Relational Parametricity for References and Recursive Types

Lars Birkedal   Kristian Støvring   Jacob Thamsborg
IT University of Copenhagen
{birkedal,kss,thamsborg}@itu.dk

Abstract

We present a possible world semantics for a call-by-value higher-order programming language with impredicative polymorphism, general references, and recursive types. The model is one of the first relationally parametric models of a programming language with all these features.

To model impredicative polymorphism we define the semantics of types via parameterized (world-indexed) logical relations over a universal domain. It is well-known that it is non-trivial to show the existence of logical relations in the presence of recursive types. Here the problems are exacerbated because of general references. We explain what the problems are and present our solution, which makes use of a novel approach to modeling references. We prove that the resulting semantics is adequate with respect to a standard operational semantics and include simple examples of reasoning about contextual equivalence via parametricity.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Denotational semantics; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords Denotational Semantics, Possible World Semantics, Relational Parametricity, Impredicative Polymorphism, General References, Recursive Types

1. Introduction

Relational parametricity was proposed by Reynolds [34] to reason about polymorphic programs, in particular, to show equivalence of polymorphic programs and to show representation independence for abstract data types. In this paper we provide one of the first[1] relationally parametric models of a programming language with recursive types and general references. We prove that the resulting semantics is adequate with respect to a standard operational semantics, which means that we can use parametricity to show contextual equivalence of expressions in the language.

Our model is based on logical relations over an untyped model of the language. The logical relations are parameterized over possible worlds which are used to capture dynamic allocation of references, much as in [7, 11, 22, 31]. It is well-known that it is non-trivial to show the existence of logical relations in the presence of recursive types [28]. Here the problems are exacerbated because of general references. We explain the problems and present our solution, which makes use of a novel approach to modeling references.

In this paper we focus on the challenge of defining an adequate semantics, in particular on the challenge pertaining to the existence of the logical relations. The resulting model can be used to prove equivalence and parametricity results for programs using references in simple ways. In future work, we plan to extend the parameters of our logical relations to accommodate local relational reasoning about programs using local state. We plan to do this using the first author's earlier work on relational reasoning for languages with references and recursive types (but not polymorphism) [11].

1.1 Background

The theory of relational parametricity was originally proposed in the setting of the second-order lambda calculus. That setting is by now fairly well-understood, see, e.g., [9, 33]. But, of course, we would like to use relational parametricity for real programs with recursion and other effects. There has been a lot of research towards this goal — the efforts can be grouped roughly into two categories: *equational type theories with effects* and *programming languages with effects*.

Work in the former category was initiated by Plotkin [32], who suggested a second-order linear type theory with a polymorphic fixed-point combinator to combine polymorphism with recursion. That approach was further investigated in [10]. One of the remarkable features of this calculus is that it allows one to encode a wide range of data types, including recursive types, with the desired universal properties following from parametricity. Hasegawa studied the combination of polymorphism and another effect, namely control [15]. Recently, this line of work was extended by Møgelberg and Simpson [25], who proposed a general polymorphic type theory for effects, as captured by computational monads. The general framework has been specialized to control effects in [26].

Work in the latter category focuses on programming languages defined using an operational semantics, specifying evaluation order, etc., and was initiated by Wadler [37]. Relational parametricity is concerned with program equivalence which is here typically defined as *contextual equivalence*: two program expressions are equivalent if they have the same observable behaviour when placed in any program context C.

It is generally quite hard to show directly that two program expressions are contextually equivalent because of the universal quantification over all contexts. Thus there has been an extensive research effort to find reasoning methods that are easier to use for establishing contextual equivalence (see, e.g., [30] for a fairly recent overview), and the work on parametricity for programming languages with effects has been closely related to the research on reasoning methods for contextual equivalence. Relationally para-

[1] Independent work [3] by Ahmed, Dreyer and Rossberg came to our attention after writing this paper, cf. section 6; we know of no other models.

metric models have been developed for languages with recursion and inductive / coinductive types, see, e.g., [8, 16, 17, 29] and, recently, also for languages with recursive types [2, 13, 23]. In addition, a number of bisimulation-based methods for proving contextual equivalence have recently been proposed; the methods most relevant for the work in this paper cover a pure language with recursive and existential types [36], untyped languages with general references and/or control operators [18, 19, 35], and a pure language with parametric polymorphism and recursive types [20, 21].

The two categories of work are, of course, related: the type theories serve as metalanguages and can be used to give semantics to programming languages. This has, e.g., been done by Møgelberg [24], who showed how to give a parametric model of the programming language FPC extended with polymorphism (i.e., a language with recursion, recursive types and polymorphism). Using a model of the type theory, adequacy wrt. the operational semantics of the programming language was proved, allowing Møgelberg to prove results about contextual equivalence using the reasoning principles of the type theory.

1.2 Overview of the technical development

In Section 2 we define the operational semantics of our programming language, which is a standard, direct-style, call-by-value higher-order language with impredicative polymorphism, recursive types, and references. The operational semantics is nondeterministic since dynamic allocation of references is modeled in the standard way via a nondeterministic choice of a new location.

In Section 3 we present an untyped denotational semantics of the language using a universal domain. In the denotational semantics we assume that the semantic set of locations is well-ordered (the set of locations is a copy of the natural numbers) and allocation is modeled by choosing the smallest free location. We use a novel form of semantic locations in the semantics; the motivation for these comes from the need to establish the existence of logical relations in the following section.

We prove that the denotational semantics is sound and adequate with respect to the operational semantics. This is done *almost* in the standard way by defining a logical formal approximation relation between the operational and denotational semantics. For the adequacy proof it suffices to give a logical relation for *closed* types and therefore, as we show, the existence of the logical relation can be proved using standard techniques [28]. The adverb 'almost' above refers to the following. For the existence proof one needs to show that the relations are suitably admissible and the standard proofs of that rely on determinacy of the operational semantics, see [28, Sec. 5, Page 81], but here we have a non-deterministic operational semantics. Intuitively, the denotational semantics should be adequate since the choice of new location should not matter for the final result of a program. In earlier domain-theoretic models of references for which adequacy have been proved [7, 11, 22], both language and denotational semantics have been defined in a monadic (continuation-passing) style; hence it was fairly easy to capture that the choice of location does not matter for the final result. Here we decide to stick to a direct-style language and operational semantics to make sure that our results do not depend on a monadic presentation and instead we define the logical relation in a continuation-passing style, which suffices for proving adequacy. In summary, the language is in direct style, but the proof of adequacy is in continuation-passing style.

The untyped semantics can only be used to establish simple forms of contextual equivalence. In Section 4 we therefore present a typed possible world semantics of the language by defining a family of parameterized logical relations over the universal domain for which we prove the fundamental theorem of logical relations. In combination with adequacy of the untyped semantics this proves adequacy for the typed semantics. To reason about parametricity we need to give a semantics not only of closed types but also of *open* types. This turns out to complicate the existence proof of the logical relation because, loosely speaking, we need to compare semantic types in the logical relation for reference types in order to check that the type for the location in the current world (a store typing) agrees with the type of the reference. We solve this problem by modeling references using a novel semantic notion of location which permits approximations to locations. The approximations are crucial for the existence proof of the logical relation. We explain what the problem is by highlighting what goes wrong if we omit such approximations.

In Section 5 we present a few examples of equivalences that can be proved using the resulting possible world semantics.

Finally, in Section 6 we conclude and briefly discuss directions for future work.

For space reasons, parts of the definitions and proofs have been omitted. A longer version of the paper is available at:

```
http://www.itu.dk/people/thamsborg/longshadow.pdf
```

2. Types and Operational Semantics

Types, expressions and values are given in Figure 1. A *context of type variables* is a list of type variables with no repeats. For any such context Ξ and any type τ we write $\Xi \vdash \tau$ if the free type variables of τ are all in Ξ and we write $\mathbf{Type}_\Xi$ for all such types. A *world* is a partial map with finite domain from $\mathbb{N}$ to the set of types; we have a partial ordering on worlds defined by setting $\Delta \sqsubseteq \Delta'$ provided $\mathrm{dom}(\Delta) \subset \mathrm{dom}(\Delta')$ and $\Delta(l) = \Delta'(l)$ for all $l \in \mathrm{dom}(\Delta)$. A *context of term variables* is a partial map with finite domain from the set of term variables to the set of types. For any context of type variables Ξ and any world Δ we write $\Xi \vdash \Delta$ if $\Xi \vdash \Delta(l)$ for all $l \in \mathrm{dom}(\Delta)$ and we let $\mathbf{World}_\Xi$ be the set of worlds with this property. We define $\Xi \vdash \Gamma$ for a context of term variables Γ similarly.

We give selected typing rules in Figure 2, a complete presentation is found in the long version of the paper. The rules assign types to expressions under assumptions of contexts of type variables, worlds and contexts of term variables. It is not hard to see that the various side conditions ensure that $\Xi \mid \Delta \mid \Gamma \vdash e : \tau$ implies $\Xi \vdash \Delta$, $\Xi \vdash \Gamma$ as well as $\Xi \vdash \tau$. Also it is worth noticing that the language is explicitly typed to ensure type uniqueness: Given $\Xi \mid \Delta \mid \Gamma \vdash e : \tau_1$ and $\Xi \mid \Delta \mid \Gamma \vdash e : \tau_2$ we can conclude that $\tau_1 = \tau_2$ and that the derivations of the judgments coincide.

As usual we identify expressions up to α-equivalence. For convenience we write $\lambda^{\tau_0 \to \tau_1} x.\ e$ for $\mathtt{fix}^{\tau_0 \to \tau_1} f(x).e$ where f is some arbitrary variable not occurring free in e.

A *syntactic store* is a partial map with finite domain from $\mathbb{N}$ to the set of values. Using that definition, we define a standard big-step *operational semantics*; selected rules are given in Figure 3, see the long version of the paper for the unabridged story. It is a quaternary relation between syntactic stores and expressions on the one hand and syntactic stores and values on the other. Notice that the memory allocator is nondeterministic in the standard way: Any free location may be picked.

For a context Ξ, a world Δ, a context Γ and a syntactic store Π we write $\Xi \mid \Delta \mid \Gamma \vdash \Pi$ to denote that $\mathrm{dom}(\Delta) = \mathrm{dom}(\Pi)$ and that for all $l \in \mathrm{dom}(\Delta)$ we have $\Xi \mid \Delta \mid \Gamma \vdash \Pi(l) : \Delta(l)$. We have the following standard proposition (see Chapter 13 of Pierce [27]):

Proposition 1 (Type Preservation)**.** *Assume $\Pi, e \Downarrow \Pi', v$. Suppose furthermore that we have $\emptyset \mid \Delta \mid \emptyset \vdash \Pi$ and $\emptyset \mid \Delta \mid \emptyset \vdash e : \tau$ for some world Δ and some type τ. Then there is $\Delta' \sqsupseteq \Delta$ such that $\emptyset \mid \Delta' \mid \emptyset \vdash \Pi'$ and $\emptyset \mid \Delta' \mid \emptyset \vdash v : \tau$.*

$$\tau ::= \alpha \mid \mathtt{unit} \mid \mathtt{int} \mid \tau\ \mathtt{ref} \mid \tau \times \tau \mid \tau + \tau \mid \mu\alpha.\tau \mid \forall\alpha.\tau \mid \tau \to \tau$$

$$\begin{aligned} e ::= {} & x \mid () \mid n \mid l \mid \mathtt{op}(e \pm e) \mid \mathtt{ifzero}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e \mid (e, e) \mid \mathtt{fst}(e) \mid \mathtt{snd}(e) \mid \mathtt{inl}^{\tau_0+\tau_1}(e) \mid \\ & \mathtt{inr}^{\tau_0+\tau_1}(e) \mid \mathtt{case}\ e\ \mathtt{of}\ \mathtt{inl}(x).\, e\ \mathtt{else}\ \mathtt{inr}(x).\, e \mid \mathtt{fold}^{\mu\alpha.\tau}(e) \mid \mathtt{unfold}^{\mu\alpha.\tau}(e) \mid \\ & \Lambda\alpha.e \mid e[\tau] \mid \mathtt{fix}^{\tau_0\to\tau_1} f(x).e \mid e(e) \mid \mathtt{ref}(e) \mid\ !\,e \mid e := e \end{aligned}$$

$$v ::= () \mid n \mid l \mid (v, v) \mid \mathtt{inl}^{\tau_0+\tau_1}(v) \mid \mathtt{inr}^{\tau_0+\tau_1}(v) \mid \mathtt{fold}^{\mu\alpha.\tau}(v) \mid \Lambda\alpha.e \mid \mathtt{fix}^{\tau_0\to\tau_1} f(x).e$$

Figure 1. Types, expressions and values.

$$\frac{}{\Xi \mid \Delta \mid \Gamma \vdash l : \tau\ \mathtt{ref}}\ (\Xi \vdash \Delta, \Xi \vdash \Gamma, l \in \mathrm{dom}(\Delta), \Delta(l) = \tau)$$

$$\frac{\Xi, \alpha \mid \Delta \mid \Gamma \vdash e : \tau}{\Xi \mid \Delta \mid \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}\ (\Xi \vdash \Delta, \Xi \vdash \Gamma)$$

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e : \forall\alpha.\tau_0}{\Xi \mid \Delta \mid \Gamma \vdash e[\tau_1] : \tau_0[\tau_1/\alpha]}\ (\Xi \vdash \tau_1)$$

$$\frac{\Xi \mid \Delta \mid \Gamma, f : \tau_0 \to \tau_1, x : \tau_0 \vdash e : \tau_1}{\Xi \mid \Delta \mid \Gamma \vdash \mathtt{fix}^{\tau_0\to\tau_1} f(x).e : \tau_0 \to \tau_1}$$

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e : \tau}{\Xi \mid \Delta \mid \Gamma \vdash \mathtt{ref}(e) : \tau\ \mathtt{ref}} \qquad \frac{\Xi \mid \Delta \mid \Gamma \vdash e : \tau\ \mathtt{ref}}{\Xi \mid \Delta \mid \Gamma \vdash\ !\,e : \tau}$$

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e_0 : \tau\ \mathtt{ref} \qquad \Xi \mid \Delta \mid \Gamma \vdash e_1 : \tau}{\Xi \mid \Delta \mid \Gamma \vdash e_0 := e_1 : \mathtt{unit}}$$

Figure 2. Select typing rules. The general form is $\Xi \mid \Delta \mid \Gamma \vdash e : \tau$ for a context of type variables Ξ, a world Δ, a context of term variables Γ, an expression e and a type τ.

$$\frac{}{\Pi, \Lambda\alpha.e \Downarrow \Pi, \Lambda\alpha.e}$$

$$\frac{}{\Pi, \mathtt{fix}^{\tau_0\to\tau_1} f(x).e \Downarrow \Pi, \mathtt{fix}^{\tau_0\to\tau_1} f(x).e}$$

$$\frac{\Pi, e \Downarrow \Pi', \Lambda\alpha.e' \qquad \Pi', e'[\tau/\alpha] \Downarrow \Pi'', v}{\Pi, e[\tau] \Downarrow \Pi'', v}$$

$$\frac{\begin{array}{c}\Pi, e_0 \Downarrow \Pi', \mathtt{fix}^{\tau_0\to\tau_1}\ f(x).e \quad \Pi', e_1 \Downarrow \Pi'', v \\ \Pi'', e[v/x, \mathtt{fix}^{\tau_0\to\tau_1}\ f(x).e/f] \Downarrow \Pi''', v'\end{array}}{\Pi, e_0(e_1) \Downarrow \Pi''', v'}$$

$$\frac{\Pi, e \Downarrow \Pi', v}{\Pi, \mathtt{ref}(e) \Downarrow \Pi'[l \mapsto v], l}\ (l \notin \mathrm{dom}(\Pi'))$$

$$\frac{\Pi, e \Downarrow \Pi', l}{\Pi, !\,e \Downarrow \Pi', v}\ (l \in \mathrm{dom}(\Pi'), \Pi'(l) = v)$$

$$\frac{\Pi, e_0 \Downarrow \Pi', l \qquad \Pi', e_1 \Downarrow \Pi'', v}{\Pi, e_0 := e_1 \Downarrow \Pi''[l \mapsto v], ()}\ (l \in \mathrm{dom}(\Pi''))$$

Figure 3. Select rules of the big-step operational semantics. The general form is $\Pi, e \Downarrow \Pi', v$ where Π and Π' are syntactic stores, e is an expression and v a value.

The proof is by induction on the structure of the derivation of the judgment. It relies on basic properties of the type system such as standard substitution lemmas for type and term variables as well as the fact that an expression of some type in one world has the same type in any larger world.

Contextual equivalence of expressions (in empty worlds) is defined in the standard manner:

Definition 2. *If $\Xi \mid \emptyset \mid \Gamma \vdash e_i : \tau$, for $i = 1, 2$, then e_1 and e_2 are* contextually equivalent*, written*

$$\Xi \mid \emptyset \mid \Gamma \vdash e_1 =_{\mathrm{ctx}} e_2 : \tau\,,$$

if, for all closing contexts $C[.] : (\Xi \mid \emptyset \mid \Gamma \vdash \tau) \Rightarrow (\emptyset \mid \emptyset \mid \emptyset \vdash \mathtt{int})$, for all n,

$$\exists \Pi_1.\, \emptyset, C[e_1] \Downarrow \Pi_1, n \Leftrightarrow \exists \Pi_2.\, \emptyset, C[e_2] \Downarrow \Pi_2, n$$

3. Untyped Denotational Semantics

We first present an 'untyped' denotational semantics of our language. By this we mean that all expressions are interpreted by means of a certain complete partial order (cpo) U, and that the interpretation essentially ignores all type information in the language. Since U must in effect allow us to model an untyped variant of our language, we have the familiar requirement for models of the untyped λ-calculus: U must contain a copy of a function space with U itself as the domain. Therefore we construct U by solving a recursive domain equation.

We work with a concrete, domain-theoretic setting: Let $\mathbf{Cppo}_\perp$ be the category of pointed ω-cpos (i.e., cpos containing a least element) and strict, continuous functions. The cpo U is constructed by solving a domain equation of the form

$$U \cong F(U, U)$$

where F is a mixed-variance functor on $\mathbf{Cppo}_\perp$ (see below).

It is not enough that U is any solution to the equation above: the standard methods for solving recursive domain equations give solutions that are so-called *minimal invariants* [28]. In our setting, minimal invariance of U means that there exist continuous functions $\pi_n : U \multimap U$ (one for each $n \in \mathbb{N}$) satisfying, among other properties, that for each $u \in U$,

$$\pi_0 u \sqsubseteq \pi_1 u \sqsubseteq \cdots \sqsubseteq \pi_n u \sqsubseteq \cdots \quad \text{and} \quad \bigsqcup_{n\in\mathbb{N}} \pi_n u = u\,.$$

We say that each element u of U is the limit of its projections $\pi_n u$. The 'projection' functions π_n therefore provide a handle for proving properties about U by induction on n. Moreover, unlike in any earlier work we are aware of, these functions are directly used in the definition of the (untyped) semantics; that will turn out to be essential when we construct our typed semantics in the next section.

We now turn to the formal development.

Definition 3. *Let $i : F(U, U) \cong U$ be a minimal invariant of the locally continuous functor $F : \mathbf{Cppo}_\perp^{op} \times \mathbf{Cppo}_\perp \to \mathbf{Cppo}_\perp$*

defined on objects by

$$F(D,E) = 1_\bot \oplus \mathbb{Z}_\bot \oplus (\mathbb{N} \times E)_\bot \oplus (E \otimes E) \oplus (E \oplus E) \oplus$$
$$E \oplus \big[(\mathbb{N} \overset{fin}{\to} D_\downarrow)_\bot \multimap (\mathbb{N} \overset{fin}{\to} E_\downarrow)_\bot \otimes E\big]_\bot \oplus$$
$$\big[(\mathbb{N} \overset{fin}{\to} D_\downarrow)_\bot \otimes D \multimap (\mathbb{N} \overset{fin}{\to} E_\downarrow)_\bot \otimes E\big]_\bot.$$

F is assembled from standard components [28] with one exception: For any pointed cpo D we define a new cpo $\mathbb{N} \overset{fin}{\to} D_\downarrow$ by having $s \sqsubseteq s'$ if $\mathrm{dom}(s) = \mathrm{dom}(s')$ and $s(l) \sqsubseteq s'(l)$ for all $l \in \mathrm{dom}(s)$. Lifting then yields the pointed cpo $(\mathbb{N} \overset{fin}{\to} D_\downarrow)_\bot$ and this endofunction on objects of $\mathbf{Cppo}_\bot$ is extended naturally – much as a smash product – to a locally continuous functor $\mathbf{Cppo}_\bot \to \mathbf{Cppo}_\bot$ which is used in the above definition. Here $A \overset{fin}{\to} B$ denotes partial maps with finite domain from a set A to a set B and $D_\downarrow$ is all but the least element of a pointed cpo D.

Notice that the minimal invariant U exists by virtue of Theorem 3.3 of [28]. In accordance with this source we define the continuous map $\delta : (U \multimap U) \to (U \multimap U)$ by $\delta(e) = i \circ F(e,e) \circ i^{-1}$ for any $e \in U \multimap U$. We then define π_n as $\delta^n(\bot)$ for any $n \in \mathbb{N}$; as discussed above, minimality of the invariant means that $\bigsqcup_{n\in\mathbb{N}} \pi_n = id_U$. Note $\pi_m \circ \pi_n = \pi_{m\wedge n}$ for any $m, n \in \mathbb{N}$.

The cpo U is our universal domain: one can intuitively think of U as the domain of all untyped semantic values, analogous to the untyped closed values of our syntactic language. We define $S = (\mathbb{N} \overset{fin}{\to} U_\downarrow)_\bot$ which intuitively is the collection of *states*. The cpo $S \multimap S \otimes U$ models *computations*, i.e., functions from an initial state either diverge or return a state and a semantic value.

From the isomorphism i and the definition of $F(U,U)$ we obtain functions for injecting integers, pairs, functions, etc. into the universal domain: $in_{\mathtt{unit}} : 1 \to U$, $in_{\mathtt{int}} : \mathbb{Z} \to U$, $in_{\mathtt{ref}} : (\mathbb{N} \times U) \to U$, $in_\times : U \otimes U \multimap U$, $in_+ : U \oplus U \multimap U$, $in_\mu : U \multimap U$, $in_\forall : (S \multimap S \otimes U) \to U$, and $in_\to : (S \otimes U \multimap S \otimes U) \to U$. The injection $in_{\mathtt{ref}}$ is explained in more detail below. We use the cpo $S \multimap S \otimes U$ of 'computations' as the domain of $in_\forall$ because, in the untyped semantics, a syntactic value $\Lambda\alpha.e$ is treated simply as a suspension of the computation e: the type argument is ignored.

We now introduce the *semantic locations* as promised:

Definition 4. *For $l \in \mathbb{N}$ we define $\Lambda_l : U \to U$ continuous and order-monic by $\lambda u \in U.\ in_{\mathtt{ref}}(l,u)$. We define $\lambda_l^n = \Lambda_l^n(\bot)$ for any $l, n \in \mathbb{N}$ and finally choose*

$$\lambda_l = \bigsqcup_{n\in\mathbb{N}} \lambda_l^n.$$

Using the the observation that $\pi_{n+1} \circ \Lambda_l = \Lambda_l \circ \pi_n$ holds for any $l, n \in \mathbb{N}$ it is not hard to prove the following properties:

Lemma 5 (Location). *For any $k, l, l', n \in \mathbb{N}$ and $u \in U$ we have:*

(i) $\pi_k(\lambda_l^n) = \lambda_l^{n\wedge k}$, $\pi_n(\lambda_l) = \lambda_l^n$.
(ii) $\lambda_l^{n+1} = \lambda_{l'}^{n+1} \Leftrightarrow l = l'$
(iii) $\pi_n(u) \sqsupseteq \lambda_l^n \Leftrightarrow \pi_n(u) = \lambda_l^n$, $u \sqsupseteq \lambda_l \Leftrightarrow u = \lambda_l$.
(iv) $u \sqsubseteq \lambda_l^n \Leftrightarrow \exists j \le n.\ u = \lambda_l^j$, $u \sqsubseteq \lambda_l \Leftrightarrow u = \lambda_l \vee \exists j.\ u = \lambda_l^j$.

Definition 6. *Any type judgment $\Xi \mid \Delta \mid \Gamma \vdash e : \tau$ is interpreted as $[\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau]\!] \in S \otimes (\mathrm{dom}(\Gamma) \to U_\downarrow)_\bot \multimap S \otimes U$ by induction on the typing derivation, important cases are in Figure 4, see the long version of the paper for a complete presentation.*

Verification of continuity is tedious but standard with the one exception that we use the Location Lemma and the particular ordering on S in the cases involving references.

If we have $\emptyset \mid \Delta \mid \emptyset \vdash v : \tau$ for a value v then there naturally is a unique $u \in U_\downarrow$ such that $[\![\emptyset \mid \Delta \mid \emptyset \vdash v : \tau]\!]^s = \lfloor s, u \rfloor$ for any $s \in S_\downarrow$, we denote this u by $[\![\Delta \vdash v : \tau]\!]$. Similarly, if we have $\emptyset \mid \Delta \mid \emptyset \vdash \Pi$ we define $[\![\Delta \vdash \Pi]\!] = \lambda l \in \mathrm{dom}(\Delta).\ [\![\Delta \vdash \Pi(l) : \Delta(l)]\!] \in S_\downarrow$. With this notation in place we are ready to prove adequacy and soundness of our untyped interpretation:

Theorem 7 (Adequacy). *For $\emptyset \mid \Delta \mid \emptyset \vdash e : \mathtt{int}$ and $\emptyset \mid \Delta \mid \emptyset \vdash \Pi$ we get that*

$$[\![\emptyset \mid \Delta \mid \emptyset \vdash e : \mathtt{int}]\!]^{[\![\Delta \vdash \Pi]\!]} \neq \bot \implies \Pi, e \downarrow.$$

Here and below we use write $\Pi, e \downarrow$ to denote *termination*, i.e., the existence of a syntactic store Π' and a value v such that $\Pi, e \Downarrow \Pi', v$.

Proposition 8 (Soundness). *For $\emptyset \mid \Delta \mid \emptyset \vdash e : \mathtt{int}$ and $\emptyset \mid \Delta \mid \emptyset \vdash \Pi$, any syntactic store Π' and any $n \in \mathbb{N}$ we get*

$$\Pi, e \Downarrow \Pi', n \implies$$
$$\exists s \in S_\downarrow.\ [\![\emptyset \mid \Delta \mid \emptyset \vdash e : \mathtt{int}]\!]^{[\![\Delta \vdash \Pi]\!]} = \lfloor s, in_{\mathtt{int}}(n) \rfloor.$$

Proving soundness is slightly nontrivial due to the nondeterministic memory allocation of the operational semantics. On the other hand, the problem intuitively comes down to location renaming, i.e., we may perform substitutions of one location for another to make the operational semantics mimic the 'least free' memory allocation of the denotational semantics. Details are deferred to the long version of the paper.

To prove adequacy we proceed along the lines of the proof of Proposition 5.1 in [28]. But since our operational semantics is not deterministic due to the nondeterministic allocation, we resort to a continuation passing style proof to ensure admissibility of the 'formal approximation' relations. We introduce *continuations* for that purpose, these are just expressions with one free term variable: For a context of type variables Ξ, a world Δ, an expression K, a variable x and types τ_0 and τ_1 we write $\Xi \mid \Delta \vdash K : (x : \tau_0 \to \tau_1)$ if we have $\Xi \mid \Delta \mid x : \tau_0 \vdash K : \tau_1$ and we refer to K as a continuation. It is a simple yet important property that for any syntactic store Π and any expression e we have

$$\Pi, (\lambda^{\tau_0 \to \tau_1} x.\ K)(e) \downarrow \Leftrightarrow \exists \Pi', v.\ \Pi, e \Downarrow \Pi', v \wedge \Pi', K[v/x] \downarrow$$

We fix some further sets of syntax: For a world Δ and a type τ with $\emptyset \vdash \Delta$ and $\emptyset \vdash \tau$ we let $\mathbf{Val}_\tau^\Delta$ and $\mathbf{Expr}_\tau^\Delta$ denote the set of values and expressions respectively that have type τ under the assumption of Δ and empty contexts. $\mathbf{SynSt}^\Delta$ is the set of syntactic stores Π with $\emptyset \mid \Delta \mid \emptyset \vdash \Pi$ and $\mathbf{Cont}^\Delta_{x:\tau_0\to\tau_1}$ is the set of continuations K with $\emptyset \mid \Delta \vdash K : (x : \tau_0 \to \tau_1)$.

Proposition 9. *There is a family of 'formal approximation' relations $\lhd_\tau^\Delta \subset U_\downarrow \times \mathbf{Val}_\tau^\Delta$ with the properties of Figure 5 and with $\{u \in U_\downarrow \mid u \lhd_\tau^\Delta v\}$ chain complete and $\lhd_\tau^\Delta \subset \lhd_\tau^{\Delta'}$ for $\Delta \sqsubseteq \Delta'$.*

Proof. Denote by $\mathbf{UAdmSub}(U)$ all uniform and admissible subsets of U in the sense that they are closed under application of π_n for any $n \in \mathbb{N}$, contain $\bot$ and are chain complete. This constitutes a complete lattice with ordinary set inclusion as ordering since all properties are preserved by intersection, and hence the following is a complete lattice too with pointwise ordering:

$$\mathcal{K} = \Big\{ f \in \prod_{(\Delta,\tau)\in\mathbf{World}_\emptyset\times\mathbf{Type}_\emptyset} \mathbf{Val}_\tau^\Delta \to \mathbf{UAdmSub}(U) \Big|$$
$$\forall \Delta, \Delta' \in \mathbf{World}_\emptyset \forall \tau \in \mathbf{Type}_\emptyset \forall v \in \mathbf{Val}_\tau^\Delta.$$
$$\Delta \sqsubseteq \Delta' \implies f(\Delta,\tau)(v) \subset f(\Delta',\tau)(v) \Big\}.$$

In Figure 6 we define a monotone map $\Phi : \mathcal{K}^{op} \times \mathcal{K} \to \mathcal{K}$ and mimicking the proof of Theorem 4.16 from [28] one can establish the existence of a fixed point, i.e., a $K \in \mathcal{K}$ with $\Phi(K,K) = K$. We write $u \lhd_\tau^\Delta v$ for $u \in K(\Delta,\tau)(v) \setminus \{\bot\}$ and are done. □

Note that in the proof above we make use of a complete lattice of *functions* from syntactic types (and worlds). This makes it

$$[\![\Xi \mid \Delta \mid \Gamma \vdash x : \tau]\!]^s_\rho = \lfloor s, \rho(x) \rfloor \qquad [\![\Xi \mid \Delta \mid \Gamma \vdash l : \tau\;\texttt{ref}]\!]^s_\rho = \lfloor s, \lambda_l \rfloor$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau]\!]^s_\rho = \big\lfloor s, \mathit{in}_\forall(\lambda s' \in S_\downarrow.[\![\Xi, \alpha \mid \Delta \mid \Gamma \vdash e : \tau]\!]^{s'}_\rho)\big\rfloor$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash e[\tau_1] : \tau_0[\tau_1/\alpha]]\!]^s_\rho = \begin{cases} \varphi(s') & [\![\Xi \mid \Delta \mid \Gamma \vdash e : \forall\alpha.\tau_0]\!]^s_\rho = \lfloor s', \mathit{in}_\forall(\varphi) \rfloor \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash \texttt{fold}^{\mu\alpha.\tau}(e) : \mu\alpha.\tau]\!]^s_\rho = \begin{cases} \lfloor s', \mathit{in}_\mu(u) \rfloor & [\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]]\!]^s_\rho = \lfloor s', u \rfloor \\ \bot & \text{otherwise} \end{cases}$$

$$\begin{aligned}[\![\Xi \mid \Delta \mid \Gamma \vdash \texttt{fix}^{\tau_0 \to \tau_1} f(x).e : \tau_0 \to \tau_1]\!]^s_\rho = \big\lfloor s, (\mathit{in}_\to \circ \mathit{fix})\big(\lambda\varphi \in S \otimes U \multimap S \otimes U.& \\ \lambda(s', u) \in S_\downarrow \times U_\downarrow.\; [\![\Xi \mid \Delta \mid \Gamma, f : \tau_0 \to \tau_1, x : \tau_0 \vdash e : \tau_1]\!]^{s'}_{\rho[f \mapsto \mathit{in}_\to(\varphi), x \mapsto u]}\big)\big\rfloor&\end{aligned}$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash \texttt{ref}(e) : \tau\;\texttt{ref}]\!]^s_\rho = \begin{cases} \lfloor s'[l \mapsto u], \lambda_l \rfloor & \left[\begin{array}{l} [\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau]\!]^s_\rho = \lfloor s', u \rfloor, \\ l \notin \mathrm{dom}(s'), \forall l' < l.\; l' \in \mathrm{dom}(s') \end{array}\right. \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash\, !e : \tau]\!]^s_\rho = \begin{cases} \lfloor s', s'(l) \rfloor & [\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau\;\texttt{ref}]\!]^s_\rho = \lfloor s', \lambda_l \rfloor,\; l \in \mathrm{dom}(s') \\ \lfloor s', \pi_n(s'(l)) \rfloor & \left[\begin{array}{l} [\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau\;\texttt{ref}]\!]^s_\rho = \lfloor s', u \rfloor,\; \pi_{n+1}(u) = \lambda_l^{n+1}, \\ \pi_{n+2}(u) \neq \lambda_l^{n+2},\; l \in \mathrm{dom}(s'),\; \pi_n(s'(l)) \neq \bot \end{array}\right. \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\Xi \mid \Delta \mid \Gamma \vdash e_0 := e_1 : \texttt{unit}]\!]^s_\rho = \begin{cases} \lfloor s''[l \mapsto u], \mathit{in}_{\texttt{unit}}(*) \rfloor & \left[\begin{array}{l} [\![\Xi \mid \Delta \mid \Gamma \vdash e_0 : \tau\;\texttt{ref}]\!]^s_\rho = \lfloor s', \lambda_l \rfloor, \\ [\![\Xi \mid \Delta \mid \Gamma \vdash e_1 : \tau]\!]^{s'}_\rho = \lfloor s'', u \rfloor,\; l \in \mathrm{dom}(s'') \end{array}\right. \\ \lfloor s''[l \mapsto \pi_n(u)], \mathit{in}_{\texttt{unit}}(*) \rfloor & \left[\begin{array}{l} [\![\Xi \mid \Delta \mid \Gamma \vdash e_0 : \tau\;\texttt{ref}]\!]^s_\rho = \lfloor s', u' \rfloor \\ [\![\Xi \mid \Delta \mid \Gamma \vdash e_1 : \tau]\!]^{s'}_\rho = \lfloor s'', u \rfloor,\; \pi_{n+1}(u') = \lambda_l^{n+1}, \\ \pi_{n+2}(u') \neq \lambda_l^{n+2},\; l \in \mathrm{dom}(s''),\; \pi_n(u) \neq \bot \end{array}\right. \\ \bot & \text{otherwise} \end{cases}$$

Figure 4. Untyped interpretation, select cases. The general form of the left hand side is $[\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau]\!]^s_\rho$ with $s \in S_\downarrow$ and $\rho \in \mathrm{dom}(\Gamma) \to U_\downarrow$. The right hand side is an element of $S \otimes U$, recall that $S = (\mathit{Loc} \xrightarrow{\mathit{fin}} U_\downarrow)_\bot$.

$$\begin{array}{rcl}
u \lhd^\Delta_{\texttt{unit}} () & \iff & u = \mathit{in}_{\texttt{unit}}(*) \\
u \lhd^\Delta_{\texttt{int}} n & \iff & u = \mathit{in}_{\texttt{int}}(n) \\
u \lhd^\Delta_{\tau\;\texttt{ref}} l & \iff & u \sqsubseteq \lambda_l \\
u \lhd^\Delta_{\tau_0 \times \tau_1} (v_0, v_1) & \iff & \exists u_0, u_1 \in U_\downarrow.\; u = \mathit{in}_\times(\lfloor u_0, u_1 \rfloor) \wedge u_0 \lhd^\Delta_{\tau_0} v_0 \wedge u_1 \lhd^\Delta_{\tau_1} v_1 \\
u \lhd^\Delta_{\tau_0 + \tau_1} \texttt{inl}^{\tau_0 + \tau_1}(v) & \iff & \exists u' \in U_\downarrow.\; u = (\mathit{in}_+ \circ \mathit{inl})(u') \wedge u' \lhd^\Delta_{\tau_0} v \\
u \lhd^\Delta_{\tau_0 + \tau_1} \texttt{inr}^{\tau_0 + \tau_1}(v) & \iff & \exists u' \in U_\downarrow.\; u = (\mathit{in}_+ \circ \mathit{inr})(u') \wedge u' \lhd^\Delta_{\tau_1} v \\
u \lhd^\Delta_{\mu\alpha.\tau} \texttt{fold}^{\mu\alpha.\tau}(v) & \iff & \exists u' \in U_\downarrow.\; u = \mathit{in}_\mu(u') \wedge u' \lhd^\Delta_{\tau[\mu\alpha.\tau/\alpha]} v \\
u \lhd^\Delta_{\forall\alpha.\tau} \Lambda\alpha.e & \iff & \exists \varphi \in S \multimap S \otimes U.\; u = \mathit{in}_\forall(\varphi) \wedge \forall \Delta' \sqsupseteq \Delta. \forall \tau' \in \mathbf{Type}_\emptyset.\; \varphi \lhd^{\Delta'}_{T\tau[\tau'/\alpha]} e[\tau'/\alpha] \\
u \lhd^\Delta_{\tau_0 \to \tau_1} \texttt{fix}^{\tau_0 \to \tau_1} f(x).e & \iff & \exists \varphi \in S \otimes U \multimap S \otimes U.\; u = \mathit{in}_\to(\varphi) \wedge \forall \Delta' \sqsupseteq \Delta. \forall u', v.\; u' \lhd^{\Delta'}_{\tau_0} v \Rightarrow \\
 & & \qquad \lambda s \in S_\downarrow.\; \varphi(\lfloor s, u' \rfloor) \lhd^{\Delta'}_{T\tau_1} e[\texttt{fix}^{\tau_0 \to \tau_1} f(x).e/f, v/x] \\
s \lhd^\Delta \Pi & \iff & \mathrm{dom}(s) = \mathrm{dom}(\Pi) \wedge \forall l \in \mathrm{dom}(s).\; s(l) \lhd^\Delta_{\Delta(l)} \Pi(l) \\
k \lhd^\Delta_{K\tau} K & \iff & \forall \Delta' \sqsupseteq \Delta. \forall u, v, s, \Pi.\; u \lhd^{\Delta'}_\tau v \wedge s \lhd^{\Delta'} \Pi \Rightarrow \big[k(\lfloor s, u \rfloor) \neq \bot \Rightarrow \Pi, (\lambda^{\tau \to \texttt{int}} x.\; K)(v) \downarrow\big] \\
\varphi \lhd^\Delta_{T\tau} e & \iff & \forall s, \Pi, k, K.\; s \lhd^\Delta \Pi \wedge k \lhd^\Delta_{K\tau} K \Rightarrow \big[k(\varphi(s)) \neq \bot \Rightarrow \Pi, (\lambda^{\tau \to \texttt{int}} x.\; K)(e) \downarrow\big]
\end{array}$$

Figure 5. Desired properties of an indexed family of 'formal approximation' relations $\lhd^\Delta_\tau \subset U_\downarrow \times \mathbf{Val}^\Delta_\tau$. Also we define three auxiliary families of relations, $\lhd^\Delta \subset S_\downarrow \times \mathbf{SynSt}^\Delta$, $\lhd^\Delta_{K\tau} \subset (S \otimes U \multimap S \otimes U) \times \mathbf{Cont}^\Delta_{x:\tau \to \texttt{int}}$ and $\lhd^\Delta_{T\tau} \subset (S \multimap S \otimes U) \times \mathbf{Expr}^\Delta_\tau$.

particularly easy to define the interpretation of (closed) recursive and polymorphic types, cf., the definition of Φ in Figure 6, and means that we find the interpretation of all types by taking one fixed point of Φ, rather than via a nested sequence of fixed points as in, e.g., [13, 14]. This idea of using a function-space lattice was also used in the first author's earlier work [11], albeit more implicitly.

Proposition 10. *Given $\Xi \mid \Delta \mid \Gamma \vdash e : \tau$ with $\Xi = \alpha_1, \ldots, \alpha_m$ and $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$. Pick $\sigma_1, \ldots, \sigma_m \in \mathbf{Type}_\emptyset$ and denote application of the substitution $[\sigma_1/\alpha_1, \ldots, \sigma_m/\alpha_m]$ by overlining. For any $\Delta_0 \in \mathbf{World}_\emptyset$ with $\Delta_0 \sqsupseteq \overline{\Delta}$ and any $u_1, \ldots, u_n \in U_\downarrow$ and any values $v_1, \ldots, v_n$ with $u_i \lhd^{\Delta_0}_{\overline{\tau_i}} v_i$ for all $1 \leq i \leq n$, we have*

$$\lambda s \in S_\downarrow.[\![\Xi \mid \Delta \mid \Gamma \vdash e : \tau]\!]^s_\rho \lhd^{\Delta_0}_{T\overline{\tau}} \overline{e}[v_1/x_1, \ldots, v_n/x_n]$$

with $\rho = [x_1 \mapsto u_1, \ldots, x_n \mapsto u_n] \in \mathrm{dom}(\Gamma) \to U_\downarrow$.

Loosely, this proposition says that any expression is related to itself. Applying the identity continuation it is not hard to see that it has adequacy as a corollary as we have $[\![\Delta \vdash v : \tau]\!] \lhd^\Delta_\tau v$ for any value v with $\emptyset \mid \Delta \mid \emptyset \vdash v : \tau$.

Proof. We prove the proposition by induction on the typing derivation, details follow for a few cases. For the case of memory alloca-

$$
\begin{array}{rcl}
\Phi(\mathcal{R},\mathcal{S})(\Delta,\mathtt{unit})(()) & = & \{\bot\}\cup\{in_{\mathtt{unit}}(*)\}\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\mathtt{int})(n) & = & \{\bot\}\cup\{in_{\mathtt{int}}(n)\}\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\tau\,\mathtt{ref})(l) & = & \{u\in U\mid u\sqsubseteq\lambda_l\}\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\tau_0\times\tau_1)((v_0,v_1)) & = & \{\bot\}\cup(in_\times\circ\lfloor-\rfloor)(\mathcal{S}(\Delta,\tau_0)(v_0)\setminus\{\bot\}\times\mathcal{S}(\Delta,\tau_1)(v_1)\setminus\{\bot\})\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\tau_0+\tau_1)(\mathtt{inl}(v)) & = & \{\bot\}\cup(in_+\circ inl)(\mathcal{S}(\Delta,\tau_0)(v)\setminus\{\bot\})\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\tau_0+\tau_1)(\mathtt{inr}(v)) & = & \{\bot\}\cup(in_+\circ inr)(\mathcal{S}(\Delta,\tau_1)(v)\setminus\{\bot\})\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\mu\alpha.\tau)(\mathtt{fold}(v)) & = & \{\bot\}\cup in_\mu(\mathcal{S}(\Delta,\tau[\mu\alpha.\tau/\alpha])(v)\setminus\{\bot\})\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\forall\alpha.\tau)(\Lambda\alpha.e) & = & \{\bot\}\cup\big\{in_\forall(\varphi)\mid\varphi\in S\multimap S\otimes U\ \wedge\\
& & \qquad\forall\Delta'\sqsupseteq\Delta.\,\forall\tau'\in\mathbf{Type}_\emptyset.\,\varphi\in\Phi^T(\mathcal{R},\mathcal{S})(\Delta',\tau[\tau'/\alpha])(e[\tau'/\alpha])\big\}\\
\Phi(\mathcal{R},\mathcal{S})(\Delta,\tau_0\to\tau_1)(\mathtt{fix}^{\tau_0\to\tau_1}f(x).e) & = & \{\bot\}\cup\big\{in_\to(\varphi)\mid\varphi\in S\otimes U\multimap S\otimes U\ \wedge\\
& & \qquad\forall\Delta'\sqsupseteq\Delta.\,\forall v\,\forall u\in\mathcal{R}(\Delta',\tau_0)(v)\setminus\{\bot\}.\\
& & \qquad\quad\lambda s\in S_\downarrow.\,\varphi(\lfloor s,u\rfloor)\in\Phi^T(\mathcal{R},\mathcal{S})(\Delta',\tau_1)\\
& & \qquad\qquad(e[\mathtt{fix}^{\tau_0\to\tau_1}f(x).e/f,v/x])\big\}\\
\Phi^S(\mathcal{S})(\Delta)(\Pi) & = & \{\bot\}\cup\big\{s\in S_\downarrow\mid\mathrm{dom}(s)\ =\ \mathrm{dom}(\Pi)\wedge\forall l\in\mathrm{dom}(s).\,s(l)\in\mathcal{S}(\Delta,\Delta(l))(\Pi(l))\setminus\{\bot\}\big\}\\
\Phi^K(\mathcal{R},\mathcal{S})(\Delta,\tau)(K) & = & \big\{k\in S\otimes U\multimap S\otimes U\mid\forall\Delta'\sqsupseteq\Delta.\,\forall v,\Pi.\\
& & \qquad\forall u\in\mathcal{R}(\Delta',\tau)(v)\setminus\{\bot\}.\,\forall s\in\Phi^S(\mathcal{R})(\Delta')(\Pi)\setminus\{\bot\}.\\
& & \qquad\quad k(\lfloor s,u\rfloor)\neq\bot\Rightarrow\Pi,(\lambda^{\tau\to\mathtt{int}}x.\,K)(v)\downarrow\big\}\\
\Phi^T(\mathcal{R},\mathcal{S})(\Delta,\tau)(e) & = & \big\{\varphi\in S\multimap S\otimes U\mid\forall\Pi,K.\\
& & \qquad\forall s\in\Phi^S(\mathcal{R})(\Delta)(\Pi)\setminus\{\bot\}.\,\forall k\in\Phi^K(\mathcal{S},\mathcal{R})(\Delta,\tau)(K).\\
& & \qquad\quad k(\varphi(s))\neq\bot\Rightarrow\Pi,(\lambda^{\tau\to\mathtt{int}}x.\,K)(e)\downarrow\big\}
\end{array}
$$

Figure 6. Definition of $\Phi:\mathcal{K}^{op}\times\mathcal{K}\to\mathcal{K}$ using three auxiliary maps $\Phi^S:\mathcal{K}\to\prod_{\Delta\in\mathbf{World}_\emptyset}\mathbf{SynSt}^\Delta\to\mathcal{P}(S)$, $\Phi^K:\mathcal{K}^{op}\times\mathcal{K}\to\prod_{(\Delta,\tau)\in\mathbf{World}_\emptyset\times\mathbf{Type}_\emptyset}\mathbf{Cont}^\Delta_{x:\tau\to\mathtt{int}}\to\mathcal{P}(S\otimes U\multimap S\otimes U)$ and $\Phi^T:\mathcal{K}^{op}\times\mathcal{K}\to\prod_{(\Delta,\tau)\in\mathbf{World}_\emptyset\times\mathbf{Type}_\emptyset}\mathbf{Expr}^\Delta_\tau\to\mathcal{P}(S\multimap S\otimes U)$.

tion, consider

$$\frac{\Xi\mid\Delta\mid\Gamma\vdash e:\tau}{\Xi\mid\Delta\mid\Gamma\vdash\mathtt{ref}(e):\tau\,\mathtt{ref}}$$

and assume that the proposition holds for the premise. Choose types $\sigma_1,\dots,\sigma_m$, $\Delta_0\sqsupseteq\overline{\Delta}$ and $u_1,\dots,u_n$ elements of $U_\downarrow$, values $v_1,\dots,v_n$ and $\rho\in\mathrm{dom}(\Gamma)\to U_\downarrow$ as stated. Furthermore pick s,Π,k and K with $s\lhd^{\Delta_0}\Pi$ and $k\lhd^{\Delta_0}_{K\overline{\tau}\,\mathtt{ref}}K$. We now assume that

$$k(\llbracket\emptyset\mid\overline{\Delta}\mid\overline{\Gamma}\vdash\mathtt{ref}(\overline{e}):\overline{\tau}\,\mathtt{ref}\rrbracket^s_\rho)\neq\bot$$

and are to prove that

$$\Pi,(\lambda^{\overline{\tau}\,\mathtt{ref}\to\mathtt{int}}x.\,K)(\mathtt{ref}(\overline{e}[v_1/x_1,\dots,v_n/x_n]))\downarrow.$$

The map $\lambda(s,u)\in S_\downarrow\times U_\downarrow.\ k(\lfloor s[l\mapsto u],\lambda_l\rfloor)$ where we choose $l\in\mathbb{N}$ minimal such that $l\notin\mathrm{dom}(s)$ defines a map $k'\in S\otimes U\multimap S\otimes U$. Also we define a continuation $K'\in\mathbf{Cont}^{\Delta_0}_{x:\overline{\tau}\to\mathtt{int}}$ by $K'=(\lambda^{\overline{\tau}\,\mathtt{ref}\to\mathtt{int}}x.\,K)(\mathtt{ref}(x))$ and by the induction hypothesis it suffices to show that $k'\lhd^{\Delta_0}_{K\overline{\tau}}K'$. According to definition we pick $\Delta'\sqsupseteq\Delta_0$ and u',v',s',Π' with $u'\lhd^{\Delta'}_{\overline{\tau}}v'$ and $s'\lhd^{\Delta'}\Pi'$, we assume that $k'(\lfloor s',u'\rfloor)\neq\bot$ and aim to prove that $\Pi',(\lambda^{\overline{\tau}\to\mathtt{int}}x.\,K')(v')\downarrow$. We remark that

$$\bot\neq k'(\lfloor s',u'\rfloor)=k(\lfloor s'[l'\mapsto u'],\lambda_{l'}\rfloor)$$

for $l'\in\mathbb{N}$ with $l'\notin\mathrm{dom}(s')$. By $s'\lhd^{\Delta'}\Pi'$ we have $l'\notin\mathrm{dom}(s')=\mathrm{dom}(\Pi')$ and hence $\Pi',\mathtt{ref}(v')\Downarrow\Pi'[l'\mapsto v'],l'$. We obviously have $\Delta'[l'\mapsto\overline{\tau}]\sqsupseteq\Delta'$ and $\lambda_{l'}\lhd^{\Delta'[l'\mapsto\overline{\tau}]}_{\overline{\tau}\,\mathtt{ref}}l'$. Also for any $l\in\mathrm{dom}(s')\cup\{l'\}$ we have $s'[l'\mapsto u'](l)\lhd^{\Delta'}_{\Delta'[l'\mapsto\overline{\tau}](l)}\Pi'[l'\mapsto v'](l)$ and hence $s'[l'\mapsto u'](l)\lhd^{\Delta'[l'\mapsto\overline{\tau}]}_{\Delta'[l'\mapsto\overline{\tau}](l)}\Pi'[l'\mapsto v'](l)$ too which means that $s'[l'\mapsto u']\lhd^{\Delta'[l'\mapsto\overline{\tau}]}\Pi'[l'\mapsto v']$ and we are done as we initially assumed that $k\lhd^{\Delta_0}_{K\overline{\tau}\,\mathtt{ref}}K$.

This case warrants some comments: It is here that we need the continuations to 'work' in all future worlds, in the other cases this property is just pushed through the proof. Also this is where we rely on the property that the formal approximations grow with larger worlds. Finally note that the operational semantics may allocate any free location, in particular we can pick the least free to match the behavior of the denotational semantics.

Consider now the case of lookup, i.e., consider

$$\frac{\Xi\mid\Delta\mid\Gamma\vdash e:\tau\,\mathtt{ref}}{\Xi\mid\Delta\mid\Gamma\vdash\,!\,e:\tau}$$

and assume that the proposition holds for the premise. Choose types $\sigma_1,\dots,\sigma_m$, a world $\Delta_0\sqsupseteq\Delta$ and $u_1,\dots,u_n$ elements of $U_\downarrow$, values $v_1,\dots,v_n$ and $\rho\in\mathrm{dom}(\Gamma)\to U_\downarrow$ as stated. Furthermore pick s,Π,k and K with $s\lhd^{\Delta_0}\Pi$ and $k\lhd^{\Delta_0}_{K\overline{\tau}}K$. We now assume that

$$k(\llbracket\emptyset\mid\overline{\Delta}\mid\overline{\Gamma}\vdash\,!\,\overline{e}:\overline{\tau}\rrbracket^s_\rho)\neq\bot$$

and are to prove that

$$\Pi,(\lambda^{\overline{\tau}\to\mathtt{int}}x.\,K)(!\,\overline{e}[v_1/x_1,\dots,v_n/x_n])\downarrow.$$

We define $k'\in S\otimes U\multimap S\otimes U$ by mapping any $(s,u)\in S_\downarrow\times U_\downarrow$ to $S\otimes U$ by copying the interpretation of lookup and applying k:

$$\begin{cases}
k(\lfloor s,s(l)\rfloor) & u=\lambda_l,\,l\in\mathrm{dom}(s)\\
k(\lfloor s,\pi_n(s(l))\rfloor) & \pi_{n+1}(u)=\lambda_l^{n+1},\pi_n(s(l))\neq\bot,\\
 & \pi_{n+2}(u)\neq\lambda_l^{n+2},l\in\mathrm{dom}(s)\\
\bot & \text{otherwise}
\end{cases}$$

Similarly we define $K'\in\mathbf{Cont}^{\overline{\Delta}}_{x:\overline{\tau}\,\mathtt{ref}\to\mathtt{int}}$ by

$$K'=(\lambda^{\overline{\tau}\to\mathtt{int}}x.\,K)(!\,x)$$

and by induction it suffices to show that $k'\lhd^{\Delta_0}_{K\overline{\tau}\,\mathtt{ref}}K'$.

For that purpose we pick $\Delta'\sqsupseteq\Delta_0$, u',v',s' and Π' with $u'\lhd^{\Delta'}_{\overline{\tau}\,\mathtt{ref}}v'$ and $s'\lhd^{\Delta'}\Pi'$, we assume that $k'(\lfloor s',u'\rfloor)\neq\bot$ and have to prove that $\Pi',(\lambda^{\overline{\tau}\,\mathtt{ref}\to\mathtt{int}}x.\,K')(v')\downarrow$. From $u'\lhd^{\Delta'}_{\overline{\tau}\,\mathtt{ref}}v'$ we deduce that there is $l'\in\mathrm{dom}(\Delta')$ with $v'=l'$, $\Delta'(l')=\overline{\tau}$ and $u'\sqsubseteq\lambda_{l'}$. Also $s'\lhd^{\Delta'}\Pi'$ yields that $l'\in\mathrm{dom}(\Delta')=\mathrm{dom}(s')=$

$\mathrm{dom}(\Pi')$ and this gives $\Pi', !\, l' \Downarrow \Pi', \Pi'(l')$ and we also have $s'(l') \lhd_{\overline{\tau}}^{\Delta'} \Pi'(l')$.

Assume now that $u' = \lambda_{l'}$. From the definition of k' we get that $\bot \neq k(\lfloor s', s'(l') \rfloor)$ and we may use the original assumption $k \lhd_{K\overline{\tau}}^{\Delta_0} K$ to prove the required. Suppose now that $u' \neq \lambda_{l'}$, i.e., that $u' = \lambda_{l'}^{n'+1}$ for some $n' \in \mathbb{N}$. We get $\bot \neq k(\lfloor s', \pi_{n'}(s'(l')) \rfloor)$ which yields $\bot \neq k(\lfloor s', s'(l') \rfloor)$ by monotonicity and we are back on the above track. □

4. Typed Denotational Semantics

In this section we present the typed possible world semantics. As mentioned in the Introduction, to reason about parametricity we need to give a semantics not only of closed types (as sufficed for proving adequacy in the previous section) but also of *open* types. This has two consequences for the technical development which we explain before proceeding with the technical development proper.

Recall first that the overall idea is to define the semantics of types by means of world-indexed binary relations over the universal domain U. These relations will be both *uniform* and *admissible*: such relations are completely determined by their elements of the form $(\pi_n u,\ \pi_n u')$. One explanation of the formal construction below is therefore the following. To define the various relations that together constitute the semantics of types, it suffices to determine for each $n \in \mathbb{N}$ whether the pairs of the form $(\pi_n u,\ \pi_n u')$ belong to the various relations; this can be done by induction on n. For all types except reference types, this approach works well due to properties of the π_n. For example, $\pi_{n+1}(in_+(inl\, u)) = in_+(inl\,(\pi_n u))$ and $\pi_{n+1}(in_\times(u_1, u_2)) = in_\times(\pi_n u_1, \pi_n u_2)$.

For the case of reference types, the idea is roughly that, for a type $\Xi \vdash \tau$, for a world $\Delta \in \mathbf{World}_\Xi$, and for semantics types $\overline{\nu}$ corresponding to the type environment Ξ,

$$(u, u') \in [\![\tau\ \mathtt{ref}]\!]_\Xi(\overline{\nu})(\Delta)$$

if and only if,

$$\exists l \in \mathrm{dom}(\Delta).\, u = u' = l \wedge [\![\tau]\!]_\Xi(\overline{\nu})(\Delta) = [\![\Delta(l)]\!]_\Xi(\overline{\nu})(\Delta).$$

That is, u and u' should be the same location l and, moreover, the interpretation of the type τ should be the same as the interpretation of the type $\Delta(l)$ found in the store type Δ. The latter is, of course, to ensure sound modelling of lookup and assignment.

The problem with the above definition is that it is not inductive: to determine whether the pair $(\pi_{n+1}\, u, \pi_{n+1}\, u')$ belongs to $[\![\tau\ \mathtt{ref}]\!]_\Xi(\overline{\nu})(\Delta)$, we need to know the entire relations $[\![\tau]\!]_\Xi(\overline{\nu})(\Delta)$ and $[\![\Delta(l)]\!]_\Xi(\overline{\nu})(\Delta)$, not just their elements of the form $(\pi_n u_0, \pi_n u_0')$. That is the reason for introducing semantic locations λ_l and λ_l^n. By means of these we can refine the above idea to what you see in Figure 8. The idea is that approximative locations λ_l^{n+1} are related in case the interpretations of types agree up to level n and that ideal locations λ_l are related in case the interpretations really are equal. (As shown, the real definition also includes a quantification over future worlds, but that is not related to what we are discussing here.)

More formally, the problem with the definition of $[\![\tau\ \mathtt{ref}]\!]$ above is that it prevents one from proving the existence of the family of logical relations constituting the semantics of types. The usual proof by fixed-point induction and minimal invariance [28] does not go through: indeed, for an earlier variant of the setup presented here, we could actually give a formal proof showing that relations satisfying such conditions do not exist.

Clearly, there are some relations between our semantic locations and step-indexed approaches to recursive types [2, 5, 6]; see Subsection 4.1 for comments on how one can attempt to make the connection formal.

The second consequence of interpreting open types is also related to the use of world-indexed relations. It has to do with how we should interpret quantified types $\forall\alpha.\, \tau$. When one is only interested in a semantics of closed types, one can define the semantics of $\forall\alpha.\, \tau$ simply as the intersection over all *syntactic types* $\bigcap_{\sigma \in \mathbf{Type}} [\![\tau[\sigma/\alpha]]\!]$, as we essentially did in the adequacy proof earlier. For a semantics of open types, one typically defines the semantics of $\forall\alpha.\, \tau$ by a big intersection over some universe ST of *semantic types* (think of ST as the set of all admissible relations) $\bigcap_{\nu \in \mathrm{ST}} [\![\tau]\!](\nu)$. However, in our case the meaning of a type *depends on* the current world. One attempt to accommodate this dependency would be to interpret a closed universal type $\forall\alpha.\tau$ essentially as an intersection over semantic types indexed by closed worlds: $\bigcap_{\nu \in \mathbf{World}_\emptyset \to \mathrm{ST}} [\![\tau]\!](\nu)$. The problem with this attempt is that in the natural Kripke-style definition of $[\![\tau]\!](\nu)$ one needs to apply ν not only to closed worlds, but to worlds containing free occurrences of α. Worse, if τ contains nested universal types, one needs to apply ν to worlds containing additional new type variables. For example, if α occurs in τ below a universal quantifier $\forall\beta$, then one needs to be able to apply ν to an arbitrary world $\Delta \in \mathbf{World}_{\alpha,\beta}$.

Informally, one attempt to interpret such an occurrence of α would be

$$[\![\alpha]\!]_{\alpha,\beta}(\nu, \nu')(\Delta) = \nu\,(\nu, \nu')\,(\Delta)\,,$$

i.e., to interpret α in a world $\Delta \in \mathbf{World}_{\alpha,\beta}$, one applies ν not only to Δ, but also to the ν and ν' that interpret α and β, respectively. But this attempt introduces a circularity: it is not clear what the formal definition of ν should be, since ν must now be applicable to itself as well as an arbitrary other ν'. To break the circularity in the above interpretation of α, we instead apply ν to *the interpretation function itself*, partially applied to (ν, ν') and Δ:

$$[\![\alpha]\!]_{\alpha,\beta}(\nu, \nu')(\Delta) = \nu\,[\lambda\tau_0 \in \mathbf{Type}_\emptyset.[\![\tau_0]\!]_{\alpha,\beta}(\nu, \nu')(\Delta)]\,.$$

In this way (ν, ν') and Δ are indirectly passed to ν. (Notice that the τ_0 on the right hand side contains fewer free type variables than α.)

In summary, we use a novel interpretation of types, where $\forall\alpha.\, \tau$ is interpreted essentially by a big intersection

$$\bigcap_{\nu \in (\mathbf{Type} \to \mathrm{ST}) \to \mathrm{ST}} [\![\tau]\!](\nu)$$

over semantic types *indexed over* a function that can interpret closed types (i.e., types with one fewer type variable than τ). This essentially allows us to *delay* the choice of semantic type until we know how the world should be interpreted.

We now continue with the formal development after which we discuss an alternative approach to dealing with the second issue mentioned above and then present some examples. In the formal development we make use of admissible relations that satisfy a couple of additional conditions, uniformity and strictness. Uniformity is typical for interpretations of polymorphism and recursive types [4]; strictness is used to capture contextual equivalence (also used in [11]).

Definition 11. *Let $\mathbf{UARel}(U)$ be the set of binary relations on U that relate $\bot$ to $\bot$ and to nothing else, are closed under π_n for any $n \in \mathbb{N}$ and are closed under taking least upper bounds of chains.*

We speak of *uniform* and *admissible* relations over U. It is not hard to see that $\mathbf{UARel}(U)$ with ordinary set inclusion constitutes a complete lattice as all properties are preserved by intersection. We can now define the *semantic closed types*:

Definition 12. *For any context of type variables Ξ we let $\mathbf{SCT}_\Xi$ be the monotone maps ν of*

$$\left[\mathbf{Type}_\Xi \to \mathbf{UARel}(U)\right] \overset{mon}{\to} \mathbf{UARel}(U)$$

for which it holds that for any two arguments $\varphi, \varphi' \in \mathbf{Type}_\Xi \to \mathbf{UARel}(U)$ and any $n \in \mathbb{N}$ we have that

$$\left[\forall \tau \in \mathbf{Type}_\Xi.\ \varphi(\tau) \overset{n}{=} \varphi'(\tau)\right] \implies \nu(\varphi) \overset{n}{=} \nu(\varphi').$$

Above and below we use $R \overset{n}{=} S$ for $n \in \mathbb{N}$ and $R, S \in \mathbf{UARel}(U)$ to mean that $\pi_n(R) \subset S$ and $\pi_n(S) \subset R$ hold; we shall also use $R \overset{n}{\subset} S$ to denote just the first of these properties. Intuitively, the demand that 'n-equality' be preserved by semantic closed types allows us to work with approximations of types – a property we need to prove the existence of the desired interpretation of types. For any context of type variables Ξ and any syntactic type $\sigma \in \mathbf{Type}_\Xi$ we furthermore define

$$\nu_\Xi(\sigma) = \lambda\varphi \in \mathbf{Type}_\Xi \to \mathbf{UARel}(U).\ \varphi(\sigma)$$

and it is easily verified that we indeed have $\nu_\Xi(\sigma) \in \mathbf{SCT}_\Xi$.

We shall need a few minor definitions: For every type in context $\alpha_1, \ldots, \alpha_m \vdash \tau$ we define the following measure

$$\#(\alpha_1, \ldots, \alpha_m \vdash \tau) = \min\{0 \le n \le m \mid \alpha_1, \ldots, \alpha_n \vdash \tau\}\,;$$

recall here that type contexts are ordered lists. For a context of type variables $\Xi = \alpha_1, \alpha_2, \ldots, \alpha_m$ we write $\mathbf{SCT}^\Xi$ as shorthand for

$$\mathbf{SCT}_\emptyset \times \mathbf{SCT}_{\alpha_1} \times \mathbf{SCT}_{\alpha_1,\alpha_2} \times \cdots \times \mathbf{SCT}_{\alpha_1,\alpha_2,\ldots,\alpha_{m-1}}.$$

Finally, assume that we have type contexts Ξ, Ξ', $\overline{\nu} \in \mathbf{SCT}^\Xi$, $\overline{\nu}' \in \mathbf{SCT}^{\Xi'}$, $\Delta \in \mathbf{World}_\Xi$ and $\Delta' \in \mathbf{World}_{\Xi'}$. We now define $[\Xi|\overline{\nu}|\Delta] \sqsubseteq [\Xi'|\overline{\nu}'|\Delta']$ to denote the existence of type variables $\beta_1, \beta_2, \ldots, \beta_m$ and semantic closed types $\nu_1 \in \mathbf{SCT}_\Xi$, $\nu_2 \in \mathbf{SCT}_{\Xi,\beta_1}$ up to $\nu_m \in \mathbf{SCT}_{\Xi,\beta_1,\ldots,\beta_{m-1}}$ such that $\Xi, \beta_1, \beta_2, \ldots, \beta_m = \Xi'$ and $(\overline{\nu}, \nu_1, \nu_2, \ldots, \nu_m) = \overline{\nu}'$ and $\Delta \sqsubseteq \Delta'$. This definition captures our typed notion of 'future' worlds: Not only does the world itself grow, we also allow extension of the context of type variables with corresponding semantic closed types. We are now ready to define the lattice $\mathcal{L}$ of type interpretations:

Definition 13. *We define a complete lattice by pointwise ordering*

$$\mathcal{L} = \Big\{ f \in \prod_\Xi \mathbf{SCT}^\Xi \to \mathbf{Type}_\Xi \to \mathbf{World}_\Xi \to \mathbf{UARel}(U) \ \Big|\ [\Xi|\overline{\nu}|\Delta] \sqsubseteq [\Xi'|\overline{\nu}'|\Delta'] \wedge \tau \in \mathbf{Type}_\Xi \implies f(\Xi)(\overline{\nu})(\tau)(\Delta) \subset f(\Xi')(\overline{\nu}')(\tau)(\Delta') \Big\}.$$

We also define $\Psi : \mathcal{L}^{op} \times \mathcal{L} \to \mathcal{L}$ monotone in Figure 7: The definition of $\Psi(\mathcal{R}, \mathcal{S})(\Xi)(\overline{\nu})(\tau)(\Delta)$ is by induction on $\#(\Xi \vdash \tau)$.

Members of $\mathcal{L}$ interpret types, and as we deal with open types we parameterize over semantic closed types to 'plug in' for the free variables as well as over worlds. The intuition behind defining Ψ by induction on $\#(\Xi \vdash \tau)$ is that $\Psi(\mathcal{R}, \mathcal{S})(\Xi)(\overline{\nu})(\tau)(\Delta)$ is obviously well defined if τ is not a type variable, and from that point on we can interpret type variables in the order they occur in Ξ. It is worth noticing that the definition of Ψ in the cases of references, polymorphic types and functions has been carefully tailored to comply with monotonicity property in the definition of $\mathcal{L}$: the quantification over 'future' worlds has been baked in.

To obtain the desired interpretation of types we could just appeal to the approach of the proof of Theorem 4.16 of [28] as done above in the proof of Theorem 7. Instead we construct the sequence of approximations and the fixed point by hand – we proceed in the style of Kleene's fixed point theorem rather than by appeal to the Knaster-Tarski fixed point theorem. The difference is merely one of presentation: The present approach is less general but arguably has a more constructive feel to it that goes well with the intuition of the semantic locations.

We define $\mathcal{R}_0 \in \mathcal{L}$ as constant $\{(\bot, \bot)\}$, $\mathcal{S}_0 \in \mathcal{L}$ as constant $\{(u, u') \in U \times U \mid \forall n \in \mathbb{N}.\ \pi_n(u) = \bot \Leftrightarrow \pi_n(u') = \bot\}$ and inductively $\mathcal{R}_{n+1} = \Psi(\mathcal{S}_n, \mathcal{R}_n) \in \mathcal{L} \ni \Psi(\mathcal{R}_n, \mathcal{S}_n) = \mathcal{S}_{n+1}$ for all $n \in \mathbb{N}$. By induction we get the crucial fact that $\mathcal{R}_n \overset{n}{=} \mathcal{S}_n$, for all $n \in \mathbb{N}$, and choosing $\nabla = \cap_{n\in\mathbb{N}} \mathcal{S}_n$ yields a fixed point of Ψ, i.e., $\Psi(\nabla, \nabla) = \nabla$. We are now able to interpret types: We shall denote $\nabla(\Xi)(\tau)(\overline{\nu})(\Delta) \setminus \{(\bot, \bot)\}$ by $[\![\tau]\!]_\Xi(\overline{\nu})(\Delta)$ and it is immediate that this interpretation has the properties listed in Figure 8. Also the following is a consequence of the construction:

Lemma 14 (Monotonicity)**.** *For $[\Xi|\overline{\nu}|\Delta] \sqsubseteq [\Xi'|\overline{\nu}'|\Delta']$ and $\tau \in \mathbf{Type}_\Xi$ we have $[\![\tau]\!]_\Xi(\overline{\nu})(\Delta) \subset [\![\tau]\!]_{\Xi'}(\overline{\nu}')(\Delta')$.*

We remark that (1) as for the adequacy proof, we again make use of a complete lattice of *functions*, cf., Definition 13; and (2) we would need to prove the existence of the logical relations using a proof as the one above, even if we had left out recursive types from the language. In that case, we could define the relation by induction on the type, for all other type constructors but `ref` – for `ref` it would not be possible, since the definition in the case for τ `ref` involves the meaning of arbitrary types in future worlds. This is typical for models of higher-order store in which one can have recursion through the store, even without recursive types.

Lemma 15 (Degenerate Substitution)**.** *With natural ranges of variables, in particular $\tau \in \mathbf{Type}_{\Xi,\alpha,\Xi'}$, $\Delta \in \mathbf{World}_{\Xi,\alpha,\Xi'}$ and $\sigma \in \mathbf{Type}_\Xi$, we have that*

$$[\![\tau]\!]_{\Xi,\alpha,\Xi'}(\overline{\nu}, \nu_\Xi(\sigma), \overline{\nu}')(\Delta) = [\![\tau[\sigma/\alpha]]\!]_{\Xi,\alpha,\Xi'}(\overline{\nu}, \nu_\Xi(\sigma), \overline{\nu}')(\Delta).$$

It is easily proved by induction that the property holds for $\mathcal{S}_n$ for all $n \in \mathbb{N}$ and the above lemma follows – the validity of this lemma is partial justification for the definition of interpretation of type variables. We refer to the lemma as *degenerate* because we perform no substitution in the world Δ and do not remove α and $\nu_\Xi(\sigma)$ on the right hand side. It is possible to state and prove a more standard substitution lemma, but we shall not need that.

The main result of this section is a 'fundamental theorem of logical relations,' intuitively stating that every well-typed term is related to itself. First some notation: For any two contexts of type variables Ξ and Ξ' we write $\Xi \subset \Xi'$ if all variables of Ξ occur in Ξ', i.e., if the inclusion holds when interpreting the contexts as sets.

Definition 16. *Two expressions in context $\Xi \mid \Delta \mid \Gamma \vdash e_i : \tau$ ($i = 1, 2$) are* semantically related*, written*

$$\Xi \mid \Delta \mid \Gamma \vdash e_1 \sim e_2 : \tau\,,$$

if for all $\Xi' \supset \Xi$, all $\overline{\nu}' \in \mathbf{SCT}^{\Xi'}$, all $\Delta' \in \mathbf{World}_{\Xi'}$ with $\Delta' \sqsupseteq \Delta$ and all $\rho, \rho' \in \mathrm{dom}(\Gamma) \to U_\downarrow$ such that $(\rho(x), \rho'(x)) \in [\![\Gamma(x)]\!]_{\Xi'}(\overline{\nu}')(\Delta')$ for each $x \in \mathrm{dom}(\Gamma)$, we have that the pair

$$\big(\lambda s \in S_\downarrow.[\![\Xi \mid \Delta \mid \Gamma \vdash e_1 : \tau]\!]^s_\rho, \lambda s \in S_\downarrow.[\![\Xi \mid \Delta \mid \Gamma \vdash e_2 : \tau]\!]^s_{\rho'}\big)$$

belongs to $[\![\tau]\!]^T_{\Xi'}(\overline{\nu}')(\Delta')$.

Theorem 17. $\Xi \mid \Delta \mid \Gamma \vdash e : \tau$ *implies* $\Xi \mid \Delta \mid \Gamma \vdash e \sim e : \tau$*.*

Proof. The proof is by induction on the typing derivation, we shall present three decisive cases. Consider the lookup case, i.e., consider

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e : \tau\ \texttt{ref}}{\Xi \mid \Delta \mid \Gamma \vdash !\,e : \tau}$$

and assume that the proposition holds for the premise. We pick arbitrary $\Xi' \supset \Xi$, $\overline{\nu}' \in \mathbf{SCT}^{\Xi'}$, $\Delta' \in \mathbf{World}_{\Xi'}$ with $\Delta' \sqsupseteq \Delta$, and $\rho, \rho' \in \mathrm{dom}(\Gamma) \to U_\downarrow$ as specified in the definition of semantic relatedness. Also we take arbitrary $(s, s') \in [\![\Delta']\!]^S_{\Xi'}(\overline{\nu}')$ and $(k, k') \in [\![\tau]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$ and we have to prove that either

$$k\big([\![\Xi \mid \Delta \mid \Gamma \vdash !\,e : \tau]\!]^s_\rho\big) = \bot = k'\big([\![\Xi \mid \Delta \mid \Gamma \vdash !\,e : \tau]\!]^{s'}_{\rho'}\big)$$

or that the left hand side and the right hand side both terminate and moreover both yield the value $in_{\texttt{int}}(n)$ for some $n \in \mathbb{N}$,

$$\begin{aligned}
\Psi(\mathcal{R},\mathcal{S})(\alpha_1,\ldots,\alpha_m)(\nu_1,\ldots,\nu_m)(\alpha_n)(\Delta) &= \nu_n\big[\lambda\tau \in \mathbf{Type}_{\alpha_1,\ldots,\alpha_{n-1}}.\ \Psi(\mathcal{R},\mathcal{S})(\alpha_1,\ldots,\alpha_m)(\nu_1,\ldots,\nu_m)(\tau)(\Delta)\big]\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\mathtt{1})(\Delta) &= \{(\bot,\bot)\} \cup \{(in_{\mathtt{unit}}(*), in_{\mathtt{unit}}(*))\}\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\mathtt{int})(\Delta) &= \{(\bot,\bot)\} \cup \{(in_{\mathtt{int}}(n), in_{\mathtt{int}}(n)) \mid n \in \mathbb{N}\}\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\tau\ \mathtt{ref})(\Delta) &= \{(\bot,\bot)\} \cup\\
&\quad \big\{(\lambda_l^{n+1},\lambda_l^{n+1}) \mid l \in \mathrm{dom}(\Delta) \wedge n \in \mathbb{N} \wedge\\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta].\ \mathcal{R}(\Xi')(\overline{\nu}')(\tau)(\Delta') \overset{n}{\subset} \mathcal{S}(\Xi')(\overline{\nu}')(\Delta'(l))(\Delta') \wedge\\
&\qquad\qquad \mathcal{R}(\Xi')(\overline{\nu}')(\Delta'(l))(\Delta') \overset{n}{\subset} \mathcal{S}(\Xi')(\overline{\nu}')(\tau)(\Delta')\big\} \cup\\
&\quad \big\{(\lambda_l,\lambda_l) \mid l \in \mathrm{dom}(\Delta) \wedge\\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta].\ \mathcal{R}(\Xi')(\overline{\nu}')(\tau)(\Delta') \subset \mathcal{S}(\Xi')(\overline{\nu}')(\Delta'(l))(\Delta') \wedge\\
&\qquad\qquad \mathcal{R}(\Xi')(\overline{\nu}')(\Delta'(l))(\Delta') \subset \mathcal{S}(\Xi')(\overline{\nu}')(\tau)(\Delta')\big\}\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\tau_0 \times \tau_1)(\Delta) &= \{(\bot,\bot)\} \cup \big\{(in_\times(\lfloor u_0,u_1\rfloor), in_\times(\lfloor u_0',u_1'\rfloor)) \mid (u_0,u_0') \in \mathcal{S}(\Xi)(\overline{\nu})(\tau_0)(\Delta) \setminus \{(\bot,\bot)\} \wedge\\
&\qquad (u_1,u_1') \in \mathcal{S}(\Xi)(\overline{\nu})(\tau_1)(\Delta) \setminus \{(\bot,\bot)\}\big\}\\
\Psi(\mathcal{R},\mathcal{S})(\overline{\nu})(\Xi)(\tau_0 + \tau_1)(\Delta) &= \{(\bot,\bot)\} \cup \big\{((in_+ \circ inl)(u), (in_+ \circ inl)(u')) \mid (u,u') \in \mathcal{S}(\Xi)(\overline{\nu})(\tau_0)(\Delta) \setminus \{(\bot,\bot)\}\big\}\\
&\quad \cup \big\{((in_+ \circ inr)(u), (in_+ \circ inr)(u')) \mid (u,u') \in \mathcal{S}(\Xi)(\overline{\nu})(\tau_1)(\Delta) \setminus \{(\bot,\bot)\}\big\}\\
\Psi(\mathcal{R},\mathcal{S})(\overline{\nu})(\Xi)(\mu\alpha.\tau)(\Delta) &= \{(\bot,\bot)\} \cup \big\{(in_\mu(u), in_\mu(u')) \mid (u,u') \in \mathcal{S}(\Xi)(\overline{\nu})(\tau[\mu\alpha.\tau/\alpha])(\Delta) \setminus \{(\bot,\bot)\}\big\}\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\forall\alpha.\tau)(\Delta) &= \{(\bot,\bot)\} \cup \big\{(in_\forall(\varphi), in_\forall(\varphi')) \mid \varphi,\varphi' \in S \multimap S \otimes U \wedge\\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta]\forall\nu \in \mathbf{SCT}_{\Xi'}.\\
&\qquad (\varphi,\varphi') \in \Psi^T(\mathcal{R},\mathcal{S})(\Xi',\alpha)(\overline{\nu}',\nu)(\tau)(\Delta')\big\}\\
\Psi(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\tau_0 \to \tau_1)(\Delta) &= \{(\bot,\bot)\} \cup\\
&\quad \big\{(in_\to(\varphi), in_\to(\varphi')) \mid \varphi,\varphi' \in S \otimes U \multimap S \otimes U \wedge\\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta]\forall(u,u') \in \mathcal{R}(\Xi')(\overline{\nu}')(\tau_0)(\Delta') \setminus \{(\bot,\bot)\}.\\
&\qquad [\lambda s \in S_\downarrow.\varphi(\lfloor s,u\rfloor), \lambda s' \in S_\downarrow.\varphi'(\lfloor s',u'\rfloor)] \in \Psi^T(\mathcal{R},\mathcal{S})(\Xi')(\overline{\nu}')(\tau_1)(\Delta')\big\}\\[1ex]
\Psi^S(\mathcal{S})(\Xi)(\overline{\nu})(\Delta) &= \{(\bot,\bot)\} \cup \big\{(s,s') \in (S_\downarrow)^2 \mid \mathrm{dom}(\Delta) = \mathrm{dom}(s) = \mathrm{dom}(s') \wedge\\
&\qquad \forall l \in \mathrm{dom}(\Delta).\ (s(l),s'(l)) \in \mathcal{S}(\Xi)(\overline{\nu})(\Delta(l))(\Delta) \setminus \{(\bot,\bot)\}\big\}\\
\Psi^K(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\tau)(\Delta) &= \big\{(k,k') \in (S \otimes U \multimap S \otimes U)^2 \mid \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta].\\
&\qquad \forall(s,s') \in \Psi^S(\mathcal{R})(\Xi')(\overline{\nu}')(\Delta') \setminus \{(\bot,\bot)\}.\\
&\qquad \forall(u,u') \in \mathcal{R}(\Xi')(\overline{\nu}')(\tau)(\Delta') \setminus \{(\bot,\bot)\}.\\
&\qquad [k(\lfloor s,u\rfloor) = \bot = k'(\lfloor s',u'\rfloor)] \vee\\
&\qquad [\exists t,t' \in S_\downarrow \exists n \in \mathbb{Z}.\ k(\lfloor s,u\rfloor) = \lfloor t, in_{\mathtt{int}}(n)\rfloor \wedge\\
&\qquad\qquad k'(\lfloor s',u'\rfloor) = \lfloor t', in_{\mathtt{int}}(n)\rfloor\big\}\\
\Psi^T(\mathcal{R},\mathcal{S})(\Xi)(\overline{\nu})(\tau)(\Delta) &= \big\{(\varphi,\varphi') \in (S \multimap S \otimes U)^2 \mid \forall(s,s') \in \Psi^S(\mathcal{R})(\Xi)(\overline{\nu})(\Delta) \setminus \{(\bot,\bot)\}.\\
&\qquad \forall(k,k') \in \Psi^K(\mathcal{S},\mathcal{R})(\Xi)(\overline{\nu})(\tau)(\Delta).\\
&\qquad [k(\varphi(s)) = \bot = k'(\varphi'(s'))] \vee\\
&\qquad [\exists t,t' \in S_\downarrow \exists n \in \mathbb{Z}.\ k(\varphi(s)) = \lfloor t, in_{\mathtt{int}}(n)\rfloor \wedge\\
&\qquad\qquad k'(\varphi'(s')) = \lfloor t', in_{\mathtt{int}}(n)\rfloor\big\}
\end{aligned}$$

Figure 7. Definition of $\Psi : \mathcal{L}^{op} \times \mathcal{L} \to \mathcal{L}$ using maps $\Psi^S : \mathcal{L} \to \prod_\Xi \mathbf{SCT}^\Xi \to \mathbf{World}_\Xi \to \mathcal{P}(S^2)$, $\Psi^K : \mathcal{L}^{op} \times \mathcal{L} \to \prod_\Xi \mathbf{SCT}^\Xi \to \mathbf{Type}_\Xi \to \mathbf{World}_\Xi \to \mathcal{P}((S \otimes U \multimap S \otimes U)^2)$ and $\Psi^T : \mathcal{L}^{op} \times \mathcal{L} \to \prod_\Xi \mathbf{SCT}^\Xi \to \mathbf{Type}_\Xi \to \mathbf{World}_\Xi \to \mathcal{P}((S \multimap S \otimes U)^2)$.

confer the definition of $[\![\tau]\!]^T_{\Xi'}(\overline{\nu}')(\Delta')$. Consider now the maps $k_0, k_0' : S \otimes U \multimap S \otimes U$ built by copying the interpretation of lookup and applying k respectively k', i.e., k_0 is obtained by mapping any $(s_0,u_0) \in S_\downarrow \times U_\downarrow$ to $S \otimes U$ as follows

$$\begin{cases} k(\lfloor s_0, s_0(l)\rfloor) & u_0 = \lambda_l, l \in \mathrm{dom}(s_0)\\ k(\lfloor s_0, \pi_n(s_0(l))\rfloor) & \begin{array}{l}\pi_{n+1}(u_0) = \lambda_l^{n+1}, \pi_n(s_0(l)) \neq \bot,\\ \pi_{n+2}(u_0) \neq \lambda_l^{n+2}, l \in \mathrm{dom}(s_0)\end{array}\\ \bot & \text{otherwise}\end{cases}$$

and k_0' is identical, with k' exchanged for k. By the induction hypothesis it suffices to prove that $(k_0,k_0') \in [\![\tau\ \mathtt{ref}]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$. For that purpose we pick $[\Xi_0|\overline{\nu}_0|\Delta_0] \sqsupseteq [\Xi'|\overline{\nu}'|\Delta']$ and we pick $(s_0,s_0') \in [\![\Delta_0]\!]^S_{\Xi_0}(\overline{\nu}_0)$ and $(u_0,u_0') \in [\![\tau\ \mathtt{ref}]\!]_{\Xi_0}(\overline{\nu}_0)(\Delta_0)$. The latter yields one of two: Either we have $u_0 = u_0' = \lambda_l^{n+1}$ for some $l \in \mathrm{dom}(\Delta_0)$ and an $n \in \mathbb{N}$ with $[\![\Delta_0(l)]\!]_{\Xi_0}(\overline{\nu}_0)(\Delta_0) \overset{n}{\subset} [\![\tau]\!]_{\Xi_0}(\overline{\nu}_0)(\Delta_0) \cup \{(\bot,\bot)\}$ or we have $u_0 = u_0' = \lambda_l$ for some $l \in \mathrm{dom}(\Delta_0)$ with $[\![\Delta_0(l)]\!]_{\Xi_0}(\overline{\nu}_0)(\Delta_0) = [\![\tau]\!]_{\Xi_0}(\overline{\nu}_0)(\Delta_0)$. And in both cases the desired follows from the definitions of k_0 and k_0' and from $(s_0,s_0') \in [\![\Delta_0]\!]^S_{\Xi_0}(\overline{\nu}_0)$ and $(k,k') \in [\![\tau]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$.

Let us now look at the case of memory allocation, i.e., consider

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e : \tau}{\Xi \mid \Delta \mid \Gamma \vdash \mathtt{ref}(e) : \tau\ \mathtt{ref}}$$

and assume that the proposition holds for the premise. We proceed as above, i.e., we pick arbitrary $\Xi' \supset \Xi$, $\overline{\nu}' \in \mathbf{SCT}^{\Xi'}$, $\Delta' \in \mathbf{World}_{\Xi'}$ with $\Delta' \sqsupseteq \Delta$, and $\rho,\rho' \in \mathrm{dom}(\Gamma) \to U_\downarrow$ as specified in the definition of semantic relatedness. Also we take arbitrary $(s,s') \in [\![\Delta']\!]^S_{\Xi'}(\overline{\nu}')$ and $(k,k') \in [\![\tau\ \mathtt{ref}]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$ and we construct $k_0, k_0' : S \otimes U \multimap S \otimes U$ by copying the interpretation of allocation and applying k respectively k', i.e., k_0 is built from the map

$$\lambda(s_0,u_0) \in S_\downarrow \times U_\downarrow.\ k(\lfloor s_0[l \mapsto u_0], \lambda_l\rfloor)$$

$$(u,u') \in \llbracket \alpha_{\mathtt{n}} \rrbracket_{\alpha_1,\dots,\alpha_m}(\nu_1,\dots,\nu_m)(\Delta) \Leftrightarrow (u,u') \in \nu_n\big[\lambda\tau \in \mathbf{Type}_{\alpha_1,\dots,\alpha_{n-1}}.\llbracket\tau\rrbracket_{\alpha_1,\dots,\alpha_m}(\nu_1,\dots,\nu_m)(\Delta) \cup \{(\bot,\bot)\}\big] \setminus \{(\bot,\bot)\}$$

$$\begin{aligned}
(u,u') \in \llbracket \mathtt{1} \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff u = u' = in_{\mathtt{unit}}(*) \\
(u,u') \in \llbracket \mathtt{int} \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \exists n \in \mathbb{Z}.\ u = u' = in_{\mathtt{int}}(n) \\
(u,u') \in \llbracket \tau\ \mathtt{ref} \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \big[\exists l \in \mathrm{dom}(\Delta)\exists n \in \mathbb{N}.\ u = u' = \lambda_l^{n+1} \wedge \\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta].\ \llbracket\tau\rrbracket_{\Xi'}(\overline{\nu}')(\Delta') \cup \{(\bot,\bot)\} \overset{n}{=} \llbracket\Delta'(l)\rrbracket_{\Xi'}(\overline{\nu}')(\Delta') \cup \{(\bot,\bot)\}\big] \vee \\
&\quad\ \big[\exists l \in \mathrm{dom}(\Delta).\ u = u' = \lambda_l \wedge \\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta].\ \llbracket\tau\rrbracket_{\Xi'}(\overline{\nu}')(\Delta') = \llbracket\Delta'(l)\rrbracket_{\Xi'}(\overline{\nu}')(\Delta')\big] \\
(u,u') \in \llbracket \tau_0 \times \tau_1 \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \exists(u_0,u_0') \in \llbracket\tau_0\rrbracket_{\Xi}(\overline{\nu})(\Delta)\exists(u_1,u_1') \in \llbracket\tau_1\rrbracket_{\Xi}(\overline{\nu})(\Delta).\ u = in_{\times}(\lfloor u_0,u_1\rfloor) \wedge u' = in_{\times}(\lfloor u_0',u_1'\rfloor) \\
(u,u') \in \llbracket \tau_0 + \tau_1 \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \big[\exists(u_0,u_0') \in \llbracket\tau_0\rrbracket_{\Xi}(\overline{\nu})(\Delta).\ u = (in_+ \circ inl)(u_0) \wedge u' = (in_+ \circ inl)(u_0')\big] \vee \\
&\quad\ \big[\exists(u_1,u_1') \in \llbracket\tau_1\rrbracket_{\Xi}(\overline{\nu})(\Delta).\ u = (in_+ \circ inr)(u_1) \wedge u' = (in_+ \circ inr)(u_1')\big] \\
(u,u') \in \llbracket \mu\alpha.\tau \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \exists(u_0,u_0') \in \llbracket\tau[\mu\alpha.\tau/\alpha]\rrbracket_{\Xi}(\overline{\nu})(\Delta).\ u = in_{\mu}(u_0) \wedge u' = in_{\mu}(u_0') \\
(u,u') \in \llbracket \forall\alpha.\tau \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \exists\varphi,\varphi' \in S \multimap S \otimes U.\ u = in_{\forall}(\varphi) \wedge u' = in_{\forall}(\varphi') \wedge \\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta]\forall\nu \in \mathbf{SCT}_{\Xi'}.\ (\varphi,\varphi') \in \llbracket\tau\rrbracket^{T}_{\Xi',\alpha}(\overline{\nu}',\nu)(\Delta') \\
(u,u') \in \llbracket \tau_0 \to \tau_1 \rrbracket_{\Xi}(\overline{\nu})(\Delta) &\iff \exists\varphi,\varphi' \in S \otimes U \multimap S \otimes U.\ u = in_{\to}(\varphi) \wedge u' = in_{\to}(\varphi') \wedge \\
&\qquad \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta]\forall(u_0,u_0') \in \llbracket\tau_0\rrbracket_{\Xi'}(\overline{\nu}')(\Delta'). \\
&\qquad\quad [\lambda s \in S_{\downarrow}.\varphi(\lfloor s,u_0\rfloor), \lambda s' \in S_{\downarrow}.\varphi'(\lfloor s',u_0'\rfloor)] \in \llbracket\tau_1\rrbracket^{T}_{\Xi'}(\overline{\nu}')(\Delta')
\end{aligned}$$

$$\begin{aligned}
(s,s') \in \llbracket\Delta\rrbracket^{S}_{\Xi}(\overline{\nu}) &\iff \mathrm{dom}(\Delta) = \mathrm{dom}(s) = \mathrm{dom}(s') \wedge \forall l \in \mathrm{dom}(\Delta).\ (s(l),s'(l)) \in \llbracket\Delta(l)\rrbracket_{\Xi}(\overline{\nu})(\Delta) \\
(k,k') \in \llbracket\tau\rrbracket^{K}_{\Xi}(\overline{\nu})(\Delta) &\iff \forall[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [\Xi|\overline{\nu}|\Delta]\forall(s,s') \in \llbracket\Delta'\rrbracket^{S}_{\Xi'}(\overline{\nu}')\forall(u,u') \in \llbracket\tau\rrbracket_{\Xi'}(\overline{\nu}')(\Delta'). \\
&\qquad \big[k(\lfloor s,u\rfloor) = \bot = k'(\lfloor s',u'\rfloor)\big] \vee \\
&\qquad \big[\exists t,t' \in S_{\downarrow}\exists n \in \mathbb{Z}.\ k(\lfloor s,u\rfloor) = \lfloor t, in_{\mathtt{int}}(n)\rfloor \wedge k'(\lfloor s',u'\rfloor) = \lfloor t', in_{\mathtt{int}}(n)\rfloor\big] \\
(\varphi,\varphi') \in \llbracket\tau\rrbracket^{T}_{\Xi}(\overline{\nu})(\Delta) &\iff \forall(s,s') \in \llbracket\Delta\rrbracket^{S}_{\Xi}(\overline{\nu})\,\forall(k,k') \in \llbracket\tau\rrbracket^{K}_{\Xi}(\overline{\nu})(\Delta). \\
&\qquad \big[k(\varphi(s)) = \bot = k'(\varphi'(s'))\big] \vee \\
&\qquad \big[\exists t,t' \in S_{\downarrow}\exists n \in \mathbb{Z}.\ k(\varphi(s)) = \lfloor t, in_{\mathtt{int}}(n)\rfloor \wedge k'(\varphi'(s')) = \lfloor t', in_{\mathtt{int}}(n)\rfloor\big]
\end{aligned}$$

Figure 8. Desired properties of interpretation of types. For $u, u' \in U_{\downarrow}$, a context Ξ, $\tau \in \mathbf{Type}_{\Xi}$, $\overline{\nu} \in \mathbf{SCT}^{\Xi}$ and $\Delta \in \mathbf{World}_{\Xi}$ we specify when $(u,u') \in \llbracket\tau\rrbracket_{\Xi}(\overline{\nu})(\Delta)$. Also we define $\llbracket\Delta\rrbracket^{S}_{\Xi}(\overline{\nu}) \subset (S_{\downarrow})^2$, $\llbracket\tau\rrbracket^{K}_{\Xi}(\overline{\nu})(\Delta) \subset (S \otimes U \multimap S \otimes U)^2$, and $\llbracket\tau\rrbracket^{T}_{\Xi}(\overline{\nu})(\Delta) \subset (S \multimap S \otimes U)^2$.

where we choose $l \in \mathbb{N}$ with $l \notin \mathrm{dom}(s_0)$ and $\forall l' < l.\ l' \in \mathrm{dom}(s_0)$ and k_0' is identical, with k' exchanged for k. It now remains to prove $(k_0,k_0') \in \llbracket\tau\rrbracket^{K}_{\Xi'}(\overline{\nu}')(\Delta')$. For that purpose we pick $[\Xi_0|\overline{\nu}_0|\Delta_0] \sqsupseteq [\Xi'|\overline{\nu}'|\Delta']$ and we pick $(s_0,s_0') \in \llbracket\Delta_0\rrbracket^{S}_{\Xi_0}(\overline{\nu}_0)$ and $(u_0,u_0') \in \llbracket\tau\rrbracket_{\Xi_0}(\overline{\nu}_0)(\Delta_0)$. From the former of these we get $k_0(\lfloor s_0,u_0\rfloor) = k(\lfloor s_0[l \mapsto u_0], \lambda_l\rfloor)$ and $k_0'(\lfloor s_0',u_0'\rfloor) = k'(\lfloor s_0'[l \mapsto u_0'], \lambda_l\rfloor)$ with l the least such that $l \notin \mathrm{dom}(\Delta_0)$. It is immediate that $(\lambda_l,\lambda_l) \in \llbracket\tau\ \mathtt{ref}\rrbracket_{\Xi_0}(\overline{\nu}_0)(\Delta_0[l \mapsto \tau])$ and for any $l' \in \mathrm{dom}(\Delta_0[l \mapsto \tau])$ we have $(s_0[l \mapsto u_0](l'), s_0'[l \mapsto u_0'](l')) \in \llbracket\Delta_0[l \mapsto \tau](l')\rrbracket_{\Xi_0}(\overline{\nu}_0)(\Delta_0)$ and hence $(s_0[l \mapsto u_0], s_0'[l \mapsto u_0']) \in \llbracket\Delta_0[l' \mapsto \tau]\rrbracket_{\Xi_0}(\overline{\nu}_0)$ by the Monotonicity Lemma. And applying the original assumption $(k,k') \in \llbracket\tau\ \mathtt{ref}\rrbracket^{K}_{\Xi'}(\overline{\nu}')(\Delta')$ we are done.

Finally we arrive at the the case of type application, this is where we require the Degenerate Substitution Lemma. We consider

$$\frac{\Xi \mid \Delta \mid \Gamma \vdash e : \forall\alpha.\tau_0}{\Xi \mid \Delta \mid \Gamma \vdash e[\tau_1] : \tau_0[\tau_1/\alpha]}\,(\Xi \vdash \tau_1)$$

and assume that the proposition holds for the premise. We proceed as usual, pick arbitrary $\Xi' \supset \Xi$, $\overline{\nu}' \in \mathbf{SCT}^{\Xi'}$, $\Delta' \in \mathbf{World}_{\Xi'}$ with $\Delta' \sqsupseteq \Delta$ and $\rho,\rho' \in \mathrm{dom}(\Gamma) \to U_{\downarrow}$ as specified in the definition of semantic relatedness. Also we take arbitrary $(s,s') \in \llbracket\Delta'\rrbracket^{S}_{\Xi'}(\overline{\nu}')$ and $(k,k') \in \llbracket\tau_0[\tau_1/\alpha]\rrbracket^{K}_{\Xi'}(\overline{\nu}')(\Delta')$ and we construct $k_0, k_0' : S \otimes U \multimap S \otimes U$ by copying the interpretation of type application and applying k respectively k', i.e., k_0 is built from the map

$$\lambda(s,u) \in S_{\downarrow} \times U_{\downarrow}.\ \begin{cases} k(\varphi(s)) & u = in_{\forall}(\varphi) \\ \bot & \text{otherwise} \end{cases}$$

and k_0' is identical, with k' exchanged for k. It now remains to prove $(k_0,k_0') \in \llbracket\forall\alpha.\tau_0\rrbracket^{K}_{\Xi'}(\overline{\nu}')(\Delta')$. For that purpose we pick $[\Xi_0|\overline{\nu}_0|\Delta_0] \sqsupseteq [\Xi'|\overline{\nu}'|\Delta']$ and we pick $(s_0,s_0') \in \llbracket\Delta_0\rrbracket^{S}_{\Xi_0}(\overline{\nu}_0)$ and $(u_0,u_0') \in \llbracket\forall\alpha.\tau_0\rrbracket_{\Xi_0}(\overline{\nu}_0)(\Delta_0)$. From the latter we get $\varphi_0,\varphi_0' \in S \multimap S \otimes U$ such that $u_0 = in_{\forall}(\varphi_0)$, $u_0' = in_{\forall}(\varphi_0')$ and $(\varphi,\varphi') \in \llbracket\tau_0\rrbracket^{T}_{\Xi_0,\alpha}(\overline{\nu}_0, \nu_{\Xi_0}(\tau_1))(\Delta_0)$, now it remains to show that we have

$$(s_0,s_0') \in \llbracket\Delta_0\rrbracket^{S}_{\Xi_0,\alpha}(\overline{\nu}_0, \nu_{\Xi_0}(\tau_1))$$

and that we have

$$(k,k') \in \llbracket\tau_0\rrbracket^{K}_{\Xi_0,\alpha}(\overline{\nu}_0, \nu_{\Xi_0}(\tau_1))(\Delta_0).$$

The first is an easy consequence of the Monotonicity lemma, for the latter we use the Degenerate Substitution Lemma to conclude

$$\llbracket\tau_0\rrbracket^{K}_{\Xi_0,\alpha}(\overline{\nu}_0, \nu_{\Xi_0}(\tau_1))(\Delta_0) = \llbracket\tau_0[\tau_1/\alpha]\rrbracket^{K}_{\Xi_0,\alpha}(\overline{\nu}_0, \nu_{\Xi_0}(\tau_1))(\Delta_0),$$

this suffices as $[\Xi_0, \alpha|(\overline{\nu}_0, \nu_{\Xi_0}(\sigma))|\Delta_0] \sqsupseteq [\Xi'|\overline{\nu}'|\Delta']$. □

Corollary 18. *Semantically related expressions in context are contextually equivalent: if* $\Xi \mid \emptyset \mid \Gamma \vdash e_1 \sim e_2 : \tau$ *then* $\Xi \mid \emptyset \mid \Gamma \vdash e_1 =_{\mathrm{ctx}} e_2 : \tau$.

Proof. This follows in the standard manner from the proof of the fundamental theorem (Theorem 17) above together with the adequacy and soundness results from the previous section. □

The theorem above, and its corollary, forms the basis for simple reasoning about parametricity using our model. A few examples are shown in Section 5.

4.1 Alternative Approach

In this subsection we briefly discuss and sketch an alternative approach to the second issue: the interpretation of open types depending on worlds as mentioned in the introduction of Section 4.

Here, we interpret quantified types as intersections over semantic closed types, the latter are members of $\mathbf{UARel}(U)$ parameterized over interpretations of types with fewer type variables. This somewhat syntactic choice goes nicely with syntactic worlds containing free type variables. The alternative approach is to have semantic worlds, mapping locations to semantic types and letting semantic types be world-indexed members of $\mathbf{UARel}(U)$. This introduces a mutual dependency between worlds and semantic types; in effect, we ask for solutions to the mutually recursive equations (recall that locations are natural numbers):

$$\begin{array}{rcl} \mathrm{ST} & = & W \to \mathbf{UARel}(U) \\ W & = & \mathbb{N} \overset{fin}{\to} \mathrm{ST} \end{array}$$

It turns out that one can solve equations similar to the above in suitable categories of complete ultra-metric spaces. Our solution relies on well-known metrics associated with partial equivalence relations [1, 4]. The alternative approach gives a more semantic understanding of open types, in particular one can interpret quantified types $\forall\alpha.\tau$ by the more standard $\bigcap_{\nu\in\mathrm{ST}}[\![\tau]\!](\nu)$. But it does come at the price of using (yet) more mathematical machinery. The present approach is a fairly (if not entirely) simple alternative. And while the two approaches yield different models, it is not immediate that either is superior in terms of proving more equivalences.

For lack of space we cannot present the details of the alternative approach here; that will be done in a forthcoming paper. With this approach we will also be able to make a more detailed comparison to the step-indexed approach to recursive types and references [2, 6]; indeed, approximations to equations similar to those shown above play a key role in recent step-indexed models [6].

5. Examples of Parametricity Reasoning

As explained in the Introduction, this paper focuses on the key technical challenges involved in defining an adequate, parametric model for a language with recursive types and general references. The main contributions of the paper are our solutions to these challenges, including the concepts of *semantic locations* and *semantic closed types*; extending the current setup to allow for more advanced applications involving local state [11] is deferred to future work (see Section 6).

As illustrated by the first example below, one can use the parametricity results in this paper to prove equivalences between different functional implementations of abstract data types in an imperative language. The proof essentially proceeds in the standard manner — but the point now is that the clients of such abstract data types may be implemented using *all* the features of the language, including general references, recursive types, etc. The remaining three examples below illustrate that one can prove simple equivalences involving imperative abstract data types and local state.

In the examples we use the standard encoding of n-ary products by means of binary products. And we refer to the type `unit` by 1.

Example 19. In the first example, we show that a client of a module that implements a counter cannot distinguish between two different, but related implementations of the module. The two implementations are very simple functional implementations, but we emphasize that the reasoning works for *any* client of the right type; the client may be implemented using all the features of the language.

The type of counter-module clients is

$$\tau_{\mathrm{cl}} = \forall\alpha.((1 \to \alpha) \times (\alpha \to \alpha) \times (\alpha \to \mathtt{int}) \to \mathtt{int})\,.$$

Intuitively, a client c of a counter module takes an unknown type α (the concrete type used internally by the module to represent counters) and three functions (the first for creating a new counter, the second for incrementing a counter, and the third for getting the value of a counter) and returns a result of type `int`.

Let the two counter implementations be given by I_1 and I_2:

$$\begin{array}{rcl} I_1 & = & (\lambda x : 1.\, 0,\, \lambda x : \mathtt{int}.\, x+1,\, \lambda x : \mathtt{int}.\, x) \\ I_2 & = & (\lambda x : 1.\, 0,\, \lambda x : \mathtt{int}.\, x-1,\, \lambda x : \mathtt{int}.\, -x). \end{array}$$

We can now use Corollary 18 to prove that

$$\emptyset \mid \emptyset \mid c : \tau_{\mathrm{cl}} \vdash c[\mathtt{int}]I_1 =_{\mathrm{ctx}} c[\mathtt{int}]I_2 : \mathtt{int}\,.$$

The proof of relatedness of $c[\mathtt{int}]I_1$ and $c[\mathtt{int}]I_2$ proceeds as expected, except that it is in continuation-passing style, and, of course, involves the definition of a relation relating each integer n to $-n$. Formally, one uses the semantic closed type $\nu_0 \in \mathbf{SCT}_\Xi$ defined by

$$\nu_0(\varphi) = \{(\bot,\bot)\} \cup \{(in_{\mathtt{int}}(n), in_{\mathtt{int}}(-n)) \mid n \in \mathbb{N}\}\,.$$

Example 20. Consider now the following type of clients of an *imperative* counter module:

$$\tau'_{\mathrm{cl}} = \forall\alpha.((1 \to \alpha) \times (\alpha \to 1) \times (\alpha \to \mathtt{int}) \to \mathtt{int})\,.$$

As in the previous example, the intuition is that a client takes an unknown type α and three functions implementing operations on counters. The difference from the previous example is that the second of the three functions has the type $\alpha \to 1$, reflecting that the 'increment' operation modifies its input and does not need to return a result.

Let the two imperative implementations be given by I'_1 and I'_2:

$$\begin{array}{rcl} I'_1 & = & (\lambda x : 1.\, \mathtt{ref}(0), \\ & & \lambda x : \mathtt{int\ ref}.\, x := \,!x + 1, \\ & & \lambda x : \mathtt{int\ ref}.\, !x) \\ I'_2 & = & (\lambda x : 1.\, \mathtt{ref}(0), \\ & & \lambda x : \mathtt{int\ ref}.\, x := \,!x - 1, \\ & & \lambda x : \mathtt{int\ ref}.\, -(!x)) \end{array}$$

We can now use Corollary 18 to prove that

$$\emptyset \mid \emptyset \mid c : \tau'_{\mathrm{cl}} \vdash c[\mathtt{int\,ref}]I'_1 =_{\mathrm{ctx}} c[\mathtt{int\,ref}]I'_2 : \mathtt{int}\,.$$

To show semantic relatedness, we let $\Delta \in \mathbf{World}_\Xi$ and $\overline{\nu} \in \mathbf{SCT}^\Xi$ and $(c_1, c_2) \in [\![\tau'_{\mathrm{cl}}]\!]_\Xi(\overline{\nu})(\Delta)$ for some arbitrary Ξ. We now exploit the fact that 'future worlds' may contain arbitrary new type variables. Pick $\alpha_0 \notin \Xi$; it suffices to show that

$$\begin{array}{l}([\![I'_1]\!], [\![I'_2]\!]) \in \\ \quad [\![(1 \to \alpha) \times (\alpha \to 1) \times (\alpha \to \mathtt{int})]\!]_{\Xi,\alpha_0,\alpha}(\overline{\nu}, \nu_0, \nu)(\emptyset)\,,\end{array}$$

where ν_0 is defined as in the previous example, and where $\nu = \nu_{(\Xi,\alpha_0)}(\alpha_0\,\mathtt{ref})$ is the semantic closed type corresponding to the syntactic type $\alpha_0\,\mathtt{ref}$.

From here, the most interesting part of the proof is the relatedness of the two implementations of the operation for creating a new counter. The core of the proof obligation is the following: given $[\Xi'|\overline{\nu}'|\Delta'] \sqsupseteq [(\Xi,\alpha_0,\alpha)|(\overline{\nu},\nu_0,\nu)|\emptyset]$, states $(s, s') \in [\![\Delta']\!]^S_{\Xi'}(\overline{\nu}')$, and continuations $(k, k') \in [\![\alpha]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$, we must show that $k\,[\![\mathtt{ref}\,0]\!]^s_\emptyset$ and $k'\,[\![\mathtt{ref}\,0]\!]^{s'}_\emptyset$ are both $\bot$ or contain the same integer component. But the characterization of $[\![\alpha]\!]^K_{\Xi'}(\overline{\nu}')(\Delta')$ in Figure 8 involves a quantification over all $\Delta'' \sqsupseteq \Delta'$: we can exploit that quantification by choosing $\Delta'' = \Delta'[l \mapsto \alpha_0]$ where l is the smallest number not in the domain of Δ'. The result easily follows.

Example 21. As in Crary and Harper [13], we can introduce the usual encoding of existential types by means of universal types:

$$\exists\alpha.\tau = \forall\beta.(\forall\alpha.\,\tau \to \beta) \to \beta\,.$$

We then revisit the previous example: the type

$$\tau_{\mathrm{m}} = \exists\alpha.\,(1 \to \alpha) \times (\alpha \to 1) \times (\alpha \to \mathtt{int})$$

can be used to model imperative counter modules.

Consider the following two module implementations, i.e., closed terms of type τ_{m}:

$$J_1 = \Lambda\beta.\lambda c.\, c[\mathtt{int\,ref}]I'_1 \quad \text{and} \quad J_2 = \Lambda\beta.\lambda c.\, c[\mathtt{int\,ref}]I'_2$$

(where I'_1 and I'_2 are defined in the previous example). We can use Corollary 18 to prove that J_1 and J_2 are contextually equivalent. The reasoning is essentially as in the previous example, except that the 'answer type' is now a universally quantified type variable β instead of the fixed type `int`.

Example 22. One can alternatively implement an imperative counter module by means of a local reference and two closures. Consider the type $\tau_{\mathrm{lr}} = 1 \to ((1 \to 1) \times (1 \to \mathtt{int}))$ and the two counter implementations

$$\begin{aligned} J &= \lambda x : 1.\, \mathtt{let}\, r \;=\; \mathtt{ref}\, 0\, \mathtt{in}\; (\lambda y : 1.\, r \;:=\, !r + 1,\; \lambda y : 1.\, !r) \\ J' &= \lambda x : 1.\, \mathtt{let}\, r \;=\; \mathtt{ref}\, 0 \\ &\qquad\qquad \mathtt{in}\; (\lambda y : 1.\, r \;:=\, !r - 1,\; \lambda y : 1.\, -(!r)) \end{aligned}$$

where the `let ... in` construct is syntactic sugar for a β-redex in the usual way. Both J and J' are closed terms of type τ_{lr}, and we can use Corollary 18 to show that the two terms are contextually equivalent. As in Example 20, the proof involves introducing a new type variable α_0, interpreted by ν_0.

6. Conclusion and Future Work

We have given a first relationally parametric possible world semantics for a call-by-value higher-order language with impredicative polymorphism, general references, and recursive types. In particular, we have discovered a technical challenge in establishing the existence of the requisite relational interpretations of types and solved the problem of existence by a novel model of references using a semantic notion of location that permits a useful approximation relation. We are convinced that the technical challenge is a real one and think that the reason it has not been observed before when modelling references with domains is that it only shows up when one insists on modeling *open* types (as needed for parametricity).

As already mentioned, the logical relations suffice for proving parametricity results for a language with recursive types and general references. They are, however, not tailored for maximal 'proof strength', rather the focus is on the underlying semantic challenges. In particular, reasoning about local state is not in general possible, we may, e.g., not prove 'garbage collection' of unused references. We plan to extend and combine the present work with earlier work on reasoning about local state [11] — this allows for formal proofs that two implementations of an abstract type using local state in different ways are related. Indeed in [12], the first author and Nina Bohr extended the techniques in [11] to a language with impredicative polymorphism and references to *closed* types (closed to avoid the technical challenges addressed in this paper), and were, e.g., able to prove two implementations of an abstract stack type related, one implementation using an ML-style list and the other using a linked list implementation for the stack [12, Sec. 5].

Finally, recent work [3] by Ahmed, Dreyer and Rossberg came to our attention after writing this paper. They too provide a relationally parametric possible world semantics of a similar language, but using a step-indexed approach rather than a domain theoretic. Also their worlds are more flexible and hence applicable to more examples. Indeed, their work extend ideas from the aforementioned work [11] but does so in a step-indexed fashion.

References

[1] M. Abadi and G. Plotkin. A per model of polymorphism and recursive types. In *Proceedings of LICS*, pages 355–365, 1990.

[2] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *Proc. of ESOP*, pages 69–83, 2006.

[3] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. To appear at POPL 2009.

[4] R. M. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1):55–85, 1991.

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

[6] A. W. Appel, P.-A. Melliès, C. D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proc. of POPL*, pages 109–122, 2007.

[7] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of TLCA*, volume 3461 of *LNCS*, 2005.

[8] G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *Proc. of HOOTS*, volume 41 of *ENTCS*, 2000.

[9] L. Birkedal and R. E. Møgelberg. Categorical models of Abadi-Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15(4):709–772, 2005.

[10] L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 2(5:1):1–33, 2006.

[11] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In *Proc. of APLAS*, pages 79–96, 2006.

[12] N. Bohr and L. Birkedal. Relational parametricity for recursive types and references of closed types. Technical report, IT University of Copenhagen, 2007. A Chapter in Nina Bohr's Ph.D. dissertation 2007.

[13] K. Crary and R. Harper. Syntactic logical relations for polymorphic and recursive types. *ENTCS*, 172:259–299, 2007.

[14] A. Filinski. On the relations between monadic semantics. *Theoretical Computer Science*, 375(1–3):41–75, 2007.

[15] M. Hasegawa. Relational parametricity and control. *Logical Methods in Computer Science*, 2(3):1–22, 2006.

[16] P. Johann. On proving the correctness of program transformations based on free theorems for higher-order polymorphic calculi. *Mathematical Structures in Computer Science*, 10(2):201–229, 2005.

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

[18] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In *Proc. of ESOP*, pages 146–161, 2006.

[19] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. of POPL*, pages 141–152, 2006.

[20] S. B. Lassen and P. B. Levy. Normal form bisimulation for parametric polymorphism. To appear at LICS 2008.

[21] S. B. Lassen and P. B. Levy. Typed normal form bisimulation. In *Proc. of CSL*, volume 4646 of *LNCS*, pages 283–297, 2007.

[22] P. Levy. Possible world semantics for general storage in call-by-value. In *Proc. of CSL*, volume 2471 of *LNCS*, pages 232–246, 2002.

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

82–91, 2005.

[24] R. Møgelberg. Interpreting polymorphic FPC into domain theoretic models of parametric polymorphism. In *Proc. of ICALP*, pages 372–383, 2006.

[25] R. Møgelberg and A. Simpson. Relational parametricity for computational effects. In *Proc. of LICS*, pages 346–355, 2007.

[26] R. Møgelberg and A. Simpson. Relational parametricity for control considered as a computational effect. In *Proc. of MFPS*, ENTCS, pages 295–312, 2007.

[27] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[28] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[29] A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in computer Science*, 10:321–359, 2000.

[30] A. M. Pitts. *Advanced Topics in Types and Programming Languages*, chapter Typed Operational Reasoning. The MIT Press, 2005.

[31] A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proceedings of MFPS*, volume 711 of *LNCS*, pages 122–141, 1993.

[32] G. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, Feb. 1993.

[33] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA*, volume 664 of *LNCS*, pages 361–375, 1993.

[34] J. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.

[35] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Proc. of POPL*, pages 161–172, 2007.

[36] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *Proc. of POPL*, pages 63–74, 2005.

[37] P. Wadler. Theorems for free! In *4th Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359, 1989.