Decomposing Logical Relations with Forcing

Guilhem Jaber, Nicolas Tabareau

Decomposing Logical Relations with Forcing

Guilhem Jaber
ENS Cachan
guilhem.jaber@ens-cachan.fr

Nicolas Tabareau
INRIA
nicolas.tabareau@inria.fr

Abstract

Logical relations have now the maturity to deal with program equivalence for realistic programming languages with features likes recursive types, higher-order references and first-class continuations. However, such advanced logical relations—which are defined with technical developments like step-indexing or heap abstractions using recursively defined worlds—can make a proof tedious. A lot of work has been done to hide step-indexing in proofs, using Gödel-Löb logic. But to date, step-indexes have still to appear explicitely in particular constructions, for instance when building recursive worlds in a stratified way. In this paper, we go one step further, proposing an extension of Abadi-Plotkin logic with forcing construction which enables to encapsulate reasoning about step-indexing or heap in different layers. Moreover, it gives a uniform and abstract management of step-indexing for recursive terms or types and for higher-order references.

1. Introduction

1.1 Logical relations

Logical relations appear in the last decade to be a powerful technique for proving program equivalence, able to deal with concrete non-trivial equivalence of programs making use of complex features like recursive types, control operators and higher-order references. For instance, it is now well known that fix point operators or recursive types [1] can be handled by logical relations using step indexing, a technique introduced in [3]. To avoid the apparent circularity in the definition of logical relations for recursive types, step-indexes are used to stratify logical relations with a natural number representing roughly the number of steps for which the programs in question behave similarly. In this way, it becomes possible to prove the equivalence of many programs that makes use of recursive types by performing a simple induction on step-indexes. But the management of step indexes during a proof appears to be—borrowing a word from N. Benton—"ugly". That's why many authors, starting from the work of A. Appel et al. [4], have proposed to hide step-indexing using Gödel-Löb logic.

More recently, D. Dreyer et al. [2, 12] have defined step-indexed Kripke logical relations with the ability to establish properties about local state that evolve during computation in some controlled fashion. Basically, instead of using local invariants as in the seminal work of A. Pitts and I. Stark [17], the evolution of the heap is constrained by a state transition system (STS).

To encompass higher-order references, D. Dreyer et al. have to extend their notion of heap relations that are used in each node of the STSs of a world to heap relations taking a world as a parameter. While all those ideas appear to be intuitively very elegant, their precise definitions still requires to make an explicit use of step-indexing and stratification where people just would like to use Gödel-Löb logic and recursively defined set of worlds.

This paper proposes a logical setting where all those different notions can be defined modularly and combined uniformly.

1.2 Forcing for set theory

Forcing is a method originally designed by P. Cohen in the 60s to prove the independence of the Continuum Hypothesis from the axiomatic set theory ZFC [8]. The main idea is to extend a ground model M to a new model $M[\mathcal{G}]$ by adding a new *generic* element $\mathcal{G}$ to M. As $M[\mathcal{G}]$ is in general really complicated, P. Cohen has proposed to control the true propositions in $M[\mathcal{G}]$ by translating them into M. To do so, he has used *forcing conditions*, which have to be seen as approximations of $\mathcal{G}$. Such forcing conditions live in M while it is not the case for the generic element $\mathcal{G}$. Thus, from a formula φ of $M[\mathcal{G}]$, the idea is to build syntactically a formula $p \Vdash \hat{\varphi}$—pronounced "*p forces $\hat{\varphi}$*"—that lives in M, and such that φ will be true when there is a "correct" approximation p of $\mathcal{G}$ such that $p \Vdash \hat{\varphi}$ in M. One key property of forcing conditions is that they are ordered. Intuitively, $p \leq q$ when p is a more precise approximation of $\mathcal{G}$ than q, i.e. contains more information, so the relation $\Vdash$ has to be monotonous for this order. In a similar manner, J.-L. Krivine has used forcing to transform into a program any proof using the dependent choice axiom and the existence of a non-trivial ultrafilter on $\mathbb{N}$ [14]. This approach has recently been clarified by A. Miquel [16], who tries to understand proof transformation induced by forcing translation through Curry-Howard isomorphism. Finally, in [9], forcing is used to prove constructively the uniform continuity of some functionals defined in intuitionistic type theory. In this article, we do not pretend to follow precisely the line drawn by those works, but rather an alternative line promoting the general slogan:

"*Forcing can be used to add logical principles modularly.*"

This slogan will be illustrated by defining logical relations in different logic layers having more and more logical principles.

1.3 Approximating logical relations with forcing conditions

In analogy to Cohen's forcing, we advocate that forcing conditions in the setting of logical relations can be used to approximate the notion of observation. This follows the well-known fact that step-indexing can be seen as an approximation of the equitermination observation $\mathcal{O}(t_1, t_2)$ which says that the term t_1 terminates if and only if t_2 terminates.

More precisely, suppose given a logic in which logical relations can be expressed and consider the formula

$$\mathcal{E}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,(t_1, t_2)$$

that says that the two terms t_1 and t_2 are related at type $\mathrm{Unit} \to \mathrm{Unit}$. In presence of complex features in the language, we sometimes want to be able to relate terms that are not absolutely equivalent but only under some condition p, written

$$p \Vdash \mathcal{E}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,(t_1, t_2).$$

Technically, the formula above is translated into a formula of the ground logic where the forcing condition p goes through the formula monotonically until the observation

$$p \Vdash \mathcal{O}(t_1, t_2)$$

which approximates the original observation. For instance, when the forcing condition is a world w, the approximated observation says that t_1 and t_2 behave the same for all heaps related by w. When the forcing condition is a step-index n, the approximated observation says that t_1 and t_2 behave the same for n steps.

1.4 Step-indexing as forcing conditions

As indicated above, step-indexes can be rephrased as forcing conditions over integers (with the usual order) that approximate the observation. In this way, it is possible to recover uniformly many definitions given in previous works. But the really interesting part is that we can formalize the logical principles that step-indexing validate in the forcing layer.

Forcing conditions can be used to approximate existing formulas but also to give some meaning to new connector in the logic. Namely, in the step-indexed ($\mathcal{SI}$) layer, one can define the so-called "later" operator as

$$n \Vdash \triangleright\varphi \quad \stackrel{def}{=} \quad \forall m < n,\ m \Vdash \varphi$$

and show that the Löb rule is valid under forcing. To be more abstract about step-indexes, let us make explicit the logic associated to the forcing layer

$$\psi_1, \ldots, \psi_k \vdash_{\mathcal{SI}} \varphi \stackrel{def}{=} \forall n, (n \Vdash \psi_1, \ldots, n \Vdash \psi_k) \vdash (n \Vdash \varphi)$$

that is, φ is true in the $\mathcal{SI}$ layer if and only if it is forced by any step-index. This formalizes the intuition that step-indexing provides approximations of logical relations whose limits are logical relation themselves. In the $\mathcal{SI}$ layer, the Löb rule is also valid and can simply be expressed by

$$\frac{\triangleright P \vdash_{\mathcal{SI}} P}{\vdash_{\mathcal{SI}} P}$$

This means that working in the $\mathcal{SI}$ layer equipped the ground logic with an induction principle for free. But we will see in Section 4 that the $\mathcal{SI}$ layer has much more logical principles, for instance the ability to define logical relations or worlds recursively without making any explicit use of stratification. But the fact that each new formula is translated in the previous layer using the forcing relation enables to validate the new reasoning principles in the previous layer, which means that:

"*The consistency of the logic defined in a forcing layer ensues from the consistency of the ground logic.*"

The idea of using forcing to prove relative consistency results in proof theory is not new, see for example [5], but here, we apply it to type theory.

While step-indexing is fitted to reason on contextual approximations, it is harder to design step-indexed logical relations to reason on contextual equivalences. In [11], then refined in [13], a method based on a *direction variable* is given, which corresponds in our framework to a simple forcing with only two conditions $\{l, r\}$.

1.5 Kripke worlds as forcing conditions

Turning back to the seminal work of A. Pitts and I. Stark [17], let us consider forcing conditions over a set of worlds defined by

$$\mathrm{World} \quad \stackrel{def}{=} \quad \mathrm{Heap} \times \mathrm{Heap} \to \mathrm{Prop}$$

A world is thus a relation that states which heaps are in relation in this world. A world w' is in the future of a world w (noted $w' \sqsupseteq w$) when w' extends w on new locations. Then, defining the relation $\mathcal{O}_w$ as t_1 and t_2 behave the same for all heaps related by w, we can recover the usual definition of Kripke logical relations (see Section 5 for details). Note that the judgement

$$w \Vdash \varphi$$

was already present in the work of A. Appel et al. [4], as the definition of Kripke model, for which truth is not absolute but relative to some appropriate notion of world.

As far as higher order references are concerned, we can no longer define worlds in this simple manner since references can point to other references whose relation depends on the world. So one would like to define world recursively as

$$\mathrm{World} \stackrel{def}{=} (\mathrm{Heap} \times \mathrm{Heap}) \to \mathrm{World} \to \mathrm{Prop}$$

However, by a cardinality argument, there is no solution to this equation in Set. This has lead many authors (see [4] or [12], for instance) to work with a stratified approximated solution to this equation. But it seems that step-indexing is coming into the picture again. Indeed, in [4] or [12], worlds are indexed by an integer that decreases when time advances.

One of the main contributions of this paper is to show that this stratified construction can be recovered by defining worlds **in the step-indexed ($\mathcal{SI}$) layer**, using directly the possibility to define recursive kinds.

2. The Logic FLR

2.1 The language $F^{\mu!}$

We consider the language $F^{\mu!}$; a standard call-by-value λ-calculus with a fixpoint construction, recursive types, universal polymorphism, and ML-like references. For simplicity, we do not have products, sums or existential types, but they could be added easily following [13]. The syntax and the operational semantics are given in in Figure 1, while the typing rules can be found in Appendix.

2.2 Contextual equivalence

Let Δ be a context for the free type variables, Υ for the locations used by the term, and Γ for the free term variables.

$$
\begin{array}{rcl}
\tau, \sigma & \stackrel{def}{=} & \text{Unit} \mid \text{Nat} \mid \alpha \mid \tau \to \sigma \mid \forall\alpha.\tau \mid \mu\alpha.\tau \mid \text{ref}\,\tau \\
v & \stackrel{def}{=} & () \mid n \mid l \mid \lambda x.M \mid \text{fix}\, f(x).M \mid \Lambda\alpha.M \mid \text{roll}\, v \\
 & & (\text{where } n \in \text{Nat}, l \in \text{Loc}) \\
M, N & \stackrel{def}{=} & v \mid x \mid \tau \mid MN \mid \text{roll}\, M \mid \text{unroll}\, M \mid \text{ref}\, M \mid \\
 & & !M \mid M := N \\
K & \stackrel{def}{=} & \circ \mid KM \mid K\tau \mid vK \mid \text{roll}\, K \mid \text{unroll}\, K \mid \\
 & & \text{ref}\, K \mid !K \mid K := M \mid v := K \\
h & \stackrel{def}{=} & \text{emp} \mid h[l \mapsto v] \mid h \bullet [l \mapsto v]
\end{array}
$$

$$
\begin{array}{rcl}
((\lambda x.M)v, h) & \mapsto & (M\{v/x\}, h) \\
(\text{fix}\, f(x).M)v, h) & \mapsto & (M\{v/x\}\{\text{fix}\, f(x).M/f\}, h) \\
((\Lambda\alpha.M)\tau, h) & \mapsto & (M\{\tau/\alpha\}, h) \\
(\text{unroll}\,(\text{roll}\, M), h) & \mapsto & (M, h) \\
(!l, h) & \mapsto & (v, h) \text{ when } h(l) = v \\
(\text{ref}\, v, h) & \mapsto & (l, h \bullet [l \mapsto v]) \text{ with } l \notin dom(h) \\
(l := v, h) & \mapsto & ((), h[l \mapsto v]) \text{ when } l \in dom(h)
\end{array}
$$

$$
\frac{(M_1, h_1) \mapsto (M_1, h_2)}{(K[M_1], h_1) \mapsto (K[M_2], h_2)}
$$

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

Contexts C are defined as terms with a hole [.] anywhere in the term, and have a type $\vdash C : (\Delta; \Upsilon; \Gamma \vdash \tau) \to (\Delta'; \Upsilon'; \Gamma' \vdash \sigma)$ saying that for any M with $\Delta; \Upsilon; \Gamma \vdash M : \tau$, we have $\Delta'; \Upsilon'; \Gamma' \vdash C[M] : \sigma$. The typing rules of context are usual and can be found for example in the additional details of [11]. The contextual equivalence $M_1 \simeq_{ctx} M_2 : \tau$ is then defined over arbitrary context C with a hole:

$$M_1 \simeq_{ctx}^{\Delta;\Upsilon;\Gamma} M_2 : \tau$$

means that $\Delta; \Upsilon; \Gamma \vdash M_1, M_2 : \tau$ and for all contexts C (such that $\vdash C : (\Delta; \Upsilon; \Gamma \vdash \tau) \to (\Delta'; \Upsilon'; \Gamma' \vdash \sigma)$), $C[M_1]$ and $C[M_2]$ equiterminate.

2.3 Syntax of FLR

Unlike Abadi-Plotkin logic [18] and all the successive developments like LSLR and LADR, which are based on second-order logic, we use higher-order logic. The resulting logic, called FLR, constitutes a language in which logical relations—and their extension using a forcing condition—can be defined.

FLR is an extension of (Curry style) $F\omega$ with implicit dependent types and a constructor for finite partial functions. We call kinds the types of the logic, to be differentiated from types of $F^{\mu!}$. Kinds are defined as :

$$
\begin{array}{l}
T, U := \text{Nat}_p \mid \text{Loc} \mid \text{Term} \mid \text{Val} \mid \text{Cont} \mid \text{Type} \mid \text{Prop} \mid \tau \mid \\
\quad \top \mid \bot \mid T \times U \mid T \to U \mid T \to_{fin} U \mid \forall x : T.U
\end{array}
$$

where $p \in \mathbb{N}$. Cont will be the kind of contexts, while Type will be the kind of types of $F^{\mu!}$.

The atomic propositions are: (1) $M = N$: syntactical equality on terms, locations and heaps; (2) $(M, h_1) \to (N, h_2)$: reduction relation on pairs of term and heap; (3) abstract notion of observation $\mathcal{O}$ on pairs of term and heap; (4) left and right approximations of the observation $\mathcal{O}_n^l, \mathcal{O}_n^r$. The idea is that $\mathcal{O}_n^l, \mathcal{O}_n^r$ are approximations of $\mathcal{O}$, controlled by the forcing conditions. Dealing with equitermination, $\mathcal{O}_n^l$ will represent the left approximation for at most n steps, i.e.

"$\mathcal{O}_n^l((t_1, h_1), (t_2, h_2))$ iff for all $k \le n$, if (t_1, h_1) can be reduced k times then (t_2, h_2) terminates."

It is axiomatized by the logical rules given in Section 2.4. Formulas are defined by :

$$
\begin{array}{rcl}
P, Q & := & n \mid l \mid M \mid K \mid \tau \mid \mathcal{O} \mid \mathcal{O}_n^l \mid \mathcal{O}_n^r \mid M = N \mid \\
 & & (M, h_1) \to (N, h_2) \mid P \wedge Q \mid P \vee Q \mid \\
 & & P \Rightarrow Q \mid \forall x : T.P \mid \exists x : T.P \mid (x).P \mid \\
 & & \pi_1 P \mid \pi_1 Q \mid \langle P, Q \rangle \mid P \in Q
\end{array}
$$

In the following, $\mathcal{C}$ denotes a logical context $\bar{P}$, i.e. a list of propositions used as hypothesis, while Γ will be a typing context. We abbreviate $Q \in P$ as $P(Q)$. The well-formedness of propositions is ensured by standard typing rules given in Figure 2.

2.4 Inference rules of FLR

Our logic comes with a congruence $\cong$ on formulas, which take into account β and η equality. It is defined as the smallest congruence satisfying the following axioms :

$$
\begin{array}{rcl}
t \in x.P & \cong & P\{t/x\} \\
x.(x \in P) & \cong & P \text{ when } x \notin FV(P) \\
\pi_1 \langle P, Q \rangle & \cong & P \\
\pi_2 \langle P, Q \rangle & \cong & Q
\end{array}
$$

There are also inference rules describing the reduction of $F^{\mu!}$, mimicking the operational semantics of Figure 1.

FLR is equipped with standard inference rules given in Figure 3. Approximated observation relations $\mathcal{O}_n^l$ and $\mathcal{O}_n^r$ satisfy specific inference rules (we only present the rules for l):

$$
\frac{\Gamma; \mathcal{C} \vdash \mathcal{O}_k^l((M_1, h_1), (M_2, h_2)) \qquad \Gamma; \mathcal{C} \vdash k < n}{\Gamma; \mathcal{C} \vdash \mathcal{O}_n^l((M_1, h_1), (M_2, h_2))}
$$

$$
\frac{\Gamma; \mathcal{C} \vdash (M_1, h_1) \to (M_1', h_1') \qquad \Gamma; \mathcal{C} \vdash \forall k < n.\mathcal{O}_k^l((M_1', h_1'), (M_2, h_2))}{\Gamma; \mathcal{C} \vdash \mathcal{O}_n^l((M_1, h_1), (M_2, h_2))}
$$

Note that in the ground logic, there is no need for inference rules for $\mathcal{O}$. The expected inference rules will appear when working in the step-indexed forcing layer.

The coherence of FLR can be ensured by constructing a model adapted from the coherent space model of Miquel [15]. Our logic is just a fragment of its logic (without universes and dependent types) with the additional presence of $\top$, $\bot\!\!\!\bot$ and Nat_p base sorts.

2.5 Partial finite function

To model heaps and world, we will need partial functions with finite support as we must guarantee to have free space in the heap[1]. That is Heap must be defined as $\text{Loc} \to_{fin} \text{Val}$.

We use the kind $A \to_{fin} B$ of partial finite functions from A to B, a predicate dom on them, plus the empty function emp and two constructors $f \bullet g$ and $[u \mapsto v]$. We also have a constructor $f[u \mapsto v]$ which enables to change the value of a partial function f.

The usual typing rules are given in Annexe and the logical rules are:

$$
\frac{\Gamma; \mathcal{C} \vdash u \notin \text{dom}(f)}{\Gamma; \mathcal{C} \vdash (f \bullet [u \mapsto v])(u) = v} \qquad \frac{\Gamma; \mathcal{C} \vdash u \in \text{dom}(f)}{\Gamma; \mathcal{C} \vdash (f[u \mapsto v])(u) = v}
$$

[1] It seems to be possible to encode them in our logic, like in [19], but we prefer to define them directly in our system.

$$\frac{q \le p}{\Gamma \vdash q : \mathrm{Nat}_p} \qquad \frac{}{\Gamma \vdash M : \mathrm{Term}} \qquad \frac{}{\Gamma \vdash v : \mathrm{Val}} \qquad \frac{}{\Gamma \vdash K : \mathrm{Cont}} \qquad \frac{}{\Gamma \vdash l : \mathrm{Loc}} \qquad \frac{}{\Gamma \vdash \tau : \mathrm{Type}} \qquad \frac{}{\Gamma \vdash P : \top}$$

$$\frac{}{\Gamma, x : \bot \vdash P : T} \quad \text{for any } P \qquad \frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : T \vdash P : U}{\Gamma \vdash x.P : T \to U} \qquad \frac{\Gamma \vdash P : T \to U \qquad \Gamma \vdash Q : T}{\Gamma \vdash Q \in P : U} \qquad \frac{\Gamma \vdash P : T \times U}{\Gamma \vdash \pi_1 P : T}$$

$$\frac{\Gamma \vdash P : T \times U}{\Gamma \vdash \pi_2 P : U} \qquad \frac{\Gamma \vdash P : T \qquad \Gamma \vdash Q : U}{\Gamma \vdash \langle P, Q \rangle : T \times U} \qquad \frac{\Gamma \vdash P : T \qquad \Gamma \vdash T <: U}{\Gamma \vdash P : U} \qquad \frac{\Gamma, x : T \vdash P : U}{\Gamma \vdash P : \forall x : T.U}$$

$$\frac{\Gamma \vdash P : \mathrm{Prop} \qquad \Gamma \vdash Q : \mathrm{Prop}}{\Gamma \vdash P \Rightarrow Q : \mathrm{Prop}} \qquad \frac{\Gamma \vdash P : \mathrm{Prop} \qquad \Gamma \vdash Q : \mathrm{Prop}}{\Gamma \vdash P \wedge Q : \mathrm{Prop}} \qquad \frac{\Gamma \vdash P : \mathrm{Prop} \qquad \Gamma \vdash Q : \mathrm{Prop}}{\Gamma \vdash P \vee Q : \mathrm{Prop}}$$

$$\frac{\Gamma, x : T \vdash P : \mathrm{Prop}}{\Gamma \vdash \forall x : T.P : \mathrm{Prop}} \qquad \frac{\Gamma, x : T \vdash P : \mathrm{Prop}}{\Gamma \vdash \exists x : T.P : \mathrm{Prop}} \qquad \frac{\Gamma \vdash M_1, M_2 : \mathrm{Term} \qquad \Gamma \vdash h_1, h_2 : \mathrm{Heap}}{\Gamma \vdash \tilde{\mathcal{O}}((M_1, h_1), (M_2, h_2)) : \mathrm{Prop}}$$

$$\frac{\Gamma \vdash M_1, M_2 : \mathrm{Term} \qquad \Gamma \vdash h_1, h_2 : \mathrm{Heap}}{\Gamma \vdash (M_1, h_1) \mapsto (M_2, h_2) : \mathrm{Prop}} \qquad \frac{\Gamma \vdash M_1, M_2}{\Gamma \vdash M_1 = M_2 : \mathrm{Prop}} \qquad \frac{\Gamma \vdash h_1, h_2 : \mathrm{Heap}}{\Gamma \vdash h_1 = h_2 : \mathrm{Prop}}$$

Figure 2. Typing rules of FLR

$$\frac{P \in \mathcal{C}}{\Gamma; \mathcal{C} \vdash P} \qquad \frac{\Gamma; \mathcal{C} \vdash P \Rightarrow Q \qquad \Gamma; \mathcal{C} \vdash P}{\Gamma; \mathcal{C} \vdash Q} \qquad \frac{\Gamma; \mathcal{C}, P \vdash Q}{\Gamma; \mathcal{C} \vdash P \Rightarrow Q} \qquad \frac{\Gamma; \mathcal{C} \vdash P \qquad P \cong P'}{\Gamma; \mathcal{C} \vdash P'} \qquad \frac{}{\Gamma, q : \mathrm{Nat}_p; \mathcal{C} \vdash q \le p}$$

$$\frac{\Gamma; \mathcal{C} \vdash P \qquad \Gamma; \mathcal{C} \vdash Q}{\Gamma; \mathcal{C} \vdash P \wedge Q} \qquad \frac{\Gamma; \mathcal{C} \vdash P \wedge Q}{\Gamma; \mathcal{C} \vdash P} \qquad \frac{\Gamma; \mathcal{C} \vdash P \wedge Q}{\Gamma; \mathcal{C} \vdash Q} \qquad \frac{\Gamma; \mathcal{C} \vdash P}{\Gamma; \mathcal{C} \vdash P \vee Q} \qquad \frac{\Gamma; \mathcal{C} \vdash Q}{\Gamma; \mathcal{C} \vdash P \vee Q}$$

$$\frac{\Gamma; \mathcal{C} \vdash P \vee Q \qquad \Gamma; \mathcal{C}, P \vdash R \qquad \Gamma; \mathcal{C}, Q \vdash R}{\Gamma; \mathcal{C} \vdash R} \qquad \frac{\Gamma, x : T; \mathcal{C} \vdash P \qquad x \notin \mathcal{C}}{\Gamma; \mathcal{C} \vdash \forall x : T.P} \qquad \frac{\Gamma; \mathcal{C} \vdash \forall x : T.P \qquad \Gamma \vdash t : T}{\Gamma; \mathcal{C} \vdash P\{t/x\}}$$

$$\frac{\Gamma; \mathcal{C} \vdash P\{t/x\}}{\Gamma; \mathcal{C} \vdash \exists x : T.P} \qquad \frac{\Gamma; \mathcal{C} \vdash \exists x : T.P \qquad \Gamma, t : T; \mathcal{C}, P\{t/x\} \vdash Q}{\Gamma; \mathcal{C} \vdash Q}$$

Figure 3. Basic Logical rules of FLR

plus the usual commutativity and associativity equality on the two constructors $\bullet$ and $[u \mapsto v]$, extending the congruence $\cong$.

2.6 Subtyping

To eliminate implicit dependent types, we use a standard subtyping relation that is a preorder, is covariant in both arguments of the product types and validates the inference rules of Figure 4. Even if a rule like $\top \to \top <: \top$ could appear to be non-standard, one can check that it is satisfied in the slightly adapted model of [15].

2.7 Logical relations

We can already build a logical relation in this logic for the pure, polymorphic fragment of $F^{\mu!}$. Logical relations, built by induction on types and defined on the following kinds

$$\mathrm{Rel} \stackrel{def}{=} (\mathrm{Term} \times \mathrm{Term}) \to \mathrm{Prop}\ , \quad \mathrm{Rel}_{\tau,\sigma} \stackrel{def}{=} (\tau \times \sigma) \to \mathrm{Prop}$$

are given in Figure 5. The definition of logical relations for polymorphic types uses an environment η which maps free types to relations. It is important to notice that the definition of $\mathcal{E}[\![\tau]\!]$ is made by biorthogonality, which is central in our work since it is the only place where the observation $\mathcal{O}$ appears. Note that for the definitions of $\mathcal{S}[\![\Upsilon]\!]$ and $\simeq_{log}$, we anticipate Section 5 and use the logical relation on reference types.

$$\frac{}{\Gamma \vdash T <: \top} \qquad \frac{}{\Gamma \vdash \top \to \top <: \top} \qquad \frac{q \le p}{\Gamma \vdash \mathrm{Nat}_q <: \mathrm{Nat}_p}$$

$$\frac{}{\Gamma \vdash \mathrm{Nat}_p <: \mathrm{Nat}} \qquad \frac{\Gamma \vdash T <: T' \qquad \Gamma \vdash U <: U'}{\Gamma \vdash T \times U <: T' \times U'}$$

$$\frac{\Gamma \vdash T' <: T \qquad \Gamma \vdash U <: U'}{\Gamma \vdash T \to U <: T' \to U'} \qquad \frac{\Gamma \vdash t : T}{\Gamma \vdash \forall x : T.U <: U\{t/x\}}$$

$$\frac{\Gamma \vdash T' <: T \qquad \Gamma, x : T' \vdash U <: U'}{\Gamma \vdash \forall x : T.U <: \forall x : T'.U'}$$

Figure 4. Subtyping rules

3. Forcing layers

To increase the power of our logic, we will extend it with new constructions, and new reasoning principles (e.g. the Löb

$$
\begin{array}{lcl}
\mathcal{V}\left[\!\left[\alpha\right]\!\right]_\eta & = & \eta(\alpha) \\
\mathcal{V}\left[\!\left[\mathrm{Unit}\right]\!\right]_\eta & = & (v_1, v_2).v_1 = v_2 = () \\
\mathcal{V}\left[\!\left[\mathrm{Nat}\right]\!\right]_\eta & = & (v_1, v_2).\exists n : \mathrm{Nat}.v_1 = v_2 = n \\
\mathcal{V}\left[\!\left[\tau \to \sigma\right]\!\right]_\eta & = & (u_1, u_2).\forall v_1, v_2 \in \mathcal{V}\left[\!\left[\tau\right]\!\right](v_1, v_2), \\
 & & (u_1 v_1, u_2 v_2) \in \mathcal{E}\left[\!\left[\sigma\right]\!\right] \\
\mathcal{V}\left[\!\left[\forall\alpha.\tau\right]\!\right]_\eta & = & (u_1, u_2).\forall\alpha_1, \alpha_2 : \mathrm{Type}, \forall r : \mathrm{Rel}_{\alpha_1,\alpha_2}. \\
 & & (u_1\alpha_1, u_2\alpha_2) \in \mathcal{E}\left[\!\left[\tau\right]\!\right]_{\eta,\alpha\mapsto r} \\
 & & \\
\mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta & = & (K_1, K_2).\forall v_1, v_2 \in \mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta, \forall h_1, h_2 : \mathrm{Heap}, \\
 & & \mathcal{O}((K_1[v_1], h_1), (K_2[v_2], h_2)) \\
\mathcal{E}\left[\!\left[\tau\right]\!\right]_\eta & = & (M_1, M_2).\forall K_1, K_2 \in \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta, \forall h_1, h_2 : \mathrm{Heap}, \\
 & & \mathcal{O}((K_1[M_1], h_1), (K_2[M_2], h_2)) \\
 & & \\
\mathcal{D}\left[\!\left[\Delta, \alpha\right]\!\right] & = & (\eta).\exists\eta' \in \mathcal{D}\left[\!\left[\Delta\right]\!\right].\exists R : \mathrm{VRel}.\eta = \eta' \bullet [\alpha \mapsto R] \\
\mathcal{G}\left[\!\left[\Gamma, x; \tau\right]\!\right]_\eta & = & (\gamma).\exists\gamma' \in \mathcal{G}\left[\!\left[\Gamma\right]\!\right].\exists(v_1, v_2) \in \mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta. \\
 & & \gamma = \gamma' \bullet [x \mapsto (v_1, v_2)] \\
\mathcal{S}\left[\!\left[\Upsilon, l : \tau\right]\!\right] & = & (l, l) \in \mathcal{V}\left[\!\left[\mathrm{ref}\,\tau\right]\!\right]_\varnothing \wedge \mathcal{S}\left[\!\left[\Upsilon\right]\!\right]
\end{array}
$$

$$
\begin{array}{lcl}
M_1 \simeq_{log}^{\Delta;\Upsilon;\Gamma} M_2 : \tau & = & \forall\eta \in \mathcal{D}\left[\!\left[\Delta\right]\!\right], \forall\gamma \in \mathcal{G}\left[\!\left[\Gamma\right]\!\right]_\eta. \\
 & & \mathcal{S}\left[\!\left[\Upsilon\right]\!\right]_\eta \Rightarrow (\eta_1\gamma_1 M_1, \eta_2\gamma_2 M_2) \in \mathcal{E}\left[\!\left[\tau\right]\!\right]_\eta
\end{array}
$$

Figure 5. Definition of logical relations

rule in the $\mathcal{SI}$ layer). Such extensions, organized in layers, will be translated in the core logic FLR. To do that, we will use forcing conditions from a partially ordered set $(\mathcal{P}\ \leq)$, and the forcing relation

$$p \Vdash \varphi$$

for any $p \in \mathcal{P}$, which carries the formula φ from a forcing layer to FLR. Then, the consistency of a forcing layer will follow directly from the consistency of FLR, since all new inference rules of a layer can be translated directly in FLR. Among others, the abstract observation $\mathcal{O}$ will take a different meaning depending on which layer it is considered. Note that sorts and terms dealing with finite functions are not forced (that is the translation is the identity) and will not appear in the rest of this section. When working in different forcing layers, the meaning of logical relations will change even if their definition will *remain the same*.

3.1 Translation of sorts

A typing judgement of a forcing layer

$$\Gamma \vdash_{\mathcal{P};\bar{F}} P : T$$

will be translated in the underlying layer as

$$p : \mathcal{P}, [\Gamma]_p \vdash_{\bar{F}} [P] : [T]_p.$$

To define the translation of a kind T, there are two choices.

In some cases, it is necessary to impose a stratification on kinds using forcing conditions, such a translation will be noted $[T]_p$. We will called it *higher-order forcing*. This will be particularly useful for the layer $\mathcal{SI}$, where we will have to translate recursive kinds.

$$
\begin{array}{lcl}
[\mathrm{Base}]_p & \stackrel{def}{=} & \mathcal{P}_p \to \mathrm{Base} \\
[T \to U]_p & \stackrel{def}{=} & \forall p' : \mathcal{P}_p.[T]_{p'} \to [U]_{p'} \\
[\forall x : T.U]_p & \stackrel{def}{=} & \forall p' : \mathcal{P}_p.\forall x : [T]_{p'}.[U]_{p'} \\
[T \times U]_p & \stackrel{def}{=} & [T]_p \times [U]_p
\end{array}
$$

where $\mathcal{P}_p = \{q \mid q \leq p\}$. Note that $\mathcal{P}_p$ is not directly definable in our logic since we do not have subset types. This explain the presence of Nat_p to deal with the layer $\mathcal{SI}$. We will always suppose to have $\mathcal{P}_p <: \mathcal{P}_q$ as soon as $p \leq q$

This translation is close to the tripos to topos construction. Indeed, the translation can be seen as a presheaf on $\mathcal{P}$. This justifies the definition of $T \to U$, since we need a property of monotonicity for functions between two presheaves.

In more simple cases, translation of kinds, noted $[T]$, shall not depend on forcing conditions :

$$
\begin{array}{lcl}
[\mathrm{Base}] & \stackrel{def}{=} & \mathcal{P} \to \mathrm{Base} \\
[T \to U] & \stackrel{def}{=} & [T] \to [U] \\
[\forall x : T.U] & \stackrel{def}{=} & \forall x : [T].[U] \\
[T \times U] & \stackrel{def}{=} & [T] \times [U]
\end{array}
$$

Notice that contrary to "traditional" forcing [16], even in this simpler setting, individuals (like Nat) are changed by forcing.

Note that $[T]$ corresponds morally to $\forall p : \mathcal{P}.[T]_p$, which means that the $[T]$ corresponds to the definition of constant presheafs. Finally, one define $[\Gamma]_p$ as

$$[(x : T), \Gamma]_p = (x : [T]_p), [\Gamma]_p$$

We now present a monotonicity property that corresponds to the presheaf condition in [6].

Proposition 1 (Monotonicity for $\mathcal{SI}$)**.** *For all type T and forcing conditions $q \leq p$, we have*

$$[T]_p <: [T]_q$$

Proof. The proof goes by induction on T.

- The atomic kinds are proved using the contravariance of subtyping relation at the left of the arrow, and the fact that $\mathcal{P}_q <: \mathcal{P}_p$.
- In the case of an arrow kind, we have to prove that
$$\left(\forall p' : \mathcal{P}_p.[T]_{p'} \to [T]_{p'}\right) <: \left(\forall p' : \mathcal{P}_q.[T]_{p'} \to [T]_{p'}\right)$$
which is direct by the definition of the subtyping.
- The implicit dependent kind is done in the same way.
- The product kind is direct by induction.

□

The next theorem states that the forcing translation on sorts is correct (we only present the theorem for higher-order forcing translation).

Theorem 1. *If $\Gamma \vdash_{\mathcal{P};\bar{F}} P : T$ then for all forcing condition p, we have in the underlying layer $p : \mathrm{Nat}, [\Gamma]_p \vdash_{\bar{F}} [P] : [T]_p$.*

Proof. The proof is done by induction on the proof of $\Gamma \vdash_{\mathcal{P};\bar{F}} P : T$ (and on p in case of $\mathcal{SI}$).

- The axiom rule is transformed into :
$$\frac{(x : [T]_p) \in [\Gamma]_p}{p : \mathrm{Nat}, [\Gamma]_p \vdash x : [T]_p}$$
- The abstraction rule is transformed into :
$$\frac{p : \mathrm{Nat}, [\Gamma]_p, x : [T]_p \vdash [P] : [U]_p}{p : \mathrm{Nat}, [\Gamma]_p \vdash x.[P] : \forall p' : \mathrm{Nat}_p.[T]_{p'} \to [U]_{p'}}$$
This rule can be mimic using the derivation
$$\cfrac{\cfrac{\cfrac{\cfrac{}{p : \mathrm{Nat}, p' : \mathrm{Nat}_p, [\Gamma]_{p'}, x : [T]_{p'} \vdash [P] : [U]_{p'}}\ (\text{Hyp and subtyp})}{p : \mathrm{Nat}, p' : \mathrm{Nat}_p, [\Gamma]_p, x : [T]_{p'} \vdash [P] : [U]_{p'}}\ (\text{Mono and subtyp})}{p : \mathrm{Nat}, p' : \mathrm{Nat}_p, [\Gamma]_p \vdash x.[P] : [T]_{p'} \to [U]_{p'}}\ (\text{Abs})}{p : \mathrm{Nat}, [\Gamma]_p \vdash x.[P] : \forall p' : \mathrm{Nat}_p[T]_{p'} \to [U]_{p'}}\ (\text{forall})$$

- The application rule is transformed into:

$$\frac{p : \mathrm{Nat}, [\Gamma]_p \vdash [P] : \forall p' : \mathrm{Nat}_p [T]_{p'} \to [U]_{p'} \quad p : \mathrm{Nat}, [\Gamma]_p \vdash [Q] : [T]_p}{p : \mathrm{Nat}, [\Gamma]_p \vdash [Q] \in [P] : [U]_p}$$

  The proof goes by instantiating forall at p and using the application of the underlying layer.
- Other cases are either direct or similar.

□

3.2 Translation of formulas

Following [16], we now present the translation of formulas. In the layer $\mathcal{P}$, the rule

$$\frac{\Gamma_1; \mathcal{C}_1 \vdash_{\mathcal{P}} \varphi_1 \qquad \dots \qquad \Gamma_n; \mathcal{C}_n \vdash_{\mathcal{P}} \varphi_n}{\Gamma; \mathcal{C} \vdash_{\mathcal{P}} \varphi}$$

will be valid if for all forcing condition p

$$\frac{[\Gamma_1]_p; p \Vdash \mathcal{C}_1 \vdash p \Vdash \varphi_1 \qquad \dots \qquad [\Gamma_n]_p; p \Vdash \mathcal{C}_n \vdash p \Vdash \varphi_n}{p \Vdash \Gamma; p \Vdash \mathcal{C} \vdash p \Vdash \varphi}$$

is valid in the underneath layer, where $p \Vdash \mathcal{C}$ is defined pointwisely.

The forcing relation of a formula P is noted $[P]$. If P is of type T then $[P]$ is of type $\mathcal{P} \to T$. It is defined by induction on the structure of the formula :

$$\begin{array}{lcl}
[x] & \stackrel{def}{=} & x \\
[P \wedge Q] & \stackrel{def}{=} & (p : \mathcal{P}).[P]p \wedge [Q]p \\
[P \vee Q] & \stackrel{def}{=} & (p : \mathcal{P}).[P]p \vee [Q]p \\
[P \Rightarrow Q] & \stackrel{def}{=} & (p : \mathcal{P}).\forall q \leq p.[P]q \Rightarrow [Q]p \\
[\forall x : T.P] & \stackrel{def}{=} & (p : \mathcal{P}).\forall q \leq p.\forall x : [T]_q.[P]q \\
[\exists x : T.P] & \stackrel{def}{=} & (p : \mathcal{P}).\exists x : [T]_q.[P]p \\
[x.P] & \stackrel{def}{=} & x.[P] \\
[Q \in P] & \stackrel{def}{=} & [Q] \in [P] \\
[\pi_i P] & \stackrel{def}{=} & \pi_i[P] \\
[\langle P, Q \rangle] & \stackrel{def}{=} & \langle [P], [Q] \rangle
\end{array}$$

and $p \Vdash P$ is defined as $p \in [P]$.

This translation satisfies these following important properties (proofs of Proposition 2 and Theorem 2 can be directly adapted from [16]).

Proposition 2 (substitution and congruence)**.**

1. $[P\{Q/x\}] = [P]\{[Q]/x\}$.
2. *If* $P \cong Q$, *then* $[P] \cong [Q]$.
3. $p \Vdash P\{Q/x\} = p \Vdash P\{[Q]/x\}$.
4. *If* $P \cong Q$, *then* $p \Vdash P \cong p \Vdash Q$.

Theorem 2 (Monotonicity)**.** *If* $p \Vdash P$ *and* $q \leq p$ *then* $q \Vdash P$, *as soon as it is the case for atomic propositions.*

Theorem 3. *If* $\Gamma, \mathcal{C} \vdash_{\mathcal{P};\bar{F}} P$ *then for all forcing condition* p, *we have in the underlying layer :* $[\Gamma]_p, p \Vdash \mathcal{C} \vdash_{\bar{F}} p \Vdash P$.

Proof. The proof goes by induction on the proof of $\Gamma, \mathcal{C} \vdash_{\mathcal{P};\bar{F}} P$. We only treat the case of forall elimination and introduction.

- Elimination of forall:

$$\frac{[\Gamma]_p; p \Vdash \mathcal{C} \vdash_{\bar{F}} \forall q \leq p.\forall x : [T]_q.q \Vdash P \qquad [\Gamma]_p \vdash_{\bar{F}} [t] : [T]_p}{[\Gamma]_p; p \Vdash \mathcal{C} \vdash_{\bar{F}} p \Vdash P\{t/x\}}$$

  using the elimination of forall in the layer $\bar{F}$ and substitutivity.

- Introduction of forall:

$$\frac{[\Gamma]_p, x : [T]_p; p \Vdash \mathcal{C} \vdash_{\bar{F}} p \Vdash P}{[\Gamma]_p; p \Vdash \mathcal{C} \vdash_{\bar{F}} \forall q \leq p.\forall x : [T]_q.q \Vdash P}$$

  using the introduction of forall in the base layer and monotonicity of the translations.

□

Note that the last theorem justifies the coherence of the logic defined in a forcing layer by the coherence of the logic in the underlying layer. This justifies the following inference rule to go from one layer to its underlying layer

$$\frac{\Gamma; \mathcal{C} \vdash_{\bar{\mathcal{F}};\mathcal{P}} P \qquad \Gamma \vdash_{\bar{\mathcal{F}};\mathcal{P}} p : \mathcal{P}}{\Gamma; \mathcal{C} \vdash_{\bar{\mathcal{F}}} (p \Vdash P)}$$

But, as we will see in the rest of the paper, the main interest of new forcing layers is that they can provide new reasoning principle that were not explicit in the underlying layer.

3.3 Composition of forcing layers

Forcing layers can be combined sequentially. To define a new layer $\mathcal{Q}$ over a layer $\mathcal{P}$, it suffices to define the forcing relation on atomic propositions and new logical connectors of $\mathcal{P}$.

For example, the step-indexed layer introduces a new logical connector $\triangleright$ for which, when building the Kripke layer over it, we have to define what $w \Vdash \triangleright\varphi$ is. In that case, it is not difficult as it is just $\triangleright(w \Vdash \varphi)$. Note that the order in which forcing conditions are composed—which will be represented by a stack of forcing names $\mathcal{P}_1; \dots; \mathcal{P}_n$ on the judgement symbol $\vdash$—really matters. For instance, building the Kripke layer before the $\mathcal{SI}$ layer provides from using recursive sorts in the definition of worlds.

4. Step-indexing as forcing with well-founded conditions

To avoid circular definitions in presence of recursion, it is now common to use *step-indexing* [3], which amounts to count the number of reduction steps. This technique is general enough to deal with the different programming features where recursion comes into the picture: (1) a fixed-point operator in the language[2], (2) recursive types or (3) higher-order references.

In our framework, step-indexing will appear as forcing conditions forming a well-founded order with smallest element noted 0 (it will usually be the set of integers). It gives rise to a forcing layer, noted $\mathcal{SI}$. The well-foundation is crucial to make inductive definitions correct. In $\mathcal{SI}$, the step-indexed notions of observations $\mathcal{O}^l_n$ and $\mathcal{O}^r_n$ are replaced by two abstract observations $\mathcal{O}^l$, $\mathcal{O}^r$ where indexes no longer show up. Before reviewing the particular formulas and kind constructions available in $\mathcal{SI}$, we present the definition of forcing for standard constructors of the logic. Forcing on basic formulas and kinds is defined as follows: (1) 0 forces any formula, (2) $n \Vdash \mathcal{O}^d \stackrel{def}{=} \mathcal{O}^d_n$ (for $n > 0$ and $d \in \{r, l\}$) and (3) definitions for other formulas and kinds are those given in Section 3.

4.1 Gödel-Löb Logic

In the $\mathcal{SI}$ layer, it is possible to define a modal operator $\triangleright$—as introduced in [4]—which, when forced, expresses the

[2] Fixpoints can be handled without step-indexing, thanks to an *unwinding lemma*, but this method is more tedious.

fact that a property has to be true in the (strict) future:

$$p \Vdash \triangleright P \quad \stackrel{def}{=} \quad \forall q : \mathrm{Pred}_p.\ q \Vdash P$$

where Pred_p is the sort of predecessors of an integer p, defined as

$$\mathrm{Pred}_0 = \bot \quad \text{and} \quad \mathrm{Pred}_{p+1} = \mathrm{Nat}_p.$$

This new connector satisfies two principles stating that the $\mathcal{SI}$ layer forms a Gödel-Löb logic:

$$\triangleright\text{-Mono}\ \frac{\Gamma; \mathcal{C}_1, \mathcal{C}_2 \vdash_{\mathcal{SI}} P}{\Gamma; \triangleright\mathcal{C}_1, \mathcal{C}_2 \vdash_{\mathcal{SI}} \triangleright P} \qquad \text{Löb}\ \frac{\Gamma; \mathcal{C}, \triangleright P \vdash_{\mathcal{SI}} P}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} P}$$

Proof. We will just prove the validity of the Löb rule, the other one is direct. Using the elimination rule of the implication before going down to the underneath layer , the rule becomes:

$$\frac{[\Gamma]_n; n \Vdash \mathcal{C} \vdash \forall m \leq n.\ (\forall k : \mathrm{Pred}_m.\ k \Vdash \varphi) \Rightarrow m \Vdash \varphi}{[\Gamma]_n; n \Vdash \mathcal{C} \vdash n \Vdash \varphi}$$

We prove this inference rule by proving $m \leq n \Rightarrow m \Vdash \varphi$ by induction on m, the case $m = 0$ being direct as $0 \Vdash \varphi$ is always true. □

Note that the well-foundation of step-indexes is central to be able to make an induction on forcing conditions.

There are also the usual distributivity rules of $\triangleright$ over the logical connectives (the symbol $\updownarrow$ indicates that the rule can be used in both directions):

$$\updownarrow\triangleright\text{-}\Rightarrow\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright(P \Rightarrow Q)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright P \Rightarrow \triangleright Q} \qquad \updownarrow\triangleright\text{-}\wedge\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright(P \wedge Q)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright P \wedge \triangleright Q}$$

$$\updownarrow\triangleright\text{-}\vee\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright(P \vee Q)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright P \vee \triangleright Q}$$

$$\triangleright\text{-}\exists\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright(\exists x : T.P)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \exists x : T. \triangleright P} \qquad \triangleright\text{-}\forall\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright(\forall x : T.P)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \forall x : T. \triangleright P}$$

Proof. Let's prove the rule $\triangleright$-$\forall$. Leaving the layer $\mathcal{SI}$, we have to prove that

$$\frac{[\Gamma]_n; n \Vdash \mathcal{C} \vdash_{\mathcal{SI}} \forall k : \mathrm{Pred}_n. \forall i : \mathrm{Nat}_k. \forall x : [T]_i. i \Vdash P(x)}{[\Gamma]_n; n \Vdash \mathcal{C} \vdash_{\mathcal{SI}} \forall k : \mathrm{Nat}_n. x : [T]_k. \forall i : \mathrm{Pred}_k. i \Vdash P(x)}$$

So let $k : \mathrm{Nat}_n, x : [T]_k$ and $i : \mathrm{Pred}_k$. By monotonicity we know that $x : [T]_i$ and instantiating the premise with $k = i$, we know that $i \Vdash P(x)$ (as $xi < n$). □

Using this modality, it is possible to reason about recursion abstractly as shown in [4]. To be able to reason about logical relations, we exhibit derived inference rules on observation, valid in $\mathcal{SI}$, such as the following rule

$$\frac{\Gamma; \mathcal{C} \vdash (M_1, h_1) \to (M_1', h_1') \quad \Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright \mathcal{O}^l((M_1', h_1'), (M_2, h_2))}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \mathcal{O}^l((M_1, h_1), (M_2, h_2))}$$

4.2 Recursive relations

Step-indexed forcing conditions enables to define recursive relations in $\mathcal{SI}$ as

$$n \Vdash \mu r.R(M_1, M_2) \stackrel{def}{=} \forall k : \mathrm{Pred}_n.\ k \Vdash R\{\mu r.R/r\}(M_1, M_2)$$

The forcing definition is well-defined thanks to the decreasing of the index. Such recursive relations satisfy the following inference rules:

$$\text{RRel-Type}\ \frac{\Gamma, r : \mathrm{Rel} \vdash_{\mathcal{SI}} R : \mathrm{Rel}}{\Gamma \vdash_{\mathcal{SI}} \mu r.R : \mathrm{Rel}}$$

$$\text{RRel-Inf}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \triangleright R\{\mu r.R/r\}(M_1, M_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{SI}} \mu r.R(M_1, M_2)}$$

Note that compared to other works, we do not require that recursive relations are *contractive*, i.e. have a $\triangleright$ in front of each occurrence of r. But to prove the membership of two terms in a recursive relation, we have to reason in the future, as the Rule RRel-Inf indicates.

As explained in the introduction, using a recursive relation, logical relations can be extended to recursive types:

$$\mathcal{V}[\![\mu\alpha.\tau]\!]_\eta \stackrel{def}{=} \mu r.\Big((v_1, v_2).\exists u_1, u_2 : \mathrm{Val}. v_1 = \mathrm{roll}\, u_1 \wedge v_2 = \mathrm{roll}\, u_2 \wedge \triangleright \mathcal{V}[\![\tau]\!]_{\eta, \alpha \mapsto r}(u_1, u_2)\Big)$$

4.3 Recursive kinds

In a recent study of step-indexing in the topos of trees [6], Birkedal et al. have introduced the notion of a *later* operator on presheaves, which indicates that we could define such an operator[3] $\blacktriangleright\ T$ (together with recursive kinds $\mu T.U$) at the level of kinds in the $\mathcal{SI}$ layer. Indeed, the following translation of kinds under step-indexed forcing conditions

$$\begin{array}{lcl} [\blacktriangleright\ T]_0 & \stackrel{def}{=} & \top \\ [\blacktriangleright\ T]_{p+1} & \stackrel{def}{=} & [T]_p \\ [\mu T.U]_0 & \stackrel{def}{=} & \top \\ [\mu T.U]_{p+1} & \stackrel{def}{=} & [U[\mu.T/T]]_p \end{array}$$

gives rise to a stratified construction of kinds. The subtyping relation is defined as the identity on the new kinds. From those definitions, one can deduce the validity of the following typing rules

$$\updownarrow\text{Rec-Kind}\ \frac{\Gamma \vdash_{\mathcal{SI}} P : \blacktriangleright\ U\{\mu T.U/T\}}{\Gamma \vdash_{\mathcal{SI}} P : \mu T.U}$$

$$\updownarrow\text{Switch-}\blacktriangleright\triangleright\ \frac{\Gamma \vdash_{\mathcal{SI}} P : \blacktriangleright\ \mathrm{Prop}}{\Gamma \vdash_{\mathcal{SI}} \triangleright P : \mathrm{Prop}}$$

$$\updownarrow\text{Commut-}\blacktriangleright\ \frac{\Gamma \vdash_{\mathcal{SI}} P : \blacktriangleright\ (T \to U)}{\Gamma \vdash_{\mathcal{SI}} P : \blacktriangleright\ T \to \blacktriangleright\ U}$$

$$\text{Mono-}\blacktriangleright\ \frac{\Gamma \vdash_{\mathcal{SI}} P : T}{\Gamma \vdash_{\mathcal{SI}} P : \blacktriangleright\ T}$$

together with rules that say that $\blacktriangleright$ vanishes in front of the atomic kinds (but Prop of course).

Proof.
- Rec-Kind : $[\ \blacktriangleright\ U\{\mu T.U/T\}]_n = [\mu T.U]_n$.
- Switch-$\blacktriangleright\ \triangleright$: For non null integer p, we have to equiderive

$$\frac{[\Gamma]_p \vdash [P] : \mathrm{Pred}_p \to \mathrm{Prop}}{[\Gamma]_p \vdash n. \forall q : \mathrm{Pred}_n. q \Vdash P : \mathrm{Nat}_p \to \mathrm{Prop}}$$

which comes from

$$\frac{[\Gamma]_p, n : \mathrm{Pred}_p \vdash p \Vdash P : \mathrm{Prop}}{[\Gamma]_p, n : \mathrm{Nat}_p, q : \mathrm{Pred}_n \vdash q \Vdash P : \mathrm{Prop}}.$$

[3] We borrow this notation from [6], to differentiate it from $\triangleright$ which is used on propositions.

The case $p = 0$ is direct using the canonical typing rules of $\bot$ and $\top$.

- Commut-$\blacktriangleright$:
  - Case $p = 0$:
    $[\blacktriangleright (T \to U)]_0 = \top <: \top \to \top = [\blacktriangleright T \to \blacktriangleright U]_0$
  - Case $p + 1$:
    $[\blacktriangleright T \to \blacktriangleright U]_{p+1} <: [\blacktriangleright T \to \blacktriangleright U]_p = [T \to U]_{p-1}$ (induction hypothesis) and $[\blacktriangleright T \to \blacktriangleright U]_{p+1} <: [T]_p \to [U]_p$. So $[\blacktriangleright T \to \blacktriangleright U]_{p+1} <: [T \to U]_p$.
    The same holds for the other direction.
- Mono-$\blacktriangleright$: We use the fact that $[T]_n <: [\blacktriangleright T]_n$. □

Such recursive kinds will be used to build recursive worlds (Section 5.2), and will explain the presence of $\triangleright$ in the different definitions of logical relations involving recursive worlds.

4.4 Equational reasoning

One of the main problem of step-indexing is that it is inherently asymmetric in the sense that going from inequational to equational logical reasoning is not automatic. Indeed, from a step-indexed order logical relation $\preccurlyeq^\tau_n$ which links two terms at level n, the induced equivalence $M_1 \simeq^\tau_n M_2$ does not satisfy the following extensional property:

If $\forall (u_1, u_2), u_1 \simeq^\tau_n u_2 \Rightarrow M_1\, u_1 \simeq^\sigma_n M_2\, u_2$ then $M_1 \simeq^{\tau\to\sigma}_n M_2$

To avoid this problem, the idea—which first appears in [11]—is to use a parameter indicating what direction of the proof is to be proved.

To embody this parametric technique in our logic, we introduce a new layer $\mathcal{D}$, whose forcing conditions is the set $\{l, r\}$, flat-ordered by equality. In this layer, there is only one notion of observation, which is $\mathcal{O}$. The forcing condition is then defined by:

$$l \Vdash \mathcal{O} \quad \stackrel{def}{=} \quad \mathcal{O}^l \quad \text{and} \qquad r \Vdash \mathcal{O} \quad \stackrel{def}{=} \quad \mathcal{O}^r$$

and is transparent for the other formulas and kinds of $\mathcal{SI}$. A reasoning in the $\mathcal{SI}; \mathcal{D}$ layer has to be true for l and r indifferently, so it corresponds to an equational reasoning. Some derivable rules valid in the layer $\mathcal{SI}, \mathcal{D}$ are shown in the Figures 6 and 7.

5. Kripke Logical Relation

We now express another use of forcing conditions—the construction of worlds for Kripke logical relations. It is well-known that Kripke model can be seen as forcing, so this should not appear as a surprise. What is interesting here is the interaction between the Kripke layer and other already defined forcing layers.

Note that compared to step-indexing, it is not possible to hide the reasoning on worlds, since there is a crucial interaction between worlds and terms. So in this layer (which will be called the $\mathcal{K}$ layer), we do not really get new logical principles as it was the case for the $\mathcal{SI}$ layer, but we rather get a new symbol $\mathcal{W}$ in the logic which enables to talk about the heap in formulas. Using this new symbol, the definition of the logical relation at type $\mathrm{ref}\,\tau$ is given by

$$\mathcal{V}\,[\![\mathrm{ref}\,\tau]\!]_\eta\,(l_1, l_2) \quad \stackrel{def}{=} \quad \forall v_1, v_2 : \tau.\ \mathcal{W}(l_1, l_2)(v_1, v_2) \iff (v_1, v_2) \in \triangleright \mathcal{V}\,[\![\tau]\!]_\eta$$

It says that two locations are related at type $\mathrm{ref}\,\tau$ if the values v_1, v_2 related in the current world at those locations are exactly the values that are related at type τ (in the future).

This means that to define Kripke logical relations over a set of worlds, we simply have to define

$$w \Vdash \mathcal{W} \quad \text{and} \quad w \Vdash \tilde{\mathcal{O}}$$

where $\tilde{\mathcal{O}}$ means any notion of observation. The rest of the definitions for formulas and kinds is straightforward. Forcing an observation with a world w is independent of the very nature of the observation, since $w \Vdash \tilde{\mathcal{O}}((M_1, h_1), (M_2, h_2))$ ise defined as

$$(h_1, h_2) : w \Rightarrow \tilde{\mathcal{O}}((M_1, h_1)(M_2, h_2)).$$

It is just a restriction of the observation to heaps related in the world w, where the formula $(h_1, h_2) : w$ in the definition above is a short-hand for

$$\forall l_1, l_2 \in \mathrm{dom}(w).\ w \Vdash \mathcal{W}(l_1, l_2)(h_1(l_1), h_2(l_2)).$$

This explains why the layer $\mathcal{K}$ can be defined above $\mathcal{SI}$ or $\mathcal{D}$ indifferently, where the new logical connectors introduced in $\mathcal{SI}$ are forced transparently as in the rule

$$w \Vdash \triangleright\varphi \quad \stackrel{def}{=} \quad \triangleright(w \Vdash \varphi).$$

Those transparent definitions ensure that the new reasoning principles available in $\mathcal{SI}$ are still valid in the layer $\mathcal{K}$.

As it is the case for Kripke models, an important fact is that the set of worlds has to be ordered (usually noted $w' \sqsupseteq w$ and pronounced w' future of w) in order to provide a good notion of forcing conditions. Note that we require the definition of forcing for the observation to be monotonous, which is the case in this paper but would not be the case for complex notion of worlds using STSs [12]. Nevertheless, it is possible to relax monotony conditions by the use of modal forcing which makes explicit any need of monotonicity (see Section 9).

5.1 First-order References

For first-order references, worlds (i.e. forcing conditions) have the kind

$$\mathrm{World} \quad \stackrel{def}{=} \quad (\mathrm{Loc} \times \mathrm{Loc}) \to_{fin} \mathrm{Rel}$$

and the meaning of $\mathcal{W}$ is simply given by

$$w \Vdash \mathcal{W}(l_1, l_2)(v_1, v_2) \quad \stackrel{def}{=} \quad w(l_1, l_2)(v_1, v_2)$$

This reconstructs logical relations defined by Pitts and Stark [17]. Note that in that case, we make no use of the new logical principles available in the $\mathcal{SI}$ layer.

5.2 Higher-order references in $\mathcal{SI}$

Dealing with higher-order references requires to define worlds recursively. This means that we must make use of recursive kinds available In the $\mathcal{SI}$ layer. For higher-order references, worlds (i.e. forcing conditions) have the kind

$$\mathrm{World} \quad \stackrel{def}{=} \quad \mu W.(\mathrm{Loc} \times \mathrm{Loc} \to_{fin} W \to \mathrm{Rel}),$$

which corresponds to the one used in the semantics model[4] of [7]. Using the typing rule of $\mathcal{SI}$, one can see that the only "typable" definition of $\mathcal{W}$ is the following (used for instance in [12]):

$$w \Vdash \mathcal{W}(l_1, l_2)(v_1, v_2) \quad \stackrel{def}{=} \quad \triangleright w(l_1, l_2)(w)(v_1, v_2)$$

[4] In [7] and previous works, the authors needs a monotonic functions space between W and Rel. It appears that—through forcing construction—any arrow in the $\mathcal{K}$ layer is monotonic so we have it for free in our framework.

Indeed, the typing derivation is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash w : \text{World}}\,\text{Ax}}{\Gamma \vdash w :\blacktriangleright (Loc \times \text{Loc} \to_{fin} \text{World} \to \text{Rel})}\,\uparrow\text{Rec-Kind}}{\Gamma \vdash w : T \quad \Gamma \vdash l_1, l_2 : \text{Loc} \quad \cdots}\,\downarrow\text{Commut-}\blacktriangleright}{\Gamma \vdash w(l_1,l_2)(w)(v_1,v_2) :\blacktriangleright \text{Prop}}\,\text{App}}{\Gamma \vdash \triangleright w(l_1,l_2)(w)(v_1,v_2) : \text{Prop}}\,\downarrow\text{Switch}\blacktriangleright\,\triangleright$$

where Γ and T are respectively

$$w : \text{World}, l_1, l_2 : \text{Loc}, v_1, v_2 : \text{Term}$$

and

$$Loc \times \text{Loc} \to_{fin} \text{World} \to (\text{Term} \times \text{Term}) \to \blacktriangleright \text{Prop}$$

5.3 Reasoning in the $\mathcal{K}$ layer

Let $\mathcal{C}' = \mathcal{C}, (h_1, h_2) : w$ and $\mathcal{O}_w = w \Vdash \mathcal{O}$. Figure 6 presents the derived rules on $\mathcal{O}_w$. As said before, those rules are not expressible in the layer $\mathcal{K}$ as a precise knowledge of the current world is required. Nevertheless, when the heap is not modified by the reduction, we can reason directly in the layer $\mathcal{K}$. Indeed, the previous inference rules can be written in the layers $\vdash_{\mathcal{K};\mathcal{SI}}$ and $\vdash_{\mathcal{D};\mathcal{K};\mathcal{SI}}$, forgetting any hypothesis about heaps h_i. Hence, inference rules on $\mathcal{E}[\![\tau]\!]$ can be deduced (see Figure 7, the proof of Step-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$ is given in Appendix).

Finally, due to the definition of forcing in $\mathcal{D}$ and $\mathcal{K}$, we can show that their forcing relations commute, which is sometime useful when we want to exit one of the two layer.

6. Properties of Logical Relations

Now that logical relations are defined for the whole language, let us look at its soundness with respect to observational equivalence. The proof of soundness for the pure fragment can be done in the $\mathcal{SI};\mathcal{D};\mathcal{K}$ layer as it can be completely abstracted with respect to worlds. Indeed, pure terms does not change heap. Such a proof can then be instantiated by any world, going down to the $\mathcal{SI};\mathcal{D}$ layer. When dealing with the impure fragment, we have to leave the $\mathcal{K}$ layer since we need to know the world under consideration precisely. And in the same way, when making an asymmetric reasoning, we will leave the layer $\mathcal{D}$ in order to work with a concrete direction (left or right).

6.1 Compatibility lemmas

To establish the correctness of our logical relation $\mathcal{E}[\![\tau]\!]$, we first prove the so-called compatibility lemmas of Figure 8. We just give the prove the compatibility for Fix, Roll and Assign. The rest of the proof can be found in appendix.

Proof. • Rule Fix :
Using the rule $\mathcal{V}$-to-$\mathcal{E}$, we just have to prove that $\Gamma;\mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{V}[\![\tau \to \sigma]\!](\text{fix}\, f(x).M_1, \text{fix}\, f(x).M_2)$. To do so, we use the Löb rule, so we have to prove

$$\Gamma; \underbrace{\mathcal{C}, \triangleright(\mathcal{V}[\![\tau \to \sigma]\!](\text{fix}\, f(x).M_1, \text{fix}\, f(x).M_2))}_{\mathcal{C}'} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{V}[\![\tau \to \sigma]\!](\text{fix}\, f(x).M_1, \text{fix}\, f(x).M_2)$$

Unwinding the definition, we will prove that

$$\Gamma, v_1 : \tau, v_2 : \tau; \mathcal{C}', \mathcal{V}[\![\tau]\!](v_1, v_2) \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}[\![\tau \to \sigma]\!]((\text{fix}\, f(x).M_1)v_1, (\text{fix}\, f(x).M_2)v_2)$$

Using the rule Step-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$, we just have to prove that

$$\Gamma'; \mathcal{C}' \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \triangleright(\mathcal{E}[\![\sigma]\!](M_1\{v_1/x\}\{f_1/f\}, M_2\{v_2/x\}\{f_1/f\}))$$

and using the rule $\triangleright$-Mono we can conclude using the premise.

• Rule Roll :
Let K_1, K_2 two contexts s.t. $\mathcal{K}[\![\mu\alpha.\tau]\!]_\eta(K_1, K_2)$, we will prove that $\mathcal{K}[\![\tau\{\mu\alpha.\tau/\alpha\}]\!]_\eta(K_1[\text{roll}\circ], K_2[\text{roll}\circ])$.
Let v_1, v_2 two values s.t. $\mathcal{V}[\![\tau\{\mu\alpha.\tau/\alpha\}]\!]_\eta(v_1, v_2)$, then we have to prove that

$$\forall h_1, h_2.\mathcal{O}((K_1[\text{roll}\, v_1], h_1), (K_2[\text{roll}\, v_2], h_2))$$

which is direct since, by monotony and substitution we have

$$\triangleright(\mathcal{V}[\![\tau]\!]_{\eta,\alpha\mapsto\mathcal{V}[\![\mu\alpha.\tau]\!]_\eta}(v_1, v_2))$$

so

$$\mathcal{V}[\![\mu\alpha.\tau]\!]_\eta(\text{roll}\, v_1, \text{roll}\, v_2).$$

• Rule Assign :
Let K_1, K_2 two contexts s.t. $\mathcal{K}[\![\text{Unit}]\!]_\eta(K_1, K_2)$, we will prove that $\mathcal{K}[\![\text{ref}\,\tau]\!]_\eta(K_1[\circ := M_1], K_2[\circ := M_2])$. So let l_1, l_2 two values s.t. $\mathcal{V}[\![\text{ref}\,\tau]\!]_\eta(l_1, l_2)$ we have to prove

$$\forall h_1, h_2.\mathcal{O}((K_1[l_1 := M_1], h_1), (K_2[l_2 := M_2], h_2))$$

To do so, we will prove that $\mathcal{K}[\![\tau]\!]_\eta(K_1[l_1 := \circ], K_2[l_2 := \circ])$. Let v_1, v_2 two values s.t. $\mathcal{V}[\![\tau]\!]_\eta(v_1, v_2)$ we have to prove

$$\forall h_1, h_2.\mathcal{O}((K_1[l_1 := v_1], h_1), (K_2[l_2 := v_2], h_2))$$

Leaving the layer $\mathcal{K}$, suppose $h_1, h_2 : w$ then from $w \Vdash \mathcal{V}[\![\text{ref}\,\tau]\!]_\eta(l_1, l_2)$ we get that $l_i \in \text{dom}(h_i)$.
Then we conclude, using the rules StepL-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$ and StepR-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$, by showing that

$$w \Vdash \mathcal{O}((K_1[()], h_1[l_1 \mapsto v_1]), (K_2[()], h_2[l_2 \mapsto v_2]))$$

using the fact that $w \Vdash \mathcal{K}[\![\text{Unit}]\!]_\eta(K_1, K_2)$ and $(h_1[l_1 \mapsto v_1), h_2[l_2 \mapsto v_2]) : w$.

□

An other crucial derived rule is the adequacy between $\mathcal{V}[\![\tau]\!]$ and $\mathcal{E}[\![\tau]\!]$ over values :

$$\mathcal{V}\text{-to-}\mathcal{E}\ \dfrac{\Gamma;\mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{V}[\![\tau]\!](M_1, M_2)}{\Gamma;\mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}[\![\tau]\!](M_1, M_2)}$$

6.2 Soundness of logical relations

Now we prove the soundness of our logical relation with respect to the contextual equivalence, following the usual schema of [13].

Theorem 4. *Fundamental property If* $\Delta;\Upsilon;\Gamma \vdash t : \tau$ *then* $\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} t \simeq_{log}^{\Delta;\Upsilon;\Gamma} t : \tau$.

By induction on the typing rule, in each case using the previous compatibility lemmas.

Theorem 5. *Congruence If* $\vdash M_1 \simeq_{log}^{\Delta;\Upsilon;\Gamma} M_2 : \tau$ *and* $\vdash C : (\Delta;\Upsilon;\Gamma \vdash \tau) \to (\Delta';\Upsilon';\Gamma' \vdash \sigma)$ *then* $\Delta;\Upsilon;\Gamma \vdash C[M_1] \simeq_{log} C[M_2] : \tau$.

The proof is done by induction on the context C, using again compatibility lemmas.

Theorem 6. *Adequacy If* $M_1 \simeq_{log}^{\cdot;\cdot;\Upsilon} M_2 : \tau$ *then for all heap* h *satisfying* Υ, $\mathcal{O}((M_1, h), (M_2, h))$.

$$\textsc{Step-}\mathcal{O}^l_w\ \frac{\Gamma;\mathcal{C}' \vdash (M_1,h_1) \to (M_1',h_1') \quad \Gamma;\mathcal{C}' \vdash_{\mathcal{SI}} (h_1',h_2) : w' \quad \Gamma;\mathcal{C}' \vdash_{\mathcal{SI}} \mathcal{O}^l_{w'}((M_1',h_1'),(M_2,h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{SI}} \mathcal{O}^l_w((M_1,h_1),(M_2,h_2))}\ \textit{and the dual rule for } r$$

$$\textsc{Step-}\triangleright\mathcal{O}^l_w\ \frac{\Gamma;\mathcal{C}' \vdash (M_1,h_1) \to (M_1',h_1') \quad \Gamma;\mathcal{C}' \vdash_{\mathcal{SI}} (h_1',h_2) : w' \quad \Gamma;\mathcal{C}' \vdash_{\mathcal{SI}} \triangleright\mathcal{O}^l_{w'}((M_1',h_1'),(M_2,h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{SI}} \mathcal{O}^l_w((M_1,h_1),(M_2,h_2))}\ \textit{and the dual rule for } r$$

$$\textsc{Step-}\triangleright\mathcal{O}_w\ \frac{\Gamma;\mathcal{C}' \vdash (M_i,h_i) \to (M_i',h_i')\ i\in\{1,2\} \quad \Gamma;\mathcal{C}' \vdash_{\mathcal{D};\mathcal{SI}} (h_1',h_2') : w' \qquad \Gamma;\mathcal{C}' \vdash_{\mathcal{D};\mathcal{SI}} \triangleright\mathcal{O}_{w'}((M_1',h_1'),(M_2',h_2'))}{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{SI}} \mathcal{O}_w((M_1,h_1),(M_2,h_2))}$$

Figure 6. Derived rules on $\mathcal{O}_w$

$$\textsc{StepL-}\mathcal{K};\mathcal{SI}\textsc{-}\mathcal{E}\ \frac{\Gamma;\mathcal{C} \vdash (M_1,h_1) \to (M_1',h_1) \qquad \Gamma;\mathcal{C} \vdash_{\mathcal{K};\mathcal{SI}} l \Vdash \mathcal{E}\,[\![\tau]\!]\,((M_1',h_1),(M_2,h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{K};\mathcal{SI}} l \Vdash \mathcal{E}\,[\![\tau]\!]\,((M_1,h_1),(M_2,h_2))}\ \textit{and the dual rule for } r$$

$$\textsc{StepL-}\mathcal{K};\mathcal{SI}\textsc{-}\triangleright\mathcal{E}\ \frac{\Gamma;\mathcal{C} \vdash (M_1,h_1) \to (M_1',h_1') \qquad \Gamma;\mathcal{C} \vdash_{\mathcal{K};\mathcal{SI}} l \Vdash \triangleright\mathcal{E}\,[\![\tau]\!]\,((M_1',h_1'),(M_2,h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{K};\mathcal{SI}} l \Vdash \mathcal{E}\,[\![\tau]\!]\,((M_1,h_1),(M_2,h_2))}\ \textit{and the dual rule for } r$$

$$\textsc{Step-}\mathcal{D};\mathcal{K};\mathcal{SI}\textsc{-}\triangleright\mathcal{E}\ \frac{\Gamma;\mathcal{C} \vdash (M_i,h_i) \to (M_i',h_i) \quad (i=1,2) \qquad \Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{K};\mathcal{SI}} \triangleright\mathcal{E}\,[\![\tau]\!]\,(M_1',M_2')}{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{K};\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]\,(M_1,M_2)}$$

Figure 7. Derived rules on $\mathcal{E}\,[\![\tau]\!]$ for the layer $\mathcal{K};\mathcal{SI}$

To prove it, we just have to show that the empty context is in $\mathcal{K}\,[\![\tau]\!]$.

Theorem 7. *Soundness If* $\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} M_1 \simeq_{log}^{\Delta;\Upsilon;\Gamma} M_2 : \tau$ *then* $\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} M_1 \simeq_{ctx}^{\Delta;\Upsilon;\Gamma} M_2 : \tau$.

Direct by combining adequacy and congruence lemmas.

7. An example : the Landin Knot

To illustrate our work, we are going to show that the well-known *Landin knot* is logically equivalent to the usual fix-point operator. Following [13], we consider

$$M_1 = \texttt{let y = ref } (\lambda x.\bot) \texttt{ in y:= } (\lambda \texttt{ x.let f = !y in e);!y}$$

and

$$M_2 = \texttt{fix } \lambda \texttt{ f(x).e}$$

and we will prove that $\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} (M_1,M_2) \in \mathcal{E}\,[\![\tau \to \sigma]\!]$. Unwinding the definition of $\mathcal{E}\,[\![\tau \to \sigma]\!]$, we have to prove that

$$(K_1,K_2) \in \mathcal{K}\,[\![\tau \to \sigma]\!] \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} ((K_1[M_1],h_1),(K_2[M_2],h_2)) \in \mathcal{O}$$

Leaving the layer $\mathcal{K}$, we have to prove that those two terms are in $\mathcal{O}_w$ for every world w. Then, we use the rule $\textsc{Step-}\mathcal{O}^l_w$ with the fact that $(K_1[M_1],h_1)$ reduce to

$$(\underbrace{\lambda\texttt{x.let f = !l in e}}_{M_1'}, \underbrace{h_1 \bullet [l \mapsto \lambda\texttt{x.let f = !l in e}]}_{h_1'})$$

So we build a world $w' \sqsupseteq w$ which restrains l to always point in h_1 to $\lambda\texttt{x.let f = !l in e}$:

$$w' = w \bullet [(l,l) \mapsto R]$$

where R is the relation $(t_1,t_2).t_1 = \lambda\texttt{x.let f = !l in e}$. So we just have to prove that $w' \Vdash (M_1',M_2) \in \mathcal{E}\,[\![\tau \to \sigma]\!]$. To do so, we use the rule $\mathcal{V}\textsc{-to-}\mathcal{E}$ and the Löb rule, and thus prove that

$$\underbrace{\triangleright(w' \Vdash (M_1',M_2) \in \mathcal{V}\,[\![\tau \to \sigma]\!])}_{\mathcal{C}_1} \vdash_{\mathcal{D};\mathcal{SI}} w' \Vdash (M_1',M_2) \in \mathcal{V}\,[\![\tau \to \sigma]\!]$$

Using the rule Abstr in the layer $\mathcal{D};\mathcal{SI}$, then the rule $\textsc{Step-}\mathcal{D};\mathcal{K};\mathcal{SI}\textsc{-}\triangleright\mathcal{E}$, this amount to prove

$$\Gamma;\triangleright\mathcal{C}_1,\mathcal{C}_2 \vdash_{\mathcal{D};\mathcal{SI}} w' \Vdash \triangleright(e\,\{v_1/x\}\,\{M_1'/f\}\,,e\,\{v_2/x\}\,\{M_2/f\}) \in \mathcal{E}\,[\![\sigma]\!]$$

where Γ is $l : \mathrm{Loc}, v_1, v_2 : \tau$ and $\mathcal{C}_2$ is $(v_1,v_2) \in \mathcal{V}\,[\![\tau]\!]$. Then, using the rule Mono-$\triangleright$, we make the $\triangleright$ disappear and we conclude using the rule App with the hypothesis $\mathcal{C}_1,\mathcal{C}_2$.

8. Related works

8.1 Syntactic models

The difficulties encountered to build logical relations for $F^{\mu!}$ also appear at the semantic model level. Indeed, those models—associating to each type a set of terms—can somehow be seen as unary version of logical relations.

The *very modal model* [4] introduces the stratification of worlds and the $\triangleright$ operator, following the idea that dereferencing will always take at least one step of reduction. In this way properties about the value stocked there only have to be proved in the strict future.

In [19], F. Pottier gives a typed store-passing translation for higher-order references. This syntactic construction is based on Nakano's system, introducing co-inductively defined kinds with well-formation defined thanks to a "later" modality on those kinds. As it is the case for [7], there are clear connections between this syntactic presentation and the logical principles available in the $\mathcal{SI}$ layer.

8.2 Topos of trees

In [6], Birkedal et al. define a semantic model of a language similar to $F^{\mu!}$, built over the topos of trees (i.e. of presheaves over $\bar{\omega}$). It appears that this topos constitutes a model of

$$\textsc{Abstr}\ \frac{\Gamma, v_1:\tau, v_2:\tau; \mathcal{C}, \mathcal{V}\,[\![\tau]\!]_\eta\,(v_1,v_2) \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\sigma]\!]_\eta\,(M_1\{v_1/x\}, M_2\{v_2/x\})}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\to\sigma]\!]_\eta\,(\lambda x.M_1, \lambda x.M_2)}$$

$$\textsc{App}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\to\sigma]\!]_\eta\,(M_1,M_2) \quad \Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_\eta\,(N_1,N_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\sigma]\!]_\eta\,(M_1N_1, M_2N_2)}$$

$$\textsc{Fix}\ \frac{\Gamma, v_1:\tau, v_2:\tau; \mathcal{C}, \mathcal{V}\,[\![\tau]\!]_\eta\,(v_1,v_2), \mathcal{V}\,[\![\tau\to\sigma]\!]_\eta\,(f_2,f_1) \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\sigma]\!]_\eta\,(M_1\{v_1/x\}\{f_1/f\}, M_2\{v_2/x\}\{f_1/f\})}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\to\sigma]\!]_\eta\,(\underbrace{\mathrm{fix}\,f(x).M_1}_{f_1}, \underbrace{\mathrm{fix}\,f(x).M_2}_{f_2})}$$

$$\textsc{Gen}\ \frac{\Gamma, \alpha_1:\mathrm{Type}, \alpha_2:\mathrm{Type}, r:\mathrm{Rel}_{\alpha_1,\alpha_2}; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_{\eta,\alpha\mapsto r}\,(M_1\{\alpha_1/\alpha\}, M_2\{\alpha_1/\alpha\})}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\forall\alpha.\tau]\!]_\eta\,(\Lambda\alpha.M_1, \Lambda\alpha.M_2)}$$

$$\textsc{Inst}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\forall\alpha.\tau]\!]_\eta\,(M_1,M_2) \qquad \Gamma \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \sigma:\mathrm{Type}}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\{\sigma/\tau\}]\!]_\eta\,(M_1\sigma, M_2\sigma)}$$

$$\textsc{Roll}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\{\mu\alpha.\tau/\alpha\}]\!]_\eta\,(M_1,M_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\mu\alpha.\tau]\!]_\eta\,(\mathrm{roll}\,M_1, \mathrm{roll}\,M_2)} \qquad \textsc{Unroll}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\mu\alpha.\tau]\!]_\eta\,(M_1,M_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau\{\mu\alpha.\tau/\alpha\}]\!]_\eta\,(\mathrm{unroll}\,M_1, \mathrm{unroll}\,M_2)}$$

$$\textsc{Alloc}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_\eta\,(M_1,M_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\mathrm{ref}\,\tau]\!]_\eta\,(\mathrm{ref}\,M_1, \mathrm{ref}\,M_2)} \qquad \textsc{Deref}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\mathrm{ref}\,\tau]\!]_\eta\,(M_1,M_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_\eta\,(!M_1, !M_2)}$$

$$\textsc{Assign}\ \frac{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\mathrm{ref}\,\tau]\!]_\eta\,(M_1,M_2) \quad \Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_\eta\,(N_1,N_2)}{\Gamma; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\,[\![\tau]\!]_\eta\,(M_1 := N_1, M_2 := N_2)}$$

Figure 8. Compatibility lemmas

our $\mathcal{SI}$ layer—even so we don't need models to testimony the consistency of our logic as we just translate it into FLR using forcing definitions. This should not appear as a surprise as Cohen forcing can be rephrased in topos theory, following the work of Lawvere and Thierney [20].

More precisely, a predicate φ in the layer $\mathcal{SI}$ is seen as an ω-valued function, mapping to the set of indexes which force it. The stratification of sorts appearing in the $\mathcal{SI}$ layer corresponds to the construction of presheaves. The stratification of Prop induced by forcing conditions corresponds exactly to the definition of the subobject classifier Ω of the topos. The fact that the translation of sorts is constant but for Prop and $\mu T.U$ follows from the idea that constant presheaves (i.e. the logic of the tripos over ω) are enough to model LSLR.

8.3 LSLR and LADR

LSLR [11] is an extension of Abadi-Plotkin logic to reason about equivalence of λ-terms with polymorphism and recursive types. LADR [13] is an extension of LSLR to prove equivalence of λ-terms with higher-order states, e.g. using Kripke logical relations. Both logics make step-indexing abstract using Gödel-Lob logic and a $\triangleright$ operator. But to build recursive worlds in LADR, the authors use an explicit stratification which makes step-indexes apparent. It should be possible to integrate their construction in our system, just like the refinement of [12], without making any step-index visible in the layer $\mathcal{SI}$, using recursive kinds.

Our approach is really different from LSLR and LADR: we do not rely on any particular model to prove the coherence of our logic. Indeed, the coherence of the different forcing layers comes directly from the translation of propositions in the core logic.

Definitions of recursive relations $\mu r.R$ in LSLR must satisfy a syntactic criterion of *contractivity*, which enforces the presence of $\triangleright$ in front of each occurrence of the variable r. This criterion is crucial to build their model. Our approach is different since the $\triangleright$ operator is not in the definition of recursive relations but appears through a particular inference rule that unfolds the definition of $\mu r.R$ only in the future.

Compared to LSLR, we do not count only roll and unroll reductions in step-indexes, but decides that every reduction matters. Then, using the rule $\triangleright$-Mono, it is possible to erase occurrences of $\triangleright$ that will not be relevant in the proof. Doing so, step-indexing can be used to deal with fixpoint operator, but it is no longer guided by the inference rules, as it is the case with the $\triangleleft$ in LSLR.

Finally, the idea of a direction parameter d to deal with the problem of equational reasoning in presence of step-indexing appeared first in LSLR, where it is simply a term variable of type bool, which always appear in the typing context Γ. This idea is then refined in LADR where the parameter appears in the definition of worlds. Here, we go one step further and define a proper forcing layer $\mathcal{D}$ where we can deal with equational reasoning.

9. Future work

Monotonicity and modal forcing What we have presented here correspond to intuitionnistic forcing, since the monotonicity is imposed by the definition of $p \Vdash P \Rightarrow Q$ and $p \Vdash \forall x:T.P$. This is enough to deal with worlds seen as invariants on heap. But recent works in [2] and [12] have developed notions of worlds which described the evolution of the heap. To deal with such works in our framework, we

would need to relax the monotonicity of our forcing relation, and rather add a modality $\Box$ just as in [13]

$$p \Vdash \Box P \stackrel{def}{=} \forall q \leq p.q \Vdash P.$$

Using this modality, we can explicitly state were monotonicity has to be enforced and where it is not relevant. In the same way we could deal with public and private transition of [12] by splitting the order of forcing condition into two parts.

Finer description of the heap It would be interesting to see if relational separation logic can be used to link abstract worlds and heaps, as in [12]. Then, temporal logic could be used to reason about STSs, which are used to restrict the possible futures of worlds. The problem with this method is that the order on worlds is eminently not linear, so we have to keep track of each choice made while moving in the future. This is done in LADR using *island contexts*. With such an extension of our framework we would be able to express useful derived rules for programs modifying their heap, without leaving the layer $\mathcal{K}$.

Implicit Complexity A recent work of Hofmann and Dal Lago on a realizability model for implicit complexity type system [10] uses a notion of resource monoid which fits well in our framework, when working with a predicative (as opposed to relational) notion of termination for the observation. This would be an example where the order on forcing conditions in the $\mathcal{SI}$ layer is not linear.

Subset types Some of the properties of our logic, like the kinds Nat_p and $T \to_{fin} U$, could be defined using usual subset types. However, adding them in our framework is not easy, since we would then have to define the forcing translation of such sorts. This seems anyway possible following the translation of proofs induced by forcing given in [16].

Formalization in Coq Formalizing this work in Coq would not be totally easy due to the lack of implicit dependent types in CIC. So this requires a *deep embedding* of the logic. An other possibility would be to avoid the use of implicit depend types by constraining the monotonicity of $[T \to U]$ directly in the definition (using record types).

10. Conclusion

In this paper, we decompose the construction of logical relations for a language with recursive types and higher-order references. This is achieved by reunderstanding Cohen's forcing as a way to increase step-by-step the power of a logical setting. The first step is to extend the logic with the Löb rule, recursive relations and recursive kinds by means of forcing with step indexes which stratify formulas and kinds. The second step is to encompass the asymmetric nature of proofs of equivalence by means of forcing with two elements representing the left and right orientation. Finally, the last step is to extend the model with a notion of recursive worlds (based on recursive kinds provided by step-indexes), which—when seeing as forcing conditions—defines directly the usual notion of Kripke logical relations. One of the main achievement of this paper is to build complex notion of worlds directly inside the logic. We believe that the use of forcing conditions opens the door to the definition of richer logical relations to deal with concurrent or design specific languages.

Acknowledgments

The authors want to thank Lars Birkedal and Alexandre Miquel for valuable discussions.

References

[1] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. *Programming Languages and Systems*, pages 69–83, 2006.

[2] A.W. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proceedings of the 36th ACM Symposium on Principles of Programming Languages*, 2009.

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

[4] A.W. Appel, P.-A. Melliès, C.D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of the 34th ACM Symposium on Principles of Programming Languages*, 2007.

[5] J. Avigad. Forcing in proof theory. *The Bulletin of Symbolic Logic*, 10(3):305–333, 2004.

[6] L. Birkedal, R. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. In *Logic in Computer Science (to appear)*, 2011.

[7] L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages*, 2011.

[8] P.J. Cohen and M. Davis. *Set theory and the continuum hypothesis*. WA Benjamin New York, 1966.

[9] T. Coquand and G. Jaber. A Note on Forcing and Type Theory. *Fundamenta Informaticae*, 100(1):43–52, 2010.

[10] Ugo Dal Lago and Martin Hofmann. A semantic proof of polytime soundness for light affine logic. *Theory of Computing Systems*, 2009. to appear.

[11] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science*, 2009.

[12] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proceedings of the 15th ACM International Conference on Functional programming*, 2010.

[13] D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages*, 2010.

[14] Jean-Louis Krivine. Structures de réalisabilité, RAM et ultrafiltre sur N.

[15] A. Miquel. A model for impredicative type systems, universes, intersection types and subtyping. In *Logic in Computer Science*, pages 18–29. IEEE, 2002.

[16] A. Miquel. Forcing as a program transformation. In *Logic in Computer Science (to appear)*, 2011.

[17] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*. CUP, 1998.

[18] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications*, 1993.

[19] F. Pottier. A typed store-passing translation for general references. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages*, 2011.

[20] M. Tierney. Sheaf theory and the continuum hypothesis. *LNM 274*, pages 13–42, 1972.

Appendix

Typing rules for partial finite functions.

$$\frac{(x:\tau)\in\Gamma}{\Delta;\Upsilon;\Gamma\vdash x:\tau} \qquad \frac{}{\Delta;\Upsilon;\Gamma\vdash():\mathrm{Unit}} \qquad \frac{}{\Delta;\Upsilon;\Gamma\vdash n:\mathrm{Nat}} \qquad \frac{\Delta;\Upsilon;\Gamma\vdash M:\tau\to\sigma \qquad \Delta;\Upsilon;\Gamma\vdash N:\tau}{\Delta;\Upsilon;\Gamma\vdash MN:\sigma}$$

$$\frac{\Delta;\Upsilon;\Gamma,x:\tau\vdash M:\sigma}{\Delta;\Upsilon;\Gamma\vdash\lambda x.M:\tau\to\sigma} \qquad \frac{\Delta;\Upsilon;\Gamma,x:\tau,f:\tau\to\sigma\vdash M:\sigma}{\Delta;\Upsilon;\Gamma\vdash\mathrm{fix}\,f(x).M:\tau\to\sigma}$$

$$\frac{\Delta;\Upsilon;\Gamma\vdash M:\forall\alpha.\tau \qquad FV(\sigma)\subseteq\Delta}{\Delta;\Upsilon;\Gamma\vdash M\sigma:\tau\,\{\sigma/\alpha\}} \qquad \frac{\Delta,\alpha;\Upsilon;\Gamma\vdash M:\tau}{\Delta;\Upsilon;\Gamma\vdash\Lambda\alpha.M:\forall\alpha.\tau}$$

$$\frac{\Delta;\Upsilon;\Gamma\vdash M:\mu\alpha.\tau}{\Delta;\Upsilon;\Gamma\vdash\mathrm{unroll}\,M:\tau\,\{\mu\alpha.\tau/\alpha\}} \qquad \frac{\Delta,\alpha;\Upsilon;\Gamma\vdash M:\tau\,\{\mu\alpha.\tau/\alpha\} \qquad FV(\tau)\subseteq\Delta\cup\{\alpha\}}{\Delta;\Upsilon;\Gamma\vdash\mathrm{roll}\,M:\mu\alpha.\tau}$$

$$\frac{}{\Delta;\Upsilon;\Gamma\vdash l:\mathrm{ref}\,\Upsilon(l)} \qquad \frac{\Delta;\Upsilon;\Gamma\vdash M:\mathrm{ref}\,\tau}{\Delta;\Upsilon;\Gamma\vdash!M:\tau} \qquad \frac{\Delta;\Upsilon;\Gamma\vdash M:\mathrm{ref}\,\tau \qquad \Delta;\Upsilon;\Gamma\vdash N:\tau}{\Delta;\Upsilon;\Gamma\vdash M:=N:\mathrm{Unit}}$$

Typing rules for partial finite functions.

$$\frac{\Gamma\vdash f:A\to_{fin}B}{\Gamma\vdash\mathrm{dom}(f):A\to\mathrm{Prop}} \qquad \frac{\Gamma\vdash f:A\to_{fin}B}{\Gamma\vdash f:A\to B} \qquad \frac{}{\Gamma\vdash\mathrm{emp}:A\to_{fin}B} \qquad \frac{\Gamma\vdash f:A\to_{fin}B \quad \Gamma\vdash u:A \quad \Gamma\vdash v:B}{\Gamma\vdash f\bullet[u\mapsto v]:A\to_{fin}B}$$

$$\frac{\Gamma\vdash f:A\to_{fin}B \qquad \Gamma\vdash u:A \qquad \Gamma\vdash v:B}{\Gamma\vdash f[u\mapsto v]:A\to_{fin}B}$$

Compatibily lemmas.

- Rule Abstr :

  Using the rule $\mathcal{V}$-TO-$\mathcal{E}$, we just have to prove that $\Gamma;\mathcal{C}\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}}\mathcal{V}[\![\tau\to\sigma]\!](\lambda x.M_1,\lambda x.M_2)$. Unwinding the definition, we will prove that

  $$\Gamma,v_1:\tau,v_2:\tau;\mathcal{C},\mathcal{V}[\![\tau]\!](v_1,v_2)\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}}\mathcal{E}[\![\tau\to\sigma]\!]((\lambda x.M_1)v_1,(\lambda x.M_2)v_2)$$

  and we conclude using the rule StepS-$\mathcal{E}$.

- Rule App :

  Let K_1,K_2 s.t. $\mathcal{K}[\![\sigma]\!](K_1,K_2)$, we will prove that $\mathcal{K}[\![\tau\to\sigma]\!](K_1[\circ N_1],K_2[\circ N_2])$. So let M_1',M_2' s.t. $\mathcal{V}[\![\tau\to\sigma]\!](\lambda x.M_1',\lambda x.M_2')$, then we must show

  $$\forall h_1,h_2.\mathcal{O}((K_1[(\lambda x.M_1')N_1],h_1),K_2[(\lambda x.M_2')N_2],h_2)$$

  Using the fact that $\mathcal{E}[\![\tau]\!](N_1,N_2)$, we just have to prove that $\mathcal{K}[\![\tau]\!](K_1[(\lambda x.M_1')\circ],K_2[(\lambda x.M_2')\circ])$. so taking v_1,v_2 s.t. $\mathcal{V}[\![\tau]\!](v_1,v_2)$ and we can conclude showing

  $$\forall h_1,h_2.\mathcal{O}((K_1[(\lambda x.M_1')v_1],h_1),K_2[(\lambda x.M_2')v_2],h_2)$$

  which comes from the fact that $\mathcal{E}[\![\sigma]\!]((\lambda x.M_1')v_1,(\lambda x.M_2')v_2)$ and $\mathcal{K}[\![\sigma]\!](K_1,K_2)$.

- Rule Fix :

  Using the rule $\mathcal{V}$-TO-$\mathcal{E}$, we just have to prove that $\Gamma;\mathcal{C}\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}}\mathcal{V}[\![\tau\to\sigma]\!](\mathrm{fix}\,f(x).M_1,\mathrm{fix}\,f(x).M_2)$. To do so, we use the Löb rule, so we have to prove

  $$\Gamma;\underbrace{\mathcal{C},\triangleright(\mathcal{V}[\![\tau\to\sigma]\!](\mathrm{fix}\,f(x).M_1,\mathrm{fix}\,f(x).M_2))}_{\mathcal{C}'}\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}}\mathcal{V}[\![\tau\to\sigma]\!](\mathrm{fix}\,f(x).M_1,\mathrm{fix}\,f(x).M_2)$$

  Unwinding the definition, we will prove that

  $$\Gamma,v_1:\tau,v_2:\tau;\mathcal{C}',\mathcal{V}[\![\tau]\!](v_1,v_2)\vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}}\mathcal{E}[\![\tau\to\sigma]\!]((\mathrm{fix}\,f(x).M_1)v_1,(\mathrm{fix}\,f(x).M_2)v_2)$$

Using the rule STEP-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$, we just have to prove that

$$\Gamma';\mathcal{C}' \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \triangleright(\mathcal{E}\left[\!\left[\sigma\right]\!\right](M_1\left\{v_1/x\right\}\left\{f_1/f\right\}, M_2\left\{v_2/x\right\}\left\{f_1/f\right\}))$$

and using the rule $\triangleright$-MONO we can conclude using the premiss.

- Rule Gen :
  Using the rule $\mathcal{V}$-TO-$\mathcal{E}$, we just have to prove $\Gamma;\mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{V}\left[\!\left[\forall\alpha.\tau\right]\!\right]_\eta(\Lambda\alpha.M_1, \Lambda\alpha.M_2)$. So let α_1, α_2 two types and $r : \mathrm{Rel}_{\alpha_1,\alpha_2}$ and we prove that

  $$\Gamma, \alpha_1 : \mathrm{Type}, \alpha_2 : \mathrm{Type}, r : \mathrm{Rel}_{\alpha_1,\alpha_2}; \mathcal{C} \vdash_{\mathcal{D},\mathcal{K},\mathcal{SI}} \mathcal{E}\left[\!\left[\tau\right]\!\right]_{\eta,\alpha\mapsto r}(((\Lambda\alpha.M_1)\alpha_1, (\Lambda\alpha.M_2)\alpha_2)$$

  using the rule STEP-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$.
- Rule Inst :
  Let K_1, K_2 two contexts s.t. $\mathcal{K}\left[\!\left[\tau\left\{\sigma/\alpha\right\}\right]\!\right](K_1, K_2)$, we will prove that

  $$\mathcal{K}\left[\!\left[\forall\alpha.\tau\right]\!\right](K_1[\circ\sigma], K_2[\circ\sigma])$$

  Indeed, let v_1, v_2 two values s.t. $\mathcal{V}\left[\!\left[\forall\alpha.\tau\right]\!\right](v_1, v_2)$, we must show :

  $$\forall h_1, h_2.\mathcal{O}((K_1[v_1\sigma], h_1), (K_2[v_2\sigma], h_2))$$

  To do so, we use the fact that $\mathcal{E}\left[\!\left[\tau\right]\!\right]_{\eta,\alpha\mapsto r}(v_1\sigma, v_2\sigma)$ for all $r : \mathrm{Rel}_\sigma$, that $\mathcal{K}\left[\!\left[\tau\left\{\sigma/\alpha\right\}\right]\!\right]_\eta(K_1, K_2)$ and the equality between $\mathcal{E}\left[\!\left[\tau\right]\!\right]_{\eta,\alpha\mapsto\mathcal{V}\left[\!\left[\sigma\right]\!\right]_\eta}$ and $\mathcal{E}\left[\!\left[\tau\left\{\tau/\alpha\right\}\right]\!\right]_\eta$.
- Rule Unroll :
  Let K_1, K_2 s.t. $\mathcal{K}\left[\!\left[\tau\left\{\mu\alpha.\tau/\alpha\right\}\right]\!\right](K_1, K_2)$, we just have to prove that $\mathcal{K}\left[\!\left[\mu\alpha.\tau\right]\!\right](K_1[\mathrm{unroll}\circ], K_2[\mathrm{unroll}\circ])$. So let v_1, v_2 s.t. $\mathcal{V}\left[\!\left[\mu\alpha.\tau\right]\!\right](\mathrm{roll}\, v_1, \mathrm{roll}\, v_2)$, we will show that
  $\forall(h_1, h_2), \mathcal{O}((K_1[\mathrm{unroll}\,\mathrm{roll}\, v_1], h_1), (K_2[\mathrm{unroll}\,\mathrm{roll}\, v_2], h_2))$.
  To do so, we know that $\triangleright\mathcal{E}\left[\!\left[\tau\left\{\mu\alpha.\tau/\alpha\right\}\right]\!\right](v_1, v_2)$, i.e. $\triangleright\mathcal{O}((K_1[v_1], h_1), (K_2[v_2], h_2))$.
  Then we conclude using the rule StepS-$\triangleright\mathcal{O}$ with the fact that $(K[\mathrm{unroll}\,\mathrm{roll}\, v_i], h_i) \to (K[v_i], h_i)$ with $i \in \{1, 2\}$.
- Rule Roll :
  Let K_1, K_2 two contexts s.t. $\mathcal{K}\left[\!\left[\mu\alpha.\tau\right]\!\right]_\eta(K_1, K_2)$, we will prove that $\mathcal{K}\left[\!\left[\tau\left\{\mu\alpha.\tau/\alpha\right\}\right]\!\right]_\eta(K_1[\mathrm{roll}\circ], K_2[\mathrm{roll}\circ])$.
  Let v_1, v_2 two values s.t. $\mathcal{V}\left[\!\left[\tau\left\{\mu\alpha.\tau/\alpha\right\}\right]\!\right]_\eta(v_1, v_2)$, then we have to prove that

  $$\forall h_1, h_2.\mathcal{O}((K_1[\mathrm{roll}\, v_1], h_1), (K_2[\mathrm{roll}\, v_2], h_2))$$

  which is direct since, by monotony and substitution we have $\triangleright(\mathcal{V}\left[\!\left[\tau\right]\!\right]_{\eta,\alpha\mapsto\mathcal{V}\left[\!\left[\mu\alpha.\tau\right]\!\right]_\eta}(v_1, v_2))$ so $\mathcal{V}\left[\!\left[\mu\alpha.\tau\right]\!\right]_\eta(\mathrm{roll}\, v_1, \mathrm{roll}\, v_2)$.
- Rule Alloc :
  The proof takes place out of the layer $\mathcal{K}$, so let w a world, we have to prove

  $$\frac{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{SI}} \forall w' \sqsupseteq w.w' \Vdash \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta(K_1', K_2') \Rightarrow \forall(h_1, h_2).\mathcal{O}_{w'}((K_1'[M_1], h_1), (K_2'[M_2], h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{SI}} \forall w' \sqsupseteq w.w' \Vdash \mathcal{K}\left[\!\left[\mathrm{ref}\,\tau\right]\!\right]_\eta(K_1, K_2) \Rightarrow \forall(h_1, h_2).\mathcal{O}_{w'}((K_1[\mathrm{ref}\, M_1], h_1), (K_2[\mathrm{ref}\, M_2], h_2))}$$

  So let K_1, K_2 two contexts s.t. $w' \Vdash \mathcal{K}\left[\!\left[\mathrm{ref}\,\tau\right]\!\right]_\eta(K_1, K_2)$. We will prove that $w' \Vdash \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta(K_1[\mathrm{ref}\circ], K_2[\mathrm{ref}\circ])$. Taking $w'' \sqsupseteq w'$ and v_1, v_2 two values s.t. $w'' \Vdash \mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta(v_1, v_2)$, we have to prove :

  $$\forall h_1, h_2.\mathcal{O}_{w''}((K_1[\mathrm{ref}\, v_1], h_1), (K_2[\mathrm{ref}\, v_2], h_2))$$

  Let h_1, h_2 two heaps then $(K_i[\mathrm{ref}\, v_i], h_i) \to (K_i[l_i], h_i \bullet [l_i \mapsto v_i])$ with $l_i \notin \mathrm{dom}(h_i)$.
  If $(h_1, h_2) : w''$ then $(l_1, l_2) \notin \mathrm{dom}(w'')$, and we can build a new world

  $$w_0 = w'' \bullet [(l_1, l_2) \mapsto (w).w \Vdash \mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta]$$

  and using the rules StepL-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$ and StepR-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$ we just have to prove that

  $$\mathcal{O}_{w_0}((K_1[l_1], h_1 \bullet [l_1 \mapsto v_1]), (K_2[l_2], h_2 \bullet [l_2 \mapsto v_2]))$$

  which comes from the fact that $(h_1 \bullet [l_1 \mapsto v_1], h_2 \bullet [l_2 \mapsto v_2]) : w_0$ and that $w_0 \Vdash \mathcal{V}\left[\!\left[\mathrm{ref}\,\tau\right]\!\right]_\eta(l_1, l_2)$, true by definition of w_0.

- Rule Deref :

  The proof takes place out of the layer $\mathcal{K}$, so let w a world, we have to prove

$$\frac{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{SI}} \forall w' \sqsupseteq w.w' \Vdash \mathcal{K}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (K_1', K_2') \Rightarrow \forall(h_1,h_2).\mathcal{O}_{w'}((K_1'[M_1],h_1),(K_2'[M_2],h_2))}{\Gamma;\mathcal{C} \vdash_{\mathcal{D};\mathcal{SI}} \forall w' \sqsupseteq w.w' \Vdash \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2) \Rightarrow \forall(h_1,h_2).\mathcal{O}_{w'}((K_1[!M_1],h_1),(K_2[!M_2],h_2))}$$

  Let K_1, K_2 two contexts s.t. $w' \Vdash \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2)$, we will prove that $w' \Vdash \mathcal{K}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (K_1[!\circ], K_2[!\circ])$. So let a world $w'' \sqsupseteq w'$ and two values l_1, l_2 s.t. $w'' \Vdash \mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (l_1,l_2)$, we have to prove :

$$w'' \Vdash \forall h_1,h_2.\mathcal{O}((K_1[!l_1],h_1),(K_2[!l_2],h_2))$$

  Suppose $h_1, h_2 : w''$ then using the fact that $w'' \Vdash \mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (l_1,l_2)$, we know that $(l_1,l_2) \in \text{dom}(w'')$ because $\mathcal{W}(l_1,l_2)$ has to be defined since $\mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta$ is not the empty relation. Then we can deduce that $l_i \in \text{dom}(h_i)$.

  So using the rule Step-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$ with the same world, we just have to prove that

$$w'' \Vdash \triangleright\mathcal{O}((K_1[h_1(l_1)],h_1),(K_2[h_2(l_2)],h_2))$$

  which come from the fact that $w'' \Vdash \triangleright\mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta (h(l_1),h(l_2))$. Indeed, $h_1, h_2 : w''$ means $w'' \Vdash \mathcal{W}(l_1,l_2)(h_1(l_1),h_2(l_2))$ and we conclude using the definition of $\mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right]$.

- Rule Assign :

  Let K_1, K_2 two contexts s.t. $\mathcal{K}\left[\!\left[\text{Unit}\right]\!\right]_\eta (K_1,K_2)$, we will prove that $\mathcal{K}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (K_1[\circ := M_1], K_2[\circ := M_2])$. So let l_1, l_2 two values s.t. $\mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (l_1,l_2)$ we have to prove :

$$\forall h_1,h_2.\mathcal{O}((K_1[l_1 := M_1],h_1),(K_2[l_2 := M_2],h_2))$$

  To do so, we will prove that $\mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1[l_1 := \circ], K_2[l_2 := \circ])$. Let v_1, v_2 two values s.t. $\mathcal{V}\left[\!\left[\tau\right]\!\right]_\eta (v_1,v_2)$ we have to prove :

$$\forall h_1,h_2.\mathcal{O}((K_1[l_1 := v_1],h_1),(K_2[l_2 := v_2],h_2))$$

  Leaving the layer $\mathcal{K}$, suppose $h_1, h_2 : w$ then from $w \Vdash \mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right]_\eta (l_1,l_2)$ we get that $l_i \in \text{dom}(h_i)$

  Then we conclude, using the rules StepL-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$ and StepR-$\mathcal{K};\mathcal{SI}$-$\mathcal{E}$, by showing that :

$$w \Vdash \mathcal{O}((K_1[()],h_1[l_1 \mapsto v_1]),(K_2[()],h_2[l_2 \mapsto v_2]))$$

  using the fact that $w \Vdash \mathcal{K}\left[\!\left[\text{Unit}\right]\!\right]_\eta (K_1,K_2)$ and $(h_1[l_1 \mapsto v_1]), h_2[l_2 \mapsto v_2]) : w$

Proof tree for rule Step-$\mathcal{D};\mathcal{K};\mathcal{SI}$-$\triangleright\mathcal{E}$.

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} (M_i,h_i) \to (M_i',h_i)}}{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} (K_1[M_i],h_i) \to (K_2[M_i'],h_i)} \quad \dfrac{\dfrac{\overline{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2)}}{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} \triangleright\mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2)} \quad \dfrac{\overline{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} \triangleright\mathcal{E}\left[\!\left[\tau\right]\!\right](M_1',M_2')}}{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} \triangleright\mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2) \Rightarrow \triangleright\mathcal{O}((K_1[M_1],h_1),(K_2[M_2],h_2))}}{\Gamma';\mathcal{C}' \vdash_{\mathcal{F}} \triangleright\mathcal{O}((K_1[M_1],h_1),(K_2[M_2],h_2))}}{\Gamma, K_1, K_2 : \text{Cont}, h_1, h_2 : \text{Heap}; \mathcal{C}, \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2) \vdash_{\mathcal{F}} \mathcal{O}((K_1[M_1],h_1),(K_2[M_2],h_2))}}{\Gamma;\mathcal{C} \vdash_{\mathcal{F}} \forall(K_1,K_2),\mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2) \Rightarrow \forall h_1,h_2.\mathcal{O}((K_1[M_1],h_1),(K_2[M_2],h_2))}$$

  where $\mathcal{F} = \mathcal{D};\mathcal{K};\mathcal{SI}$ and $\Gamma';\mathcal{C}' = \Gamma, K_1, K_2 : \text{Cont}, h_1, h_2 : \text{Heap}; \mathcal{C}, \mathcal{K}\left[\!\left[\tau\right]\!\right]_\eta (K_1,K_2)$.