-ABS and $\square$-INTRO-TERM). Conversely, a $\square$'d proposition is of course true in the current world ($\square$-ELIM). Finally, rule $\triangleright\square$-SWAP asserts that the two modalities commute.

The rules ELEM and ELEM-μ define inhabitation of relations. A recursive relation is equivalent to its expansion.

The next rules deal with populations. Because populations are required to grow monotonically in the model, membership in one is a monotone property (rule POP-MONO—see Section 9.2 for a good example of its use). Furthermore, if A represents the current population of an island, then the island's population law B holds for it (rule POP-LAW). Rule POP-SNAP allows us to take an absolute "snapshot" a of the current population of an island (intuitively, we can do this because the population of an island remains constant under time and width extension of its surrounding world).

An island proposition $\propto a.(B, H)$ can only be introduced if an island with population and heap laws equivalent to B and H exists in the context (rule $\propto$-INTRO). Conversely, eliminating an island proposition (rule $\propto$-ELIM) requires a case distinction: either (1) it talks about an island p that we do not know yet, in which case we just add p to the island context, or (2) it refers to one of the islands in $\mathcal{L}$, in which case said island must have population and heap laws equivalent to B and H.

The next four rules in Figure 8 are but a small sample of the many rules for heap relation entailments $H \Rrightarrow J$. Many more rules appear in the appendix [17], and most of those are completely standard rules from intuitionistic separation logic. The rules we show here are interesting in that they involve the atomic propositions of the form $\square P$. The first rule says that any $\square P$ that we can prove in the main judgment may serve as the conclusion of an entailment. The second rule says that if $\square P$ is true, it is true of the empty heap. The third rule enables $\square P$ to be shifted between the context and the antecedent of the entailment. The fourth rule establishes that when e_1 points to e_2, they must both be values, and e_1 must be a location.

$\boxed{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}$

$$\frac{\mathcal{C} \vdash B[A/a] \quad \mathcal{C}, p, p \propto a.(B, H'), p \equiv A \vdash \{H\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{isl-new} \qquad \frac{\mathcal{C}.\mathcal{L} = \overline{p \propto a.(B,H)} \quad \mathcal{C} \vdash \bigwedge \overline{p \subseteq A} \quad \mathcal{C} \vdash \bigwedge \overline{B[A/a]} \quad \mathcal{C}' = \dagger\mathcal{C}, \overline{p \equiv A} \quad \mathcal{C}' \vdash H \Rrightarrow * \overline{H[A/a]} \quad \mathcal{C}' \vdash (e_1, e_2) \in \uparrow R}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{isl-upd}$$

$$\frac{\mathcal{C} \vdash \{H\}\ e_1' \approx e_2'\ \{R\} \quad \mathcal{C} \vdash e_1 \rightsquigarrow^* e_1' \quad \mathcal{C} \vdash e_2 \rightsquigarrow^* e_2'}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{expand} \qquad \frac{\mathcal{C} \vdash \{H\}\ e_1' \approx e_2'\ \{R\} \quad \mathcal{C} \vdash e_1' \rightsquigarrow^0 e_1 \quad \mathcal{C} \vdash e_2' \rightsquigarrow^0 e_2}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{reduce}$$

$$\frac{\mathcal{C} \vdash \{H\}\ e_1 \overset{\mathcal{C}_1}{\longmapsto}_1 e_1'\ \{H'\} \quad \mathcal{C}, \mathcal{C}_1 \vdash \{H'\}\ e_1' \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{step-l} \qquad \frac{\mathcal{C} \vdash \{H\}\ e_2 \overset{\mathcal{C}_2}{\longmapsto}_2 e_2'\ \{H'\} \quad \mathcal{C}, \mathcal{C}_2 \vdash \{H'\}\ e_1 \approx e_2'\ \{R\}}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{step-r}$$

$$\frac{\mathcal{C} \vdash \{H_1\}\ e_1 \overset{\mathcal{C}_1}{\longmapsto}_1 e_1'\ \{H_1'\} \quad \mathcal{C} \vdash \{H_2\}\ e_2 \overset{\mathcal{C}_2}{\longmapsto}_2 e_2'\ \{H_2'\} \quad \triangleleft\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2 \vdash \{H_1' * H_2'\}\ e_1' \approx e_2'\ \{R\}}{\mathcal{C} \vdash \{H_1 * H_2\}\ e_1 \approx e_2\ \{R\}}\ \textsc{step-lr}$$

$$\frac{\mathcal{C} \vdash H \Rrightarrow H' \quad \mathcal{C} \vdash \{H'\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{sep-entail} \qquad \frac{\mathcal{C} \vdash \Box P \quad \mathcal{C} \vdash \{H * \Box P\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{sep-cut}$$

$$\frac{\mathcal{C}, \Box P \vdash \{H\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H * \Box P\}\ e_1 \approx e_2\ \{R\}}\ \Box\textsc{-shift} \qquad \frac{\mathcal{C} \vdash \{H_1\}\ e_1 \approx e_2\ \{R\} \quad \mathcal{C} \vdash \{H_2\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{H_1 \vee H_2\}\ e_1 \approx e_2\ \{R\}}\ \textsc{sep-}\vee \qquad \frac{\mathcal{C}, \mathcal{X} \vdash \{H\}\ e_1 \approx e_2\ \{R\}}{\mathcal{C} \vdash \{\exists \mathcal{X}.H\}\ e_1 \approx e_2\ \{R\}}\ \textsc{sep-}\exists$$

$\boxed{\mathcal{C} \vdash \{H\}\ e \overset{\mathcal{C}'}{\longmapsto}_i e'\ \{H'\}}$

$$\frac{\mathcal{C} \vdash e \rightsquigarrow^1 e'}{\mathcal{C} \vdash \{H\}\ e \longmapsto_i e'\ \{H\}}\ \textsc{unroll} \qquad \frac{\mathcal{C} \vdash e \in \mathrm{Val}_i \quad \mathcal{C} \vdash A \subseteq \mathrm{Val}_i \quad \mathcal{C}' = x, x \in \mathrm{Val}_i}{\mathcal{C} \vdash \{H\}\ E[\mathsf{ref}\ e] \overset{\mathcal{C}'}{\longmapsto}_i E[x]\ \{H * x \hookrightarrow_i e * \Box(x \notin A)\}}\ \textsc{alloc}$$

$$\frac{\mathcal{C} \vdash H \Rrightarrow e_1 \hookrightarrow_i e_2}{\mathcal{C} \vdash \{H\}\ E[!e_1] \longmapsto_i E[e_2]\ \{H\}}\ \textsc{deref} \qquad \frac{\mathcal{C} \vdash e_2 \in \mathrm{Val}_i}{\mathcal{C} \vdash \{H * e_1 \hookrightarrow_i e_2'\}\ E[e_1 := e_2] \longmapsto_i E[\langle\rangle]\ \{H * e_1 \hookrightarrow_i e_2\}}\ \textsc{assign}$$

Figure 9. Key Inference Rules of LADR (Part 2): Separation and Small-Step Judgments

The remaining rules of the main judgment deal with relatedness of terms. The first three ($\uparrow$-EXPAND, $\uparrow$-REDUCE, and $\uparrow$-UNROLL) consider closure of relatedness under pure conversion, and are again straight out of LSLR. As they are essentially pure versions of the separation judgment rules EXPAND, REDUCE, and STEP-LR, we refer the reader to the discussion of those rules below.

Rules $\uparrow$-RETURN and $\uparrow$-BIND can be regarded as *monadic* rules for reasoning about computations in a sequentialized manner. The former represents the base case: if two values are related by R, then they are also related by $\uparrow R$. The latter is useful when we have terms in evaluation position (e_1 and e_2 in the rule) that are not syntactically reducible to values, but that we know belong to $\uparrow S$ for some S. The rule allows us to invent variables to represent their values, and to assume the variables are related by S in a future world. See Sections 8 and 9.2 for examples where $\uparrow$-BIND is key.

In order to reason about terms involving impure reduction steps, rule $\uparrow$-IMPURE enables us to switch to the separation judgment. The heap laws from all islands in the island context are converted (via separating conjunction) into a precondition, under which we proceed to prove the terms related. Since the heap laws may mention the current population, we also pick fresh absolute a's to represent the current populations of all the islands and instantiate the heap laws with those a's.

This leads us to the separation judgment, defined in Figure 9. Rule ISL-NEW allows us to extend the world (widthwise) with a new island. It requires us to pick an initial population A and prove that it obeys the population law. Two terms can then be proven related under the assumption that the new island exists and that A is its (current) population.

Rule ISL-UPD allows us to prove a separation judgment by switching back to the main judgment, at which point we have the opportunity to advance to a future world by extending the population on any island. For each island, the respective new population A must be shown to be a superset of the current one, and it must obey the corresponding population law. The world is then updated with the new populations, by removing all assumptions about the previous world from the context (courtesy of the $\dagger\mathcal{C}$ notation), and adding the knowledge about the new populations. In this new world, the heap laws of all known islands (joined again by a separating conjunction) must be entailed by the precondition H.

The main purpose of the separation judgment is to enable us to prove the equivalence of terms by symbolically stepping through their evaluations. Rules EXPAND, STEP-L, and STEP-R describe closure of the separation judgment under expansion. REDUCE describes closure under reduction, as long as the reduction takes zero "serious" steps. Rule STEP-LR is another rule for closure under expansion, but since both sides expand with a "serious" step, we can drop the $\triangleright$'s from any $\triangleright P$ assumptions in the context. See Sections 8 and 9.3 for examples of how this is critically useful.

All three STEP rules use the auxiliary stepping judgment to perform the actual step and update the heap assertion H accordingly. The only slightly unusual rule of this judgment is ALLOC: it provides a means of picking a set A of well-formed values with respect to which the variable x (representing the new location) is assumed to be fresh in the postcondition. This is reminiscent of Aydemir *et al.*'s "cofinite quantification" style of reasoning about fresh identifiers in mechanized metatheory [5], for although A is not necessarily finite, its absoluteness suffices to show it contains finitely many locations. The freshness postcondition is useful, *e.g.,* when reasoning about equality of locations. See the name generator example in Section 4.1 of our online appendix [17] for details.

The remaining rules of the separation judgment are straightforward: SEP-ENTAIL is one half of the rule of consequence. (We have not needed the other half, but it would be easy to prove.) With rule SEP-CUT, derivable monotone propositions can be "cut into" the

heap precondition, and with rule $\Box$-SHIFT they can be shifted back. The two remaining rules are standard separation logic rules.

8. LADR: Soundness of the Logical Relation

We now present our main results concerning soundness of the logical relation with respect to contextual equivalence. The first step is to define a logical equivalence *judgment*:

Definition 8.1 (Logical Equivalence Judgment)
Given $\Gamma; \Sigma \vdash e_1 : \tau$ and $\Gamma; \Sigma \vdash e_2 : \tau$, with $\Gamma = \overline{\alpha}, \overline{x : \tau}$ and $\Sigma = \overline{l : \sigma}$, define

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

where

$$\begin{array}{ll} \gamma_i = \overline{\alpha \mapsto \alpha_i}, \overline{x \mapsto x_i} & \mathcal{X} = \overline{\alpha_1}, \overline{\alpha_2}, \overline{x_1}, \overline{x_2} \\ \mathcal{R} = \overline{r}, \overline{p} & \rho = \overline{\alpha \mapsto r} \\ \multicolumn{2}{l}{\mathcal{L} = \overline{p \propto a.(a \equiv \{(l,l)\}, \exists y_1, y_2.\ l \hookrightarrow_1 y_1 * l \hookrightarrow_2 y_2 * \overline{\Box\triangleright(y_1, y_2) \in \mathcal{V}[\![\sigma]\!]})}} \\ \multicolumn{2}{l}{\mathcal{P} = \overline{r : \mathrm{Type}}, \overline{(x_1, x_2) \in \mathcal{V}[\![\tau]\!]\rho}, \overline{l \in \mathrm{Val}_1}, \overline{l \in \mathrm{Val}_2}} \end{array}$$

The basic building blocks of the soundness proof are the *compatibility* lemmas [30], which state that the logical relation is closed under each of the language's constructs. Together, they yield the *fundamental property* of the logical relation and the fact that the logical relation is a congruence wrt. language contexts C. In contrast to ADR, our proofs for compatibility lemmas are much shorter, because we can express them, at a much higher level of abstraction, in terms of the proof rules we have given in Section 7. As an example, we show the compatibility lemma for dereferencing, which was multiple pages long in ADR. See the online appendix for the proofs of the other state-related compatibility lemmas [17].

Lemma 8.2 (Compatibility: Dereference)
If $\Gamma; \Sigma \vdash e_1 \approx^{log} e_2 : \mathsf{ref}\ \tau$, then $\Gamma; \Sigma \vdash !e_1 \approx^{log} !e_2 : \tau$.

Proof: Unfolding the definition of $\approx^{log}$, this boils down to deriving $\mathcal{C}_0 \vdash (e_1, e_2) \in \mathcal{E}[\![\mathsf{ref}\ \tau]\!]\rho \Rightarrow (!e_1, !e_2) \in \mathcal{E}[\![\tau]\!]\rho$ for some $\mathcal{C}_0$ in which $e_1 \in \mathrm{Term}_1$ and $e_2 \in \mathrm{Term}_2$. Using rule $\mathcal{L}$-WEAKEN, we may assume that $\mathcal{C}_0.\mathcal{L} = \emptyset$. Starting with rule $\uparrow$-BIND, we need to show $\mathcal{C}_1 \vdash (!x_1, !x_2) \in \mathcal{E}[\![\tau]\!]\rho$, where $\mathcal{C}_1 = \dagger\mathcal{C}_0, x_1, x_2, (x_1, x_2) \in \mathcal{V}[\![\mathsf{ref}\ \tau]\!]\rho$. The latter tells us that $\propto a.(B, H)$, where $H = \exists y_1, y_2.\ x_1 \hookrightarrow_1 y_1 * x_2 \hookrightarrow_2 y_2 * \Box\triangleright(y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho$. With the help of rule $\propto$-ELIM and the fact that our $\mathcal{L}$ is empty, we can extend $\mathcal{C}_1$ to $\mathcal{C}_2 = \mathcal{C}_1, p, p \propto a.(B, H)$ and then use rule $\uparrow$-IMPURE to enter the separation judgment. Here we are required to show $\{H\}\ !x_1 \approx !x_2\ \{\mathcal{V}[\![\tau]\!]\rho\}$. We first extend $\mathcal{C}_2$ with $y_1, y_2, \triangleright\Box(y_1, y_2) \in \mathcal{V}[\![\tau]\!]\rho$ (using rules SEP-$\exists$, $\Box$-SHIFT, and $\triangleright\Box$-SWAP). By combining rules DEREF and STEP-LR, the proof goal reduces to $\{H\}\ y_1 \approx y_2\ \{\mathcal{V}[\![\tau]\!]\rho\}$, but under a context where $\triangleright$ is removed from the assumption concerning the relatedness of y_1 and y_2. We now switch back to the regular judgment using rule ISL-UPD, without actually updating any island. Consequently, we need to show $(y_1, y_2) \in \uparrow\mathcal{V}[\![\tau]\!]\rho$, which follows from the assumption in the context and rules $\uparrow$-RETURN and $\Box$-ELIM. ■

Theorem 8.3 (Fundamental Property)
If $\Gamma; \Sigma \vdash e : \tau$, then $\Gamma; \Sigma \vdash e \approx^{log} e : \tau$.

Theorem 8.4 (Congruence)
If $\Gamma; \Sigma \vdash e_1 \approx^{log} e_2 : \tau$ and $\vdash C : (\Gamma; \Sigma; \tau) \rightsquigarrow (\Gamma'; \Sigma'; \tau')$, then $\Gamma'; \Sigma' \vdash C[e_1] \approx^{log} C[e_2] : \tau'$.

Given the Congruence Theorem, the only missing piece in the soundness proof is *adequacy* of the logical relation [30], *i.e.,* the fact that if two closed terms are logically related, then they coterminate under any (well-formed) heap. For this lemma we need to reason directly in the model, and the proof follows the one in ADR; details are in the online appendix [17].

Lemma 8.5 (Adequacy)
If $\cdot; \Sigma \vdash e_1 \approx^{log} e_2 : \tau$ and $\vdash h : \Sigma$, then $(h; e_1)\Downarrow$ iff $(h; e_2)\Downarrow$.

Theorem 8.6 (Soundness w.r.t. Contextual Equivalence)
If $\Gamma; \Sigma \vdash e_1 \approx^{log} e_2 : \tau$, then $\Gamma; \Sigma \vdash e_1 \approx^{ctx} e_2 : \tau$.

9. Examples

In this section, we demonstrate how to use LADR to prove some interesting contextual equivalences. We can use LADR to prove all the examples in ADR, save for one which we discuss in Section 10.

9.1 Twin Abstraction

Recall the "twin abstraction" example from Section 3.1:

$$\begin{array}{rcl} \tau & = & \exists\alpha, \beta.\ (\mathsf{unit} \to \alpha) \times (\mathsf{unit} \to \beta) \times (\alpha \times \beta \to \mathsf{bool}) \\ C & = & \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in} \\ & & \mathsf{pack}\,\mathsf{int}, \mathsf{int}, \langle \lambda\_.{+}{+}x, \lambda\_.{+}{+}x, \lambda y.[\bullet]\rangle\ \mathsf{as}\ \tau \\ e_1 & = & C[\mathsf{fst}\ y = \mathsf{snd}\ y] \\ e_2 & = & C[\mathsf{false}] \end{array}$$

To prove e_1 and e_2 equivalent, we want to show $\vdash (e_1, e_2) \in \mathcal{E}[\![\tau]\!]$, or, by $\uparrow$-IMPURE, $\vdash \{\Box\top\}\ e_1 \approx e_2\ \{\mathcal{V}[\![\tau]\!]\}$. By rules STEP-LR, ALLOC and EXPAND, we need to show

$$\{x_1 \hookrightarrow_1 0 * x_2 \hookrightarrow_2 0\}\ e_1'[x_1/x] \approx e_2'[x_2/x]\ \{\mathcal{V}[\![\tau]\!]\}$$

where e_1' and e_2' are the bodies of the respective let expressions and we omit the pure heap assertions produced by ALLOC.

Using rule ISL-NEW we introduce an island $p \propto a.(B, H)$, whose population is a finite mapping of the symbols generated so far (which will be the same on both sides) to the respective abstract type they inhabit, *i.e.,* 1 for α and 2 for β:

$$\begin{array}{rcl} B & = & \exists n.\ \mathrm{funmap}(a, n, 2) \\ H & = & \exists n.\ x_1 \hookrightarrow_1 n * x_2 \hookrightarrow_2 n * \Box\mathrm{maxdom}(a, n) \end{array}$$

The population law B states that the population a is the relational map of a function from $\{1, \ldots, n\}$ to $\{1, 2\}$, for some n. The heap law H then verifies that the states of x_1 and x_2 always are in sync with the maximum n in the domain of a. (The absolute predicates, funmap and maxdom, are easy to define in LADR.) Given $A = \emptyset$ for the initial population, it is easy to verify that $B[A/a]$ holds.

Both $e_1'[x_1/x]$ and $e_2'[x_2/x]$ are values, so we want to apply rule ISL-UPD immediately to get back into the pure judgment, *i.e.,* prove $(e_1'[x_1/x], e_2'[x_2/x]) \in \mathcal{V}[\![\tau]\!]$ under the island context $p \propto a.(B, H)$. We instantiate that rule with $A = \emptyset$ as before, and so the only new thing to prove is $x_1 \hookrightarrow_1 0 * x_2 \hookrightarrow_2 0 \Rrightarrow H[\emptyset/a]$, which involves the straightforward proof of $\Box\mathrm{maxdom}(\emptyset, 0)$.

Now, we unroll the definition of $\mathcal{V}[\![\tau]\!] = \mathcal{V}[\![\exists\alpha.\exists\beta.\_]\!]$ and pick

$$\begin{array}{rcl} R_\alpha & = & (x_1 \in \mathrm{Const}_{\mathsf{int}}, x_2 \in \mathrm{Const}_{\mathsf{int}}).\ x_1 = x_2 \wedge (x_1, 1) \in p \\ R_\beta & = & (x_1 \in \mathrm{Const}_{\mathsf{int}}, x_2 \in \mathrm{Const}_{\mathsf{int}}).\ x_1 = x_2 \wedge (x_1, 2) \in p \end{array}$$

We need to prove $R_\alpha : \mathrm{Type}$ and $R_\beta : \mathrm{Type}$. Both properties follow straightforwardly from rule POP-MONO. Now let $\rho = \alpha \mapsto R_\alpha, \beta \mapsto R_\beta$. By definition of $\mathcal{V}[\![\_ \times \_]\!]\,\rho$, we have to show:

1. $\mathcal{C} \vdash (\lambda\_.{+}{+}x_1, \lambda\_.{+}{+}x_2) \in \mathcal{V}[\![\mathsf{unit} \to \alpha]\!]\rho$
2. $\mathcal{C} \vdash (\lambda\_.{+}{+}x_1, \lambda\_.{+}{+}x_2) \in \mathcal{V}[\![\mathsf{unit} \to \beta]\!]\rho$
3. $\mathcal{C} \vdash (\lambda y.\mathsf{fst}\ y = \mathsf{snd}\ y, \lambda y.\mathsf{false}) \in \mathcal{V}[\![\alpha \times \beta \to \mathsf{bool}]\!]\rho$

where $\mathcal{C} = x_1, x_2; p; p \propto a.(B, H); \Box(x_1 \in \mathrm{Val}_1), \Box(x_2 \in \mathrm{Val}_2)$.

For (1), we unroll the definition of $\mathcal{V}[\![\_ \to \_]\!]\,\rho$, and apply the introduction rules for $\Box$, $\forall$ and $\Rightarrow$, producing the goal

$$((\lambda\_.{+}{+}x_1)\,y_1, (\lambda\_.{+}{+}x_2)\,y_2) \in \uparrow\mathcal{V}[\![\alpha]\!]\rho = R_\alpha$$

with $(y_1, y_2) \in \mathcal{V}[\![\mathsf{unit}]\!]\rho$ in the assumptions. By $\uparrow$-EXPAND and $\uparrow$-IMPURE we get the name $a \equiv p$ for the current population. By $\exists$-INTRO, we have to show:

$$\{x_1 \hookrightarrow_1 n * x_2 \hookrightarrow_2 n * \Box\mathrm{maxdom}(a, n)\}\ {+}{+}x_1 \approx {+}{+}x_2\ \{R_\alpha\}$$

We can step through both computations simultaneously using rule STEP-LR, record $n + 1 \rightsquigarrow^0 n'$ in the context (for a freshly bound variable n'), and reach:

$$\{x_1 \hookrightarrow_1 n' * x_2 \hookrightarrow_2 n' * \Box\mathrm{maxdom}(a, n)\}\ n' \approx n'\ \{R_\alpha\}$$

Now that the new symbols have been generated, we update the island with the new population $A = a \cup \{(n', 1)\}$, using rule ISL-UPD. To do so, we first have to show $p \subseteq A$, which is easy given $p \equiv a$. Then we have to show that the population law still holds, *i.e.,* $\exists n''.\,\mathrm{funmap}(A, n'', 2)$. This is easy to derive by picking $n'' = n'$ and using the assumption $\mathrm{maxdom}(a, n)$. Likewise, we have to prove that the final heap assertion entails the heap law $H[A/a]$ under the assumption $p \equiv A$. Choosing n' for the existential variable, this reduces to showing $\mathrm{maxdom}(A, n')$, which is easy.

The final step in this part is to show $(n', 1) \in \uparrow R_\alpha$. Unfolding the definition, this follows directly from rule $\uparrow$-RETURN, given that the updated population $p \equiv A = a \cup \{(n', 1)\}$.

For part (2) we proceed analogously, the only difference being that we choose $A = a \cup \{(n', 2)\}$ for the new population.

Part (3) is relatively straightforward now. We start as before, yielding the goal

$$((\lambda y.\mathsf{fst}\, y = \mathsf{snd}\, y)\, y_1, (\lambda y.\mathsf{false})\, y_2) \in \uparrow\mathcal{V}[\![\mathsf{bool}]\!]$$

where $(y_1, y_2) \in \mathcal{V}[\![\alpha \times \beta]\!]\rho$. We can unfold the definition of $\mathcal{V}[\![\alpha \times \beta]\!]\rho$ and eliminate the top-level existential bindings, thus producing the assumptions $y_1{=}\langle y_1', y_1''\rangle$, $y_2{=}\langle y_2', y_2''\rangle$ and $(y_1', y_2') \in R_\alpha, (y_1'', y_2'') \in R_\beta$ for some fresh y_1', y_2', y_1'', y_2''. Given this, y_1 can be replaced by the pair $\langle y_1', y_1''\rangle$, and by $\uparrow$-EXPAND we are left to prove $(y_1' = y_1'', \mathsf{false}) \in \uparrow\mathcal{V}[\![\mathsf{bool}]\!]$. At this point, the proof essentially boils down to the absolute proposition

$$\forall a, y', y''.\, B \Rightarrow (y', 1) \in a \Rightarrow (y'', 2) \in a \Rightarrow y' \neq y''$$

which is straightforward to prove given the definition of B. We can then conclude that $y_1' = y_1''$ reduces to false, and we are done.

9.2 Irreversible State Change

Consider Pitts and Stark's "awkward" example [31], two equivalent functions of type $(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}$:

$$\begin{array}{lcl} e_1 & = & \mathsf{let}\, x = \mathsf{ref}\, 0\, \mathsf{in}\, \lambda f.(x := 1; f\,\langle\rangle; !x) \\ e_2 & = & \lambda f.(f\,\langle\rangle; 1) \end{array}$$

In e_1, the local reference x can be in only two states: either 0, before the function f has been called for the first time, or 1 after that event. However, once it has changed to 1, it will never change back, not even through reentrant calls inside the callback f.

In the proof we can represent the two states by an island p with

$$\begin{array}{lcl} B & = & a \subseteq \{1\} \\ H & = & (\Box a \equiv \emptyset * x \hookrightarrow_1 0) \vee (\Box a \equiv \{1\} * x \hookrightarrow_1 1) \end{array}$$

Note that we encode the state 0 by the empty set, because populations can only grow monotonically.

The proof first proceeds somewhat similarly to the previous one, until we reach the point of showing the function bodies equivalent:

$$\{H\}\ (x := 1; f_1\,\langle\rangle; !x) \approx (f_2\,\langle\rangle; 1)\ \{\mathcal{V}[\![\mathsf{int}]\!]\}$$

where $(f_1, f_2) \in \mathcal{V}[\![\mathsf{unit} \to \mathsf{unit}]\!]$. Here, we apply rule SEP-$\vee$.

In either case, we step forward the computations until the calls to f_1 and f_2—both cases will now have the heap assertion $x \hookrightarrow_1 1$. At this point, we apply ISL-UPD to update the population to $\{1\}$ and get back into the pure judgment. We can now prove $\Box(1 \in p)$ using rule POP-MONO, and cut that into the context. Then, we use $\uparrow$-BIND to bind the results of the calls to variables y_1, y_2, and proceed with proving $((y_1; !x), (y_2; 1)) \in \mathcal{E}[\![\mathsf{int}]\!]$. We apply $\uparrow$-IMPURE to return into the separation judgment and finish the proof by again applying SEP-$\vee$. In the first subcase, the new hypotheses $a \equiv \emptyset$ and $a \equiv p$ will yield a contradiction with $\Box(1 \in p)$, which remained as an assumption in the context by virtue of its being $\Box$'d. The other subcase tells $!x \rightsquigarrow 1$ right away.

9.3 Landin's Knot

We want to prove that Landin's knot—the construction of a fixed-point using backpatching—works. That is, we want to prove the equivalence of the following two expressions of type $\tau_1 \to \tau_2$:

$$\begin{array}{lcl} e_1 & = & \mathsf{let}\, z = \mathsf{ref}\, (\lambda x.\bot)\, \mathsf{in}\, (z := (\lambda x.\mathsf{let}\, f = !z\, \mathsf{in}\, e); !z) \\ e_2 & = & \mathsf{fix}\, f(x).\, e \end{array}$$

where fix is a standard call-by-value fixed-point operator, which can be defined in our language as follows:

$$\begin{array}{rcl} \mathsf{fix}\, f(x).\, e & = & \lambda x.(\mathsf{unroll}\, v)\, v\, x \\ \text{where } v & = & \mathsf{roll}\, \lambda f'.(\lambda f.\lambda x.e)\, (\lambda x.(\mathsf{unroll}\, f')\, f'\, x) \end{array}$$

We need to show $\vdash (e_1, e_2) \in \mathcal{E}[\![\tau_1 \to \tau_2]\!]$.

Let F be the function that z ends up being assigned. There are three interesting points in the proof:

1. To record the fact that z will contain F forever after the assignment, we introduce an island with the heap law $H = z \hookrightarrow_1 F$ (the populations are not relevant for this proof).
2. After that, we invoke the LÖB rule to prove that F and e_2 (the results of evaluating e_1 and e_2) are related in $\mathcal{V}[\![\tau_1 \to \tau_2]\!]$ under the inductive assumption $\triangleright(F, e_2) \in \mathcal{V}[\![\tau_1 \to \tau_2]\!]$.
3. Further into the proof, we reach a point where we have to show

   $$\{z \hookrightarrow_1 F\}\ (\mathsf{let}\, f = !z\, \mathsf{in}\, e[y_1/x]) \approx e_2(y_2)\ \{\mathcal{V}[\![\tau_2]\!]\}$$

   for some $(y_1, y_2) \in \mathcal{V}[\![\tau_1]\!]$. At this point, both computations are ready to make a "serious" step, so we can apply rule STEP-LR, using DEREF on the left and UNROLL on the right, to reduce the goal to proving

   $$\{z \hookrightarrow_1 F\}\ e[F/f][y_1/x] \approx e[e_2/f][y_2/x]\ \{\mathcal{V}[\![\tau_2]\!]\}$$

   under a $\triangleleft$'d context, *i.e.,* where the $\triangleright$-operator has been removed from the LÖB-inductive assumption relating F and e_2.

Finally, we can apply the rules ISL-UPD and $\uparrow$-REDUCE (backwards) to β-expand the two terms to $(\lambda f.\lambda x.e)(F)(y_1)$ and $(\lambda f.\lambda x.e)(e_2)(y_2)$. The result then follows from the Fundamental Property (which says that $\lambda f.\lambda x.e$ is related to itself), together with the relatedness of (F, e_2) and (y_1, y_2).

10. Conclusion and Related Work

In this paper, we have presented a relational modal logic, LADR, for reasoning about program equivalence in a language with higher types, type abstraction, recursive types, and (local) higher-order state. To our knowledge, this is the first logic for reasoning syntactically about contextual equivalence of programs in such a rich language. The model of the logic is based closely on the step-indexed Kripke logical relation developed recently by Ahmed, Dreyer, and Rossberg [2], but by using the high-level proof rules of the logic, we can abstract away many of the messy details of the ADR model.

Comparison to ADR Before relating LADR to ADR in detail, let us begin by noting that there is a huge amount of prior work on using logical relations to reason about contextual equivalence in higher-order languages with some subset of recursive types, existential types, and local state (*e.g.,* [1, 8, 10, 23, 30, 31, 37]). There has also been significant work on using bisimulations for the same purpose (*e.g.,* [21, 22, 39, 41, 43]). We refer the reader to [2] for detailed comparisons between the ADR model and previous approaches. More recently, Sumii has developed an *environmental bisimulation* method that handles all the examples in ADR and more [42], although some of them involve tricky proofs that reason explicitly about induction on program contexts. We are very interested in exploring the connection between step-indexed Kripke logical relations and environmental bisimulations, and hope that the present work may serve as a stepping stone to a unified, logical understanding of both methods.

It is difficult to precisely compare the expressiveness of LADR with the original ADR model, or indeed to precisely compare the expressiveness of any two methods developed in this area (including all the aforementioned logical relations and bisimulation techniques). It is usually impossible to pinpoint exactly what class of equivalences can be proven in any model/logic, partly because it depends on what one means by "can be proven". For instance, many of the bisimulation techniques are touted as being "complete" with respect to contextual equivalence, but completeness *per se* gives no indication of whether a technique is effective at proving anything. (After all, contextual equivalence is complete with respect to itself!) For this reason, the literature tends to be very example-driven, with each paper attempting to handle all or most of the previously proposed examples, as well as some new class of examples.

The ADR paper, to which some of the present authors contributed, presented a whole range of new and interesting examples (concerning stateful ADTs), precisely in order to better suggest what the ADR model was capable (and incapable) of. But ADR, being state-of-the-art, is also capable of handling the other standard examples from the literature, such as the Meyer-Sieber examples [24]. For the present paper, our aim was for LADR to handle all the examples from the ADR paper (that ADR can handle) with the exception of one (described below), and for the LADR proofs to be much shorter and cleaner (which they are). Moreover, like ADR, LADR also handles the other well-known examples from the literature (*e.g.,* those in [24, 31]).

The one example that the ADR paper shows how to prove, but that LADR (we believe) cannot, is the "callback with lock" example, based on the reentrant callback example of Banerjee and Naumann [6]. The ADR proof for that example is, one might say, rather hacky—it "messes with the steps" in order to fake a very poor man's temporal logic. It is therefore not expressible in our logic, which treats step-indices in a more abstract way. As the authors of ADR note, the proof was included in the paper primarily because it was surprising that the model was capable of handling the example at all. However, the proof is quite inelegant, and rather than develop a method for expressing the ADR proof for "callback with lock" more abstractly, we are actively working on developing an altogether different, cleaner proof for it.

Our model of LADR is very similar to the ADR model. The primary characteristics of the ADR model (namely, the treatment of populations and the use of step-indexing to stratify the worlds) are essentially identical in both models. The differences between the models are minor ones aimed at simplifying unnecessary or overcomplicated aspects of the ADR model.

For example, the ADR model enforced "separation" of islands in worlds by attaching to each island a store typing that explicitly listed the set of memory locations governed by that island. We instead define the heap-world satisfaction relation ($h_1, h_2 : W$ in Figure 5) separation-logic-style, *i.e.,* there must be *some* way of splitting up the heaps so that each disjoint pair of heaps satisfies a corresponding island. We adopted this approach here because it seems to simplify the model and permits "ownership transfer" of locations between islands (although we are not yet sure how to exploit this feature).

Another difference is that the ADR model was constructed from syntactically well-typed terms, whereas our present model does not make that restriction. We diverged from ADR on this point as an experiment, in order to see if the syntactic typing restriction in ADR was really necessary or was just cluttering up proofs. The short answer is that, for the purpose of the ADR proofs, it is not necessary. That said, we know of some other examples where syntactic typing is a useful assumption (*e.g.,* so that one can make use of "syntactic" properties like "canonical forms"). In any case, it seems straightforward to define a variant of LADR that is restricted to well-typed terms, if so desired.

Other Related Work Honsell, Mason, Smith, and Talcott [20] and Yoshida, Honda, and Berger [45] have proposed logics for reasoning about higher-order programs with local state, the former equational and the latter Hoare-style. Neither deals with type abstraction or the kinds of reasoning afforded by ADR's populations.

Separation logic, a highly influential variant of Hoare logic for reasoning about programs with pointers, was originally developed by Reynolds, O'Hearn, and others for low-level languages (see the survey in [38]). However, in recent years, variants of separation logic have also been developed for modular verification of OO languages [14, 28], as well as for languages with higher types and/or higher-order state [11, 26, 36]. All of these approaches, however, essentially only handle *strong* memory updates, not ML-style reference types, which enjoy stronger invariants.

A key idea in separation logic is the *frame* rule, which allows one to use separating conjunction to "frame-in" additional resources to the pre- and post-conditions of Hoare triples. O'Hearn, Yang, and Reynolds [27] showed how to extend the original (first-order) frame rule to second-order, and subsequently Birkedal, Torp-Smith, and Yang [11] discovered how to prove higher-order frame rules sound by means of a possible-world semantics for specifications. Their approach is to effectively bake the frame rule into the interpretation of Hoare triples. In LADR, the model of our "separation" judgment involves a similar baking-in of the frame rule.

Birkedal and Yang [12] devised a relational model of separation logic, for a language with higher types but flat store. Their model was explicitly intended as a first step towards a logic for stateful ADTs. However, they only developed the model, not a corresponding logic for syntactic reasoning. They proposed, however, that a logic for syntactic reasoning might employ ideas from Yang's earlier work on *relational separation logic* [44], as we have done here.

Concurrently with our work, Hobor, Dockins, and Appel [19] have investigated a more abstract description of step-indexed models by means of certain section-retraction pairs. Their `squash` and `unsquash` functions seem to be at least superficially similar to our $\triangleright$ and $\triangleleft$ operations on worlds, respectively, so it would be interesting to see if their approach can be usefully applied to give a simpler account of the (L)ADR models. We believe there are some non-trivial challenges because propositions in the (L)ADR models are required to be monotone with respect to time and width extension; on the face of it, Hobor *et al.*'s setup does not accommodate such monotonicity requirements directly.

Also concurrently with our work, Pilkiewicz and Pottier [29] propose a way to reason about monotonic state changes in an expressive capability type system. The basic capability type system (borrowed from Charguéraud and Pottier [13]) incorporates strong references, whose types can be updated via assignment; capabilities are used to restrict sharing of such strong references for the purpose

of ensuring soundness. The idea of Pilkiewicz and Pottier is that it is sound for multiple clients to share references if the state evolves in a monotonic way and clients only rely on such monotonic state changes. Monotonic state changes are described using so-called fates and laws, which are related to our use of populations and laws. Using this idea of monotonic state change, and by applying a generalized version of Pottier's *anti-frame rule* for hiding local state [34, 35], Pilkiewicz and Pottier are able to give types that express the behavior of the example programs considered in ADR. Pilkiewicz and Pottier's type system cannot, however, be used to prove contextual equivalence of programs, and it has not yet been proven sound. Recently, Schwinghammer *et al.* have proven the anti-frame rule sound in a separation-logic setting [40], and there is hope that this result can be extended to a soundness proof of Pilkiewicz and Pottier's system, but there are also some technical difficulties ahead (*e.g.,* modeling the generalized anti-frame rule).

We view logical relations and separation logic as largely complementary tools, and the design of LADR is (we think) particularly interesting because it shows how both approaches can cooperate effectively within one logic. Admittedly, our use of separation logic is fairly limited, but this is deliberate on our part (as a way of simplifying matters at first). We are keen to explore ways of generalizing LADR to support richer forms of separation logic reasoning, and to explore further the connection with higher-order separation logic.

Acknowledgments

The first author would like to thank Amal Ahmed for helpful conversations early in the development of this work.

References

[1] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *ESOP*, 2006.

[2] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *POPL*, 2009.

[3] A. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *TOPLAS*, 23(5):657–683, 2001.

[4] A. Appel, P.-A. Melliès, C. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *POPL*, 2007.

[5] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *POPL*, 2008.

[6] A. Banerjee and D. A. Naumann. State based ownership, reentrance, and encapsulation. In *ECOOP*, 2005.

[7] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL*, 2004.

[8] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *TLCA*, 2005.

[9] N. Benton and N. Tabareau. Compiling functional types to relational specifications for low level imperative code. In *TLDI*, 2009.

[10] L. Birkedal, K. Støvring, and J. Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In *FOSSACS*, 2009.

[11] L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules. *LMCS*, 2(5:1), 2006.

[12] L. Birkedal and H. Yang. Relational parametricity and separation logic. *LMCS*, 4(2:6), 2008.

[13] A. Charguéraud and F. Pottier. Functional translation of a calculus of capabilities. In *ICFP*, 2008.

[14] W.-N. Chin, C. David, H. H. Nguyen, and S. Qin. Enhancing modular OO verification with separation logic. In *POPL*, 2008.

[15] R. Dockins, A. W. Appel, and A. Hobor. Multimodal separation logic for reasoning about operational semantics. In *MFPS*, 2008.

[16] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *LICS*, 2009.

[17] D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs (Technical appendix), 2009. URL: `http://www.mpi-sws.org/~dreyer/papers/ladr/`.

[18] A. Hobor, A. Appel, and F. Zappa Nardelli. Oracle semantics for concurrent separation logic. In *ESOP*, 2008.

[19] A. Hobor, R. Dockins, and A. Appel. A theory of indirection via approximation. In *POPL*, 2010.

[20] F. Honsell, I. A. Mason, S. Smith, and C. Talcott. A variable typed logic of effects. *Inf. Comput.*, 119(1):55–90, 1995.

[21] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL*, 2006.

[22] S. B. Lassen and P. B. Levy. Typed normal form bisimulation for parametric polymorphism. In *LICS*, 2008.

[23] P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS*, 2005.

[24] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables. In *POPL*, 1988.

[25] H. Nakano. A modality for recursion. In *LICS*, 2000.

[26] A. Nanevski, A. Ahmed, G. Morrisett, and L. Birkedal. Abstract predicates and mutable ADTs in Hoare Type Theory. In *ESOP*, 2007.

[27] P. W. O'Hearn, H. Yang, and J. C. Reynolds. Separation and information hiding. In *POPL*, 2004.

[28] M. Parkinson and G. Bierman. Separation logic, abstraction and inheritance. In *POPL*, 2008.

[29] A. Pilkiewicz and F. Pottier. The essence of monotonic state. Submitted for publication, 2009.

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

[31] A. Pitts and I. Stark. Operational reasoning for functions with local state. In *HOOTS*, 1998.

[32] A. M. Pitts. Relational properties of domains. *Inf. Comput.*, 127:66–90, 1996.

[33] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *TLCA*, 1993.

[34] F. Pottier. Hiding local state in direct style: a higher-order anti-frame rule. In *LICS*, 2008.

[35] F. Pottier. Generalizing the higher-order frame and anti-frame rules. Unpublished, 2009.

[36] B. Reus and J. Schwinghammer. Separation logic for higher-order store. In *CSL*, 2006.

[37] J. C. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 1983.

[38] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, 2002.

[39] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *LICS*, 2007.

[40] J. Schwinghammer, H. Yang, L. Birkedal, F. Pottier, and B. Reus. A semantic foundation for hidden state. Submitted for publication, 2009.

[41] K. Støvring and S. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *POPL*, 2007.

[42] E. Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In *CSL*, 2009.

[43] E. Sumii and B. Pierce. A bisimulation for type abstraction and recursion. *JACM*, 54(5):1–43, 2007.

[44] H. Yang. Relational separation logic. *TCS*, 375(1–3):308–334, 2007.

[45] N. Yoshida, K. Honda, and M. Berger. Logical reasoning for higher-order functions with local state. *LMCS*, 4(4:2), 2008.