A Kripke Logical Relation for Effect-Based Program Transformations

Jacob Thamsborg
IT University of Copenhagen, Denmark
thamsborg@itu.dk

Lars Birkedal
IT University of Copenhagen, Denmark
birkedal@itu.dk

Abstract

We present a Kripke logical relation for showing the correctness of program transformations based on a type-and-effect system for an ML-like programming language with higher-order store and dynamic allocation.

We show how to use our model to verify a number of interesting program transformations that rely on effect annotations.

Our model is constructed as a step-indexed model over the standard operational semantics of the programming language. It extends earlier work [7, 8] that has considered, respectively, dynamically allocated first-order references and higher-order store for global variables (but no dynamic allocation). It builds on ideas from region-based memory management [21], and on Kripke logical relations for higher-order store, e.g. [12, 14].

Our type-and-effect system is region-based and includes a region-masking rule which allows to hide local effects. One of the key challenges in the model construction for dynamically allocated higher-order store is that the meaning of a type may change since references, conceptually speaking, may become dangling due to region-masking. We explain how our Kripke model can be used to show correctness of program transformations for programs involving references that, conceptually, are dangling.

Categories and Subject Descriptors D.3.1 [*PROGRAMMING LANGUAGES*]: Formal Definitions and Theory; F.3.1 [*LOGICS AND MEANINGS OF PROGRAMS*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords Kripke logical relation, ultrametric spaces, step-indexed model, type and effect system, program transformations, effect masking, dangling pointers

1. Introduction

A type system for a programming language classifies programs according to properties that the programs satisfy. An effect system is a type system that, in particular, classifies programs according to which side effects the programs may have. A variety of effect systems have been proposed for higher-order programming languages, e.g., [15, 18, 21], see [17] for a recent overview. Effect systems can often be understood as specifying the results of a static analysis, in the sense that it is possible to automatically infer types and effects. Effect systems can be used for different purposes: they were originally proposed by Lucassen and Gifford [18] for parallelization purposes but they have also, e.g., been used as the basis for implementing ML using a stack of regions for memory management [9, 21]. In a recent series of papers, Benton et. al. have argued that another important point of effect systems is that they can be used as the basis for effect-based program transformations, e.g., compiler optimizations, [5–8]. The idea is that certain program transformations are only sound under additional assumptions about which effects program phrases may, or rather may not, have. For example, in a higher-order language with references it is only sound to hoist an expression out a lambda abstraction if it is known that the expression neither allocates new references, nor reads or writes references.

While it is intuitively clear that effect information is important for validating program transformations, it is surprisingly challenging to develop semantic models that can be used to rigorously justify effect-based transformations. In earlier work, Benton et. al. developed semantic relational models of effect systems for a higher-order language with dynamically allocated first-order references (ground store) [7] and for a higher-order language with global variables for higher-order store (but no dynamic allocation) [8].

In this paper we present a Kripke logical relations model of a region-based effect system for a higher-order language with higher-order store *and* dynamic allocation, i.e., with general ML-like references. As pointed out in [8], this is a particularly challenging extension and one that is important for soundness of effect-based transformations for realistic ML-like languages. We now explain what the main challenges are; in Section 3 we give an intuitive overview of how we address these challenges.

The main challenge arises from effect masking: Our region-based effect system includes an effect masking rule that allows one to hide local uses of effects. This makes it possible to view a computation as pure even if it uses effects locally and makes the effect system stronger (it can justify more program transformations). To model effect masking we need to model references that, conceptually speaking, become dangling because they point into a region that is masked away. We say "conceptually speaking" because in the real operational semantics there is no deallocation of references (the operational semantics is completely standard); but in our model we have to reason *as if* regions could actually be allocated and deallocated.

Here is a simple concrete example: Consider the following expression e, typed as indicated:

$$\texttt{let x} = \texttt{ref}_{\rho}\ 7\ \texttt{in let y} = \texttt{ref}_{\sigma}\ \texttt{x in}$$
$$\lambda z.\texttt{y} := \texttt{x} : \texttt{1} \xrightarrow{\{wr_{\sigma}\}} \texttt{1}, \{al_{\sigma}\}$$

ICFP'11, September 19–21, 2011, Tokyo, Japan.

The program allocates two references, binds them to x and y, and then returns a trivial function from unit to unit that assigns x to y. The types indicate that x will be bound to a location in region ρ, say location 0, and that y will be bound to a location in region σ, say location 1. At location 0 the value 7 is stored and at location 1 the location 0 is stored. The so-called latent effect $\{wr_\sigma\}$ of the function type of the whole expression describes which effects the function may have when called. Finally the entire expression has the effect $\{al_\sigma\}$ as it allocates a location in region σ. It performs allocation in region ρ as well, but this effect has been masked out: since no ρ appears in the return type, the expression cannot leak location 0, and we can, conceptually, deallocate all locations in region ρ once the computation has run; this is what the masking rule captures. This particular example, however, was chosen to stress test the masking rule, as location 0 is in fact leaked: it is in the function closure that the expression returns. The function neither reads not writes location 0, but it does write it to the heap; our model must be able to cope with such conceptually dangling pointers. On the level of types, the meaning of the type $\texttt{ref}_\rho\,\texttt{int}$ of x changes: after the allocation of values for x and y, it contains location 0, but what should it contain after region ρ has been masked out? If, e.g., $\texttt{ref}_\rho\,\texttt{int}$ is taken to be empty, then the function is most likely no longer well-typed. We explain our answer to this question in Section 3.

Note that the effect annotations in the types are just annotations; the operational semantics is completely standard and regions only exist in our semantic model, not in the operational semantics. Further note that the issues in the above example do not arise for a language with only ground store.

Another challenge arises from the fact that since our language includes dynamically allocated general references, the existence of the logical relation is non-trivial; in particular, the set of worlds must be recursively defined. Here we build on our earlier work [12] and define the worlds as a solution to a recursive metric-space equation.

One may worry whether our resulting model now becomes much more complicated than the earlier, already non-trivial, models of Benton et. al. [7, 8]. As we shall explain in Section 8, that is not the case. Indeed, our model is arguably a bit simpler even though it applies to a richer language. Moreover, it also becomes simpler to verify equivalences using the model.

Our relational model is built directly over the operational semantics, using our metric approach to step-indexing. We could also have defined the model using a denotational model of the programming language; see the discussion in Section 8 — here we preferred the operational approach because it is perhaps more widely accessible.

We use our model to validate a number of interesting effect-based program transformations that, to the best of our knowledge, have not been proved correct before, see Section 7.

To focus the presentation on the core challenges, we here consider a monomorphically typed higher-order programming language with general references, but leave out universal and existential types as well as recursive types. However, we want to stress that since our semantic techniques (step-indexed Kripke logical relations over recursively defined worlds) do indeed scale well to universal, existential, and recursive types, e.g. [12, 14], it is straightforward to extend our model to a language with such types.[1] We conjecture that it is also possible to extend our model to richer effect systems involving region and effect polymorphism, but we have not done so yet.

[1] With type abstractions as values, we would have a latent effect on a polymorphic type, which would thus be of the form $\forall^\varepsilon\alpha.\tau$.

Throughout the paper we aim to explain the underlying intuitions of the technical definitions and how the different concepts interact. In particular, we have included a couple of selected proof cases, which serve to illuminate central points. A version of the paper with appendices containing more proof cases can be found online at `www.itu.dk/people/thamsborg/longcarnival.pdf`.

2. Language and Effect System

The expressions and values of our model language are defined in the grammar below. We use e to range over the set of expressions $\mathcal{E}$, v to range over the set of values $\mathcal{V}$, l to range over a countably infinite set of locations $\mathcal{L}$, and n to range over integers.

$$e ::= x \mid l \mid \underline{n} \mid () \mid (e_1, e_2) \mid \pi_1\, e \mid \pi_2\, e \mid \texttt{fix}\, f(x).e \mid e_1 e_2 \mid \texttt{ref}\, e \mid e_1 := e_2 \mid !\, e$$

$$v ::= l \mid \underline{n} \mid () \mid (v_1, v_2) \mid \texttt{fix}\, f(x).e$$

Note that expressions are not explicitly typed (we could also have worked with an explicitly typed language). The operational semantics is given by a small-step operational semantics in the standard way. We omit the full definition; suffice it to say that heaps $\mathcal{H}$ are finite maps from locations to closed values, and, for $e \in \mathcal{E}$, $h \in \mathcal{H}$ and $j \geq 0$, we write $\langle e, h\rangle \xrightarrow{j} \langle e', h'\rangle$ if $\langle e, h\rangle$ reduces to $\langle e', h'\rangle$ in j steps, counting all reduction steps. We write $\mathrm{irr}\langle e, h\rangle$ to state that no further reductions of $\langle e, h\rangle$ are possible, either because e is in fact a value or because we are stuck. To make some of the later proofs a little bit simpler, we assume that the allocation of a new location by evaluation of $\texttt{ref}\ e$ is deterministic. Runtime errors (such as trying to look up a non-existing location or trying to apply a number as a function) are modelled by the evaluation getting stuck.

Our effect system ensures that programs are well-typed in the standard sense of not getting stuck and, moreover, tracks dependencies and side-effects of computations on the heap.

The typing rules for our effect system are given in Figure 1. Types are defined by the following grammar:

$$\tau ::= \texttt{int} \mid \texttt{1} \mid \tau_1 \times \tau_2 \mid \tau_1 \xrightarrow{\varepsilon} \tau_2 \mid \texttt{ref}_\rho\, \tau,$$

where ρ ranges over a countably infinite set of region variables $\mathcal{RV}$ and ε is ranges over the set of effects. An effect is a finite set of primitive effects, and a primitive effect is of one of the forms rd_ρ, wr_ρ, or al_ρ. The primitive effect rd_ρ specifies a read effect on region ρ. Likewise wr_ρ specifies a write effect and al_ρ specifies an allocation effect (that locations may be allocated in region ρ).

Note that, as usual, the function type $\tau_1 \xrightarrow{\varepsilon} \tau_2$ includes a *latent* effect ε; this is the effect that the function may have when called.

The effect type judgement defined in Figure 1 takes the form

$$\Pi \mid \Gamma \vdash e : \tau, \varepsilon,$$

with Γ a finite map from program variables to types, e an expression, τ a type, ε an effect, and Π a finite set of region variables, including all the region variables in Γ, τ, and ε.

The effect system is fairly standard. Note the inclusion of the effect masking rule T-MASKING; here $\varepsilon - \rho$ is defined as $\varepsilon - \{rd_\rho, wr_\rho, al_\rho\}$ and Π, ρ denotes the union of Π and $\{\rho\}$. We write $\mathrm{FRV}(\Gamma, \tau)$ for the region variables in the types of Γ and in τ. The masking rule expresses that to mask out primitive effects on ρ from ε, the region variable ρ must not be free in Γ, nor in τ. The idea is that in this case the remaining part of the computation cannot access ρ and thus we may hide the effects on ρ.

Also note that there is a standard notion of effect subtyping, given by finite set inclusion, which also yields a notion of subtyping (defined by the last three rules in the figure).

$$\frac{}{\Pi \mid \Gamma, x{:}\tau \vdash x : \tau, \emptyset} \qquad \frac{}{\Pi \mid \Gamma \vdash () : \mathtt{1}, \emptyset} \qquad \frac{}{\Pi \mid \Gamma \vdash \underline{n} : \mathtt{int}, \emptyset} \qquad \frac{\Pi \mid \Gamma \vdash e : \mathtt{ref}_\rho \tau, \varepsilon}{\Pi \mid \Gamma \vdash !\, e : \tau, \varepsilon \cup \{rd_\rho\}}$$

$$\frac{\Pi \mid \Gamma \vdash e_1 : \mathtt{ref}_\rho \tau, \varepsilon_1 \qquad \Pi \mid \Gamma \vdash e_2 : \tau, \varepsilon_2}{\Pi \mid \Gamma \vdash e_1 {:=} e_2 : \mathtt{1}, \varepsilon_1 \cup \varepsilon_2 \cup \{wr_\rho\}} \qquad \frac{\Pi \mid \Gamma \vdash e : \tau, \varepsilon \qquad \rho \in \Pi}{\Pi \mid \Gamma \vdash \mathtt{ref}\ e : \mathtt{ref}_\rho \tau, \varepsilon \cup \{al_\rho\}}$$

$$\frac{\Pi \mid \Gamma \vdash e_1 : \tau_1, \varepsilon_1 \qquad \Pi \mid \Gamma \vdash e_2 : \tau_2, \varepsilon_2}{\Pi \mid \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2, \varepsilon_1 \cup \varepsilon_2} \qquad \frac{\Pi \mid \Gamma \vdash e : \tau_1 \times \tau_2, \varepsilon}{\Pi \mid \Gamma \vdash \pi_1\, e : \tau_1, \varepsilon} \qquad \frac{\Pi \mid \Gamma \vdash e : \tau_1 \times \tau_2, \varepsilon}{\Pi \mid \Gamma \vdash \pi_2\, e : \tau_2, \varepsilon}$$

$$\frac{\Pi \mid \Gamma, f{:}\tau_1 \xrightarrow{\varepsilon} \tau_2, x{:}\tau_1 \vdash e : \tau_2, \varepsilon}{\Pi \mid \Gamma \vdash \mathtt{fix}\ f(x).e : \tau_1 \xrightarrow{\varepsilon} \tau_2, \emptyset} \qquad \frac{\Pi \mid \Gamma \vdash e_1 : \tau_1 \xrightarrow{\varepsilon} \tau_2, \varepsilon_1 \qquad \Pi \mid \Gamma \vdash e_2 : \tau_1, \varepsilon_2}{\Pi \mid \Gamma \vdash e_1 e_2 : \tau_2, \varepsilon \cup \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Pi, \rho \mid \Gamma \vdash e : \tau, \varepsilon \qquad \rho \notin \mathrm{FRV}(\Gamma, \tau)}{\Pi \mid \Gamma \vdash e : \tau, \varepsilon - \rho}\ \text{T-MASKING}$$

$$\frac{\Pi \mid \Gamma \vdash e : \tau_1, \varepsilon_1 \qquad \Pi \mid \tau_1 \leq \tau_2 \qquad \varepsilon_1 \subseteq \varepsilon_2}{\Pi \mid \Gamma \vdash e : \tau_2, \varepsilon_2}\ \text{T-SUB}$$

$$\frac{}{\Pi \mid \tau \leq \tau} \qquad \frac{\Pi \mid \tau_1 \leq \tau_1' \qquad \Pi \mid \tau_2 \leq \tau_2'}{\Pi \mid \tau_1 \times \tau_2 \leq \tau_1' \times \tau_2'} \qquad \frac{\Pi \mid \tau_1' \leq \tau_1 \qquad \Pi \mid \tau_2 \leq \tau_2' \qquad \varepsilon_1 \subseteq \varepsilon_2}{\Pi \mid \tau_1 \xrightarrow{\varepsilon_1} \tau_2 \leq \tau_1' \xrightarrow{\varepsilon_2} \tau_2'}$$

Figure 1. Effect System

Consider the example expression e from the introduction. Note that it is well-typed (`let`-expressions are definable as usual) since the judgement $\{\sigma\} \mid \emptyset \vdash e : \mathtt{1} \xrightarrow{\{wr_\sigma\}} \mathtt{1}, \{al_\sigma\}$ is derivable in the effect system. The last rule applied is the T-MASKING rule, which allows us to hide the local effect on ρ. Specifically, we apply the masking rule like this:

$$\frac{\{\sigma, \rho\} \mid \emptyset \vdash e : \mathtt{1} \xrightarrow{\{wr_\sigma\}} \mathtt{1}, \{al_\rho, al_\sigma\}}{\{\sigma\} \mid \emptyset \vdash e : \mathtt{1} \xrightarrow{\{wr_\sigma\}} \mathtt{1}, \{al_\sigma\}}$$

Thus we hide the allocation and use of the local reference bound to x.

Effect masking makes it possible to do more optimizations: Consider the familiar example of an efficient implementation *fib* of the fibonacci function using two local references. We can use the masking rule to give it type and effect $\mathtt{int} \xrightarrow{\emptyset} \mathtt{int}, \emptyset$. This allows us to view the imperative implementation as pure, and thus, e.g, by Theorem 7.1 we find that it is sound to optimize two identical calls to *fib* into one call. This sounds like a simple optimization, but the point is that a compiler can perform it automatically, just based on the effect types — and our model justifies that it is sound to do so. See [7, 8] for more examples.

Note though, that these earlier works either have ground store (only integers in memory) or no dynamic allocation, so many non-trivial uses of memory cannot be expressed. By contrast, we can write, say, a function that sorts a (functional) list of integers by doing an internal (imperative) heap sort, i.e., by building a heap in fresh memory, and returning a sorted list. Using the masking rule, we could type such a function as a pure function with no effects, and would know, e.g., that sorting the same list twice serves no purpose.

3. Overview of the Technical Development

Our point of departure is the Kripke logical relation reading of types for an ML-like programming langauge that has been explored extensively by the authors and others [2, 11, 12, 14]. The common approach is to augment the semantics — be it operational or denotational — with *worlds* that keep track of the layout of the heap: it knows the types of values stored at the allocated locations of the heap.[2] We then index type interpretations by worlds; this makes it possible to interpret $\mathtt{ref}\,\tau$ as the set of allocated locations that hold values of type τ. A computation has the "precondition" that all locations hold well-typed values according to the world, and this property must be re-established after running the computation, i.e., it also serves as a "postcondition". As more locations are allocated, the world grows and the interpretation of types is set up to grow as well: it is a crucial property that interpreting types in future worlds yields more (or at least no fewer) values, this we refer to as *type monotonicity*.

Barring the masking rule, it is a mostly straightforward, if lengthy, exercise to extend this approach to the present region-based type-and-effect system. The worlds have to be partitioned into regions and interpreting $\mathtt{ref}_\rho\,\tau$ only considers region ρ of the world for locations that hold values of type τ. Also, computations now have effects and hence their behavior is more restricted: as "precondition" they only assume well-typedness of values at locations in the regions with read effects; as "postcondition" they are required to have performed only well-typed writes and allocations and, importantly, only in such regions as permitted, respectively, by the write and allocation effects.

The masking rule, however, introduces a new dimension to the development of worlds: in addition to adding new locations with types to existing regions as sketched above, we may now introduce new regions as well as *mask out* existing ones. One can loosely think of masking out region ρ as deallocating the locations of that region. A slightly revised intuition is that we just stop caring about region ρ: we expect never to touch it again and so we may safely relinquish control. On the level of worlds, we discard the region, i.e., lose the locations and the associated types. Whether the locations are actually deallocated or just float around in the surroundings does not matter: from our point of view they are gone.

As argued by example above, this leaves us the issue of dangling pointers. Phrased more concretely: how should the interpretation of

[2] In general, worlds may contain complex invariants of the heap; they may even vary over time and may thus be represented by state transition systems [14]. Here, we need only the invariant that values stored at locations belong to certain types.

types cope with region masking? What is, say, the interpretation of $\texttt{ref}_\rho\,\tau$ in a world where region ρ has been masked out? A natural choice is the empty set — after all, we know nothing about region ρ so what locations could we possibly choose. We, however, take a different approach: We generally interpret a type as all values with the properties you expect from that type. The more properties, the fever values and vice versa. A value v in $\texttt{ref}_\rho\,\tau$ is a reference to a location that we used to — but no longer — control. The value at the location may have changed or it may even have been deallocated: v is conceptually *dangling*. What can we reasonably do with a dangling pointer? We can neither read nor write it, indeed, we can do only things that go for all values, e.g., put it into a pair, project it out, etc. In other words, we expect no properties of dangling pointers and correspondingly interpret $\texttt{ref}_\rho\,\tau$ as the set of all values. We take a similar stand on functions: interpreting a function type that has effects on masked out regions gives the set of all values; such functions are somehow dangling too. Running such a function relies on preconditions outside our control, so we make no promises about the behavior.

This approach goes well with type monotonicity: the interpretation of types should not shrink under masking and, indeed, masking out region ρ enlarges $\texttt{ref}_\rho\,\tau$ to hold all values. There is a catch, though, since future worlds may introduce new regions as well, and reintroducing region ρ would not do. This is obvious from type monotonicity; a more conceptual explanation, however, is this: Assume that some function reads region ρ and that region ρ holds location l that stores values of some type. To run the function, we need to establish the precondition that a value of the proper type is stored at location l, because the function could very well read l. We then mask out region ρ, losing all information about l in the process, and afterwards reintroduce it. To run the function now, we verify that all locations in region ρ hold values of appropriate type — this is, after all, the precondition of the function — but as l is no longer in ρ, we cannot expect proper behavior of the function.

We solve this issue by tracking, in the worlds, the regions that have been masked out and prohibit their reintroduction. One viewpoint is that, as the world develops over time, a region goes through a life-cycle: Initially, it is unknown to the world, but at some point it gets initialized, joining the set of *live* regions. With further development of the world, locations with associated types are added to the region. This proceeds until the region is masked out; it loses all content and is moved to the set of *dead* regions. And this is a one-way street: once you are gone, you can't come back.

Local Reasoning Conceptually, a world does not describe the entire heap, just the part of it that the program sees or controls. On the level of definitions, this comes down to a frame property of our computations: a computation in a world runs in a world-adhering heap extended with a frame, and the latter remains unchanged. In addition, a computation may allocate locations that are not tracked by the ensuing world; these locations are conceptually transferred to the surroundings, they become part of the frame of the following computation. This is the fate of locations in regions that are masked out. In summa, we achieve a form of local reasoning by quantifying over frames, similarly to models of separation logic for higher-order languages [10].

4. Metric Spaces and Type-World Circularity

As argued above, we intend to augment our semantics with worlds that track the layout of the heap; hence we are faced with the construction of such worlds. Defining them directly will not do since, loosely, a world holds semantic types whilst semantic types are parameterized over worlds; this is the *type-world circularity* observed already by Ahmed in her thesis [1].

In recent work, Reus, Schwinghammer, Støvring, Yang and the authors [12] have proposed a general solution to such circularities, applying metric-space theory. The notion of worlds we require here is sufficiently simple that this solution is applicable off-the-shelf, so to speak. So we omit the machinery of the construction here and just present the worlds that is the result; after all, this is presently our object of interest. Details are given in the appendix of the long version of the paper. In addition, we will largely ignore the fact that we actually deal in metric spaces and not just plain sets. We emphasize that this is just for presentation purposes, worlds and types *are* metric spaces with certain properties; this is necessary to solve the circularity and must be taken into account, e.g., when interpreting types, see the appendix of the long version of the paper.

We face an additional challenge here: As described above, we intend to track dead regions to avoid recycling them. But if an expression is well-typed by the masking rule, then in terms of the development of worlds, it initializes and eventually masks out the very same region on each evaluation, and maybe we would like to run it more than once. Our solution to this is to introduce a layer of indirection, following ideas of earlier work [9, 21]: On the level of syntactic types, we have the region variables $\mathcal{RV}$ introduced above. In the worlds, however, we work with a countably infinite set of *region names* $\mathcal{RN}$. To interpret types we have *region environments*, i.e., injective maps $\mathcal{RV} \rightharpoonup_{\mathit{fin}} \mathcal{RN}$ with adequate domains; on each evaluation of an expression that performs masking, we then map the same region variable to a fresh region name. In the textual explanations below, however, we purposely blur the distinction between region variables, region names and the regions themselves.

The definition of worlds and types is given in Figure 2; the ordering on worlds relies on the world transitions given in Figure 3. Both warrant a few comments. $\mathrm{ParBij}(X, Y, \mathbf{Z})$ are finite partial bijections between X and Y decorated with elements from $\mathbf{Z}$; we write $\mathrm{dom}_1(P)$ for the set of first coordinates and $\mathrm{dom}_2(P)$ for the set of second coordinates. Worlds $\mathbf{W}$ have two components: the first is the *live* regions, these are partial bijections; the second the *dead* regions. No region can be both live and dead, nor can any location belong to more than one region.

Worlds may develop over time according to the transitions in Figure 3. The first transition adds a location pair with associated type to a live region; this corresponds to an actual allocation in the operational semantics and is a standard notion of world extension. The second and third transitions are orthogonal to the first. They give the region dynamics and have no counterpart in the operational semantics; they are, however, intimately connected to the masking rule. The first initializes a new empty region, the second masks out a live one, losing the partial bijection in the process. After being masked out, a region is considered dead and cannot be initialized once more.

The reflexive, transitive closure of the transitions is a preorder on worlds. We require of our types $\mathbf{T}$ that they are monotone with respect to that preorder and the standard set-theoretic inclusion on $\mathit{URel}(\mathcal{V})$; this is the type monotonicity. For any set X, $\mathit{URel}(X)$ is the set of indexed, downwards closed relations on X, i.e.,

$$\mathit{URel}(X) = \{R \subseteq \mathbb{N} \times X \times X \mid \\ \forall (k, x_1, x_2) \in R.\, \forall j < k.\, (j, x_1, x_2) \in R\}.$$

The downwards closure is essential, it prevents values from fleeing types as the operational semantics progresses.[3] One minor issue remains: the types that decorate the partial bijections that are the regions belong to $\widehat{\mathbf{T}}$ and must be coerced into $\mathbf{T}$ by the isomor-

[3] The set $\mathit{URel}(X)$ has a natural metric [12] and the functions in $\mathbf{T}$ are not only monotone but also non-expansive; see the appendix of the long version of the paper for details.

$$
\begin{aligned}
\mathrm{ParBij}(X,Y,\mathbf{Z}) = \{P \subseteq_{fin} X \times Y \times \mathbf{Z} \mid \forall (x_1,y_1,z_1),(x_2,y_2,z_2) \in P.\\
(x_1 = x_2 \Rightarrow y_1 = y_2 \wedge z_1 = z_2) \wedge (y_1 = y_2 \Rightarrow x_1 = x_2 \wedge z_1 = z_2)\}\\
\mathbf{W} = \{(\varphi,\psi) \in (\mathcal{RN} \rightharpoonup_{fin} \mathrm{ParBij}(\mathcal{L},\mathcal{L},\widehat{\mathbf{T}})) \times \mathcal{P}_{fin}(\mathcal{RN}) \mid \mathrm{dom}(\varphi) \cap \psi = \emptyset \wedge\\
\forall r,s \in \mathrm{dom}(\varphi).\, r \neq s \Rightarrow \mathrm{dom}_1(\varphi(r)) \cap \mathrm{dom}_1(\varphi(s)) = \emptyset = \mathrm{dom}_2(\varphi(r)) \cap \mathrm{dom}_2(\varphi(s))\}\\
w \sqsubseteq w' \iff \exists n \in \mathbb{N}.\, \exists w_0, w_1, \ldots, w_n \in \mathbf{W}.\, w = w_0 \wedge w' = w_n \wedge\\
\forall 0 \leq i < n.\, (\exists r \in \mathcal{RN}.\, \exists l_1, l_2 \in \mathcal{L}.\, \exists \mu \in \widehat{\mathbf{T}}.\, w_i \rightarrow_{al(r,l_1,l_2,\mu)} w_{i+1}) \vee\\
(\exists r \in \mathcal{RN}.\, w_i \rightarrow_{reg(r)} w_{i+1}) \vee (\exists r \in \mathcal{RN}.\, w_i \rightarrow_{mask(r)} w_{i+1})\\
\mathbf{T} = \mathbf{W} \rightarrow_{mon} \mathit{URel}(\mathcal{V}), \quad \iota : \widehat{\mathbf{T}} \cong \frac{1}{2}\mathbf{T}
\end{aligned}
$$

Figure 2. Our semantic footing: worlds and types. The former comes equipped with the preorder that is the reflexive, transitive closure of the transitions in Figure 3

$$
\begin{aligned}
(\varphi,\psi) \rightarrow_{al(r,l_1,l_2,\mu)} (\varphi',\psi) &\iff r \in \mathrm{dom}(\varphi) \wedge l_1 \notin \mathrm{dom}_1(\varphi(r)) \wedge l_2 \notin \mathrm{dom}_2(\varphi(r)) \wedge\\
&\quad \mathrm{dom}(\varphi') = \mathrm{dom}(\varphi) \wedge \varphi'(r) = \varphi(r) \cup \{(l_1,l_2,\mu)\} \wedge\\
&\quad \forall s \in \mathrm{dom}(\varphi) \setminus \{r\}.\, \varphi'(s) = \varphi(s).\\
(\varphi,\psi) \rightarrow_{reg(r)} (\varphi',\psi) &\iff r \in \mathcal{RN} \setminus (\mathrm{dom}(\varphi) \cup \psi) \wedge \mathrm{dom}(\varphi') = \mathrm{dom}(\varphi) \cup \{r\} \wedge\\
&\quad \varphi'(r) = \emptyset \wedge \forall s \in \mathrm{dom}(\varphi).\, \varphi'(s) = \varphi(s).\\
(\varphi,\psi) \rightarrow_{mask(r)} (\varphi',\psi') &\iff r \in \mathrm{dom}(\varphi) \wedge \mathrm{dom}(\varphi') = \mathrm{dom}(\varphi) \setminus \{r\} \wedge\\
&\quad \psi' = \psi \cup \{r\} \wedge \forall s \in \mathrm{dom}(\varphi) \setminus \{r\}.\, \varphi'(s) = \varphi(s).
\end{aligned}
$$

Figure 3. The three worlds transitions. We have $r \in \mathcal{RN}$ in all of them and additionally $l_1, l_2 \in \mathcal{L}$ and $\mu \in \widehat{\mathbf{T}}$ in the first. Transitions are, once parameters are given, partial maps from $\mathbf{W}$ to $\mathbf{W}$.

phism ι before they can be applied to a world. This isomorphism is what we get from the solution to the type-world circularity.

A bit of world-related notation will come in handy: For a world $w = (\varphi,\psi)$ we set $\mathrm{dom}(w) = \mathrm{dom}(\varphi)$ and $|w| = \mathrm{dom}(\varphi) \cup \psi$, i.e., $\mathrm{dom}(w)$ is the set of live regions and $|w|$ is the combined set of live and dead regions. We write $\mathrm{dom}_1(w)$ for the union of all first coordinates of all live regions; $\mathrm{dom}_2(w)$ is defined similarly. For $r \in \mathrm{dom}(w)$ we abuse notation and write $w(r)$ for $\varphi(r)$.

5. Types and the Logical Relation

The relational, world-indexed interpretation of types is given in Figure 4; the interpretation of function types relies on the interpretation of computations given in Figure 5. It is worthwhile to comment a bit on this. Note that both the interpretation of types and computations take a number of parameters, the requirements on those are given in the captions of the figures.

Overall, we follow the intuition laid out in Section 3. Integer, unit and product types are standard. Looking at the reference type there are three cases: If we interpret $\mathtt{ref}_\rho\,\tau$ in a world where $R(\rho)$ is unknown, we get the empty set. This we never do; it is just a dummy clause that is neutral with respect to type monotonicity, but we need it since we, for technical convenience, want interpreted types to be applicable to all worlds. The middle case is the proper one, here $R(\rho)$ is a live region and we go through it in search for location pairs that hold values of a type semantically identical to τ. The quantification over future worlds ensure type monotonicity and the k-equality is necessary for the step-indexed setup; both are quite standard. The latter means that the sets we compare are equal if we restrict to elements with index strictly less than k. In the last case we look for references to a masked region; these are conceptually dangling pointers that we make no assumption about, hence we return all (pairs of) values.

The function type proceeds along the same lines. The middle case is the proper one and it too is quite standard, except that we quantify only over future worlds in which $R(\mathrm{FRV}(\varepsilon))$ remain live. In worlds where this fails, we make no promise about the behavior.

The interpretations of computations is crucial. Note first that $\mathrm{rds}\,\varepsilon$ are all region variables with read effects in ε, $\mathrm{wrs}\,\varepsilon$ and $\mathrm{als}\,\varepsilon$ are defined similarly. $\mathbf{P}^R_\varepsilon\, w$ gives the precondition on heaps that computations running in world w with effects ε have: that all location pairs in all regions with read effects do, in fact, hold values of the appropriate type. If, now, the precondition holds and the left hand side terminates, then so does the right hand side, and there is a future world w' such that the results are in the desired type at w' and the resulting heaps satisfy the postcondition $\mathbf{Q}^R_\varepsilon\, w, w'$. The latter states, that any writes to existing locations are of the correct type and are permitted by a write effect; also any newly allocated locations tracked by w' hold well-typed values and are in regions with an allocation effect. That the postcondition speaks of the initial heaps as well as the final ones is seen in, e.g., Hoare Type Theory [19] as well. Also, since we do local reasoning, there may be parts of the heap outside our control; these are the frames f_1 and f_2, they must remain unmodified. And the computations may allocate locations that the future world does not track, these are the additional frames f_1' and f_2'.

Finally, it is worthwhile to remark that the region-dynamics of computations is quite restricted: the future world w' must have the same live regions as w. In other words, computations cannot mask out existing regions and if they initialize any new regions, they are obliged to mask them out before they terminate. Here we take inspiration from work on region-based memory management [9,

$$[\![\mathtt{1}]\!]^R w = \{(k,(),()) \mid k \in \mathbb{N}\} \qquad [\![\mathtt{int}]\!]^R w = \{(k,n,n) \mid k \in \mathbb{N} \wedge n \in \mathbb{Z}\}$$

$$[\![\tau_1 \times \tau_2]\!]^R w = \{(k,(v_{1,1},v_{2,1}),(v_{1,2},v_{2,2})) \mid (k,v_{1,1},v_{1,2}) \in [\![\tau_1]\!]^R w \wedge (k,v_{2,1},v_{2,2}) \in [\![\tau_2]\!]^R w\}$$

$$[\![\mathtt{ref}_\rho\, \tau]\!]^R w = \begin{cases} \emptyset & R(\rho) \notin |w| \\ \left\{(k,l_1,l_2) \mid \exists \mu \in \widehat{\mathbf{T}}.\, (l_1,l_2,\mu) \in w(R(\rho)) \wedge \forall w' \sqsupseteq w.\, [\![\tau]\!]^R w' \stackrel{k}{=} (\iota\,\mu)(w')\right\} & R(\rho) \in \mathrm{dom}(w) \\ \{(k,v_1,v_2) \mid k \in \mathbb{N} \wedge v_1,v_2 \in \mathcal{V}\} & R(\rho) \in |w| \setminus \mathrm{dom}(w). \end{cases}$$

$$[\![\tau_1 \xrightarrow{\varepsilon} \tau_2]\!]^R w = \begin{cases} \emptyset & R(\mathrm{FRV}(\varepsilon)) \notin |w| \\ \left\{ (k, \mathtt{fix}\, f(x).e_1,\ \mathtt{fix}\, f(x).e_2) \;\middle|\; \begin{array}{r} \forall w' \sqsupseteq w.\, R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w') \Rightarrow \\ \forall j \le k. \forall (v_1,v_2) \in \mathcal{V} \times \mathcal{V}.\, (j,v_1,v_2) \in [\![\tau_1]\!]^R w' \Rightarrow \\ (j, (\mathtt{fix}\, f(x).e_1)\, v_1, (\mathtt{fix}\, f(x).e_2)\, v_2) \in [\![\mathtt{T}_\varepsilon\, \tau_2]\!]^R w' \end{array} \right\} & R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w) \\ \{(k,v_1,v_2) \mid k \in \mathbb{N} \wedge v_1,v_2 \in \mathcal{V}\} & \mathrm{dom}(w) \not\supseteq R(\mathrm{FRV}(\varepsilon)) \subseteq |w|. \end{cases}$$

Figure 4. Interpretation of types. We require $R : \mathcal{RV} \rightharpoonup_{fin} \mathcal{RN}$ injective with $\mathrm{FRV}(\tau) \subseteq \mathrm{dom}(R)$ and get $[\![\tau]\!]^R : \mathbf{W} \to_{mon} URel(\mathcal{V})$.

21] where regions are allocated and deallocated following a stack discipline.

Proposition 5.1. We have $[\![\tau]\!]^R \in \mathbf{T}$, i.e., it is a non-expansive and monotone map from $\mathbf{W}$ to $URel(\mathcal{H})$.

The proof of this proposition is deferred to the appendix of the long version of the paper.

When interpreting a type, we use region environments with possibly excessive domains. Clearly, the value of the region environments outside the region variables of the type does not matter; this is captured in the following lemma that we need below:

Lemma 5.2 (Environment Extension)**.** Let $R_1, R_2 : \mathcal{RV} \rightharpoonup_{fin} \mathcal{RN}$ be injective with $\mathrm{dom}(R_1) \supseteq \mathrm{FRV}(\tau) \subseteq \mathrm{dom}(R_2)$. If we have $R_1(\rho) = R_2(\rho)$ for all $\rho \in \mathrm{FRV}(\tau)$, then $[\![\tau]\!]^{R_1} = [\![\tau]\!]^{R_2}$.

The logical relation on expressions is defined in Figure 6. Π is a syntactic over-approximation of all region variables; together with the condition $R : \Pi \hookrightarrow |w|$ it ensures that we deal in live and dead regions only, not unknown ones. As for functions and computations, we require that the regions of the effects are live.

The logical relation is asymmetrical: the left hand side *approximates* the right hand side. We write $\Pi \mid \Gamma \models e_1 \sim e_2 : \tau, \varepsilon$ if the approximation goes both ways and consider the computations *equivalent* in that case. Our logical relation and subtyping are sound in the following, standard, sense:

Theorem 5.3 (Compatibility)**.** The logical relation in Figure 6 is compatible with the typing rules of Figure 1. That is, the formation of expressions according to the typing rules respects the logical relation.

Proposition 5.4. Interpreting subtypes gives subsets, i.e., we have that $\Pi \mid \tau_1 \le \tau_2$ implies $[\![\tau_1]\!]^R w \subseteq [\![\tau_2]\!]^R w$ for any region environment $R : \Pi \hookrightarrow \mathcal{RN}$ and any world $w \in \mathbf{W}$.

The proof of the theorem relies naturally on the proposition. The proofs are deferred to the next section and the appendix of the long version of the paper.

We have the Fundamental Lemma as corollary:

Lemma 5.5. (Fundamental)

$$\Pi \mid \Gamma \vdash e : \tau, \varepsilon \implies \Pi \mid \Gamma \models e \preceq e : \tau, \varepsilon.$$

Another standard corollary is *contextual equivalence*: Two equivalent computations that are placed inside the same closing, integer context, will co-terminate with the same value, when run in any two heaps. Observe, though, that the context must be typed according to the typing rules here, i.e., with effect annotations.

6. Soundness

In this section, we present some of the cases of the proof of the Compatibility Theorem stated just above. First off, two lemmas to handle some of the details, then we give detailed proofs of lookup, masking, and fixed points. See the appendix of the long version of the paper for more cases.

Lemma 6.1 (Precondition Strengthening)**.**

$$\varepsilon \subseteq \varepsilon' \implies \mathbf{P}^R_\varepsilon\, w \supseteq \mathbf{P}^R_{\varepsilon'}\, w.$$

Lemma 6.2 (Postcondition Weakening)**.**

$$\varepsilon \subseteq \varepsilon' \implies \mathbf{Q}^R_\varepsilon\, w, w' \subseteq \mathbf{Q}^R_{\varepsilon'}\, w, w'.$$

Lemma 6.3 (Lookup)**.** $\Pi \mid \Gamma \models e_1 \preceq e_2 : \mathtt{ref}_\rho\, \tau, \varepsilon$ implies $\Pi \mid \Gamma \models !\, e_1 \preceq !\, e_2 : \tau, \varepsilon \cup \{rd_\rho\}$.

Proof. We unroll the definition of the logical relation: Let $k \in \mathbb{N}$, $w \in \mathbf{W}$, $R : \Pi \hookrightarrow |w|$ and $\gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}$ be arbitrary. Assume $R(\mathrm{FRV}(\varepsilon \cup \{rd_\rho\})) \subseteq \mathrm{dom}(w)$ and that $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w$. We now must show that

$$(k, !\, e_1[\gamma_1/\Gamma], !\, e_2[\gamma_2/\Gamma]) \in [\![\mathtt{T}_{\varepsilon \cup \{\mathrm{rd}_\rho\}}\, \tau]\!]^R w.$$

We proceed to unroll the definition of computations: Let $j \le k$, $e_1'' \in \mathcal{E}$ and $h_1, h_2, f_1, f_2, g_1'' \in \mathcal{H}$ be arbitrary. Assume $(k, h_1, h_2) \in \mathbf{P}^R_{\varepsilon \cup \{rd_\rho\}}\, w$, that

$$\langle !\, e_1[\gamma_1/\Gamma], h_1 \cdot f_1 \rangle \xrightarrow{j} \langle e_1'', g_1'' \rangle$$

and $\mathrm{irr}\langle e_1'', g_1'' \rangle$.

By the operational semantics, there must be $0 \le i \le j$, $e_1' \in \mathcal{E}$ and $g_1' \in \mathcal{H}$, such that

$$\langle e_1[\gamma_1/\Gamma], h_1 \cdot f_1 \rangle \xrightarrow{i} \langle e_1', g_1' \rangle, \quad \langle !\, e_1', g_1' \rangle \xrightarrow{j-i} \langle e_1'', g_1'' \rangle$$

and $\mathrm{irr}\langle e_1', g_1' \rangle$ holds. The assumption of the lemma and Preconditioning Strengthening gives us $w' \sqsupseteq w$ with $\mathrm{dom}(w') = \mathrm{dom}(w)$ as well as $e_2' \in \mathcal{E}$ and $g_2', h_1', h_2', f_1', f_2'$ such that

$$\langle e_2[\gamma_2/\Gamma], h_2 \cdot f_2 \rangle \xrightarrow{*} \langle e_2', g_2' \rangle,$$

with $g_1' = h_1' \cdot f_1' \cdot f_1$, $g_2' = h_2' \cdot f_2' \cdot f_2$ and $(k-i, e_1', e_2') \in [\![\mathtt{ref}_\rho\, \tau]\!]^R w'$ as well as $(k-i, h_1, h_2, h_1', h_2') \in \mathbf{Q}^R_\varepsilon\, w, w'$.

Let us ponder the fact $(k-i, e_1', e_2') \in [\![\mathtt{ref}_\rho\, \tau]\!]^R w'$. By assumption, $R(\rho) \in \mathrm{dom}(w) = \mathrm{dom}(w')$ and so we must have

$$
\begin{aligned}
&[\![\mathtt{T}_\varepsilon \tau]\!]^R w = \Big\{ (k, e_1, e_2) \mid \\
&\quad \forall j \le k.\, \forall e_1' \in \mathcal{E}.\, \forall h_1, h_2, f_1, f_2, g_1' \in \mathcal{H}. \\
&\quad \Big[(k, h_1, h_2) \in \mathbf{P}_\varepsilon^R\, w \wedge \\
&\quad\quad \langle e_1, h_1 \cdot f_1 \rangle \xrightarrow{j} \langle e_1', g_1' \rangle \wedge \mathrm{irr}\langle e_1', g_1' \rangle \Big] \implies \\
&\quad \Big[\exists w' \sqsupseteq w.\, \exists e_2' \in \mathcal{E}.\, \exists g_2', h_1', h_2', f_1', f_2' \in \mathcal{H}. \\
&\quad\quad \mathrm{dom}(w) = \mathrm{dom}(w') \wedge \\
&\quad\quad \langle e_2, h_2 \cdot f_2 \rangle \xrightarrow{*} \langle e_2', g_2' \rangle \wedge \\
&\quad\quad g_1' = h_1' \cdot f_1' \cdot f_1 \wedge g_2' = h_2' \cdot f_2' \cdot f_2 \wedge \\
&\quad\quad (k - j, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^R\, w, w' \wedge \\
&\quad\quad (k - j, e_1', e_2') \in [\![\tau]\!]^R w' \Big] \Big\}
\end{aligned}
$$

$$
\begin{aligned}
&(k, h_1, h_2) \in \mathbf{P}_\varepsilon^R\, w \iff \\
&\quad \mathrm{dom}(h_1) = \mathrm{dom}_1(w) \wedge \mathrm{dom}(h_2) = \mathrm{dom}_2(w) \wedge \\
&\quad \big(\forall \rho \in \mathrm{rds}\, \varepsilon.\, \forall (l_1, l_2, \mu) \in w(R(\rho)). \\
&\quad\quad k > 0 \Rightarrow (k-1, h_1(l_1), h_2(l_2)) \in (\iota\, \mu)(w) \big)
\end{aligned}
$$

$$
\begin{aligned}
&(k, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^R\, w, w' \iff \\
&\quad \mathrm{dom}(h_1) = \mathrm{dom}_1(w) \wedge \mathrm{dom}(h_2) = \mathrm{dom}_2(w) \wedge \\
&\quad \mathrm{dom}(h_1') = \mathrm{dom}_1(w') \wedge \mathrm{dom}(h_2') = \mathrm{dom}_2(w') \wedge \\
&\quad \big(\forall l_1 \in \mathrm{dom}(h_1).\, h_1(l_1) \neq h_1'(l_1) \Rightarrow \\
&\quad\quad \exists \rho \in \mathrm{wrs}\, \varepsilon.\, \exists (m_1, l_2, \mu) \in w(R(\rho)).\, l_1 = m_1 \wedge \\
&\quad\quad\quad k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w') \big) \wedge \\
&\quad \big(\forall l_2 \in \mathrm{dom}(h_2).\, h_2(l_2) \neq h_2'(l_2) \Rightarrow \\
&\quad\quad \exists \rho \in \mathrm{wrs}\, \varepsilon.\, \exists (l_1, m_2, \mu) \in w(R(\rho)) \wedge l_2 = m_2 \wedge \\
&\quad\quad\quad k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w') \big) \wedge \\
&\quad \big(\forall r \in \mathrm{dom}(w).\, \forall (l_1, l_2, \mu) \in w'(r) \setminus w(r).\, r \in R(\mathrm{als}\, \varepsilon) \wedge \\
&\quad\quad k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w') \big)
\end{aligned}
$$

Figure 5. Interpretation of computations with pre- and postconditions. There are implicit disjointness requirements on heaps: the heap compositions must all be well-defined. In all three cases we have $R : \mathcal{RV} \rightharpoonup_{fin} \mathcal{RN}$ injective and $w \in \mathbf{W}$. For $[\![\mathtt{T}_\varepsilon\, \tau]\!]^R w$ we require $\mathrm{FRV}(\varepsilon, \tau) \subseteq \mathrm{dom}(R)$ and $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w)$. $\mathbf{P}_\varepsilon^R\, w$ is defined for $\mathrm{FRV}(\varepsilon) \subseteq \mathrm{dom}(R)$ and $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w)$, $\mathbf{Q}_\varepsilon^R\, w, w'$ additionally requires $w' \in \mathbf{W}$ and $w \sqsubseteq w'$ with $\mathrm{dom}(w) = \mathrm{dom}(w')$.

$$
\begin{aligned}
&\Pi \mid \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon \\
&\quad \iff \\
&\forall k \in \mathbb{N}.\, \forall w \in \mathbf{W}.\, \forall R : \Pi \hookrightarrow |w|.\, \forall \gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}. \\
&\big[R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w) \wedge (k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w \big] \Rightarrow \\
&(k, e_1[\gamma_1/\Gamma], e_2[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\, \tau]\!]^R w.
\end{aligned}
$$

Figure 6. The logical relation. We require $\mathrm{FRV}(\Gamma, \tau, \varepsilon) \subseteq \Pi$ and, as always, $\mathrm{FV}(e_1, e_2) \in \Gamma$.

$l_1, l_2 \in \mathcal{L}$ and $\mu \in \widehat{\mathbf{T}}$ such that $e_1' = l_1$, $e_2' = l_2$, $(l_1, l_2, \mu) \in w'(R(\rho))$ and $[\![\tau]\!]^R w' \overset{k-i}{=} (\iota\, \mu)(w')$. Since $l_1 \in \mathrm{dom}_1(w') = \mathrm{dom}(h_1') \subseteq \mathrm{dom}(g_1')$ we must have $i < j$ since the alternative disproves $\mathrm{irr}\langle e_1'', g_1'' \rangle$. Indeed, we must have $i = j - 1$ and the entire left hand side reduction must look like

$$\langle !\, e_1[\gamma_1/\Gamma], h_1 \cdot f_1 \rangle \xrightarrow{j-1} \langle !\, l_1, g_1' \rangle \to \langle h_1'(l_1), g_1' \rangle,$$

in particular $e_1'' = h_1'(l_1)$ and $g_1'' = g_1'$. Moreover, since

$$\langle e_2[\gamma_2/\Gamma], h_2 \cdot f_2 \rangle \xrightarrow{*} \langle l_2, g_2' \rangle,$$

and $l_2 \in \mathrm{dom}_2(w') = \mathrm{dom}(h_2') \subseteq \mathrm{dom}(g_2')$ we get a similar reduction on the right hand side

$$\langle !\, e_2[\gamma_2/\Gamma], h_2 \cdot f_2 \rangle \xrightarrow{*} \langle !\, l_2, g_2' \rangle \to \langle h_2'(l_2), g_2' \rangle.$$

By Postcondition Weakening we are left to verify only that $(k - j, h_1'(l_1), h_2'(l_2)) \in [\![\tau]\!]^R w'$. Since $[\![\tau]\!]^R w' \overset{k-j+1}{=} (\iota\, \mu)(w')$, it suffices to prove $(k - j, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w')$. We now branch on whether $(l_1, l_2, \mu) \in w(R(\rho))$ holds, i.e., whether locations we look up existed prior to the entire computation or were allocated along the way. If $(l_1, l_2, \mu) \in w'((R(\rho)) \setminus w(R(\rho))$, we get $(k - j, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w')$ directly from $(k - i, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^R\, w, w'$. If, on the other hand, it is the case that $(l_1, l_2, \mu) \in w(R(\rho))$ holds, then we get $(k - 1, h_1(l_1), h_2(l_2)) \in (\iota\, \mu)(w)$ from $\mathbf{P}_{\varepsilon \cup \{rd_\rho\}}^R$ because of the read effect rd_ρ. If the locations are not written during the computation, i.e., if $h_1(l_1) = h_1'(l_1)$ and $h_2(l_2) = h_2'(l_2)$, we get $(k - j, h_1'(l_1), h_2'(l_2)) \in (\iota\, \mu)(w')$ immediately. If, on the other hand, they are written, then we still get this, but this time from $(k - i, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^R\, w, w'$. □

Lemma 6.4 (Masking)**.** $\Pi, \rho \mid \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon$ implies $\Pi \mid \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon - \rho$ provided that $\rho \notin \mathrm{FRV}(\Gamma, \tau)$.

Proof. We unroll the definition of the logical relation: Let $k \in \mathbb{N}$, $w \in \mathbf{W}$, $R : \Pi \hookrightarrow |w|$ and $\gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}$ be arbitrary. Assume $R(\mathrm{FRV}(\varepsilon - \rho)) \subseteq \mathrm{dom}(w)$ and that $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w$. We now must show that

$$(k, e_1[\gamma_1/\Gamma], e_2[\gamma_2/\Gamma]) \in [\![\mathtt{T}_{\varepsilon - \rho}\, \tau]\!]^R w.$$

We proceed to unroll the definition of computations: Let $j \le k$, $e_1' \in \mathcal{E}$ and $h_1, h_2, f_1, f_2, g_1' \in \mathcal{H}$ be arbitrary. Assume that $(k, h_1, h_2) \in \mathbf{P}_{\varepsilon - \rho}^R\, w$, that $\langle e_1, h_1 \cdot f_1 \rangle \xrightarrow{j} \langle e_1', g_1' \rangle$ and that $\mathrm{irr}\langle e_1', g_1' \rangle$. We now have to prove a range of things; to do so we make use of the assumptions of the lemma.

Pick $r \in \mathcal{RN} \setminus |w|$ and define $w' \in \mathbf{W}$ by the transition $w \to_{reg(r)} w'$. Let $R' = R[\rho \to r]$ be the corresponding extension of R. We have $R' : \Pi, \rho \to |w'|$, $R'(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w')$ and $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^{R'} w'$, the latter by Environment Extension and monotonicity of semantic types. But then the assumption of the lemma buys us that

$$(k, e_1[\gamma_1/\Gamma], e_2[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\, \tau]\!]^{R'} w'.$$

Observe, now, that $(k, h_1, h_2) \in \mathbf{P}_\varepsilon^{R'}\, w'$ because of type monotonicity and since $w'(R(\rho)) = \emptyset$. This means, that we have $w'' \sqsupseteq w'$ with $\mathrm{dom}(w'') = \mathrm{dom}(w')$ as well as $e_2' \in \mathcal{E}$, $g_2', h_1', h_2', f_1', f_2' \in \mathcal{H}$ such that $\langle e_2, h_2 \cdot f_2 \rangle \xrightarrow{*} \langle e_2', g_2' \rangle$, $g_1' = h_1' \cdot f_1' \cdot f_1$, $g_2' = h_2' \cdot f_2' \cdot f_2$, $(k - j, e_1', e_2') \in [\![\tau]\!]^{R'} w''$ and $(k - j, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^{R'}\, w', w''$.

Define $w''' \in \mathbf{W}$ by the transition $w'' \to_{mask(r)} w'''$, i.e., we mask out region r. We have $w''' \sqsupseteq w$ by transitivity. Also

$$\mathrm{dom}(w''') = \mathrm{dom}(w'') \setminus \{r\} = \mathrm{dom}(w') \setminus \{r\} = \mathrm{dom}(w).$$

By Environment Extension and monotonicity we get

$$(k-j, e_1', e_2') \in [\![\tau]\!]^{R'} w'' \subseteq [\![\tau]\!]^R w'''.$$

Write $h_1' = h_1^\dagger \cdot f_1^\dagger$ with $h_1^\dagger, f_1^\dagger \in \mathcal{H}$ and $\mathrm{dom}(f_1^\dagger) = \mathrm{dom}_1(w''')$. And similarly, write $h_2' = h_2^\dagger \cdot f_2^\dagger$ with $h_2^\dagger, f_2^\dagger \in \mathcal{H}$ and $\mathrm{dom}(f_2^\dagger) = \mathrm{dom}_2(w''')$. It remains to prove that $(k-j, h_1, h_2, h_1^\dagger, h_2^\dagger) \in \mathbf{Q}^R_{\varepsilon-\rho} w, w'''$; note that we have transferred to the frame whatever locations that were allocated in region r.

Take $l_1 \in \mathrm{dom}(h_1)$ and assume $h_1(l_1) \neq h_1^\dagger(l_1)$. We get $\sigma \in \mathrm{wrs}\,\varepsilon$ and $(m_1, l_2, \mu) \in w'(R(\sigma))$ with $l_1 = m_1$ and

$$k - j > 0 \Rightarrow (k-j-1, h_1'(l_1), h_2'(l_2)) \in (\iota\,\mu)(w'').$$

Since $w'(R(\rho)) = w'(r) = \emptyset$, we conclude $\sigma \neq \rho$ and so $\sigma \in \mathrm{wrs}\,\varepsilon - \rho$ and $(l_1, l_2, \mu) \in w(R(\sigma))$; all that remains is an application of type monotonicity. Writes to the right hand side heap is treated similarly. Assume, finally, that there is $s \in \mathrm{dom}(w)$ and $(l_1, l_2, \mu) \in w'''(s) \setminus w(s)$. Since $s \neq r$ we have $(l_1, l_2, \mu) \in w''(s) \setminus w'(s)$ and so there is $s \in R(\mathrm{als}\,\varepsilon)$ and

$$k - j > 0 \Rightarrow (k-j-1, h_1'(l_1), h_2'(l_2)) \in (\iota\,\mu)(w'').$$

Since we had $s \neq r = R(\rho)$, we must have $s \in R(\mathrm{als}\,\varepsilon - \rho)$ as well; type monotonicity takes us the last way if $k - j > 0$ happens to hold. □

Lemma 6.5 (Fix)**.** $\Pi \mid \Gamma, f : \tau_1 \xrightarrow{\varepsilon} \tau_2, x : \tau_1 \models e_1 \preceq e_2 : \tau_2, \varepsilon$ implies $\Pi \mid \Gamma \models \mathtt{fix}\, f(x).e_1 \preceq \mathtt{fix}\, f(x).e_2 : \tau_1 \xrightarrow{\varepsilon} \tau_2, \emptyset$.

Proof. We unroll the definition of the logical relation: Let $k \in \mathbb{N}$, $w \in \mathbf{W}$, $R : \Pi \hookrightarrow |w|$ and $\gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}$ be arbitrary. The assumption $R(\mathrm{FRV}(\emptyset)) \subseteq \mathrm{dom}(w)$ gives us nothing, but we do get that $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w$. Write $e_1' = \mathtt{fix}\, f(x).e_1[\gamma_1/\Gamma]$ and $e_2' = \mathtt{fix}\, f(x).e_2[\gamma_2/\Gamma]$, we must show that

$$(k, e_1', e_2') \in [\![\mathtt{T}_\emptyset\, \tau_1 \xrightarrow{\varepsilon} \tau_2]\!]^R w.$$

As both expressions are, in fact, values it will suffice to show

$$(k, e_1', e_2') \in [\![\tau_1 \xrightarrow{\varepsilon} \tau_2]\!]^R w.$$

Notice first that $R(\mathrm{FRV}(\varepsilon)) \subseteq |w|$. If $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w)$ fails, then there is nothing left to show. Assume, hence, that the inclusion holds, we aim to prove by induction that for all $0 \leq j \leq k$ we have

$$(j, e_1', e_2') \in [\![\tau_1 \xrightarrow{\varepsilon} \tau_2]\!]^R w.$$

The base case is easy, since there is no termination in 0 steps here. So assume the above for $0 \leq j < k$, we must prove that

$$(j+1, e_1', e_2') \in [\![\tau_1 \xrightarrow{\varepsilon} \tau_2]\!]^R w.$$

is good too. Pick $w' \sqsupseteq w$ with $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w')$. Let $i \leq j+1$ and $(v_1, v_2) \in \mathcal{V} \times \mathcal{V}$ with $(i, v_1, v_2) \in [\![\tau_1]\!]^R w'$ be arbitrary. We must show

$$(i, e_1' v_1, e_2' v_2) \in [\![\mathtt{T}_\varepsilon\, \tau_2]\!]^R w'.$$

So, let $i' \leq i$, $e_1'' \in \mathcal{E}$ and $h_1, h_2, f_1, f_2, g_1'' \in \mathcal{H}$ be arbitrary and assume $(i, h_1, h_2) \in \mathbf{P}^R_\varepsilon w'$ and

$$\langle e_1' v_1, h_1 \cdot f_1 \rangle \xrightarrow{i'} \langle e_1'', g_1'' \rangle$$

with $\mathrm{irr}\langle e_1'', g_1'' \rangle$.

Clearly, $i' > 0$, so the reduction must go like

$$\langle e_1' v_1, h_1 \cdot f_1 \rangle \to \langle e_1[\gamma_1/\Gamma, e_1'/f, v_1/x], h_1 \cdot f_1 \rangle \xrightarrow{i'-1} \langle e_1'', g_1'' \rangle$$

It is now time to make use of the assumption. Since $i' - 1 \leq i - 1 \leq j$, we can apply the induction hypothesis to get

$$(i'-1, e_1[\gamma_1/\Gamma, e_1'/f, v_1/x], e_2[\gamma_2/\Gamma, e_2'/f, v_2/x]) \in [\![\mathtt{T}_\varepsilon\, \tau_2]\!]^R w'.$$

All that is left is gathering the consequences. □

7. Applications

We now show applications of our logical relations model: we verify four effect-based program transformations. These transformations are also considered in [7], but only for a language with ground store.

Theorem 7.1 (Idempotent Computation)**.** A computation with disjoint read and write effects and no allocation effects is idempotent. More precisely, assume that we have

$$\Pi \mid \Gamma \vdash e : \tau, \varepsilon$$

with $\mathrm{rds}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon = \emptyset = \mathrm{als}\,\varepsilon$. Then it holds that

$$\Pi \mid \Gamma \models \mathtt{let\ x\ } = e \mathtt{\ in let\ y\ } = e \mathtt{\ in\ } (\mathtt{x}, \mathtt{y}) \sim \mathtt{let\ x\ } = e \mathtt{\ in\ } (\mathtt{x}, \mathtt{x}) : \tau \times \tau, \varepsilon.$$

Proof. We just prove that the left hand side approximates the right hand side, the other way round proceeds similarly. Let $k \in \mathbb{N}$, $w \in \mathbf{W}$, $R : \Pi \hookrightarrow |w|$ and $\gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}$ be arbitrary. Assume that $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w)$ and that $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w$. We set

$$e_1 = \mathtt{let\ x\ } = e[\gamma_1/\Gamma] \mathtt{\ in let\ y\ } = e[\gamma_1/\Gamma] \mathtt{\ in\ } (\mathtt{x}, \mathtt{y})$$

and

$$e_2 = \mathtt{let\ x\ } = e[\gamma_2/\Gamma] \mathtt{\ in\ } (\mathtt{x}, \mathtt{x})$$

and have to prove $(k, e_1, e_2) \in [\![\mathtt{T}_\varepsilon\, \tau \times \tau]\!]^R w$.

We proceed to unroll the definition of computations: Let $j \leq k$, $e_1''' \in \mathcal{E}$ and $h_1, h_2, f_1, f_2, g_1''' \in \mathcal{H}$ be arbitrary. Assume that $(k, h_1, h_2) \in \mathbf{P}^R_\varepsilon w$, that

$$\langle e_1, h_1 \cdot f_1 \rangle \xrightarrow{j} \langle e_1''', g_1''' \rangle$$

and that $\mathrm{irr}\langle e_1''', g_1''' \rangle$.

By the definition of the operational semantics there must be $0 \leq i \leq j$, $e_1' \in \mathcal{E}$ and $g_1' \in \mathcal{H}$ such that the above reduction can be split into

$$\langle e_1, h_1 \cdot f_1 \rangle \xrightarrow{i} \langle \mathtt{let\ x\ } = e_1' \mathtt{\ in let\ y\ } = e[\gamma_1/\Gamma] \mathtt{\ in\ } (\mathtt{x}, \mathtt{y}), g_1' \rangle \xrightarrow{j-i} \langle e_1''', g_1''' \rangle$$

with

$$\langle e[\gamma_1/\Gamma], h_1 \cdot f_1 \rangle \xrightarrow{i} \langle e_1', g_1' \rangle$$

and $\mathrm{irr}\langle e_1', g_1' \rangle$.

From the Fundamental Lemma we get $(k, e[\gamma_1/\Gamma], e[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\, \tau]\!]^R w$. But then we get $w' \sqsupseteq w$ with $\mathrm{dom}(w) = \mathrm{dom}(w')$ as well as $e_2' \in \mathcal{E}$ and $g_2', h_1', h_2', f_1', f_2' \in \mathcal{H}$ such that

$$\langle e[\gamma_2/\Gamma], h_2 \cdot f_2 \rangle \xrightarrow{*} \langle e_2', g_2' \rangle,$$

$g_1' = h_1' \cdot f_1' \cdot f_1, g_2' = h_2' \cdot f_2' \cdot f_2, (k-i, h_1, h_2, h_1', h_2') \in \mathbf{Q}^R_\varepsilon w, w'$ and $(k-i, e_1', e_2') \in [\![\tau]\!]^R w'$. The latter implies $e_1' \in \mathcal{V}$ and so we must have $i < j$ as the alternative would invalidate $\mathrm{irr}\langle e_1''', g_1''' \rangle$. Indeed, there must be $0 \leq i' \leq j - i - 1$, $e_1'' \in \mathcal{E}$ and $g_1'' \in \mathcal{H}$ such that we can split the last $j - i$ steps further

$$\begin{aligned} &\langle \mathtt{let\ x\ } = e_1' \mathtt{\ in let\ y\ } = e[\gamma_1/\Gamma] \mathtt{\ in\ } (\mathtt{x}, \mathtt{y}), g_1' \rangle \\ &\quad \to \langle \mathtt{let\ y\ } = e[\gamma_1/\Gamma] \mathtt{\ in\ } (e_1', \mathtt{y}), g_1' \rangle \\ &\quad \xrightarrow{i'} \langle \mathtt{let\ y\ } = e_1'' \mathtt{\ in\ } (e_1', \mathtt{y}), g_1'' \rangle \\ &\quad \xrightarrow{j-i-1-i'} \langle e_1''', g_1''' \rangle \end{aligned}$$

with

$$\langle e[\gamma_1/\Gamma], g_1' \rangle \xrightarrow{i'} \langle e_1'', g_1'' \rangle$$

and $\mathrm{irr}\langle e_1'', g_1'' \rangle$.

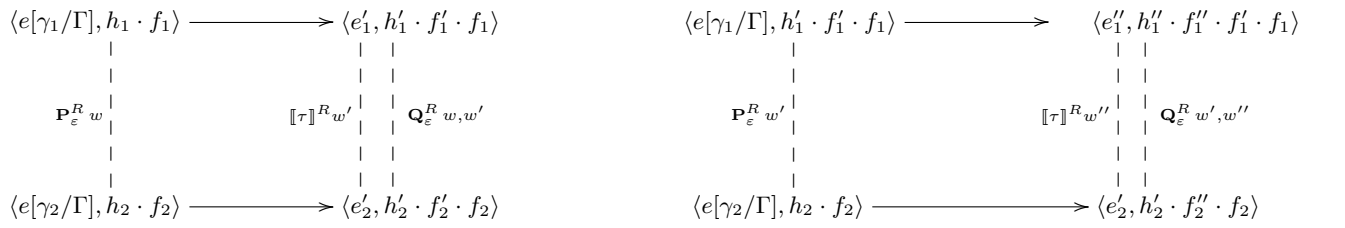

Figure 7. Illustrated proof of the Idempotent Computation Theorem.

Now for something odd: we reset the right hand side to the initial state. More precisely, we argue that $(k-i-1, h_1', h_2) \in \mathbf{P}_\varepsilon^R\, w'$, this is the crux of the entire proof: Notice initially, that not only is it the case that $\mathrm{dom}(w) = \mathrm{dom}(w')$, we also have $\forall r \in \mathrm{dom}(w).\, w(r) = w'(r)$ since $\mathrm{als}\,\varepsilon = \emptyset$; in particular we get $\mathrm{dom}_2(w) = \mathrm{dom}_2(w')$. Combining now the facts

$$\forall \rho \in \mathrm{rds}\,\varepsilon.\, \forall (l_1, l_2, \mu) \in w(R(\rho)).\\ k > 0 \Rightarrow (k-1, h_1(l_1), h_2(l_2)) \in (\iota\,\mu)(w)$$

and

$$\forall l_1 \in \mathrm{dom}(h_1).\, h_1(l_1) \neq h_1'(l_1) \Rightarrow\\ \exists \rho \in \mathrm{wrs}\,\varepsilon.\, \exists (m_1, l_2, \mu) \in w(R(\rho)).\, l_1 = m_1 \wedge\\ k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\,\mu)(w'),$$

with $\mathrm{rds}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon = \emptyset$ buys us

$$\forall \rho \in \mathrm{rds}\,\varepsilon.\, \forall (l_1, l_2, \mu) \in w'(R(\rho)).\\ k-i-1 > 0 \Rightarrow (k-i-2, h_1'(l_1), h_2(l_2)) \in (\iota\,\mu)(w').$$

Loosely speaking, the locations we are permitted to read held values of the correct type from the beginning and were not changed by the computation seen so far.

We proceed to use the fact that $(k-i-1, e[\gamma_1/\Gamma], e[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\,\tau]\!]^R w'$ by a second application of the Fundamental Lemma. This yields $w'' \sqsupseteq w'$ with $\mathrm{dom}(w') = \mathrm{dom}(w'')$ as well as $e_2'' \in \mathcal{E}$ and $g_2'', h_1'', h_2'', f_1'', f_2'' \in \mathcal{H}$ such that

$$\langle e[\gamma_2/\Gamma], h_2 \cdot f_2\rangle \xrightarrow{*} \langle e_2'', g_2''\rangle,$$

$g_1'' = h_1'' \cdot f_1'' \cdot f_1' \cdot f_1$, $g_2'' = h_2'' \cdot f_2'' \cdot f_2$, $(k-i-1-i', h_1', h_2, h_1'', h_2'') \in \mathbf{Q}_\varepsilon^R\, w', w''$ and $(k-i-1-i', e_1'', e_2'') \in [\![\tau]\!]^R w''$. The latter implies $e_1'' \in \mathcal{V}$ and so we must have $i' < j-i-1$ since an equality would conflict with $\mathrm{irr}\langle e_1'', g_1''\rangle$. Even more precisely, we must have $j-i-1-i' = 1$ and the final step in the entire reduction $\langle e_1, h_1 \cdot f_1\rangle \xrightarrow{j} \langle e_1''', g_1'''\rangle$ must be

$$\langle \mathtt{let\ y\ } = e_1''\ \mathtt{in\ } (e_1', \mathtt{y}), g_1''\rangle \to \langle (e_1', e_1''), g_1''\rangle,$$

in particular $e_1''' = (e_1', e_1'')$ and $g_1''' = g_1''$.

We immediately get

$$\langle e_2, h_2 \cdot f_2\rangle \xrightarrow{*} \langle (e_2'', e_2''), g_2''\rangle.$$

In the proof, it now remains to prove $(k-j, h_1, h_2, h_1'', h_2'') \in \mathbf{Q}_\varepsilon^R\, w, w''$ and that $(k-j, (e_1', e_1''), (e_2'', e_2'')) \in [\![\tau]\!]_\varepsilon^R$ w". Notice initially, that by the determinism of the operational semantics and the fact that $e_2', e_2'' \in \mathcal{V}$ we have $e_2' = e_2''$ and $g_2' = g_2''$. Since $\mathrm{dom}_2(w') = \mathrm{dom}_2(w'')$ by $\mathrm{als}\,\varepsilon = \emptyset$ we furthermore get $h_2' = h_2''$ and $f_2' = f_2''$. Also recall $k-j = k-i-i'-2$.

On the first obligation, we take $l_1 \in \mathrm{dom}(h_1)$ and assume that $h_1(l_1) \neq h_1''(l_1)$. If also $h_1'(l_1) \neq h_1''(l_1)$ then the desired follows from $(k-i-1-i', h_1', h_2, h_1'', h_2'') \in \mathbf{Q}_\varepsilon^R\, w', w''$. Otherwise, we must have $h_1(l_1) \neq h_1'(l_1) = h_1''(l_1)$, but then $(k-i, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^R\, w, w'$ paves the way. And the second obligation is met by recalling $(k-i, e_1', e_2') \in [\![\tau]\!]^R w'$ and $(k-i-1-i', e_1'', e_2'') \in [\![\tau]\!]^R w''$. □

Theorem 7.2 (Commuting Computations)**.** Two computations commute if neither reads a region that the other writes, and there is no region they both write. More precisely, assume that we have

$$\Pi \mid \Gamma \vdash e : \tau, \varepsilon, \quad \Pi \mid \Gamma \vdash e' : \tau', \varepsilon'$$

with $\mathrm{rds}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon' = \mathrm{rds}\,\varepsilon' \cap \mathrm{wrs}\,\varepsilon = \mathrm{wrs}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon' = \emptyset$. Then we have the following equivalence:

$$\Pi \mid \Gamma \models \mathtt{let\ x\ } = e\ \mathtt{in\ let\ y\ } = e'\ \mathtt{in\ } (\mathtt{x}, \mathtt{y}) \sim\\ \mathtt{let\ y\ } = e'\ \mathtt{in\ let\ x\ } = e\ \mathtt{in\ } (\mathtt{x}, \mathtt{y}) : \tau \times \tau', \varepsilon \cup \varepsilon'.$$

Proof. Again we give the details only one way: that the left hand side approximates the right hand side. Let $k \in \mathbb{N}$, $w \in \mathbf{W}$, $R : \Pi \hookrightarrow |w|$ and $\gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}$ be arbitrary. Assume that $R(\mathrm{FRV}(\varepsilon \cup \varepsilon')) \subseteq \mathrm{dom}(w)$ and that $(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w$. Set

$$e_1 = \mathtt{let\ x\ } = e[\gamma_1/\Gamma]\ \mathtt{in\ let\ y\ } = e'[\gamma_1/\Gamma]\ \mathtt{in\ } (\mathtt{x}, \mathtt{y})$$

and

$$e_2 = \mathtt{let\ y\ } = e'[\gamma_2/\Gamma]\ \mathtt{in\ let\ x\ } = e[\gamma_2/\Gamma]\ \mathtt{in\ } (\mathtt{x}, \mathtt{y});$$

we have to prove $(k, e_1, e_2) \in [\![\mathtt{T}_{\varepsilon\cup\varepsilon}\,\tau \times \tau']\!]^R w$.

We proceed to unroll the definition of computations: Let $j \leq k$, $e_1''' \in \mathcal{E}$ and $h_1, h_2, f_1, f_2, g_1''' \in \mathcal{H}$ be arbitrary. Assume that $(k, h_1, h_2) \in \mathbf{P}^R_{\varepsilon\cup\varepsilon'}\, w$, that

$$\langle e_1, h_1 \cdot f_1\rangle \xrightarrow{j} \langle e_1''', g_1'''\rangle$$

and that $\mathrm{irr}\langle e_1''', g_1'''\rangle$.

By the definition of the operational semantics there must be $0 \leq i \leq j$, $e_1' \in \mathcal{E}$ and $g_1' \in \mathcal{H}$ such that the above reduction can be split into

$$\langle e_1, h_1 \cdot f_1\rangle \xrightarrow{i} \langle\, \mathtt{let\ x\ } = e_1'\ \mathtt{in\ let\ y\ } = e'[\gamma_1/\Gamma]\ \mathtt{in\ } (\mathtt{x}, \mathtt{y}), g_1'\rangle\\ \xrightarrow{j-i} \langle e_1''', g_1'''\rangle$$

with

$$\langle e[\gamma_1/\Gamma], h_1 \cdot f_1\rangle \xrightarrow{i} \langle e_1', g_1'\rangle$$

and $\mathrm{irr}\langle e_1', g_1'\rangle$.

From the Fundamental Lemma we get $(k, e[\gamma_1/\Gamma], e[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\,\tau]\!]^R w$. By Precondition Strengthening we get a whole range of stuff, but we need only the following two facts: that $e_1' \in \mathcal{V}$ and that for $l_1 \in \mathrm{dom}(h_1)$ such that there is $r \in R(\mathrm{rds}\,\varepsilon')$ and $(m_1, l_2, \mu) \in w(r)$ with $m_1 = l_1$ we have $g_1'(l_1) = h_1(l_1)$. The latter is a consequence of $\mathrm{wrs}\,\varepsilon \cap \mathrm{rds}\,\varepsilon' = \emptyset$.

We must have $i < j$ as the alternative would invalidate $\mathrm{irr}\langle e_1''', g_1'''\rangle$. Indeed, there must be $0 \leq i' \leq j-i-1$, $e_1'' \in \mathcal{E}$

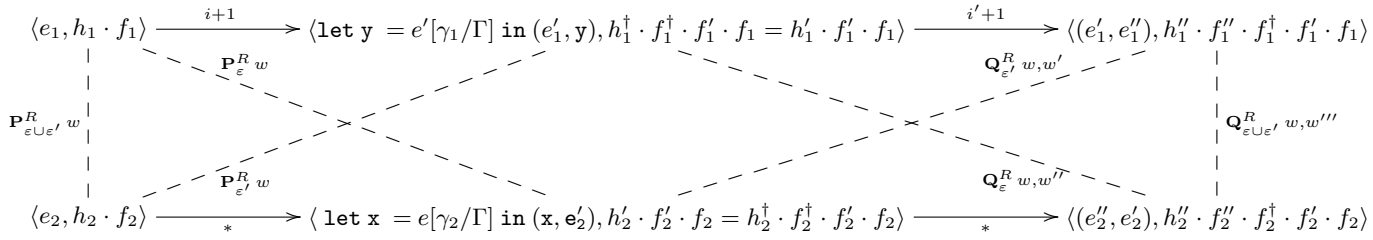

Figure 8. Illustrated proof of the Commuting Computations Theorem. Most of the dashed lines are slightly off, i.e., they do not point at the exact right subheaps.

and $g_1'' \in \mathcal{H}$ such that we can split the last $j - i$ steps further into

$$\begin{aligned}&\langle \texttt{let x } = e_1' \texttt{ in let y } = e'[\gamma_1/\Gamma] \texttt{ in } (\texttt{x}, \texttt{y}), g_1'\rangle \\ &\quad \to \langle \texttt{let y } = e'[\gamma_1/\Gamma] \texttt{ in } (e_1', \texttt{y}), g_1'\rangle \\ &\quad \xrightarrow{i'} \langle \texttt{let y } = e_1'' \texttt{ in } (e_1', \texttt{y}), g_1''\rangle \\ &\quad \overset{j-i-1-i'}{\to} \langle e_1''', g_1'''\rangle\end{aligned}$$

with

$$\langle e'[\gamma_1/\Gamma], g_1'\rangle \xrightarrow{i'} \langle e_1'', g_1''\rangle$$

and $\operatorname{irr}\langle e_1'', g_1''\rangle$.

Much as in the previous proof, we now ditch our right hand side progress, but unlike that, we also lose our new future world. Notice first that the Fundamental Lemma gives us $(k - i - 1, e'[\gamma_1/\Gamma], e'[\gamma_2/\Gamma]) \in [\![\mathtt{T}_{\varepsilon'}\, \tau']\!]^R w$, we would like to apply that. Write $g_1' = h_1^\dagger \cdot h_1^\circ$ for $h_1^\dagger, h_1^\circ \in \mathcal{H}$ with $\operatorname{dom}(h_1^\dagger) = \operatorname{dom}_1(w)$; the crucial observation now is that we have $(k-i-1, h_1^\dagger, h_2) \in \mathbf{P}^R_{\varepsilon'}\, w$. Hence we get $w' \sqsupseteq w$ with $\operatorname{dom}(w) = \operatorname{dom}(w')$ as well as $e_2' \in \mathcal{E}$ and $g_2', h_1'', h_2', f_1'', f_2' \in \mathcal{H}$ such that

$$\langle e'[\gamma_2/\Gamma], h_2 \cdot f_2\rangle \xrightarrow{*} \langle e_2', g_2'\rangle,$$

$g_1'' = h_1'' \cdot f_1'' \cdot f_1^\circ, g_2' = h_2' \cdot f_2' \cdot f_2, (k-i-1-i', h_1^\dagger, h_2, h_1'', h_2') \in \mathbf{Q}^R_{\varepsilon'}\, w, w'$ and $(k - i - 1 - i', e_1'', e_2') \in [\![\tau']\!]^R w'$.

Much as we have argued before, we now know that $e_1'' \in \mathcal{V}$, hence $j - i - 1 - i' = 1$ and the entire left hand side reduction must look like

$$\langle e_1, h_1 \cdot f_1\rangle \xrightarrow{j} \langle (e_1', e_1''), g_1''\rangle,$$

in particular, $e_1''' = (e_1', e_1'')$ and $g_1''' = g_1''$.

We have now, in some sense, considered the situation from the point of view of e', it is time to turn the tables. There are $h_2^\dagger, f_2^\dagger$ with $\operatorname{dom}(h_2^\dagger) = \operatorname{dom}_2(w)$ such that $h_2' = h_2^\dagger \cdot f_2^\dagger$. So we have $(k, h_1, h_2^\dagger) \in \mathbf{P}^R_\varepsilon\, w$ by arguments as above, in particular we apply $\operatorname{wrs}\varepsilon' \cap \operatorname{rds}\varepsilon = \emptyset$. Now, we still have $(k, e[\gamma_1/\Gamma], e[\gamma_2/\Gamma]) \in [\![\mathtt{T}_\varepsilon\, \tau]\!]^R w$ and so there is $w'' \sqsupseteq w$ with $\operatorname{dom}(w) = \operatorname{dom}(w'')$ as well as $e_2'' \in \mathcal{E}$ and $g_2'', h_1', h_2'', f_1', f_2'' \in \mathcal{H}$ such that

$$\langle e[\gamma_2/\Gamma], h_2^\dagger \cdot f_2^\dagger \cdot f_2' \cdot f_2\rangle \xrightarrow{*} \langle e_2'', g_2''\rangle,$$

$g_1' = h_1' \cdot f_1' \cdot f_1, g_2'' = h_2'' \cdot f_2'' \cdot f_2^\dagger \cdot f_2' \cdot f_2, (k-i, h_1, h_2^\dagger, h_1', h_2'') \in \mathbf{Q}^R_\varepsilon\, w, w''$ and $(k - i, e_1', e_2'') \in [\![\tau]\!]^R w^\dagger$. Write $h_1' = h_1^\dagger \cdot f_1^\dagger$ for $f_1^\dagger \in \mathcal{H}$, in particular $g_1' = h_1^\dagger \cdot f_1^\dagger \cdot f_1' \cdot f_1$ and $g_1'' = h_1'' \cdot f_1'' \cdot f_1^\dagger \cdot f_1' \cdot f_1$; this is notation we need soon. Observe first, though, that we have the right hand side reduction

$$\langle e_2, h_2 \cdot f_2\rangle \xrightarrow{*} \langle (e_2', e_2''), g_2''\rangle.$$

We now build a world w''' with $\operatorname{dom}(w''') = \operatorname{dom}(w)$ and with both $w''' \sqsupseteq w'$ and $w''' \sqsupseteq w''$, i.e., a common future world. The natural choice is for the dead regions of w''' to be the dead regions of both w' and w''. For $r \in \operatorname{dom}(w)$ we set $w'''(r) = w'(r) \cup w''(r)$, but we must take care not to wreck the partial bijections nor their mutual disjointness: Let $r, s \in \operatorname{dom}(w)$ and take $(l_1', l_2', \mu') \in w'(r) \setminus w(r)$ and $(l_1'', l_2'', \mu'') \in w''(s) \setminus w(s)$; it will suffice to show $l_1' \neq l_1''$ and $l_2' \neq l_2''$. Now, we know that $l_1' \in \operatorname{dom}_1(w') = \operatorname{dom}(h_1'')$, in particular we have $l_1' \notin \operatorname{dom}(f_1^\dagger)$. Also $l_1' \notin \operatorname{dom}_1(w) = \operatorname{dom}(h_1^\dagger)$. But $l_1'' \in \operatorname{dom}_1(w'') = \operatorname{dom}(h_1')$ and since $h_1' = h_1^\dagger \cdot f_1^\dagger$ we must have $l_1' \neq l_1''$. Proving $l_2' \neq l_2''$ proceeds similarly.

It shall now suffice to show

$$(k - j, h_1, h_2, h_1'' \cdot f_1^\dagger, h_2'' \cdot f_2^\dagger) \in \mathbf{Q}^R_{\varepsilon\cup\varepsilon'}\, w, w'''$$

as the remaining obligations are discharged already. We have

$$\begin{aligned}\operatorname{dom}_1(w''') &= \operatorname{dom}_1(w') \cup \operatorname{dom}_1(w'') \\ &= \operatorname{dom}(h_1'') \cup \operatorname{dom}(h_1^\dagger \cdot f_1^\dagger) \\ &= \operatorname{dom}(h_1'' \cdot f_1^\dagger)\end{aligned}$$

and get $\operatorname{dom}_2(w''') = \operatorname{dom}(h_2'' \cdot f_2^\dagger)$ similarly. So take $l_1 \in \operatorname{dom}(h_1)$ and assume that $h_1(l_1) \neq h_1''(l_1)$. If $h_1^\dagger(l_1) \neq h_1''(l_1)$ we get $\rho \in \operatorname{wrs}\varepsilon'$ and $(m_1, l_2, \mu) \in w(R(\rho))$ with $l_1 = m_1$ and $(k - j, h_1''(l_1), h_2'(l_2)) \in (\iota\, \mu)(w')$. And since $\operatorname{wrs}\varepsilon \cap \operatorname{wrs}\varepsilon = \emptyset$ we know that $h_2''(l_2) = h_2'(l_1)$. If, on the other hand, $h_1^\dagger(l_1) = h_1''(l_1)$ then we must have $h_1(l_1) \neq h_1^\dagger(l_1)$ and we get $\rho \in \operatorname{wrs}\varepsilon$ and $(m_1, l_2, \mu) \in w(R(\rho))$ with $l_1 = m_1$ and $(k - i - 1, h_1^\dagger(l_1), h_2''(l_2)) \in (\iota\, \mu)(w'')$.

It remains to consider allocation. So take $r \in \operatorname{dom}(w)$ and $(l_1, l_2, \mu) \in w'''(r) \setminus w(r)$. By the construction of w''' we get $(l_1, l_2, \mu) \in w'(r) \setminus w(r)$ or $(l_1, l_2, \mu) \in w'(r) \setminus w(r)$; in both cases we proceed similarly, so assume the former holds. Then we know $r \in R(\operatorname{als}\varepsilon')$ and that $(k - j, h_1''(l_1), h_2'(l_2)) \in (\iota\, \mu)(w')$. But as $l_2 \notin \operatorname{dom}_2(w)$ we must have $l_2 \in \operatorname{dom}(f_2^\dagger)$ and so $(h_2'' \cdot f_2^\dagger)(l_2) = f_2^\dagger(l_2) = h_2'(l_2)$ and we are done. □

In the remaining two applications we can prove only approximation, not equivalence, because of the possibility of non-termination. We defer the proofs to the appendix of the long version of the paper.

Theorem 7.3 (Neutral Computation)**.** A computation that performs no writes and has return type unit approximates the trivial computation. More precisely, having

$$\Pi \mid \Gamma \vdash e : \mathtt{1}, \varepsilon$$

with $\operatorname{wrs}\varepsilon = \emptyset$ implies

$$\Pi \mid \Gamma \models e \preceq () : \mathtt{1}, \varepsilon.$$

Theorem 7.4 (Pure Lambda Hoist)**.** A pure computation evaluated as part of a function can, up to approximation, be evaluated once and the result cached. More precisely, having

$$\Pi \mid \Gamma \vdash e : \tau_1, \emptyset, \quad \Pi \mid \Gamma, y : \tau_2, x : \tau_1 \vdash e' : \tau_3, \varepsilon.$$

gives us

$$\Pi \mid \Gamma \models \texttt{let x} = e \texttt{ in } \lambda y.\, e' \preceq \lambda y.\, \texttt{let x} = e \texttt{ in } e' : \tau_2 \xrightarrow{\varepsilon} \tau_3, \emptyset$$

8. Discussion

8.1 Work by Benton et. al

An important point of reference for our work is the relational model by Benton et. al. [7] of an effect system for a higher-order langauge with dynamic allocation and ground store, i.e., only integers in the heap. Indeed, apart from our extension to higher-order store, the type systems and examples considered are roughly the same.

Having said so, we remark that our take on the issue of masking is novel; in particular it is different from that of Benton et. al. Their approach does not scale easily, if at all, to the higher-order store setting: The pivot is the *Masking Lemma* [7, Lemma 3], stating that the interpretation of both types and computations in a world are preserved up to equality under masking, *provided* that the region masked out is not, syntactically, in the type respectively in the computation. Combined with ground store, this makes short work of soundness of the masking rule.

The Masking Lemma, however, does not scale easily to a higher-order store setting. Consider the computation from the introduction: The returned function has latent effect $\{wr_\sigma\}$ and correspondingly writes to location 1 in region σ. But the values stored at location 1 must be of type $\texttt{ref}_\rho\,\texttt{int}$, and that depends on the type $\texttt{int}$ associated with location 0 in region ρ. In other words, the interpretation of a type may depend on regions that do not occur syntactically in the type; this is the antithesis to the Masking Lemma.

As described above, we take the different approach that interpretations of types should grow (or at least not shrink) under any application of masking, and only when we actually perform reads, writes and allocations do we require that the regions in question are still live. This also means that we can get by without a *silent region* [7, Sections 5 and 6]. This is a designated region of the world that tracks inaccessible parts of the heap in an untyped way; in *loc.cit.* it is necessary for the Masking Lemma to hold for computations.

That we have no silent region is not just a technical convenience: it means that our model permits *any* action on locations that have been masked out, including garbage collection or ownership transfer. The locations have left the world and we make no further assumptions on them, whatsoever. By contrast, locations in a silent region are still in the world and computations may assume that they remain allocated, even if the stored value cannot be changed. Indeed, computations may actually access such locations in extensionally invisible ways: we could, say, read a location in the silent region as long as we make no use of the read value.

Benton et. al.: Higher Order Store In more recent work, Benton et. al. [8] have given a relational model for a language with higher order store. The language has no dynamic allocation though, nor is there any masking rule.

Unlike our approach, however, they work with a denotational semantics of the language; in particular they prove existence of the logical relation by solving a non-trivial mixed-variance domain equation, extending the standard technique by Pitts [20] to do so. One advantage is transitivity of their logical relation; that is something we do not get. This is a general issue with step-indexed models observed first by Ahmed [3]; there are fixes, but a more proper take is to reason in a logic suited for step-indexing as done, e.g., by Dreyer et. al. [13]. We are convinced, however, that using the techniques developed by Støvring and the authors [11], we could also define a model based on a denotational semantics of the programming language, hence achieving transitivity.

Preservation of All Store Relations Throughout their work, Benton et. al. interpret computations by requiring preservation of *all* relations on heaps that respect the effects of the computations, i.e., that ensure well-typed reads at locations with read effects and is closed under well-typed writes to locations with write effects. Our approach is more simple-minded; we do, in some sense, just preserve one relation. But it is unclear what the additional relations buys, we know of no equivalences that fail due to our approach; indeed the two alternatives may very well be equivalent. What is clear, however, is that losing the many relations, as well as the absence of a silent region, simplify proofs considerably: writing out all details, as we do, obviously generates some mileage but in essence one may argue in simple diagrams as illustrated.

8.2 On Expressiveness

Consider the following variant of the so-called awkward example, discussed in detail in [14]. Let $\tau = (\texttt{1} \xrightarrow{\varepsilon_0} \texttt{1}) \xrightarrow{\varepsilon_1} \texttt{int}$ with $\varepsilon_0 = \{rd_\rho, wr_\rho\}$ and $\varepsilon_1 = \{rd_\rho, wr_\rho, rd_\sigma, wr_\sigma\}$ and define expressions e_1 and e_2, which both have type τ and effect $\{al_\sigma\}$, by

$$\begin{aligned} e_1 &= \texttt{let x} = \texttt{ref}_\sigma\ \texttt{0 in} \\ &\quad\ \lambda f.(x := 0; f(); x := 1; f(); !\,x) \\ e_2 &= \texttt{let x} = \texttt{ref}_\sigma\ \texttt{0 in} \\ &\quad\ \lambda f.(f(); f(); x := 1; !\,x). \end{aligned}$$

We have used the subscript σ to indicate that x will have the type $\texttt{ref}_\sigma\,\texttt{int}$. Since ρ and σ are distinct, f cannot read / write the reference bound to x; this ensures that both the left and the right hand side functions always return 1. Indeed we can show that e_1 and e_2 are contextually equivalent. We remark that if, on the other hand, we had typed e_1 and e_2 with the same region variable instead of two distinct region variables, then our semantic model would not be expressive enough to show that e_1 and e_2 are contextually equivalent. For that, we could extend our model using ideas from the model for non-effect-annotated types in [14]. In summa, effect information can be used to restrict the applicable contexts and can thus make it easier to show two expressions equivalent (for those restricted contexts). Note that a standard unification-based algorithm for inferring effects, would infer the type τ for e_1 with ρ and σ distinct.

9. Conclusion and Future Work

We have presented a solution to the open problem of constructing a relational model for an effect system for a higher-order language with dynamically allocated higher-order store. We have demonstrated that the model can be used to rigorously justify effect-based program transformations.

Future work includes extending the model to region and effect polymorphism, as found, e.g., in [21]. Moreover, we think our model could be adapted to a situation where the conceptual regions do not necessarily follow a stack-discipline [4, 16] by a slight modification of the interpretation of computations. In this paper we have focused on a relational model for verifying effect-based program transformations. We believe that the ideas of our model construction can also be used to construct models for other applications of effect systems for languages with higher-order store.

Acknowledgments

We thank Nick Benton, Lennart Beringer, Martin Hofmann and Andrew Kennedy for useful discussions. Kasper Svendsen and Filip Sieczkowski gave useful comments on a late draft of the paper. This research was funded in part by the ToMeSo project from the Danish Agency for Science, Technology and Innovation (grant number 274–08–0412).

References

[1] A. Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, 2004.

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

[3] A. J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In P. Sestoft, editor, *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006. ISBN 3-540-33095-X.

[4] A. Aiken, M. Fähndrich, and R. Levien. Better static memory management: Improving region-based analysis of higher-order languages. In *Prcoeedings of PLDI*, 1995.

[5] N. Benton and P. Buchlovsky. Semantics of an effect analysis for exceptions. In *Proceedings of TLDI*, 2007.

[6] N. Benton, A. Kenney, M. Hofmann, and L. Beringer. Reading, writing and relations: Towards extensional semantics for effect analyses. In *Proceedings of APLAS*, 2006.

[7] N. Benton, L. Beringer, M. Hofmann, and A. Kennedy. Relational semantics for effect-based program transformations with dynamic allocation. In *Proceedings of PPDP*. ACM, 2007.

[8] N. Benton, L. Beringer, M. Hofmann, and A. Kennedy. Relational semantics for effect-based program transformations: Higher-order store. In *Proceedings of PPDP*. ACM, 2009.

[9] L. Birkedal, M. Tofte, and M. Vejlstrup. From region inference to von Neumann machines via region representation inference. In *Proceedings of POPL*, 1996.

[10] L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules for algol-like langauges. *Logical Methods in Computer Science*, 2(5:1):1–33, 2006.

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

[12] L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *Proceedings of POPL*, pages 119–132, 2011.

[13] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *LICS*, pages 71–80. IEEE Computer Society, 2009. ISBN 978-0-7695-3746-7.

[14] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proceedings of ICFP*, 2010.

[15] D. Gifford and J. Lucassen. Integrating functional and imperative programming. In *ACM Conference of LISP and Functional Programming*, 1986.

[16] F. Henglein, M. Makholm, and H. Niss. A direct approach to control-flow sensitive region-based memory management. In *Prcoeedings of PPDP*, 2001.

[17] F. Henglein, H. Makholm, and H. Niss. Effect types and region-based memory management. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*. MIT Press, 2005.

[18] J. Lucassen and D. Gifford. Polymorphic effect systems. In *Proceedings of POPL*, 1988.

[19] A. Nanevski, G. Morrisett, and L. Birkedal. Polymorphism and separation in hoare type theory. In J. H. Reppy and J. L. Lawall, editors, *ICFP*, pages 62–73. ACM, 2006. ISBN 1-59593-309-3.

[20] A. M. Pitts. Relational properties of domains. *Inf. Comput.*, 127(2): 66–90, 1996.

[21] M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *Proceedings of POPL*, 1994.