A Typed Semantics for Languages with Higher-Order Store and Subtyping

JAN SCHWINGHAMMER
INFORMATICS, UNIVERSITY OF SUSSEX, BRIGHTON, UK
`j.schwinghammer@sussex.ac.uk`

ABSTRACT. We consider a call-by-value language, with higher-order functions, records, references to values of arbitrary type, and subtyping. We adapt an intrinsically typed denotational model for a similar language based on a possible-world semantics, recently given by Levy [29], and relate it to an untyped model by a logical relation. Following the methodology of Reynolds [45], this relation is used to establish coherence of the typed semantics, with a coercion interpretation of subtyping. Moreover, we demonstrate that this technique scales to ML-like polymorphic type schemes. We obtain a typed denotational semantics of (imperative) object-oriented languages, both class-based and object-based ones.

Contents

1 Introduction

Languages such as Standard ML [32] and Scheme [4] allow to store values of arbitrary types, including function types. Essentially the same effect is pervasive in object-based languages (see [1, 44]), where objects are created on-the-fly and arbitrary method code needs to be kept in the store. This feature is often referred to as *higher-order store* or *general references*, and complicates the semantics (and logics) of such languages considerably [42]: Besides introducing recursion to the language [28], higher order store in fact requires the semantic domain to be defined by a *mixed-variant* recursive equation. So far, only few models of (typed) languages with general references appeared in the literature [5, 6, 29], and most of the work done on semantics of storage does not readily apply to languages with higher-order store [48].

In a recent paper, Paul Levy proposed a typed semantics for a language with higher-order functions and higher-order store [29]. This is a possible worlds model, explicating the dynamic allocation of new (typed) storage locations in the course of a computation. We recall this model below, and extend it to accommodate subtyping by using coercion maps. In the terminology of Reynolds [45], we obtain an *intrinsic* semantics: Meaning is given to derivations of typing judgements, rather than to terms, with the consequence that

- ill-typed phrases are meaningless,
- terms satisfying several judgements will be assigned several meanings, and
- coherence between the meaning of several derivations of the *same* judgement must be established.

Due to the addition of subtyping to Levy's model, derivations are indeed no longer unique and we must prove coherence. A standard approach for such proofs is to transform derivations into a normal form while preserving their semantics. This can be quite involved, even for purely functional languages (see, e.g., [12, 33]).

In contrast to intrinsic semantics, an *extrinsic* semantics gives meaning to *all* terms. Types (and typing judgements) are interpreted as, e.g., (admissible) predicates or partial equivalence relations over an untyped model. Usually, the interpretation of subtyping is straightforward in such models. In [45], Reynolds uses a logical relation between intrinsic and extrinsic cpo models of a lambda calculus with subtyping (but no state) to prove coherence. The proof essentially relies on the fact that (the denotations of) all derivations of a judgement $\Gamma \triangleright e : A$ are related to the denotation $[\![e]\!]$ of e in the untyped model underlying the extrinsic semantics, via the *basic lemma* of logical relations (e.g., [33]). A family of retractions between intrinsic and extrinsic semantics is then used to obtain the meaning of $[\![\Gamma \triangleright e : A]\!]$ in terms of Γ, $[\![e]\!]$ and A alone, i.e., independent of any particular derivation of the judgement.

We apply the same ideas to obtain a coherence proof for the language considered here. Two modifications have to be made: Firstly, because of the indexing by worlds we use a *Kripke logical relation* [34] to relate intrinsic and extrinsic semantics — this is straightforward. Secondly, due to the mixed-variant recursion forced by the higher-order store we can no longer use induction over the type structure to establish properties of the relations. In fact even the existence of the Kripke logical relation requires a non-trivial proof — we use the framework of Pitts [40] to deal with this complication.

While the combination of higher-order storage and subtyping is interesting in its own right, we see the current work as a step toward our longer-term goal of investigating logics for languages involving higher-order store. In particular, we are interested in semantics and reasoning principles for object-oriented programs, and it should be noted that a number of object encodings used a target language similar to the one considered here [2, 26, 10]. Some evidence that the model of this paper can indeed serve as basis for such logics is provided: We give a semantics to the object calculus of Abadi and Cardelli [1], and to a simple class-based language. This is done using a typed variant of Kamin and Reddy's "closure model" [26]. To the best of our knowledge this is the first (intrinsically typed) domain-theoretic model of the imperative object calculus.

Previously we have given a denotational semantics for a logic of objects [3], where an untyped cpo model was used [43]. This logic has a built-in notion of invariance which makes it very similar to a type system. The semantic structure of function types used in [43] very much resembles that of various possible worlds models for languages with dynamic allocation [29, 41, 47]. We compare the semantics of *loc. cit.* with an extrinsic semantics derived from the Kripke logical relation, again following a construction of Reynolds [45].

Finally, we show that the theory smoothly extends to a model of general references and predicative, bounded polymorphic types. After establishing coherence of this system, we use the model to sketch a semantics of generic classes, similar to those found in the Java language [11].

In summary, our technical contributions here are

- we present a model of a language that includes general references, subtyping and bounded "let"-polymorphism,
- we successfully apply the ideas of Reynolds [45] to prove coherence of the interpretation, and
- we provide the first (intrinsically typed) model of the imperative object calculus of Abadi and Cardelli [1], based on cpos.

In addition, we shed some light on a construction used to establish the existence of store specifications in [43].

TABLE 1. Typing

$$\frac{\Gamma \triangleright e : A \quad A \prec: B}{\Gamma \triangleright e : B} \qquad \frac{x{:}A \in \Gamma}{\Gamma \triangleright x : A}$$

$$\frac{\Gamma \triangleright e_1 : B \quad \Gamma, x{:}B \triangleright e_2 : A}{\Gamma \triangleright \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 : A} \qquad \frac{}{\Gamma \triangleright \mathsf{true} : \mathsf{bool}}$$

$$\frac{\Gamma \triangleright x : \mathsf{bool} \quad \Gamma \triangleright e_1 : A \quad \Gamma \triangleright e_2 : A}{\Gamma \triangleright \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A} \qquad \frac{}{\Gamma \triangleright \mathsf{false} : \mathsf{bool}}$$

$$\frac{\Gamma \triangleright x_i : A_i \quad \forall i \in I}{\Gamma \triangleright \{\mathsf{m}_i = x_i\}_{i \in I} : \{\mathsf{m}_i : A_i\}_{i \in I}} \qquad \frac{\Gamma \triangleright x : \{\mathsf{m}_i : A_i\}_{i \in I}}{\Gamma \triangleright x.\mathsf{m}_j : A_j}\ (j \in I)$$

$$\frac{\Gamma, x{:}A \triangleright e : B}{\Gamma \triangleright \lambda x.e : A \Rightarrow B} \qquad \frac{\Gamma \triangleright x : A \Rightarrow B \quad \Gamma \triangleright y : A}{\Gamma \triangleright x(y) : B}$$

$$\frac{\Gamma \triangleright x : A}{\Gamma \triangleright \mathsf{new}_A\ x : \mathsf{ref}\ A} \qquad \frac{\Gamma \triangleright x : \mathsf{ref}\ A}{\Gamma \triangleright \mathsf{deref}\ x : A}$$

$$\frac{\Gamma \triangleright x : \mathsf{ref}\ A \quad \Gamma \triangleright y : A}{\Gamma \triangleright x{:=}y : \mathbf{1}}$$

STRUCTURE OF THE REPORT. In the next section, language and type system are introduced. Then, in Sects. 3 and 4, typed and untyped models are presented. The logical relation is defined next, and retractions between types of the intrinsic semantics and the untyped value space are used to prove coherence in Section 6. In Section 7 both a derived per semantics and the relation to our earlier work on an interpretation of objects are discussed. Section 8 presents the applications of the theory, providing a semantics of classes and objects, as well as an example specification and verification of a non-trivial program. In Section 9 the type system is enriched with (predicative) polymorphism and proved useful in obtaining a semantics of generic collection classes. Finally, Section 10 discusses related work.

2 Language

We consider a single base type of booleans, bool, records $\{\mathsf{m}_i : A_i\}_{i \in I}$ with labels $\mathsf{m} \in \mathbb{L}$, and function types $A \Rightarrow B$. We set $\mathbf{1} \overset{def}{=} \{\}$ for the (singleton) type of empty records. Finally, we have a type ref A of mutable references to values of type A. Term forms include constructs for creating, dereferencing and updating of storage locations. The syntax of types and terms is given by the grammar:

$$
\begin{aligned}
A, B \in \mathit{Type} &::= \mathsf{bool} \mid \{\mathsf{m}_i : A_i\}_{i \in I} \mid A \Rightarrow B \mid \mathsf{ref}\ A \\
v \in \mathit{Val} &::= x \mid \mathsf{true} \mid \mathsf{false} \mid \{\mathsf{m}_i = x_i\}_{i \in I} \mid \lambda x.e \\
e \in \mathit{Exp} &::= v \mid \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 \mid \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid x.\mathsf{m} \mid x(y) \\
&\quad \mid \mathsf{new}_A\ x \mid \mathsf{deref}\ x \mid x{:=}y
\end{aligned}
$$

Subterms in most of these term forms are restricted to variables in order to simplify the statement of the semantics in the next section: There, we can exploit the fact that subterms that exhibit side-effects only appear in the let-construct. However, in subsequent examples we will use a more generous syntax. The reduction of such syntax sugar to the expressions above should always be immediate.

The subtyping relation $A \prec: B$ is the least reflexive and transitive relation closed under the rules

$$
\frac{A_i \prec: A'_i \quad \forall i \in I' \quad I' \subseteq I}{\{\mathsf{m}_i : A_i\}_{i \in I} \prec: \{\mathsf{m}_i : A'_i\}_{i \in I'}} \qquad \frac{A' \prec: A \quad B \prec: B'}{A{\Rightarrow}B \prec: A'{\Rightarrow}B'}
$$

Note that there is no rule for reference types as these need to be invariant, i.e., $\mathsf{ref}\ A \prec: \mathsf{ref}\ B$ only if $A \equiv B$. A type inference system is given in Table 1, where contexts Γ are finite sets of variable-type pairs, with each variable occurring at most once. As usual, in writing $\Gamma, x{:}A$ we assume x does not occur in Γ. A subsumption rule is used to for subtyping of terms.

3 Intrinsic Semantics

In this section we recall the possible worlds model of [29]. Its extension with records is straightforward, and we interpret the subsumption rule using coercion maps.

WORLDS. For each $A \in \mathit{Type}$ let Loc_A be mutually disjoint, countably infinite sets of *locations*. We let l range over $\mathsf{Loc} \overset{def}{=} \bigcup_{A \in \mathit{Type}} \mathsf{Loc}_A$, and may use the notation l_A to emphasize that $l \in \mathsf{Loc}_A$. A *world* w is a finite set of locations $l_A \in \mathsf{Loc}$. A world w' extends w, written $w' \geq w$, if $w' \supseteq w$. We write $\mathcal{W} = (\mathcal{W}, \leq)$ for the poset of worlds.

SEMANTIC DOMAIN. Let **pCpo** be the category of cpos (not necessarily containing a least element) and partial continuous functions. For a partial continuous function f we write $f(a)\downarrow$ if the application is defined, and $f(a)\uparrow$ otherwise. Let **Cpo** be the subcategory of **pCpo** where the morphisms are *total* continuous functions.

Informally, a world describes the shape of the store, i.e., the number of locations of each type allocated in the store. In the semantics we want a cpo S_w of w-stores for each $w \in \mathcal{W}$, and a cpo $[\![A]\!]_w$ of values of type A.

In fact, we require that each $[\![A]\!]$ denotes a co-variant functor from $\mathcal{W}$ to **Cpo**, formalising the intuition that values can always be used with larger stores.[1] We write the image of $w \leq w'$ under $[\![A]\!]$ as $[\![A]\!]_w^{w'}$.

The cpo of w-stores is defined as $S_w = \prod_{l_A \in w} [\![A]\!]_w$. For worlds $w \in \mathcal{W}$, $[\![\mathsf{bool}]\!]_w = \mathsf{BVal}$ denotes the set $\{\mathit{true}, \mathit{false}\}$ of truth values considered as flat cpo, and similarly, $[\![\mathsf{ref}\ A]\!]_w = \{l_A \mid l_A \in w\}$ is the discretely ordered cpo of A-locations allocated in w-stores. Further, $[\![\{\mathsf{m}_i : A_i\}_{i \in I}]\!]_w = \{\!|\mathsf{m}_i : [\![A_i]\!]_w\,|\!\}$ is the cpo of records $\{\!|\mathsf{m}_i = a_i|\!\}_{i \in I}$ with component m_i in $[\![A_i]\!]_w$, ordered pointwise. On morphisms $w \leq w'$, $[\![\mathsf{bool}]\!]_w^{w'} = \mathsf{id}_{\mathsf{BVal}}$ is the identity map, and $[\![\mathsf{ref}\ A]\!]_w^{w'}$ is the inclusion $[\![\mathsf{ref}\ A]\!]_w \subseteq [\![\mathsf{ref}\ A]\!]_{w'}$. For records, $[\![\{\mathsf{m}_i : A_i\}]\!]_w^{w'} = \lambda r.\{\!|\mathsf{m}_i = [\![A_i]\!]_w^{w'}(r.\mathsf{m}_i)|\!\}$ acts pointwise on the components. The type of functions $A \Rightarrow B$ is the most interesting, since it involves the store S:

$$[\![A \Rightarrow B]\!]_w \quad = \quad \prod_{w' \geq w} (S_{w'} \times [\![A]\!]_{w'} \rightharpoonup \sum_{w'' \geq w'} (S_{w''} \times [\![B]\!]_{w''})) \qquad (1)$$

This says that a function $f \in [\![A \Rightarrow B]\!]_w$ may be applied in any future (larger) store w' to a w'-store s and value $v \in [\![A]\!]_{w'}$. The computation $f_{w'}(s, v)$ may allocate new storage, and upon termination it yields a store and value in a yet larger world $w'' \geq w'$. For a morphism $w \leq w'$, $[\![A \Rightarrow B]\!]_w^{w'}(f) = \lambda_{w'' \geq w'} f_{w''}$ is the restriction to worlds $w'' \geq w'$.

Equation (1) clearly shows the effect of allowing higher order store: Since functions $A \Rightarrow B$ can also be stored, S and $[\![A \Rightarrow B]\!]$ are mutually recursive. Due to the use of S in both positive and negative positions in (1) a mixed-variant domain equation for S must be solved. To this end, in [29] a *bilimit-compact* category $\mathcal{C}$ is considered, i.e.,

- $\mathcal{C}$ is **Cpo**-enriched and each hom-cpo $\mathcal{C}(A, B)$ has a least element $\bot_{A,B}$ s.t. $\bot \circ f = \bot = g \circ \bot$
- $\mathcal{C}$ has an initial object
- in the category $\mathcal{C}^E$ of embedding-projection pairs of $\mathcal{C}$, every ω-chain $\Delta = D_0 \to D_1 \to \ldots$ has an O-colimit [46], i.e., a cocone $\langle e_i, p_i \rangle_{i \in \mathbb{N}} : \Delta \to D$ in $\mathcal{C}^E$ s.t. $\bigsqcup_i e_i \circ p_i = \mathsf{id}_D$ in $\mathcal{C}(D, D)$

It follows that every locally continuous functor $F : \mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$ has a minimal invariant, i.e., an object D in $\mathcal{C}$ s.t. $F(D, D) = D$ (omitting isomorphisms) and id_D is the least fixed point of the continuous endofunction $\delta : \mathcal{C}(D, D) \to \mathcal{C}(D, D)$ given by $\delta(e) = F(e, e)$ [40].

[1] In contrast, S is not assumed to be (co- or contra-variant) functorial: In general there is no obvious way to enlarge or restrict a store [30].

Following [29] the semantics of types can now be obtained as minimal invariant of the locally continuous functor $F : \mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$ (derived from the domain equations for types by separating positive and negative occurrences of the store) given in Table 2. Here, $\mathcal{C}$ is the bilimit-compact category

$$\mathcal{C} \overset{def}{=} \prod_{w \in \mathcal{W}} \mathbf{pCpo} \times \prod_{A \in \mathit{Type}} [\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}] \tag{2}$$

where $[\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}]$ denotes the category with objects the functors $A, B : \mathcal{W} \to \mathbf{Cpo}$ and morphisms the partial natural transformations $\mu : A \dot{\rightharpoonup} B$, i.e., for $A, B : \mathcal{W} \to \mathbf{Cpo}$ the diagram

$$\begin{tikzcd} A_w \arrow[r, harpoon, "\mu_w"] \arrow[d, "A_w^{w'}"'] & B_w \arrow[d, "B_w^{w'}"] \\ A_{w'} \arrow[r, harpoon, "\mu_{w'}"'] & B_{w'} \end{tikzcd} \tag{3}$$

commutes. The first component of the product in (2) is used to obtain $S_w \overset{def}{=} D_{Sw}$ from the minimal invariant $D = \langle \{D_{Sw}\}_w, \{D_A\}_A \rangle$, and the second component yields $[\![A]\!] \overset{def}{=} D_A$.

In fact, for every type $A \in \mathit{Type}$ the minimal invariant D provides isomorphisms $F(D, D)_A = D_A$ in the category $[\mathcal{W}, \mathbf{Cpo}]$ of functors $\mathcal{W} \to \mathbf{Cpo}$ and *total* natural transformations.

SEMANTICS. Each subtyping derivation $A \prec: B$ determines a *coercion*, which is in fact a (total) natural transformation from $[\![A]\!]$ to $[\![B]\!]$, defined in Table 3: We follow the notation of [45] and write $\mathcal{P}(J)$ to distinguish a *derivation* of judgement J from the judgement itself.

In the following we write $[\![\Gamma]\!]_w$ for the set of environments, i.e., maps from variables to $\bigcup_A [\![A]\!]_w$ s.t. $\rho(x) \in [\![A]\!]_w$ for all $x{:}A \in \Gamma$. For $w \le w'$, $[\![\Gamma]\!]_w^{w'}(\rho)$ denotes the environment such that $[\![\Gamma]\!]_w^{w'}(\rho)(x) = [\![A]\!]_w^{w'}(\rho(x))$ for $x{:}A$ in Γ. The semantics of (derivations of) typing judgments can now be presented,

$$[\![\Gamma \triangleright e : A]\!]_w : [\![\Gamma]\!]_w \to S_w \rightharpoonup \sum_{w' \ge w} (S_{w'} \times [\![A]\!]_{w'})$$

As observed in Levy's paper, each *value* $\Gamma \triangleright v : A$ determines a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$ in $[\mathcal{W}, \mathbf{Cpo}]$. Here this is a consequence of the fact that (i) values do not affect the store and (ii) coercion maps determine (total) natural transformations. We make use of this fact in the statement of the semantics. For example, in the case of records we do not have to fix an order for the evaluation of the components.

The semantics of subtyping judgements is used for the interpretation

TABLE 2. Defining $F : \mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$

On $\mathcal{C}$-objects D, E

$$
\begin{array}{lcl}
F(D,E)_{Sw} & = & \prod_{l_A \in w} E_{Aw} \\[1ex]
F(D,E)_{\mathsf{bool}w} & = & \mathsf{BVal} = \{\mathit{true}, \mathit{false}\} \\
F(D,E)_{\mathsf{bool}(w \le w')} & = & \mathsf{id}_{\mathsf{BVal}} \\[1ex]
F(D,E)_{\{\mathsf{m}_i : A_i\}w} & = & \{\!|\mathsf{m}_i : E_{A_i w}|\!\} \\
F(D,E)_{\{\mathsf{m}_i : A_i\}(w \le w')} & = & \lambda r. \{\!|\mathsf{m}_i = E_{A_i(w \le w')}(r.\mathsf{m}_i)|\!\} \\[1ex]
F(D,E)_{A \Rightarrow B w} & = & \prod_{w' \ge w} (D_{Sw'} \times D_{Aw'} \rightharpoonup \sum_{w'' \ge w'} (E_{Sw''} \times E_{Bw''})) \\
F(D,E)_{A \Rightarrow B(w \le w')} & = & \lambda f \lambda w'' \ge w'. f_{w''} \\[1ex]
F(D,E)_{\mathsf{ref}\ Aw} & = & \{l_A \mid l_A \in w\} \\
F(D,E)_{\mathsf{ref}\ A(w \le w')} & = & \lambda l.l
\end{array}
$$

On $\mathcal{C}$-morphisms $h : D' \longrightarrow D$ and $k : E \longrightarrow E'$ by

$$
\begin{array}{lcl}
F(h,k)_{Sw} & = & \lambda s. \begin{cases} l_A \mapsto k_{Sw}(s)_{l_A} & \text{if } k_{Sw}(s)_{l_A}\!\downarrow \text{ for all } l_A \in w \\ \text{undefined} & \text{otherwise} \end{cases} \\
F(h,k)_{\mathsf{bool}w} & = & \mathsf{id}_{\mathsf{BVal}} \\[1ex]
F(h,k)_{\{\mathsf{m}_i : A_i\}w} & = & \lambda r. \begin{cases} \{\!|\mathsf{m}_i = k_{A_i w}(r.\mathsf{m}_i)|\!\} & \text{if } k_{A_i w}(r.\mathsf{m}_i)\!\downarrow \text{ for all } i \\ \text{undefined} & \text{otherwise} \end{cases} \\
F(h,k)_{A \Rightarrow B w} & = & \lambda f \lambda w' \ge w\, \lambda \langle s, a \rangle. \\
 & & \begin{cases} \langle w'', \langle k_{Sw''}(s''), k_{Bw''}(b) \rangle\rangle \\ \qquad \text{if } h_{Sw'}(s)\!\downarrow \text{ and } h_{Aw'}(a)\!\downarrow \text{ and} \\ \qquad f_{w'}(h_{Sw'}(s), h_{Aw'}(a)) = \langle w'', \langle s'', b \rangle\rangle\!\downarrow \\ \qquad \text{and } k_{Sw''}(s'')\!\downarrow \text{ and } k_{Bw''}(b)\!\downarrow \\ \text{undef. otherwise} \end{cases} \\
F(h,k)_{\mathsf{ref}\ Aw} & = & \lambda l.l
\end{array}
$$

of the subsumption rule,

$$
\left[\!\!\left[\frac{\mathcal{P}(\Gamma \rhd e : A) \;\; \mathcal{P}(A \prec: B)}{\Gamma \rhd e : B} \right]\!\!\right]_w \rho s
$$
$$
= \begin{cases} \langle w', \langle s', [\![\mathcal{P}(A \prec: B)]\!]_{w'}\, a \rangle\rangle & \text{if } [\![\mathcal{P}(\Gamma \rhd e : A)]\!]_w\, \rho s = \langle w', \langle s', a \rangle\rangle\!\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}
$$

As explained above, the semantics of functions is parameterised over extensions of the current world w,

$$
\left[\!\!\left[\frac{\mathcal{P}(\Gamma, x : A \rhd e : B)}{\Gamma \rhd \lambda x.e : A \Rightarrow B} \right]\!\!\right]_w \rho s
$$
$$
= \langle w, \langle s, \lambda w' \ge w \lambda \langle s', a \rangle. \, [\![\mathcal{P}(\Gamma, x{:}A \rhd e : B)]\!]_{w'} \, ([\![\Gamma]\!]_w^{w'}\, \rho)[x := a]\, s' \rangle\rangle
$$

TABLE 3. Coercion maps

$$\left[\!\!\left[\frac{}{A \prec: A}\right]\!\!\right]_w = \mathsf{id}_{[\![A]\!]_w}$$

$$\left[\!\!\left[\frac{\mathcal{P}(A \prec: A') \quad \mathcal{P}(A' \prec: B)}{A \prec: B}\right]\!\!\right]_w = [\![\mathcal{P}(A' \prec: B)]\!]_w \circ [\![\mathcal{P}(A \prec: A')]\!]_w$$

$$\left[\!\!\left[\frac{I' \subseteq I \quad \mathcal{P}(A_i \prec: A'_i)\ \forall i \in I'}{\{\mathsf{m}_i : A_i\}_{i\in I} \prec: \{\mathsf{m}_i : A'_i\}_{i \in I'}}\right]\!\!\right]_w = \lambda r.\{\!|\mathsf{m}_i = [\![\mathcal{P}(A_i \prec: A'_i)]\!]_w\,(r.\mathsf{m}_i)|\!\}_{i\in I'}$$

$$\left[\!\!\left[\frac{\mathcal{P}(A' \prec: A) \quad \mathcal{P}(B \prec: B')}{A \Rightarrow B \prec: A' \Rightarrow B'}\right]\!\!\right]_w$$

$$= \lambda f \lambda_{w' \geq w}\, \lambda\langle s, x\rangle. \begin{cases} \langle w'', \langle s', [\![\mathcal{P}(B \prec: B')]\!]_{w''}\, x'\rangle\rangle \\ \qquad\qquad \text{if } f_{w'}\langle s, [\![\mathcal{P}(A' \prec: A)]\!]_{w'}\,(x)\rangle = \langle w'', \langle s', x'\rangle\rangle\downarrow \\ \text{undefined otherwise} \end{cases}$$

Function application is

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \rhd x : A \Rightarrow B) \quad \mathcal{P}(\Gamma \rhd y : A)}{\Gamma \rhd x(y) : B}\right]\!\!\right]_w \rho s = f_w(s, a)$$

where the premiss of the rule yields $\langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : A \Rightarrow B)]\!]_w\, \rho s$ and $\langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd y : A)]\!]_w\, \rho s$. Most of the remaining cases are similarly straightforward; Tables 4 and 5 contain the complete definition.

4 An Untyped Semantics

We give an untyped semantics of the language in the (bilimit-compact) category **pCpo**. Let Val satisfy

$$\mathsf{Val} \;=\; \mathsf{BVal} + \mathsf{Loc} + \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) + (\mathsf{St} \times \mathsf{Val} \rightharpoonup \mathsf{St} \times \mathsf{Val}) \qquad (4)$$

where $\mathsf{St} \overset{def}{=} \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Val})$ denotes the cpo of records with labels from Loc, ordered by $r_1 \sqsubseteq r_2$ iff $\mathsf{dom}(r_1) = \mathsf{dom}(r_2)$ and $r_1.\mathsf{m} \sqsubseteq r_2.\mathsf{m}$ for all $\mathsf{m} \in \mathsf{dom}(r_1)$. The interpretation of terms, $[\![e]\!] : \mathsf{Env} \to \mathsf{St} \rightharpoonup \mathsf{St} \times \mathsf{Val}$, is essentially straightforward, typical cases are those of abstraction and application:

$$[\![\lambda x.e]\!]\,\eta\sigma = \langle \sigma, \lambda\langle\sigma', v\rangle.\ [\![e]\!]\,\eta[x := v]\,\sigma'\rangle$$

$$[\![x(y)]\!]\,\eta\sigma = \begin{cases} \eta(x)\langle\sigma, \eta(y)\rangle & \text{if } \eta(x) \in [\mathsf{St} \times \mathsf{Val} \rightharpoonup \mathsf{St} \times \mathsf{Val}] \text{ and } \eta(y)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

Compared to the intrinsic semantics of the previous section, there are now many more possibilities of undefinedness if things "go wrong", e.g., if evaluation of x in $x(y)$ does not yield a function value.

The semantics of new_A may be slightly surprising as there is still some

TABLE 4. Semantics of typing judgements

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright e : A) \quad \mathcal{P}(A \prec: B)}{\Gamma \triangleright e : B}\right]\!\!\right]_w \rho s = \begin{cases} \langle w', \langle s', [\![\mathcal{P}(A \prec: B)]\!]_{w'}\, a\rangle\rangle & \text{if } [\![\mathcal{P}(\Gamma \triangleright e : A)]\!]_w\, \rho s = \langle w', \langle s', a\rangle\rangle \downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\left[\!\!\left[\frac{}{\Gamma \triangleright x : A}\right]\!\!\right]_w \rho s = \langle w, \langle s, \rho(x)\rangle\rangle$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright e_1 : B) \quad \mathcal{P}(\Gamma, x{:}B \triangleright e_2 : A)}{\Gamma \triangleright \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 : A}\right]\!\!\right]_w \rho s = \begin{cases} \mathcal{P}([\![\Gamma, x{:}B \triangleright e_2 : A]\!])_{w'}\,([\![\Gamma]\!]_w^{w'}\, \rho)[x := b]\, s' \\ \qquad\qquad \text{if } [\![\mathcal{P}(\Gamma \triangleright e_1 : B)]\!]_w\, \rho s = \langle w', \langle s', b\rangle\rangle \downarrow \\ \text{undefined} \quad \text{otherwise} \end{cases}$$

$$\left[\!\!\left[\frac{}{\Gamma \triangleright \mathsf{true} : \mathsf{bool}}\right]\!\!\right]_w \rho s = \langle w, \langle s, \mathit{true}\rangle\rangle$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright x : \mathsf{bool}) \quad \mathcal{P}(\Gamma \triangleright e_i : A) \quad i = 1,2}{\Gamma \triangleright \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A}\right]\!\!\right]_w \rho s = \begin{cases} [\![\mathcal{P}(\Gamma \triangleright e_1 : A)]\!]_w\, \rho s & \text{if } [\![\mathcal{P}(\Gamma \triangleright x : \mathsf{bool})]\!]_w\, \rho s = \langle w, \langle s, \mathit{true}\rangle\rangle \\ [\![\mathcal{P}(\Gamma \triangleright e_2 : A)]\!]_w\, \rho s & \text{if } [\![\mathcal{P}(\Gamma \triangleright x : \mathsf{bool})]\!]_w\, \rho s = \langle w, \langle s, \mathit{false}\rangle\rangle \end{cases}$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright x_i : A_i) \quad \forall i \in I}{\Gamma \triangleright \{\mathsf{m}_i = x_i\}_{i\in I} : \{\mathsf{m}_i : A_i\}_{i\in I}}\right]\!\!\right]_w \rho s = \langle w, \langle s, \{\!|\mathsf{m}_i = a_i|\!\}_{i\in I}\rangle\rangle$$
$$\text{where } \langle w, \langle s, a_i\rangle\rangle = [\![\mathcal{P}(\Gamma \triangleright x_i : A_i)]\!]_w\, \rho s$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright x : \{\mathsf{m}_i : A_i\}_{i\in I})}{\Gamma \triangleright x.\mathsf{m}_i : A_i}\right]\!\!\right]_w \rho s = \langle w, \langle s, a.\mathsf{m}\rangle\rangle$$
$$\text{where } \langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \triangleright x : \{\mathsf{m}_i : A_i\})]\!]_w\, \rho s$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma, x : A \triangleright e : B)}{\Gamma \triangleright \lambda x.e : A \Rightarrow B}\right]\!\!\right]_w \rho s = \langle w, \langle s, \lambda w' \geq w \lambda\langle s', a\rangle.\, [\![\mathcal{P}(\Gamma, x{:}A \triangleright e : B)]\!]_{w'}\, ([\![\Gamma]\!]_w^{w'}\, \rho)[x := a]\, s'\rangle\rangle$$

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \triangleright x : A \Rightarrow B) \quad \mathcal{P}(\Gamma \triangleright y : A)}{\Gamma \triangleright x(y) : B}\right]\!\!\right]_w \rho s = f_w(s, a)$$
$$\text{where } \langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \triangleright x : A \Rightarrow B)]\!]_w\, \rho s$$
$$\text{and } \langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \triangleright y : A)]\!]_w\, \rho s$$

TABLE 5. Semantics of typing judgements (continued)

$$\left[\!\left[\frac{\mathcal{P}(\Gamma \rhd x : A)}{\Gamma \rhd \mathsf{new}_A\ x : \mathsf{ref}\ A}\right]\!\right]_w \rho s = \langle w', \langle s', l_A\rangle\rangle$$

$$\text{where } \langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : A)]\!]_w\, \rho s,$$
$$w' = w \cup \{l_A\} \text{ for } l_A \in \mathsf{Loc}_A \setminus \mathsf{dom}(w) \text{ and for all } l' \in w' :$$
$$s'.l' = \begin{cases} [\![A']\!]_w^{w'}\ (s.l') & \text{for } l' \in w \cap \mathsf{Loc}_{A'} \\ [\![A]\!]_w^{w'}\ (a) & \text{for } l' = l_A \end{cases}$$

$$\left[\!\left[\frac{\mathcal{P}(\Gamma \rhd x : \mathsf{ref}\ A)}{\Gamma \rhd \mathsf{deref}\ x : A}\right]\!\right]_w \rho s = \langle w, \langle s, s.l\rangle\rangle$$

$$\text{where } \langle w, \langle s, l\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : \mathsf{ref}\ A)]\!]_w\, \rho s$$

$$\left[\!\left[\frac{\mathcal{P}(\Gamma \rhd x : \mathsf{ref}\ A) \quad \mathcal{P}(\Gamma \rhd y : A)}{\Gamma \rhd x{:=}y : \mathbf{1}}\right]\!\right]_w \rho s = \langle w, \langle \hat{s}, \{\!|\,|\!\}\rangle\rangle$$

$$\text{where } \langle w, \langle s, l\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : \mathsf{ref}\ A)]\!]_w\, \rho s,$$
$$\langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd y : A)]\!]_w\, \rho s \text{ and for } l' \in w :$$
$$\hat{s}.l' = \begin{cases} a & \text{if } l' = l \\ s.l' & \text{if } l' \neq l \end{cases}$$

type information in the choice of locations:

$$[\![\mathsf{new}_A\ x]\!]\,\eta\sigma = \langle \sigma + \{\!|l_A = \eta(x)|\!\}, l_A\rangle \qquad \text{where } l_A \in \mathsf{Loc}_A \setminus \mathsf{dom}(\sigma)$$

if $\eta(x)\downarrow$, and undefined otherwise. Informally, the worlds of the intrinsic semantics are encoded in the domain of untyped stores. Although σ with $\mathsf{dom}(\sigma) = w$ need not necessarily correspond to a (typed) w-store in any sense, this will be the case for stores being derived from well-typed terms. This is one of the results of Section 5 below; see also the discussion in Section 7.1.

The remaining cases of the definition are given in Table 6.

5 A Kripke Logical Relation

While in [45] a logical relation between typed and untyped models was used to establish coherence, here this must be slightly generalised to a Kripke logical relation because of the possible worlds semantics of types. Kripke logical relations have appeared in work on Kripke lambda models [34], and more recently in connection with dynamic name creation in the nu-calculus [51, 22]. Kripke logical relations are not only indexed by types but also by possible worlds, subject to a monotonicity condition (Lemma 5.4 below) stating that related elements remain in relation when moving to a larger world.

In Table 7 we now define such a family of *Type*- and $\mathcal{W}$-indexed relations $R_w^A \subseteq [\![A]\!]_w \times \mathsf{Val}$. Note that the existence of this family R is not

TABLE 6. Interpretation of untyped terms

$$[\![x]\!]\,\eta\sigma = \begin{cases} \langle\sigma, \eta(x)\rangle & \text{if } \eta(x)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2]\!]\,\eta\sigma = \begin{cases} [\![e_2]\!]\,\eta[x := v]\,\sigma' & \text{if } [\![e_1]\!]\,\eta\sigma = \langle\sigma', v\rangle\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{true}]\!]\,\eta\sigma = \langle\sigma, \mathit{true}\rangle$$

$$[\![\mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\,\eta\sigma = \begin{cases} [\![e_1]\!]\,\eta\sigma & \text{if } \eta(x) = \mathit{true}\downarrow \\ [\![e_2]\!]\,\eta\sigma & \text{if } \eta(x) = \mathit{false}\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![\{\mathsf{m}_i = x_i\}]\!]\,\eta\sigma = \begin{cases} \langle\sigma, \{\!|\mathsf{m}_i = \eta(x_i)|\!\}\rangle & \text{if } \eta(x_i)\downarrow \text{ for all } i \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![x.\mathsf{m}]\!]\,\eta\sigma = \begin{cases} \langle\sigma, \eta(x).\mathsf{m}\rangle & \text{if } \eta(x) \in \mathsf{Rec}_{\mathcal{M}}(\mathsf{Val}) \text{ and } \eta(x).\mathsf{m}\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![\lambda x.a]\!]\,\eta\sigma = \langle\sigma, \lambda\langle\sigma', v\rangle.\ [\![a]\!]\,\eta[x := v]\,\sigma'\rangle$$

$$[\![x(y)]\!]\,\eta\sigma = \begin{cases} \eta(x)\langle\sigma, \eta(y)\rangle & \text{if } \eta(x) \in [\mathsf{St}\times\mathsf{Val} \rightharpoonup \mathsf{St}\times\mathsf{Val}] \\ & \text{and } \eta(y)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{new}_A\ x]\!]\,\eta\sigma = \langle\sigma + \{\!|l_A = \eta(x)|\!\}, l_A\rangle,\ \text{where } l_A \in \mathsf{Loc}_A \setminus \mathsf{dom}(\sigma)$$

$$[\![\mathsf{deref}\ x]\!]\,\eta\sigma = \begin{cases} \langle\sigma, \sigma.\eta(x)\rangle & \text{if } \eta(x) \in \mathsf{Loc} \text{ and } \sigma.\eta(x)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$[\![x{:=}y]\!]\,\eta\sigma = \begin{cases} \langle\sigma', \{\!|\,|\!\}\rangle & \text{if } \eta(x) \in \mathsf{Loc}, \sigma.\eta(x)\downarrow \text{ and } \eta(y)\downarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\text{where } \sigma'.l = \begin{cases} \eta(y) & \text{if } l = \eta(x) \\ \sigma.l & \text{otherwise} \end{cases}$$

straightforward: There are both positive and negative occurrences of R^{St}_w in the clause for function types $A \Rightarrow B$. Consequently, R cannot be defined by induction on the type structure, nor does it give rise to a monotone operation (on the complete lattice of admissible predicates).

TABLE 7. Kripke logical relation

$$\langle x,y\rangle \in R_w^{\mathsf{bool}} \quad \overset{def}{\iff} \quad y \in \mathsf{BVal} \ \wedge \ x = y$$

$$\langle r,s\rangle \in R_w^{\{\mathsf{m}_i:A_i\}} \quad \overset{def}{\iff} \quad s \in \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) \ \wedge \ \forall i.\ (s.\mathsf{m}_i\!\downarrow \ \wedge \ \langle r.\mathsf{m}_i, s.\mathsf{m}_i\rangle \in R_w^{A_i})$$

$$\begin{aligned}\langle f,g\rangle \in R_w^{A\Rightarrow B} \quad \overset{def}{\iff} \quad & g \in [\mathsf{St}\times\mathsf{Val} \rightharpoonup \mathsf{St}\times\mathsf{Val}] \ \wedge \\ & \forall w' \geq w\ \forall\langle s,\sigma\rangle \in R_{w'}^{St}\ \forall\langle x,y\rangle \in R_{w'}^{A} \\ & \quad (f_{w'}(s,x)\!\uparrow \ \wedge g(\sigma,y)\!\uparrow) \\ & \vee\ \exists w'' \geq w'\ \exists s' \in S_{w'}\ \exists x' \in [\![B]\!]_{w'}\ \exists\sigma' \in \mathsf{St}\ \exists y' \in \mathsf{Val}. \\ & \qquad (f_{w'}(s,x) = \langle w'', \langle s',x'\rangle\rangle \ \wedge \ g(\sigma,y) = \langle\sigma',y'\rangle \\ & \qquad \wedge\ \langle s',\sigma'\rangle \in R_{w''}^{St} \ \wedge \ \langle x',y'\rangle \in R_{w''}^{B})\end{aligned}$$

$$\langle x,y\rangle \in R_w^{\mathsf{ref}\ A} \quad \overset{def}{\iff} \quad y \in w \cap \mathsf{Loc}_A \ \wedge \ x = y$$

with the auxiliary relation $R_w^{St} \subseteq S_w \times \mathsf{St}$,

$$\langle s,\sigma\rangle \in R_w^{St} \quad \overset{def}{\iff} \quad \mathsf{dom}(s) = w = \mathsf{dom}(\sigma) \ \wedge \ \forall l_A \in w.\ \langle s.l_A, \sigma.l_A\rangle \in R_w^A$$

5.1 Existence of R_w^A

To establish the existence of such a relation one uses Pitts' technique for the bilimit-compact product category $\mathcal{C}\times\mathbf{pCpo}$. Let $G : \mathbf{pCpo}^{op}\times\mathbf{pCpo} \longrightarrow \mathbf{pCpo}$ be the locally continuous functor for which (4) is the minimal invariant,

$$G(D,E) \ = \ \mathsf{BVal} + \mathsf{Loc} + \mathsf{Rec}_{\mathbb{L}}(E) + (\mathsf{Rec}_{\mathsf{Loc}}(D)\times D \rightharpoonup \mathsf{Rec}_{\mathsf{Loc}}(E)\times E)$$

and let F be the functor defined in Table 2 on page 8. Therefore $\langle D, \mathsf{Val}\rangle$ is the minimal invariant of $F\times G$. A (normal) relational structure $\mathcal{R}$ on the category $\mathcal{C}\times\mathbf{pCpo}$, in the sense of [40], is given by the following data.

- For each object $\langle X,Y\rangle$ of $\mathcal{C}\times\mathbf{pCpo}$, let $\mathcal{R}(X,Y)$ consist of the type- and world-indexed families R of admissible relations, where $R_w^A \subseteq X_{Aw}\times Y$ and $R_w^{St} \subseteq X_{Sw}\times\mathsf{Rec}_{\mathsf{Loc}}(Y)$.
- For morphisms $f = \langle f_1, f_2\rangle : \langle X,Y\rangle \to \langle X',Y'\rangle$, and relations $R \in \mathcal{R}(X,Y)$ and $S \in \mathcal{R}(X',Y')$, we define $\langle f_1,f_2\rangle : R \subset S$ *iff* for all $w \in \mathcal{W}$, $A \in$ *Type*, for all $x \in X_{Aw}$, $y \in Y$, $s \in X_{Sw}$ and $\sigma \in \mathsf{Rec}_{\mathsf{Loc}}(Y)$:

$$\langle x,y\rangle \in R_w^A \implies \begin{cases} f_{1\,Aw}(x)\!\uparrow \wedge f_2(y)\!\uparrow \ \text{ or} \\ f_{1\,Aw}(x)\!\downarrow \wedge f_2(y)\!\downarrow \wedge \langle f_{1\,Aw}(x), f_2(y)\rangle \in S_w^A \end{cases}$$

$$\langle s,\sigma\rangle \in R_w^{St} \implies \begin{cases} f_{1\,Sw}(x)\!\uparrow \wedge \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\!\uparrow \ \text{ or} \\ f_{1\,Sw}(x)\!\downarrow \wedge \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\!\downarrow \\ \qquad \wedge\ \langle f_{1\,Sw}(x), \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma)\rangle \in S_w^{St} \end{cases}$$

Firstly, it is easy to see that this relational structure $\mathcal{R}$ on $\mathcal{C}\times\mathbf{pCpo}$ indeed satisfies the axioms of [40]:

(Identity) $\langle \mathsf{id}_X, \mathsf{id}_Y \rangle : R \subset R$ for all objects $\langle X, Y\rangle$ and all $\mathcal{R}$-relations $R \in \mathcal{R}(X,Y)$

(Composition) for composable f, g with $f : S \subset T$ and $g : R \subset S$ we have $f \circ g : R \subset T$

(Normality) if $\mathsf{id} : R \subset S$ and $\mathsf{id} : S \subset R$ then $R = S$

Further, for objects $\langle X, Y\rangle$, $\langle X', Y'\rangle$ and relations $R \in \mathcal{R}(X,Y)$ and $S \in \mathcal{R}(X',Y')$, the subset

$$[R,S] \overset{\mathit{def}}{=} \{\langle f,g\rangle \mid \langle f,g\rangle : R \subset S\}$$

of $[X \rightharpoonup X'] \times [Y \rightharpoonup Y']$ has a least element and is chain-closed:

- Clearly, the pair $\bot = \langle \bot, \bot\rangle$ of maps that are undefined everywhere satisfies $\bot : R \subset S$ and is below every other $f \in [R,S]$.
- Let $\langle f_0, g_0\rangle \sqsubseteq \langle f_1, g_1\rangle \sqsubseteq \ldots$ in $[R,S]$ and $\langle f, g\rangle = \bigsqcup_i \langle f_i, g_i\rangle$. Now suppose $\langle x, y\rangle \in X_{Aw} \times Y$ s.t. $\langle x,y\rangle \in R^A_w$ and $f_{Aw}(x)\downarrow$ ($g(y)\downarrow$, resp.). Then ${f_i}_{Aw}(x)\downarrow$ ($g_i(y)\downarrow$, resp.) for all sufficiently large i, which entails that also $g_i(y)\downarrow$ (${f_i}_{Aw}(x)\downarrow$, resp.) and $\langle {f_i}_{Aw}(x), g_i(y)\rangle \sqsubseteq \langle {f_{i+1}}_{Aw}(x), g_{i+1}(y)\rangle \sqsubseteq \ldots$ in S^A_w. Hence $g(y)\downarrow$ ($f_{Aw}(x)\downarrow$, resp.) and by admissibility of S^A_w also $\langle f_{Aw}(x), g(y)\rangle \in S^A_w$.

  A similar line of reasoning shows that $\langle f(x)_{Sw}, \mathsf{Rec}_{\mathsf{Loc}}(g)(\sigma)\rangle \in S^{\mathit{St}}_w$ whenever $\langle x_{Sw}, \sigma\rangle \in R^{\mathit{St}}_w$ and either $f(x)\downarrow$ or $\mathsf{Rec}_{\mathsf{Loc}}(g)(\sigma)\downarrow$. Hence, $\langle f,g\rangle : R \subset S$ which we needed to show.

Next, in Table 8, we define a map $\Phi(R^-, R^+)$ on the relational structure corresponding to the equations for the Kripke logical relation R above (separating positive and negative occurrences) such that for $R \in \mathcal{R}(X,Y)$ and $S \in \mathcal{R}(X',Y')$ we have $\Phi(R,S) \in \mathcal{R}((F \times G)(\langle X,Y\rangle, \langle X',Y'\rangle))$.

It is not hard to show that Φ is indeed well-defined: admissibility of each S^{St}_w and S^A_w entails admissibility of the corresponding component of $\Phi(R,S)$:

Lemma 5.1. *Let $R \in \mathcal{R}(X',Y')$ and $S \in \mathcal{R}(X,Y)$. Then, for all $w \in \mathcal{W}$ and $A \in \mathit{Type}$,*

$$\Phi(R,S)^A_w \subseteq X_{Aw} \times Y \quad \textit{and} \quad \Phi(R,S)^{\mathit{St}}_w \subseteq X_{Sw} \times \mathsf{Rec}_{\mathsf{Loc}}(Y)$$

are admissible subsets of the cpos $X_{Aw} \times Y$ and $X_{Sw} \times \mathsf{Rec}_{\mathsf{Loc}}(Y)$, resp.

Moreover, Φ is an admissible action of the functor $F \times G$ on $\mathcal{R}$, in the following sense:

Lemma 5.2. *For all $e = \langle e_1, e_2\rangle, f = \langle f_1, f_2\rangle$ and R, R', S, S', if $e : R' \subset R$ and $f : S \subset S'$ then $(F \times G)(e,f) : \Phi(R,S) \subset \Phi(R',S')$.*

TABLE 8. The functional Φ

At A, w the map Φ is defined according to

$$\langle x,y\rangle \in \Phi(R,S)^{\mathsf{bool}}_w \overset{def}{\iff} y \in \mathsf{BVal} \text{ and } x=y$$

$$\langle r,s\rangle \in \Phi(R,S)^{\{\mathsf{m}_i:A_i\}}_w \overset{def}{\iff} s \in \mathsf{Rec}_{\mathcal{M}}(Y') \text{ and } \forall i\ s.\mathsf{m}_i\downarrow \wedge \langle r.\mathsf{m}_i, s.\mathsf{m}_i\rangle \in S^{A_i}_w$$

$$\begin{aligned}\langle f,g\rangle \in \Phi(R,S)^{A\Rightarrow B}_w \overset{def}{\iff} & g \in [\mathsf{Rec}_{\mathsf{Loc}}(Y)\times Y \rightharpoonup \mathsf{Rec}_{\mathsf{Loc}}(Y')\times Y'] \text{ and}\\ & \forall w'\geq w\ \forall\langle s,\sigma\rangle \in R^{St}_{w'}\ \forall\langle x,y\rangle\in R^{A}_{w'}\\ & \quad (f_{w'}(s,x)\uparrow \wedge g(\sigma,y)\uparrow) \text{ or}\\ & \quad (f_{w'}(s,x)=\langle w'',\langle s',x'\rangle\rangle\downarrow \wedge g(\sigma,y)=\langle\sigma',y'\rangle\downarrow\\ & \qquad \wedge\langle s',\sigma'\rangle\in S^{St}_{w''}\wedge\langle x',y'\rangle\in S^{B}_{w''})\end{aligned}$$

$$\langle x,y\rangle \in \Phi(R,S)^{\mathsf{ref}\ A}_w \overset{def}{\iff} y \in w\cap \mathsf{Loc}_A \text{ and } x=y$$

and at S_w it is given by

$$\langle s,\sigma\rangle \in \Phi(R,S)^{St}_w \overset{def}{\iff} \mathsf{dom}(s)=w=\mathsf{dom}(\sigma) \text{ and } \forall l_A\in w.\ \langle s.l,\sigma.l\rangle\in S^{A}_w$$

According to [40], Lemma 5.2 guarantees that Φ has a unique fixed point $\mathit{fix}(\Phi)$ in $\mathcal{R}(D,\mathsf{Val})$, and we obtain the Kripke logical relation $R \overset{def}{=} \mathit{fix}(\Phi)$ satisfying $R=\Phi(R,R)$ as required.

Theorem 5.3 (Existence, [40]). *The functional Φ has a unique fixed point.*

Proof of Lemma 5.2. Let $w\in\mathcal{W}$ and $A\in \mathit{Type}$. We consider cases for A.

- A is bool: By definition of the functors F and G, $(F\times G)(e,f)=\langle F(e_1,f_1),G(e_2,f_2)\rangle$ with

  $$F(e_1,f_1)_{\mathsf{bool}w}=\mathsf{id}_{\mathsf{BVal}} \text{ and } G(e_2,f_2)=\mathsf{id}_{\mathsf{BVal}}$$

  Now if $\langle x,y\rangle\in\Phi(R,S)^{\mathsf{bool}}_w$ then $y\in\mathsf{BVal}$ and $x=y$, hence,

  $$\langle F(e_1,f_1)_{\mathsf{bool}w}(x),G(e_2,f_2)(y)\rangle=\langle x,y\rangle\in\Phi(R',S')^{\mathsf{bool}}_w$$

- A is $\{\!|\mathsf{m}_i:A_i|\!\}$. Suppose $\langle x,y\rangle\in\Phi(R,S)^{A}_w$, i.e., $\langle x.\mathsf{m}_i,y.\mathsf{m}_i\rangle\in S^{A_i}_w$ for all i. By assumption, $f_{1A_iw}(x.\mathsf{m}_i)\downarrow$ if and only if $f_2(y.\mathsf{m}_i)$, and then

  $$\langle f_{1A_iw}(x.\mathsf{m}_i),f_2(y.\mathsf{m}_i)\rangle\in {S'}^{A_i}_w \tag{5}$$

  By definition of F and G, $F(e_1,f_1)_{Aw}(x)\downarrow$ if and only if $f_{1A_iw}(x.\mathsf{m}_i)\downarrow$ for all i, which by the above is equivalent to $f_2(y.\mathsf{m}_i)\downarrow$ for all i, i.e., $G(e_2,f_2)(y)\downarrow$, and then

  $$F(e_1,f_1)_{Aw}(x).\mathsf{m}_i=f_{1A_iw}(x.\mathsf{m}_i) \text{ and } G(e_2,f_2)(y).\mathsf{m}_i=f_2(y.\mathsf{m}_i)$$

  Hence, (5) shows $\langle F(e_1,f_1)_{Aw}(x),G(e_2,f_2)(y)\rangle\in\Phi(R',S')^{A}_w$.

- A is $B \Rightarrow B'$. Suppose $\langle h, k\rangle \in \Phi(R,S)_w^{B\Rightarrow B'}$, we have to show that
$$\langle F(e_1, f_1)_{B\Rightarrow B'\,w}(h), G(e_2, f_2)(k)\rangle \in \Phi(R', S')_w^{B\Rightarrow B'} \tag{6}$$
So let $w' \geq w$, $\langle s, \sigma\rangle \in {R'}^{\mathit{St}}_{w'}$ and $\langle x, y\rangle \in {R'}^{B}_{w'}$. By assumption,
$$e_{1\,Sw'}(s)\downarrow \textit{ iff } \mathsf{Rec}_{\mathsf{Loc}}(e_2)(\sigma)\downarrow \text{ and then } \langle e_{1\,Sw'}(s), \mathsf{Rec}_{\mathsf{Loc}}(e_2)(\sigma)\rangle \in R^{\mathit{St}}_{w'}$$
$$e_{1\,Bw'}(x)\downarrow \textit{ iff } e_2(y)\downarrow \text{ and then } \langle e_{1\,Bw'}(x), e_2(y)\rangle \in R^{B}_{w'}$$
From $\langle h, k\rangle \in \Phi(R,S)_w^{B\Rightarrow B'}$ we then obtain
$$h_{w'}(e_{1\,Sw'}(s), e_{1\,Bw'}(x)) = \langle w'', \langle s', x'\rangle\rangle\downarrow$$
$$\text{iff}\quad k(\mathsf{Rec}_{\mathsf{Loc}}(e_2)(\sigma), e_2(y)) = \langle \sigma', y'\rangle\downarrow$$
and then both $\langle s', \sigma'\rangle \in S^{\mathit{St}}_{w''}$ and $\langle x', y'\rangle \in S^{B'}_{w''}$. By assumption
$$f_{1\,Sw''}(s')\downarrow \textit{ iff } \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma')\downarrow \text{ and then } \langle f_{1\,Sw''}(s'), \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma')\rangle \in {S'}^{\mathit{St}}_{w''}$$
$$f_{1\,B'w''}(x')\downarrow \textit{ iff } f_2(y')\downarrow \text{ and then } \langle f_{1\,B'w''}(x'), f_2(y')\rangle \in {S'}^{B'}_{w''}$$
By definition of F, G, we have
$$F(f_1, e_1)(h)_{w'}(s, x)\downarrow \implies \begin{cases} F(f_1, e_1)(h)_{w'}(s, x) \\ \quad = \langle w'', \langle f_{1\,Sw''}(s'), f_{1\,B'w''}(x')\rangle\rangle \end{cases}$$
$$G(e_2, f_2)(k)(\sigma, y)\downarrow \implies G(e_2, f_2)(k)(\sigma, y) = \langle \mathsf{Rec}_{\mathsf{Loc}}(f_2)(\sigma'), f_2(y')\rangle$$
so by the above considerations (6) holds.

- A is $\mathsf{ref}\ B$. By definition of F and G,
$$F(e_1, f_1)_{\mathsf{ref}\ B\,w} = \mathsf{id}_{w\cap\mathsf{Loc}_B} \text{ and } G(e_2, f_2) = \mathsf{id}_{\mathsf{Loc}}$$
So if $\langle x, y\rangle \in \Phi(R,S)_w^{\mathsf{ref}\ B}$ then necessarily $y \in w \cap \mathsf{Loc}_A$ and $x = y$, hence,
$$\langle F(e_1, f_1)_{\mathsf{ref}\ B\,w}(x), G(e_2, f_2)(y)\rangle = \langle x, y\rangle \in \Phi(R', S')_w^{\mathsf{ref}\ B}$$

Finally, the case for $\langle s, \sigma\rangle \in \Phi(R,S)_w^{\mathit{St}}$ proceeds analoguously to the case for records $\{\!|\mathsf{m}_i : A_i|\!\}$ above. □

5.2 The Basic Lemma

We establish the following monotonicity properties before proving the basic lemma of logical relations for the Kripke logical relation R:

Lemma 5.4 (Kripke Monotonicity). *Suppose $\langle a, u\rangle \in R^A_w$ and $w' \geq w$. Then $\langle [\![A]\!]_w^{w'}(a), u\rangle \in R^A_{w'}$.*

Proof. By induction on A (note that this is possible here because, in the case of function types, $[\![A \Rightarrow B]\!]_w^{w'}$ does *not* depend on the store).

- A is bool. This follows immediately from $[\![\mathsf{bool}]\!]_w^{w'}(x) = \mathsf{id}(x) = x$.

- A is $\{\mathsf{m}_i : A_i\}_{i\in I}$. By definition of R^A_w we know $y \in \mathsf{Rec}_{\mathcal{M}}(\mathsf{Val})$ and $\langle x.\mathsf{m}_i, y.\mathsf{m}_i\rangle \in R^{A_i}_w$ for all $i \in I$. So by induction hypothesis, $\langle [\![A_i]\!]^{w'}_w (x.\mathsf{m}_i), y.\mathsf{m}_i\rangle \in R^{A_i}_w$ for all i, and $\langle [\![A]\!]^{w'}_w (x), y\rangle \in R^A_{w'}$ follows since
$$[\![A]\!]^{w'}_w (x).\mathsf{m}_i = [\![A_i]\!]^{w'}_w (x.\mathsf{m}_i)$$
- A is $B \Rightarrow B'$. By definition, $[\![B \Rightarrow B']\!]^{w'}_w (x) = \lambda_{w''\geq w'} x_{w''}$, so the result follows directly from the definition of $R^{B\Rightarrow B'}_{w'}$ and the assumption $\langle x, y\rangle \in R^{B\Rightarrow B'}_w$.
- A is $\mathsf{ref}\ B$. Immediately from $[\![\mathsf{ref}\ B]\!]^{w'}_w (x) = x$.

□

Lemma 5.5 (Subtype Monotonicity). *Let $w \in \mathcal{W}$, $A \prec: B$ and $\langle a, u\rangle \in R^A_w$. Then $\langle [\![A \prec: B]\!]_w (a), u\rangle \in R^B_w$.*

Proof. By a straightforward induction on the derivation of $A \prec: B$: Suppose $\langle x, y\rangle \in R^A_w$. If the last step in $A \prec: B$ is

- (Reflexivity). In this case, $A \equiv B$ and $[\![A \prec: B]\!]_w (x) = x$, so that $\langle [\![A \prec: B]\!]_w (x), y\rangle \in R^B_w$ is immediate.
- (Transitivity). Assume $A \prec: B$ was derived from $A \prec: A'$ and $A' \prec: B$. Applying the induction hypothesis, $\langle [\![A \prec: A']\!]_w (x), y\rangle \in R^{A'}_w$ and again by induction hypothesis,
$$\langle [\![A' \prec: B]\!]_w ([\![A \prec: A']\!]_w (x)), y\rangle \in R^B_w$$
as required.
- (Arrow). Write $x' := [\![A{\Rightarrow}B \prec: A' \Rightarrow B']\!]_w (x)$, we must show $\langle x', y\rangle \in R^{A'\Rightarrow B'}_w$. Let $w' \geq w$, $\langle s, \sigma\rangle \in R^{St}_{w'}$ and $\langle u, u'\rangle \in R^{A'}_{w'}$.

  By induction, $\langle [\![A' \prec: A]\!]_{w'} (u), u'\rangle \in R^A_{w'}$, and therefore we have $x_{w'}(s, [\![A' \prec: A]\!]_{w'} (v))\!\downarrow$ if and only if $y(\sigma, v')\!\downarrow$, by assumption $\langle x, y\rangle \in R^{A\Rightarrow B}_w$. Moreover, if both are defined then
$$x_{w'}(s, [\![A' \prec: A]\!]_{w'} (u)) = \langle w'', \langle s', v\rangle\rangle \quad \text{and} \quad y(\sigma, u') = \langle \sigma', v'\rangle$$
  s.t. $\langle s', \sigma'\rangle \in R^{St}_{w''}$ and $\langle v, v'\rangle \in R^B_{w''}$. By induction hypothesis, the latter entails $\langle [\![B \prec: B']\!]_{w''} (v), v'\rangle \in R^{B'}_{w''}$ and we can conclude $\langle x', y\rangle \in R^{A'\Rightarrow B'}_w$.
- (Record). Suppose $\{\mathsf{m}_i : A_i\}_{i\in I} \prec: \{\mathsf{m}_i : A'_i\}_{i\in I'}$ has been derived from $I' \subseteq I$ and $A_i \prec: A'_i$, for all $i \in I'$. Assume $\langle x, y\rangle \in R^{\{\mathsf{m}_i:A_i\}_{i\in I}}_w$. If we let $x' := [\![\{\mathsf{m}_i : A_i\}_{i\in I} \prec: \{\mathsf{m}_i : A'_i\}_{i\in I'}]\!]_w (x)$ we must show $\langle x', y\rangle \in R^{\{\mathsf{m}_i:A'_i\}_{i\in I'}}_w$.

By assumption $y \in \mathsf{Rec}_{\mathcal{M}}(\mathsf{Val})$ and $\langle x.\mathsf{m}_i, y.\mathsf{m}_i\rangle \in R_w^{A_i}$, for all $i \in I$. By induction hypothesis, $\langle [\![A_i \prec: A_i']\!]_w (x.\mathsf{m}_i), y.\mathsf{m}_i\rangle \in R_w^{A_i}$ for all $i \in I' \subseteq I$. The result follows, since by definition $x'.\mathsf{m}_i = [\![A_i \prec: A_i']\!]_w (x.\mathsf{m}_i)$.

□

Lemmas 5.4 and 5.5 show a key property of the relation R, which lies at the heart of the coherence proof: For $\langle a, u\rangle \in R_w^A$ we can apply coercions to a and enlarge the world w while remaining in relation with $u \in \mathsf{Val}$.

We extend R to contexts Γ in the natural way by defining $\langle \rho, \eta\rangle \in R_w^\Gamma$ if and only if $\langle \rho(x), \eta(x)\rangle \in R_w^A$, for all $x{:}A$ in Γ. It is tedious, but not difficult, to prove the fundamental property of logical relations. It states that the (typed and untyped) denotations of well-typed terms compute related results.

Lemma 5.6 (Basic Lemma). *Suppose $\Gamma \rhd e : A$, $w \in \mathcal{W}$, $\langle \rho, \eta\rangle \in R_w^\Gamma$ and $\langle s, \sigma\rangle \in R_w^{St}$. Then*

- *either $[\![\Gamma \rhd e : A]\!]_w \rho s \uparrow$ and $[\![e]\!] \eta\sigma \uparrow$, or*
- *there are $w' \geq w, s', a, \sigma', u$ s.t. $[\![\Gamma \rhd e : A]\!]_w \rho s = \langle w', \langle s', a\rangle\rangle \downarrow$ and $[\![e]\!] \eta\sigma = \langle \sigma', u\rangle \downarrow$ s.t. $\langle s', \sigma'\rangle \in R_{w'}^{St}$ and $\langle a, u\rangle \in R_{w'}^A$.*

Proof. The proof is by induction on the derivation of $\Gamma \rhd e : A$, using Lemmas 5.4 and 5.5.

- (Subsumption). In this case, $\Gamma \rhd e : B$ has been derived from premisses $\Gamma \rhd e : A$ and $A \prec: B$. By induction hypothesis, either both $[\![\Gamma \rhd e : A]\!]_w \rho s \uparrow$ and $[\![e]\!] \eta\sigma \uparrow$, or $[\![\Gamma \rhd e : A]\!]_w \rho s = \langle w', \langle s', x\rangle\rangle \downarrow$ and $[\![e]\!] \eta\sigma = \langle \sigma', y\rangle$ where $\langle s', \sigma'\rangle \in R_{w'}^{St}$ and $\langle x, y\rangle \in R_{w'}^A$. But then $\langle [\![A \prec: B]\!]_{w'} (x), y\rangle \in R_{w'}^B$, by the previous Lemma, which concludes this case using the semantics of the subsumption rule.
- (Var). By assumption $\langle \rho, \eta\rangle \in R_w^\Gamma$, and by the premiss of the variable rule $x : A \in \Gamma$, which entails $\langle \rho(x), \eta(x)\rangle \in R_w^A$. The result now follows immediately from the definitions of $[\![\Gamma \rhd x : A]\!]$ and $[\![x]\!]$, and the assumption $\langle s, \sigma\rangle \in R_w^{St}$.
- (Let). Assume we have derived $\Gamma \rhd \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 : B$ by the rule (Let). By induction hypothesis, either both $[\![\Gamma \rhd e_1 : A]\!]_w \rho s \uparrow$ and $[\![e_1]\!] \eta\sigma \uparrow$, or $[\![\Gamma \rhd e_1 : A]\!]_w \rho s = \langle w', \langle s', u\rangle\rangle \downarrow$ and $[\![e_1]\!] \eta\sigma = \langle \sigma', v\rangle$ where $\langle s', \sigma'\rangle \in R_{w'}^{St}$ and $\langle u, v\rangle \in R_{w'}^A$.

  In the latter case, observe that $\langle [\![\Gamma]\!]_w^{w'} (\rho), \eta\rangle \in R_{w'}^\Gamma$ by Kripke Monotonicity, and therefore $\langle [\![\Gamma]\!]_w^{w'} (\rho)[x := u], \eta[x := v]\rangle \in R_{w'}^{\Gamma, x:A}$. Applying the inductive hypothesis to $\Gamma, x : A \triangleright e_2 : B$ we obtain that either both $[\![e_2]\!]\, \eta[x := v]\sigma' \uparrow$ and $[\![\Gamma, x{:}A \triangleright e_2 : B]\!]_{w'} ([\![\Gamma]\!]_w^{w'}(\rho)[x := u])s' \uparrow$, or

- $[\![\Gamma, x{:}A \triangleright e_2 : B]\!]_{w'} ([\![\Gamma]\!]_w^{w'}(\rho)[x := u])s' = \langle w'', \langle s'', u'\rangle\rangle \downarrow$ and
- $[\![e_2]\!]\, \eta[x := v]\sigma' = \langle \sigma'', v'\rangle$

where $\langle s'', \sigma''\rangle \in R^{St}_{w''}$ and $\langle u', v'\rangle \in R^{B}_{w''}$. Using the definition of $[\![\Gamma \triangleright \mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2 : B]\!]$ and $[\![\mathsf{let}\ x{=}e_1\ \mathsf{in}\ e_2]\!]$, this is all that needed to be shown.

- (Const) Suppose we have derived $\Gamma \triangleright \mathsf{true} : \mathsf{bool}$ by the rule for constant true. The result follows directly from $[\![\Gamma \triangleright \mathsf{true} : \mathsf{bool}]\!]_w\, \rho s = \langle s, \mathit{true}\rangle$ and $[\![\mathsf{true}]\!]\, \eta\sigma = \langle \sigma, \mathit{true}\rangle$, the assumption $\langle s, \sigma\rangle \in R^{St}_w$ and the definition of R^{bool}_w. The case where $\Gamma \triangleright \mathsf{false} : \mathsf{bool}$ is analogous.

- (If) By induction hypothesis on the premiss $\Gamma \triangleright x : \mathsf{bool}$, the assumption $\langle \rho, \eta\rangle \in R^{\Gamma}_w$ and the definition of the semantics, $[\![\Gamma \triangleright x : \mathsf{bool}]\!]_w\, \rho s = \langle w, \langle s, u\rangle\rangle$ and $[\![x]\!]\, \eta\sigma = \langle \sigma, v\rangle$ s.t. $\langle u, v\rangle \in R^{\mathsf{bool}}_w$, for all $\langle s, \sigma\rangle \in R^{St}_w$. By definition this means $u, v \in \mathsf{BVal}$ and $u = v$.

  We consider the case where $u = \mathit{true} = v$, the case where both equal false is analogous. By induction hypothesis on $\Gamma \triangleright e_1 : A$, either both $[\![\Gamma \triangleright e_1 : A]\!]_w\, \rho s \uparrow$ and $[\![e_1]\!]\, \eta\sigma \uparrow$, or $[\![\Gamma \triangleright e_1 : A]\!]_w\, \rho s = \langle w', \langle s', u'\rangle\rangle \downarrow$ and $[\![e_1]\!]\, \eta\sigma = \langle \sigma', v'\rangle$ where $\langle s', \sigma'\rangle \in R^{St}_{w'}$ and $\langle u', v'\rangle \in R^{A}_{w'}$. The result follows now by observing that $[\![\Gamma \triangleright \mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A]\!]_w\, \rho s = \langle w', \langle s', u'\rangle\rangle$ and $[\![\mathsf{if}\ x\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\, \eta\sigma = \langle \sigma', v'\rangle$.

- (Record) For all $i \in I$, by induction hypothesis and from the fact that $[\![x_i]\!]\, \eta\sigma = \langle \sigma, \eta(x_i)\rangle$ one obtains $[\![\Gamma \triangleright x_i : A_i]\!]_w\, \rho s = \langle w, \langle s, u_i\rangle\rangle$ s.t. $\langle u_i, \eta(x_i)\rangle \in R^{A_i}_w$. By definition, $[\![\Gamma \triangleright \{\mathsf{m}_i = x_i\} : \{\mathsf{m}_i : A_i\}]\!]_w\, \rho s = \langle w, \langle s, \{\!|\mathsf{m}_i = u_i|\!\}\rangle\rangle$ and $[\![\{\mathsf{m}_i = x_i\}]\!]\, \eta\sigma = \langle \sigma, \{\!|\mathsf{m}_i = \eta(x_i)|\!\}\rangle$, so the result follows directly from the definition of $R^{\{\mathsf{m}_i : A_i\}}_w$.

- (Selection) By induction hypothesis we have $\langle u, \eta(x)\rangle \in R^{\{\mathsf{m}_i : A_i\}}_w$ provided $\langle w, \langle s, u\rangle\rangle = [\![\Gamma \triangleright x : \{\mathsf{m}_i : A_i\}]\!]_w\, \rho s$. Therefore, by the definition of $R^{\{\mathsf{m}_i : A_i\}}_w$, this entails $\langle u.\mathsf{m}_j, \eta(x).\mathsf{m}_j\rangle \in R^{A_j}_w$ and the result follows from the semantics of $x.\mathsf{m}_j$.

- (Lambda). This case is similar to the case for (Let), using Kripke Monotonicity when constructing an extended context. We know that there exist f, g s.t. $[\![\Gamma \triangleright \lambda x.e : A \Rightarrow B]\!]_w\, \rho s = \langle s, f\rangle$ and $[\![\lambda x.e]\!]\, \eta\sigma = \langle \sigma, g\rangle$, so all that needs to be shown is $\langle f, g\rangle \in R^{A \Rightarrow B}_w$. Let $w' \geq w$, $\langle u, v\rangle \in R^{A}_{w'}$ and $\langle s', \sigma'\rangle \in R^{St}_{w'}$. By Lemma 5.4, $\langle [\![\Gamma]\!]_w^{w'}(\rho)[x := u], \eta[x := v]\rangle \in R^{\Gamma, x:A}_{w'}$. By induction hypothesis for the premiss $\Gamma, x{:}A \triangleright e : B$, either both $[\![\Gamma, x{:}A \rhd e : B]\!]_{w'} ([\![\Gamma]\!]_w^{w'} (\rho)[x := u])s' \uparrow$ and $[\![e]\!]\, \eta[x := v]\sigma' \uparrow$, or

$$[\![\Gamma, x{:}A \rhd e : B]\!]_{w'} ([\![\Gamma]\!]_w^{w'} (\rho)[x := u])s' = \langle w'', \langle s'', u' \rangle\rangle \downarrow$$

and $[\![e]\!]\, \eta[x := v]\sigma' = \langle \sigma'', v' \rangle$ where $\langle s'', \sigma'' \rangle \in R^{St}_{w''}$ and $\langle u', v' \rangle \in R^B_{w''}$. But this is just the definition of $\langle f, g \rangle \in R^{A \Rightarrow B}_w$.

- (Application) Suppose the derivation of $\Gamma \rhd x(y) : B$ ends with the application rule. By the premiss of the rule, $\Gamma \rhd x : A \Rightarrow B$ and $\Gamma \rhd y : A$, so by induction $[\![\Gamma \rhd x : A \Rightarrow B]\!]_w\, \rho s = \langle w, \langle s, f \rangle\rangle$ s.t. $\langle f, \eta(x) \rangle \in R^{A \Rightarrow B}_w$, and $[\![\Gamma \rhd y : A]\!]_w\, \rho s = \langle w, \langle s, u \rangle\rangle$ s.t. $\langle u, \eta(y) \rangle \in R^A_w$.

  By definition of $R^{A \Rightarrow B}_w$, either both $[\![\Gamma \rhd x(y) : B]\!]_w\, \rho s = f_w(s, u) \uparrow$ and $[\![x(y)]\!]\, \eta\sigma = \eta(x)(\sigma, \eta(y)) \uparrow$, or

$$[\![\Gamma \rhd x(y) : B]\!]_w\, \rho s = \langle w', \langle s', u' \rangle\rangle \downarrow \text{ and } [\![x(y)]\!]\, \eta\sigma = \langle \sigma', v' \rangle \downarrow$$

  with $\langle s', \sigma' \rangle \in R^{St}_{w'}$ and $\langle u', v' \rangle \in R^B_{w'}$.

- (New) By definition, we have $[\![\Gamma \rhd \mathsf{new}_A\ x : \mathsf{ref}\ A]\!]_w\, \rho s = \langle w{,}l_A, \langle s', l_A \rangle\rangle$ and $[\![\mathsf{new}_A\ x]\!] = \langle \sigma', l_A \rangle$ with $l_A \in \mathsf{Loc}_A$ and s', σ' as in Table 5 in Section 3 and the semantic equations given in Table 6 in Section 4, resp. All that remains to show is $\langle s', \sigma' \rangle \in R^{St}_{w'}$, where $w' = w{,}l_A$.

  From the assumption $\langle s, \sigma \rangle \in R^{St}_w$ we immediately find $\mathsf{dom}(s') = w' = \mathsf{dom}(\sigma')$. Moreover, by induction we have $[\![\Gamma \rhd x : A]\!]_w\, \rho s = \langle w, \langle s, u \rangle\rangle$ with $\langle u, \eta(x) \rangle \in R^A_w$. By Kripke Monotonicity this entails $\langle [\![A]\!]_w^{w'} (u), \eta(x) \rangle \in R^A_{w'}$. Also, by assumption $\langle s, \sigma \rangle \in R^{St}_w$ we have $\langle s.l, \sigma.l \rangle \in R^{A'}_w$ for all $l \in w \cap \mathsf{Loc}_{A'}$. By Kripke Monotonicity this gives $\langle [\![A']\!]_w^{w'} (s.l), \sigma.l \rangle \in R^{A'}_{w'}$, and we have shown $\langle s'.l', \sigma'.l' \rangle \in R^{A'}_{w'}$ for all $l' \in w'$. Thus $\langle s', \sigma' \rangle \in R^{St}_{w'}$ as required.

- (Deref) By induction hypothesis, $[\![\Gamma \rhd x : \mathsf{ref}\ A]\!]_w\, \rho s = \langle w, \langle s, l \rangle\rangle$ s.t. $\langle l, \eta(x) \rangle \in R^{\mathsf{ref}\ A}_w$, i.e., $\eta(x) \in w \cap \mathsf{Loc}_A$ and $l = \eta(x)$. Then $\langle s, \sigma \rangle \in R^{St}_w$ shows $\langle s.l, \sigma.l \rangle \in R^A_w$. The result follows since $[\![\Gamma \rhd \mathsf{deref}\ x : A]\!]_w\, \rho s = \langle w, \langle s, s.l \rangle\rangle$ and $[\![\mathsf{deref}\ x]\!]\, \eta\sigma = \langle \sigma, \sigma.\eta(x) \rangle$.

- (Update) By definition and the assumptions $\langle \rho, \eta \rangle \in R^\Gamma_w$ and $\langle s, \sigma \rangle \in R^{St}_w$ we necessarily have $[\![x{:=}y]\!]\, \eta\sigma = \langle \sigma', \{\|\} \rangle \downarrow$ and $[\![\Gamma \rhd x{:=}y : \mathbf{1}]\!]_w\, \rho s = \langle w, \langle s', \{\|\} \rangle\rangle \downarrow$, with s' and σ' as in Tables 5 and 6, resp.

  Clearly $\langle \{\|\}, \{\|\} \rangle \in R^{\mathbf{1}}_w$, and all that remains to be shown is $\langle s', \sigma' \rangle \in R^{\mathbf{1}}_w$. From $\langle s, \sigma \rangle \in R^{St}_w$ and the definition of s', σ' one sees $\mathsf{dom}(s') = w = \mathsf{dom}(\sigma')$. So let $l \in w \cap \mathsf{Loc}_B$. Then, by the induction hypothesis, $[\![\Gamma \rhd x : \mathsf{ref}\ A]\!]_w\, \rho s = \langle w, \langle s, l_0 \rangle\rangle$ s.t. $\langle l_0, \eta(x) \rangle \in R^{\mathsf{ref}\ A}_w$, i.e., $l_0 = \eta(x) \in w \cap \mathsf{Loc}_A$.

  Thus, if $l \neq l_0$ then $\langle s'.l, \sigma'.l \rangle = \langle s.l, \sigma.l \rangle \in R^B_w$, by assumption $\langle s, \sigma \rangle \in R^{St}_w$. Finally, for $l = l_0$ we have $\langle s'.l, \sigma'.l \rangle = \langle u, \eta(y) \rangle \in$

TABLE 9. Bracketing maps

$$
\begin{array}{lcl}
\phi_w^{\mathsf{bool}}(b) & = & b \\
\psi_w^{\mathsf{bool}}(v) & = & \begin{cases} v & \text{if } v \in \mathsf{BVal} \\ \text{undefined} & \text{otherwise} \end{cases} \\
\phi_w^{\{\mathsf{m}_i : A_i\}}(r) & = & \{\!|\mathsf{m}_i = \phi_w^{A_i}(r.\mathsf{m}_i)|\!\} \\
\psi_w^{\{\mathsf{m}_i : A_i\}}(v) & = & \begin{cases} \{\!|\mathsf{m}_i = \psi_w^{A_i}(v.\mathsf{m}_i)|\!\} \\ \qquad \text{if } v \in \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) \text{ and } \psi_w^{A_i}(v.\mathsf{m}_i)\downarrow \text{ for all } i \\ \text{undefined otherwise} \end{cases} \\
\phi_w^{A \Rightarrow B}(f) & = & \lambda\langle \sigma, v\rangle . \begin{cases} \langle \phi_{w''}^{St}(s), \phi_{w''}^{B}(b)\rangle \\ \qquad \text{if } \mathsf{dom}(\sigma) = w' \in \mathcal{W}, \psi_{w'}^{St}(\sigma)\downarrow, \psi_{w'}^{A}(v)\downarrow \\ \qquad \text{and } f_{w'}(\psi_{w'}^{St}(\sigma), \psi_{w'}^{A}(v)) = \langle w'', \langle s, b\rangle\rangle \\ \text{undefined otherwise} \end{cases} \\
\psi_w^{A \Rightarrow B}(g) & = & \lambda_{w' \geq w}\, \lambda\langle s, a\rangle . \begin{cases} \langle w'', \langle \psi_{w''}^{St}(\sigma), \psi_{w''}^{B}(v)\rangle\rangle \\ \qquad \text{if } g(\phi_{w'}^{St}(s), \phi_{w'}^{A}(a)) = \langle \sigma, v\rangle\downarrow \\ \qquad \mathsf{dom}(\sigma) = w'' \in \mathcal{W}, \\ \qquad \psi_{w''}^{St}(\sigma)\downarrow \text{ and } \psi_{w''}^{B}(v)\downarrow \\ \text{undefined otherwise} \end{cases} \\
\phi_w^{\mathsf{ref}\ A}(l) & = & l \\
\psi_w^{\mathsf{ref}\ A}(v) & = & \begin{cases} v & \text{if } v \in \mathsf{Loc}_A \\ \text{undefined} & \text{otherwise} \end{cases} \\
\phi_w^{St}(s) & = & \{\!|l_A = \phi_w^{A}(s.l_A)|\!\}_{l_A \in w} \\
\psi_w^{St}(\sigma) & = & \begin{cases} \{\!|l_A = \psi_w^{A}(\sigma.l_A)|\!\}_{l_A \in w} & \text{if } \psi_w^{A}(\sigma.l_A)\downarrow \text{ for all } l_A \in w \\ \text{undefined} & \text{otherwise} \end{cases}
\end{array}
$$

R_w^A, since by induction hypothesis, $[\![\Gamma \triangleright y : A]\!]_w \,\rho s = \langle w, \langle s, u\rangle\rangle$ with $\langle u, \eta(y)\rangle \in R_w^A$.

□

5.3 *Bracketing*

Next, in Table 9, we define families of "bracketing" maps ϕ_w, ψ_w,

$$\begin{tikzcd} {[\![A]\!]_w} \arrow[r, shift left, "\phi_w^A"] & \mathsf{Val} \arrow[l, shift left, "\psi_w^A"] \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} S_w \arrow[r, shift left, "\phi_w^{St}"] & \mathsf{St} \arrow[l, shift left, "\psi_w^{St}"] \end{tikzcd}$$

such that $\psi_w^A \circ \phi_w^A = \mathsf{id}_{[\![A]\!]_w}$, i.e., each $[\![A]\!]_w$ is a retract of the untyped model Val. As in [45], the retraction property follows from a more general result which justifies the term "bracketing",

$$\phi_w^A \subseteq R_w^A \quad \text{and} \quad R_w^A \subseteq (\psi_w^A)^{op}$$

relating the (graphs of the) bracketing maps and the Kripke logical relation of the previous section.

Theorem 5.7 (Bracketing). *For all $w \in \mathcal{W}$ and $A \in$ Type,*

1. *for all $x \in [\![A]\!]_w$. $\langle x, \phi^A_w(x)\rangle \in R^A_w$,*
2. *for all $s \in S_w$. $\langle s, \phi^{St}_w(s)\rangle \in R^{St}_w$*
3. *for all $\langle x, y\rangle \in R^A_w$. $x = \psi^A_w(y)$,*
4. *for all $\langle s, \sigma\rangle \in R^{St}_w$. $s = \psi^{St}_w(\sigma)$*

Compared to Reynolds work, the proof of Theorem 5.7 is more involved, again due to the (mixed-variant) type recursion caused by the use of higher-order store. Therefore we first show a preliminary lemma, which uses the projection maps that come with the minimal invariant solution D of the endofunctor F on $\mathcal{C}$: For $\delta(e) = F(e,e)$ we set $\pi^{Aw}_n \overset{def}{=} \delta^n(\bot)_{Aw}$, and similarly $\pi^{Sw}_n \overset{def}{=} \delta^n(\bot)_{Sw}$. Note that by definition of the minimal invariant solution,

$$\bigsqcup_n \pi^{Aw}_n = (\bigsqcup_n \delta^n(\bot))_{Aw} = (\mathsf{lfp}(\delta))_{Aw} = \mathsf{id}_{Aw}$$

follows. Similarly, $\bigsqcup_n \pi^{Sw}_n = \mathsf{id}_{Sw}$ holds.

Lemma 5.8. *For all $n \in \mathbb{N}$, $w \in \mathcal{W}$, $A \in$ Type,*

1. $\forall x \in [\![A]\!]_w$. $\pi^{Aw}_n(x)\downarrow \implies \langle \pi^{Aw}_n(x), \phi^A_w(\pi^{Aw}_n(x))\rangle \in R^A_w$
2. $\forall s \in S_w$. $\pi^{Sw}_n(s)\downarrow \implies \langle \pi^{Sw}_n(s), \phi^{St}_w(\pi^{Sw}_n(s))\rangle \in R^{St}_w$
3. $\forall \langle x, y\rangle \in R^A_w$. $\pi^{Aw}_n(x)\downarrow \implies \pi^{Aw}_n(x) = \pi^{Aw}_n(\psi^A_w(y))$
4. $\forall \langle s, \sigma\rangle \in R^{St}_w$. $\pi^{Sw}_n(s)\downarrow \implies \pi^{Sw}_n(s) = \pi^{Sw}_n(\psi^{St}_w(\sigma))$

Proof. By a simultaneous induction on n, considering cases for A in parts 1 and 3. Clearly the result holds for $n = 0$ since then π^{Aw}_0 and π^{Sw}_0 are undefined everywhere. For the case $n > 0$:

1. We consider cases for A:
   - A is bool: By definition, $\pi^{\mathsf{bool}w}_n(x) = x \in \mathsf{BVal}$, and therefore $\phi^{\mathsf{bool}}_w(\pi^{\mathsf{bool}w}_n(x)) = \pi^{\mathsf{bool}w}_n(x) = x \in \mathsf{BVal}$. Hence,
   $$\langle \pi^{\mathsf{bool}w}_n(x), \phi^{\mathsf{bool}}_w(\pi^{\mathsf{bool}w}_n(x))\rangle = \langle x, x\rangle \in R^{\mathsf{bool}}_w$$
   by the definition of R^{bool}_w.
   - A is $\{\!|\mathsf{m}_i : A_i|\!\}$: We know $\pi^{\{\!|\mathsf{m}_i:A_i|\!\}}_n(x) = \{\!|\mathsf{m}_i = \pi^{A_i w}_{n-1}(x.\mathsf{m}_i)|\!\}$. By induction hypothesis,
   $$\langle \pi^{A_i w}_{n-1}(x.\mathsf{m}_i), \phi^{A_i}_w(\pi^{A_i w}_{n-1}(x.\mathsf{m}_i))\rangle \in R^{A_i}_w$$

for all i and, by the definition of $\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}$ and $\phi_w^{\{\!|\mathsf{m}_i:A_i|\!\}}$,

$$\begin{aligned}\phi_w^{\{\!|\mathsf{m}_i:A_i|\!\}}(\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}}(x)) &= \phi_w^{\{\!|\mathsf{m}_i:A_i|\!\}}(\{\!|\mathsf{m}_i = \pi_{n-1}^{A_i w}(x.\mathsf{m}_i)|\!\}) \\ &= \{\!|\mathsf{m}_i = \phi_w^{A_i}(\pi_{n-1}^{A_i w}(x.\mathsf{m}_i))|\!\}\end{aligned}$$

Therefore $\langle \pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x).\mathsf{m}_i, \phi_w^{\{\!|\mathsf{m}_i:A_i|\!\}}(\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x)).\mathsf{m}_i\rangle \in R_w^{A_i}$ for all i, i.e.,

$$\langle \pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x), \phi_w^{\{\!|\mathsf{m}_i:A_i|\!\}}(\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x))\rangle \in R_w^{\{\!|\mathsf{m}_i:A_i|\!\}}$$

as required.

- A is $B \Rightarrow B'$: By definition of $\pi_n^{B\Rightarrow B'\,w}$ and $\phi_w^{B\Rightarrow B'}$,

$$\pi_n^{B\Rightarrow B'\,w}(x) = \lambda w' \geq w\, \lambda\langle s, v\rangle.\, \begin{cases} \langle \pi_{n-1}^{Sw''}(s'), \pi_{n-1}^{B'\,w''}(v')\rangle \\ \qquad \text{if } x_{w'}(\pi_{n-1}^{Sw'}(s), \pi_{n-1}^{Bw'}(v)) \\ \qquad\quad = \langle w'', \langle s', v'\rangle\rangle\downarrow \\ \qquad \text{and } \pi_{n-1}^{Sw''}(s')\downarrow \\ \qquad \text{and } \pi_{n-1}^{B'\,w''}(v')\downarrow \\ \text{undefined otherwise}\end{cases}$$

and $\phi_w^{B\Rightarrow B'}(\pi_n^{B\Rightarrow B'\,w}(x))$ equals

$$\lambda\langle \sigma, v\rangle.\, \begin{cases} \langle \phi_{w''}^{St}(\pi_{n-1}^{Sw''}(s')), \phi_{w''}^{B'}(\pi_{n-1}^{B'\,w''}(v'))\rangle \\ \qquad \text{if } \mathsf{dom}(\sigma) = w' \geq w, \psi_{w'}^{St}(\sigma)\downarrow, \psi_{w'}^{B}(v)\downarrow \\ \qquad \text{and } x_{w'}(\pi_{n-1}^{Sw'}(\psi_{w'}^{St}(s)), \pi_{n-1}^{Bw'}(\psi_{w'}^{B}(v))) = \langle w'', \langle s', v'\rangle\rangle\downarrow \\ \text{undefined} \\ \qquad \text{otherwise}\end{cases}$$

In order to show $\langle \pi_n^{B\Rightarrow B'\,w}(x), \phi_w^{B\Rightarrow B'}(\pi_n^{B\Rightarrow B'\,w}(x))\rangle \in R_w^{B\Rightarrow B'}$, let $w' \geq w$, $\langle s, \sigma\rangle \in R_{w'}^{St}$ and $\langle u, v\rangle \in R_{w'}^{B}$.

If either of $\pi_{n-1}^{Sw'}(s)$ or $\pi_{n-1}^{Bw'}(u)$ is undefined then we immediately have both $\pi_n^{B\Rightarrow B'\,w}(x)_{w'}(s, u)\uparrow$ and $\phi_w^{B\Rightarrow B'}(\pi_n^{B\Rightarrow B'\,w}(x))_{w'}(\sigma, v)\uparrow$. So without loss of generality assume both $\pi_{n-1}^{Sw'}(s)\downarrow$ and $\pi_{n-1}^{Bw'}(u)\downarrow$ in the following. Then, by part 3 and 4, resp., of the induction hypothesis, $\pi_{n-1}^{Bw'}(u) = \pi_{n-1}^{Bw'}(\psi_{w'}^{B}(v))$ and $\pi_{n-1}^{Sw'}(s) = \pi_{n-1}^{Sw'}(\psi_{w'}^{St}(\sigma))$. Now if

$$x_{w'}(\pi_{n-1}^{Sw'}(s), \pi_{n-1}^{Bw'}(u)) = \langle w'', \langle s', u'\rangle\rangle\downarrow$$

then the induction hypothesis entails $\langle \pi_{n-1}^{Sw''}(s'), \phi_{w''}^{St}(\pi_{n-1}^{Sw''}(s'))\rangle \in R_{w''}^{St}$ and $\langle \pi_{n-1}^{B'\,w''}(v'), \phi_{w''}^{B'}(\pi_{n-1}^{B'\,w''}(v'))\rangle \in R_{w''}^{B'}$ whenever $\pi_{n-1}^{Sw''}(s')\downarrow$ and $\pi_{n-1}^{B'\,w''}(v')\downarrow$ both hold.

Thus we have established $\langle \pi_n^{B\Rightarrow B'\,w}(x), \phi_w^{B\Rightarrow B'}(\pi_n^{B\Rightarrow B'\,w}(x))\rangle \in R_w^{B\Rightarrow B'}$, by definition of $R_w^{B\Rightarrow B'}$.

- A is $\mathsf{ref}\ B$: By definition, $\pi_n^{\mathsf{ref}\ Bw}(x) = x \in \mathsf{Loc}_A$. Thus we have

$\phi_w^{\mathsf{ref}\ B}(\pi_n^{\mathsf{ref}\ Bw}(x)) = \phi_w^{\mathsf{ref}\ B}(x) = x \in \mathsf{Loc}$, which entails

$$\langle \pi_n^{\mathsf{ref}\ Bw}(x), \phi_w^{\mathsf{ref}\ B}(\pi_n^{\mathsf{ref}\ Bw}(x))\rangle = \langle x, x\rangle \in R_w^{\mathsf{ref}\ B}$$

This concludes this part of the proof.

2. Suppose $\pi_n^{Sw}(s)\downarrow$ and let $s_n = \pi_n^{Sw}(s) = \{\!|l_A = \pi_{n-1}^{Aw}(s.l_A)|\!\}_{l_A \in w}$, and so

$$\begin{aligned}\phi_w^{\mathit{St}}(s_n) &= \{\!|l_A = \phi_w^{A}(s_n.l_A)|\!\}_{l_A \in w}\\ &= \{\!|l_A = \phi_w^{A}(\pi_{n-1}^{Aw}(s.l_A))|\!\}_{l_A \in w}\end{aligned}$$

Then $\mathsf{dom}(s_n) = w = \mathsf{dom}(\phi_w^{\mathit{St}}(s_n))$. Moreover, the first part of the induction hypothesis yields $\langle s_n.l_A, \phi_w^{\mathit{St}}(s_n).l_A\rangle \in R_w^{A}$, for all $l_A \in w$, i.e., $\langle s_n, \phi_w^{\mathit{St}}(s_n)\rangle \in R_w^{\mathit{St}}$ as required.

3. Again, we consider cases for A:

- A is bool: By the definition of R_w^{bool}, $y \in \mathsf{BVal}$ and $x = y$. The result follows immediately from $\pi_n^{\mathsf{bool}w}(x) = x = y = \pi_n^{\mathsf{bool}w}(y) = \pi_n^{\mathsf{bool}w}(\psi_w^{\mathsf{bool}}(y))$.

- A is $\{\!|\mathsf{m}_i : A_i|\!\}$: Suppose $\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x)\downarrow$. In particular, since by definition $\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x) = \{\!|\mathsf{m}_i = \pi_{n-1}^{A_i w}(x.\mathsf{m}_i)|\!\}$ holds this implies $\pi_{n-1}^{A_i w}(x.\mathsf{m}_i)\downarrow$ for all i. The assumption $\langle x, y\rangle \in R_w^{\{\!|\mathsf{m}_i:A_i|\!\}}$ gives $\langle x.\mathsf{m}_i, y.\mathsf{m}_i\rangle \in R_w^{A_i}$ for all i, and so by induction hypothesis

$$\pi_{n-1}^{A_i}(x.\mathsf{m}_i) = \pi_{n-1}^{A_i}(\psi_w^{A_i}(y.\mathsf{m}_i))$$

From this the result follows by calculating

$$\begin{aligned}\pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(x) &= \{\!|\mathsf{m}_i = \pi_{n-1}^{A_i w}(x.\mathsf{m}_i)|\!\}\\ &= \{\!|\mathsf{m}_i = \pi_{n-1}^{A_i}(\psi_w^{A_i}(y.\mathsf{m}_i))|\!\}\\ &= \pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(\{\!|\mathsf{m}_i = \psi_w^{A_i}(y.\mathsf{m}_i)|\!\})\\ &= \pi_n^{\{\!|\mathsf{m}_i:A_i|\!\}w}(\psi_w^{\{\!|\mathsf{m}_i:A_i|\!\}w}(y))\end{aligned}$$

- A is $B \Rightarrow B'$: By definition, for all $w' \geq w$, $s \in S_{w'}$ and $u \in [\![B]\!]_{w'}$,

$$\pi_n^{B\Rightarrow B'\ w}(x)_{w'}(s,u) = \begin{cases}\langle w'', \langle \pi_{n-1}^{Sw''}(s'), \pi_{n-1}^{B'w''}(u')\rangle\rangle\\ \qquad \text{if } x_{w'}(\pi_{n-1}^{Sw'}(s), \pi_{n-1}^{Bw'}(u)) = \langle w'', \langle s', u'\rangle\rangle\downarrow\\ \qquad \text{and } \pi_{n-1}^{Sw''}(s')\downarrow, \pi_{n-1}^{B'w''}(u')\downarrow\\ \text{undefined}\\ \qquad \text{otherwise}\end{cases}$$

and

$$\pi_n^{B\Rightarrow B'\,w}(\psi_w^{B\Rightarrow B'}(y))_{w'}(s,u) = \begin{cases} \langle w'', \langle \pi_{n-1}^{Sw''}(\psi_{w''}^{St}(\sigma)), \pi_{n-1}^{B'w''}(\psi_{w''}^{B'}(v))\rangle\rangle \\ \quad \text{if } y(\phi_{w'}^{St}(\pi_{n-1}^{Sw'}(s)), \phi_{w'}^{B}(\pi_{n-1}^{Bw'}(u))) \\ \quad = \langle\sigma, v\rangle\downarrow \text{ and } \mathsf{dom}(\sigma) = w'' \geq w' \\ \quad \text{and } \pi_{n-1}^{Sw''}(\psi_{w''}^{St}(\sigma))\downarrow \\ \quad \text{and } \pi_{n-1}^{B'w''}(\psi_{w''}^{B'}(v))\downarrow \\ \text{undefined} \\ \quad \text{otherwise} \end{cases}$$

By the first and second parts of the induction hypothesis we know that for $\hat{s} = \pi_{n-1}^{Sw'}(s)$ and $\hat{u} = \pi_{n-1}^{Bw'}(u)$

$$\langle \hat{u}, \phi_{w'}^{B}(\hat{u})\rangle \in R_{w'}^{B} \text{ and } \langle \hat{s}, \phi_{w'}^{St}(\hat{s})\rangle \in R_{w'}^{St}$$

So the assumption $\langle x, y\rangle \in R_w^{B\Rightarrow B'}$ yields

$$x_{w'}(\hat{s},\hat{u}) = \langle w'', \langle s', u'\rangle\rangle\downarrow \textit{ iff } y(\phi_{w'}^{St}(\hat{s}), \phi_{w'}^{B}(\hat{u})) = \langle \sigma, v\rangle\downarrow$$

where $\langle s', \sigma\rangle \in R_{w''}^{St}$ and $\langle u', v\rangle \in R_{w''}^{B'}$.

If either $\pi_{n-1}^{Sw''}(s')\uparrow$ or $\pi_{n-1}^{B'w''}(u')\uparrow$ then, by definition, clearly both $\pi_n^{B\Rightarrow B'\,w}(x)_{w'}(s,u)\uparrow$ and $\pi_n^{B\Rightarrow B'\,w}(\psi_w^{B\Rightarrow B'}(y))_{w'}(s,u)\uparrow$. So without loss of generality $\pi_{n-1}^{Sw''}(s')\downarrow$ and $\pi_{n-1}^{B'w''}(u')\downarrow$, and by parts 3 and 4 of the induction hypothesis we obtain

$$\pi_{n-1}^{Sw''}(s) = \pi_{n-1}^{Sw''}(\psi_{w''}^{St}(\sigma)) \text{ and } \pi_{n-1}^{B'w''}(v) = \pi_{n-1}^{B'w''}(\psi_{w''}^{B'}(v))$$

Since this holds for all $w' \geq w$, s and u we have in fact shown $\pi_n^{B\Rightarrow B'\,w}(x) = \pi_n^{B\Rightarrow B'\,w}(\psi_w^{B\Rightarrow B'}(y))$ as required.

- A is ref B: By definition $\langle x, y\rangle \in R_w^{\mathsf{ref}\ B}$ implies $y \in \mathsf{Loc}_A$ and $x = y$. Similar to the case for bool,

$$\pi_n^{\mathsf{ref}\ Bw}(x) = x = y = \pi_n^{\mathsf{ref}\ Bw}(\psi_w^{\mathsf{ref}\ B}(y))$$

as required.

4. Let $\langle s, \sigma\rangle \in R_w^{St}$, and assume $\pi_n^{Sw}(s)\downarrow$, i.e.

$$\pi_n^{Sw}(s) = \{\!| l_A = \pi_{n-1}^{Aw}(s.l_A) |\!\}_{l_A\in w}$$

In particular $\pi_{n-1}^{Aw}(s.l_A)\downarrow$ for all $l_A \in w$. By definition of R, $\mathsf{dom}(s) = \mathsf{dom}(\sigma)$ and $\langle s.l_A, \sigma.l_A\rangle \in R_w^A$ for all $l_A \in w$. Thus, part 3 of the induction hypothesis entails (since $\pi_{n-1}^{Aw}(s.l_A)\downarrow$) that $\pi_{n-1}^{Aw}(s.l_A) = \pi_{n-1}^{Aw}(\psi_w^A(\sigma.l_A))$ for all $l_A \in w$ and we obtain

$$\begin{aligned} \pi_n^{Sw}(s) &= \{\!| l_A = \pi_{n-1}^{Aw}(s.l_A) |\!\}_{l_A\in w} \\ &= \{\!| l_A = \pi_{n-1}^{Aw}(\psi_w^A(\sigma.l_A)) |\!\}_{l_A\in w} \\ &= \pi_n^{Sw}(\psi_w^{St}(\sigma)) \end{aligned}$$

as required.

□

Proof of Theorem 5.7. For the first part, let $x \in [\![A]\!]_w$. As observed above we have $x = \bigsqcup_n \pi_n^{Aw}(x)$, and in particular $\pi_n^{Aw}(x)\downarrow$ for sufficiently large $n \in \mathbb{N}$. By Lemma 5.8,

$$\langle \pi_n^{Aw}(x), \phi_w^A(\pi_n^{Aw}(x))\rangle \in R_w^A$$

for all sufficiently large n. Since this forms an increasing chain in the cpo $[\![A]\!]_w \times \mathsf{Val}$, completeness of R_w^A and continuity of ϕ_w^A shows

$$\begin{aligned}\langle x, \phi_w^A(x)\rangle &= \langle \textstyle\bigsqcup_n \pi_n^{Aw}(x), \phi_w^A(\bigsqcup_n \pi_n^{Aw}(x))\rangle \\ &= \textstyle\bigsqcup_n \langle \pi_n^{Aw}(x), \phi_w^A(\pi_n^{Aw}(x))\rangle \ \in R_w^A\end{aligned}$$

as required. The other parts are similar. □

6 Coherence of the Intrinsic Semantics

We have now all the parts assembled in order to prove coherence (which proceeds exactly as in [45]): Suppose $\mathcal{P}_1(\Gamma \triangleright e : A)$ and $\mathcal{P}_2(\Gamma \triangleright e : A)$ are derivations of the judgement $\Gamma \triangleright e : A$. We show that their semantics agree. Let $w \in \mathcal{W}$, $\rho \in [\![\Gamma]\!]_w$ and $s \in S_w$. By Theorem 5.7 parts (1) and (2), $\langle \rho, \phi_w^\Gamma(\rho)\rangle \in R_w^\Gamma$ and $\langle s, \phi_w^{St}(s)\rangle \in R_w^{St}$. Hence, by two applications of the Basic Lemma of logical relations, either

$$[\![\mathcal{P}_1(\Gamma \triangleright e : A)]\!]_w\, \rho s\uparrow \ \wedge\ [\![e]\!]\,(\phi_w^\Gamma(\rho))(\phi_w^{St}(s))\uparrow \ \wedge\ [\![\mathcal{P}_2(\Gamma \triangleright e : A)]\!]_w\, \rho s\uparrow$$

or else there exist w_i, s_i, v_i and σ, v such that

$$\begin{aligned}&[\![\mathcal{P}_1(\Gamma \triangleright e : A)]\!]_w\, \rho s = \langle w_1, \langle s_1, v_1\rangle\rangle \\ \wedge\ &[\![e]\!]\,(\phi_w^\Gamma(\rho))(\phi_w^{St}(s)) = \langle \sigma, v\rangle \\ \wedge\ &[\![\mathcal{P}_2(\Gamma \triangleright e : A)]\!]_w\, \rho s = \langle w_2, \langle s_2, v_2\rangle\rangle\end{aligned}$$

where $\langle s_i, \sigma\rangle \in R_{w_i}^{St}$ and $\langle v_i, v\rangle \in R_{w_i}^A$, for $i = 1, 2$. The definition of the relation $R_{w_i}^{St}$ entails $w_1 = \mathsf{dom}(\sigma) = w_2$, and by Theorem 5.7 parts (3) and (4), $s_1 = \psi_{w_1}^{St}(\sigma) = \psi_{w_2}^{St}(\sigma) = s_2$ and $v_1 = \psi_{w_1}^A(v) = \psi_{w_2}^A(v) = v_2$. We have therefore shown

Theorem 6.1 (Coherence). *All derivations of a judgement $\Gamma \triangleright e : A$ have the same meaning in the intrinsic semantics.*

Note that this result does not hold if the type annotation A in new_A was removed. In particular, there would then be two different derivations of the judgement

$$x{:}\{\mathsf{m} : \mathsf{bool}\} \triangleright \mathsf{new}\ x; \mathsf{true} : \mathsf{bool} \tag{7}$$

one without use of subsumption, and one where x is coerced to type $\mathbf{1}$ before allocation. The denotations of these two derivations are *different* (clearly not even the resulting extended worlds are equal). It could be argued that, at least in this particular case, this is a defect of the underlying model: The use of a global store does not reflect the fact that the cell allocated in (7) above remains *local* and cannot be accessed by any enclosing program. However, in the general case we do not know if the lack of locality is the only reason preventing coherence for terms without type annotations.

7 A PER Model of Higher-Order Storage and Subtyping

We consider two consequences of the preceding technical development in more detail. Firstly, the results can be used to obtain an (extrinsic) semantics over the untyped model, based on partial equivalence relations. Secondly, we discuss how this relates to a model of Abadi and Leino's logic for objects that was considered in [43].

7.1 Extrinsic PER Semantics

Apart from proving coherence, Reynolds used (his analogue of) Theorem 5.7 to develop an *extrinsic* semantics of types for the (purely applicative) language considered in [45]. Besides Theorem 5.7 this only depends on the Basic Lemma, and we can do exactly the same here. More precisely, the binary relation $||A||_w := (R^A_w)^{op} \circ R^A_w$, i.e.,

$$||A||_w \overset{def}{=} \left\{ \langle u, v\rangle \in \mathsf{Val} \times \mathsf{Val} \,\middle|\, \exists a \in [\![A]\!]_w.\ \langle a, u\rangle \in R^A_w \ \wedge\ \langle a, v\rangle \in R^A_w \right\} \qquad (8)$$

is a partial equivalence relation (per) on $\mathsf{Val} \times \mathsf{Val}$. Note that a direct proof of transitivity is non-trivial, but it follows easily with part (3) of Theorem 5.7: In case of existence, the existentially quantified a in (8) is uniquely determined as $\psi^A_w(u) = \psi^A_w(v)$.

This definition induces a per $||w|| \subseteq \mathsf{St} \times \mathsf{St}$ for every $w \in \mathcal{W}$ by $\langle \sigma, \sigma'\rangle \in ||w||$ iff $\mathsf{dom}(\sigma) = w = \mathsf{dom}(\sigma')$ and $\langle \sigma.l_A, \sigma'.l_A\rangle \in ||A||_w$ for all $l_A \in w$. The Basic Lemma then shows that the semantics is well-defined on $||{-}||$-equivalence classes, in the sense that if $\Gamma \triangleright e : A$ then for all $w \in \mathcal{W}$, for all $\langle \eta, \eta'\rangle \in ||\Gamma||_w$ and all $\langle \sigma, \sigma'\rangle \in ||w||$,

$$[\![e]\!]\,\eta\sigma\downarrow \vee [\![e]\!]\,\eta'\sigma'\downarrow \implies \begin{cases} [\![e]\!]\,\eta\sigma = \langle \sigma_1, u\rangle \wedge [\![e]\!]\,\eta'\sigma' = \langle \sigma_1', u'\rangle \wedge \\ \exists w' \geq w.\, \langle \sigma_1, \sigma_1'\rangle \in ||w'|| \wedge \langle u, u'\rangle \in ||A||_{w'} \end{cases} \qquad (9)$$

The resulting per model satisfies some of the expected typed equations: For instance, $\{|\mathsf{f} = \mathit{true}, \mathsf{g} = \mathit{true}|\}$ and $\{|\mathsf{f} = \mathit{true}, \mathsf{g} = \mathit{false}|\}$ are equal at $\{\mathsf{f} : \mathsf{bool}\}$. Unfortunately, no non-trivial equations involving store hold in this model; in particular, locality and information hiding are not captured. This is no surprise since we work with a global store, and the failure of various desirable equations has already been observed for the underlying typed model [29].

However, locality is a fundamental assumption underlying many reasoning principles about programs, such as object and class invariants in object-oriented programming. The work of Reddy and Yang [41], and Benton and Leperchey [7], shows how more useful equivalences can be built in into typed models of languages with storable references. We plan to investigate in how far these ideas carry over to full higher-order store.

We remark that, unusually, the per semantics sketched above does not seem to work over a "completely untyped" partial combinatory algebra: The construction relies on the partition of the location set $\mathsf{Loc} = \bigcup_A \mathsf{Loc}_A$. In particular, the definition of the pers $||A||_w$ depends on this rather arbitrary partition. The amount of type information retained by using typed locations allows to express the invariance required for references in the presence of subtyping. We have been unable to find a more "semantic" condition. In view of this, the "untyped" model could be viewed simply as a means to an end, facilitating the definition of the logical relation and bracketing maps in order to prove coherence.

Nevertheless, as pointed out to us by Bernhard Reus, the per model may be useful for providing a semantics of languages with *down-casts*, for example in the form of a construct

$$\frac{\Gamma \triangleright x : A \quad \Gamma \triangleright e_1 : B \Rightarrow C \quad \Gamma \triangleright e_2 : A \Rightarrow C}{\Gamma \triangleright \mathsf{try}\ (B)x\ \mathsf{in}\ e_1\ \mathsf{else}\ e_2 : C}\ (B \prec: A)$$

The intrinsic semantics of Section 3 is not suitable for this purpose: For instance, due to the use of coercions, it is impossible to recover "forgotten" fields of a record.

7.2 On Abadi and Leino's Logic of Objects

Further, it is interesting to observe the rôle that the typed "witness" of $\langle x_1, x_2 \rangle \in ||A||_w$ play, i.e., the unique element $a \in [\![A]\!]_w$ with $\langle a, x_i \rangle \in R^A_w$: Crucially, a determines the world $w' \geq w$ over which the result store and value are to be interpreted in the case of application. This is closely related to the soundness proof of a program logic for objects [43]. In [43] an untyped domain

$$O = \mathsf{Rec}_{\mathcal{F}}(\mathsf{Val}) \times \mathsf{Rec}_{\mathcal{M}}(S \rightharpoonup S \times \mathsf{Val})$$

was employed, representing objects consisting of records of fields $\mathsf{f} \in \mathcal{F}$ and (parameterless) methods $\mathsf{m} \in \mathcal{M}$, where $S = \mathsf{Rec}_{\mathsf{Loc}}(O)$ are object stores. Types (and more generally, specifications) are interpreted as predicates over O. Due to the use of the (higher-order) store S, types are lifted to a notion of store typing w, leading to recursively defined semantics of types as in the present paper. The semantics of object types involves a clause for methods m of type A, similar to (9) above: For all w and all

w-stores σ,

$$m(\sigma) = \langle \sigma', v \rangle \implies \exists w' \geq w. v \in [\![A]\!]_{w'} \text{ and } \sigma' \text{ is a } w\text{-store} \tag{10}$$

where $[\![A]\!]_{w'}$ is the appropriate denotation of type A. But the use of an existential quantification is problematic: It does not preserve admissibility and therefore precludes the use of the machinery of [40].

In [43] the workaround for this problem was to construct a domain of "choice functions" ϕ that track computations in the untyped model on the level of worlds $\mathcal{W}$ to provide a witness w. Then, in (10) above, the existential quantifier can be replaced by "$\phi(w, \sigma) = w'$ for some $w' \geq w$ such that…". Regarding the setting of the present paper, the tracking of the computation on $\mathcal{W}$ is hard-wired into the witnesses coming from the typed model.

8 A Semantics of Objects and Classes

In this section we show that our simple notion of subtyping is useful in obtaining a pleasingly straightforward semantics of the object calculus [1]. We proceed as follows.

- The technical results of previous sections are extended to also cover fixed point combinators. Some care is necessary in formalising such combinators in the presence of side-effects.
- A semantics of objects is given, transferring the definitive work of Kamin and Reddy [26] into a typed world.
- It is demonstrated how to prove (non-trivial) properties of programs using higher-order store in the model: We consider an object-oriented, "circular" implementation of the factorial function (a similar example has been considered in [24]).
- The advantage of using an intrinsically typed model is evidenced by revisiting the simple call-back protocol considered in [44]: A more concise specification of such call-backs is provided. We prove an implementation correct with respect to this specification.
- Referring to the results in [26] again (and also Cook and Palsberg's work in [17]), a semantics of class-based languages is discussed.

8.1 Recursive Functions

As a first step, we show how to interpret explicit recursion in the model (as opposed to recursion through the store due to self-application). Recursive functions will be used in subsequent parts of this section to resolve the dependence of methods on the "self" parameter.

Call-by-value languages do not provide a fixed-point operator at all types. A common restriction is to add a constant

$$\Gamma \rhd \mathsf{fix}_A : (A \Rightarrow A) \Rightarrow A$$

for functional types $A \equiv B \Rightarrow B'$ only. The existence of fixed points is then guaranteed by the pointedness of (the denotations) of such types in **pCpo** (but see also Boudol's recent work on a more generous "safe" value recursion [10]). However, in the presence of computational effects there remains the question about the meaning of

$$\Gamma \rhd \mathsf{fix}_{A \Rightarrow B}\,(\lambda f.e) : A \Rightarrow B \tag{11}$$

It could be taken to be equivalent to the unfolding $e[(\mathsf{fix}_{A \Rightarrow B}\,\lambda f.e)/f]$, thus duplicating the side-effects of e. An alternative semantics would be to evaluate e to a value v first, and then define (11) to be equivalent to $v[(\mathsf{fix}_{A \Rightarrow B}\,\lambda f.v)/f]$. In the latter case the side-effects are performed only once. These issues are discussed in [19, 36]. Here we decided to avoid this problem and follow the simpler approach of the Standard ML language: Essentially, (11) is well-formed only if $e \in \mathsf{Val}$ (i.e., e must be an abstraction). We capture this restriction by extending the syntax of values by a case for recursive functions,

$$v \in \mathsf{Val} ::= \ldots \mid \mu f(x).e$$

which can be thought of as $\mathsf{fix}_{A \Rightarrow B}\,(\lambda f \lambda x.e)$. We add the new type inference rule

$$\frac{\Gamma, f{:}A \Rightarrow B, x{:}A \rhd e : B}{\Gamma \rhd \mu f(x).e : A \Rightarrow B}$$

As for the semantics, note that each derivation $\mathcal{P}(\Gamma, f{:}A \Rightarrow B, x{:}A \rhd e : B)$ determines a total natural transformation $F : [\![\Gamma]\!] \times [\![A \Rightarrow B]\!] \longrightarrow [\![A \Rightarrow B]\!]$ in $[\mathcal{W}, \mathbf{Cpo}]$, given by

$$\begin{aligned} F_w(\rho, h) = \lambda_{w' \geq w} \lambda \langle a, s \rangle. \\ [\![\mathcal{P}(\Gamma, f{:}A \Rightarrow B, x{:}A \rhd e : B)]\!]_{w'}\, (([\![\Gamma, f{:}A \Rightarrow B]\!]_w^{w'}\ \rho[f{:=}h])[x{:=}a])s \end{aligned}$$

Moreover, every cpo $[\![A \Rightarrow B]\!]_w$ is pointed, with least element $\perp_w$ given by the function that is everywhere undefined, $\perp_w = \lambda_{w' \geq w} \lambda \langle a, s \rangle \uparrow$. By monotonicity of each F_w,

$$\perp_w \sqsubseteq G_{w\rho}(\perp_w) \sqsubseteq G^2_{w\rho}(\perp_w) \sqsubseteq \ldots$$

for all $w \in \mathcal{W}$ and $\rho \in [\![\Gamma]\!]_w$, where $G_{w\rho} = \lambda h.F_w(\rho, h)$. Hence by continuity of F_w the least fixed point exists in $[\![A \Rightarrow B]\!]_w$,

$$\mathsf{lfp}(G_{w\rho}) = \bigsqcup_n G^n_{w\rho}(\perp_w) = G_{w\rho}(\mathsf{lfp}(G_{w\rho})) = F_w(\rho, \mathsf{lfp}(G_{w\rho}))$$

Further, by induction it follows that

$$[\![A \Rightarrow B]\!]_w^{w'} \circ G^n_{w\rho}(\perp_w) = G^n_{w' [\![\Gamma]\!]_w^{w'}(\rho)}(\perp_{w'})$$

for all $n \in \mathbb{N}$. Thus $[\![A{\Rightarrow}B]\!]_w^{w'} \circ \mathsf{lfp}(G_{w\rho}) = \mathsf{lfp}(G_{w'[\![\Gamma]\!]_w^{w'}(\rho)})$ and $\rho \mapsto \mathsf{lfp}(G_{w\rho})$ is a natural transformation $[\![\Gamma]\!] \longrightarrow [\![A \Rightarrow B]\!]$. Given the notation as above, we now set

$$\left[\!\!\left[\frac{\Gamma, f{:}A{\Rightarrow}B, x{:}A \triangleright e : B}{\Gamma \triangleright \mu f(x).e : A \Rightarrow B}\right]\!\!\right]_w \rho s = \langle w, \langle s, \mathsf{lfp}(G_{w\rho})\rangle\rangle \in \textstyle\sum_{w' \geq w} S_{w'} \times [\![A{\Rightarrow}B]\!]_{w'}$$

to obtain a semantics for recursive functions in the typed model. In the untyped model, we simply set

$$[\![\mu f(x).e]\!]\,\eta\sigma = \langle \sigma, \mathsf{lfp}(\lambda h.\, [\![\lambda x.e]\!]\,\eta[f := h])\rangle$$

Finally, we turn to the proof of the Basic Lemma, which extends to the case of recursive functions, too.

Proof of Lemma 5.6, continued. Let $\langle s, \sigma\rangle \in R_w^{St}$ and $\langle \rho, \eta\rangle \in R_w^{\Gamma}$. We know that by definition,

$$\left[\!\!\left[\frac{\Gamma, f{:}A{\Rightarrow}B, x{:}A \triangleright e : B}{\Gamma \triangleright \mu f(x).e : A \Rightarrow B}\right]\!\!\right]_w \rho s = \langle w, \langle s, \mathsf{lfp}(G_{w\rho})\rangle\rangle$$

and

$$[\![\mu f(x).e]\!]\,\eta\sigma = \langle \sigma, \mathsf{lfp}(\lambda h.\, [\![\lambda x.e]\!]\,\eta[f := h])\rangle$$

By assumption, $\langle s, \sigma\rangle \in R_w^{St}$, and it remains to show that the two fixed points are related by $R_w^{A\Rightarrow B}$.

To see this, first observe that $\langle \rho[f := h], \eta[f := h']\rangle \in R_w^{\Gamma, f:A\Rightarrow B}$ for all $\langle h, h'\rangle \in R_w^{A\Rightarrow B}$. Therefore as in the case (Lambda) of non-recursive functions, from the induction hypothesis $\Gamma, f{:}A{\Rightarrow}B, x{:}A \triangleright e : B$ it follows that

$$\langle G_{w\rho}(h), [\![\lambda x.e]\!]\,\eta[f := h]\rangle \in R_w^{A\Rightarrow B} \tag{12}$$

for all $\langle h, h'\rangle \in R_w^{A\Rightarrow B}$. From the definition of $R_w^{A\Rightarrow B}$ it is immediate that $\langle \bot_w, \bot\rangle \in R_w^{A\Rightarrow B}$ where $\bot = \lambda\langle\sigma, u\rangle\!\uparrow$ is the everywhere-undefined function in Val. Therefore induction on n and (12) shows

$$\langle G_{w\rho}^n(\bot_w), (\lambda h.\, [\![\lambda x.e]\!]\,\eta[f := h])^n(\bot)\rangle \in R_w^{A\Rightarrow B}$$

for all $n \in \mathbb{N}$. Thus, $\langle \mathsf{lfp}(G_{w\rho}), \mathsf{lfp}(\lambda h.\, [\![\lambda x.e]\!]\,\eta[f := h])\rangle \in R_w^{A\Rightarrow B}$ by admissibility of $R_w^{A\Rightarrow B}$. □

8.2 Objects

Next, we sketch how to give a semantics to Abadi and Cardelli's imperative object calculus with first-order types [1, 3], where we distinguish between fields and methods (with parameters). Fields are mutable, but methods cannot be updated. The type of objects with fields f_i of type A_i and methods m_j of type C_j (with self parameter y_j) and parameter z_j of type B_j, is written $[\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j}$. The introduction rule is

$$\frac{\begin{array}{c} A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j} \\ \Gamma \rhd x_i : A_i \ \forall i \quad \Gamma, y_j{:}A, z_j{:}B_j \rhd b_j : C_j \ \forall j \end{array}}{\Gamma \rhd [\mathsf{f}_i = x_i, \mathsf{m}_j = \varsigma(y_j)\lambda z_j.\, b_j]_{i,j} : A} \tag{13}$$

Subtyping on objects is by width, and for methods also by depth:

$$\frac{B_j{\Rightarrow}C_j \prec: B'_j{\Rightarrow}C'_j \ \forall j \in J' \quad I' \subseteq I \quad J' \subseteq J}{[\mathsf{f}_i : A_i, \mathsf{m}_j : B_j \Rightarrow C_j]_{i \in I, j \in J} \prec: [\mathsf{f}_i : A_i, \mathsf{m}_j : B'_j \Rightarrow C'_j]_{i \in I', j \in J'}} \tag{14}$$

The following is essentially a (syntactic) presentation of the *fixed-point* (or *closure*) *model* of objects [26], albeit in a typed setting: Objects of type $A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j}$ are simply interpreted as *records* of the corresponding record type $A^* \equiv \{\mathsf{f}_i{:}\mathsf{ref}\ A_i^*, \mathsf{m}_j{:}B_j^*{\Rightarrow}C_j^*\}_{i,j}$. Note that the self parameter does not play any part in this type (in contrast to functional interpretations of objects, see [14] for instance), and soundness of the subtyping rule (14) follows directly from the rules of Section 2.

A new object $[\mathsf{f}_i{=}x_i, \mathsf{m}_j{=}\varsigma(y_j)\lambda z_j.\, b_j]_{i,j}$ of type A is created by allocating a state record s and defining the methods by mutual recursion (using obvious syntax sugar),

$$\mathsf{let}\ s = \{\mathsf{f}_i = \mathsf{new}_{A_i}(x_i)\}_{i \in I}\ \mathsf{in}\ \mathit{Meth}_A(s)(\{\mathsf{m}_j = \lambda y_j \lambda z_j.\, b_j\}_{j \in J})$$

where $\mathit{Meth}_A : \{\mathsf{f}_i{:}\mathsf{ref}\ A_i\}_{i \in I} \Rightarrow \{\mathsf{m}_j{:}A^*{\Rightarrow}B_j{\Rightarrow}C_j\}_{j \in J} \Rightarrow A^*$ is given by

$$\mathit{Meth}_A \equiv \mu f(s).\lambda m.\ \{\mathsf{f}_i = s.\mathsf{f}_i, \mathsf{m}_j = \lambda z_j.\,(m.\mathsf{m}_j(f(s)(m)))(z_j)\}_{i \in I, j \in J}$$

Soundness of the introduction rule (13) follows immediately from this interpretation of objects and object types.

The semantics of field selection and field update are simply dereferencing and update, resp., of the corresponding field of the record. The reduction $(-)^*$ of objects to the procedural language of Section 2 is summarized in Table 10.

8.3 Reasoning about Higher-order Store and Objects

One of the main motivations for devising a denotational semantics is to provide proof principles. It should enable us to specify, and reason about, concrete programs.

We look at two small case studies in this section: Firstly, *recursion through the store*, exemplified by an object-based implementation of the factorial function, where the recursion is resolved by calling the method through an object stored in a member field. This calls for recursively defined predicates whose well-definedness has to be established first (similar to the existence proof for the Kripke logical relation of Section 5). Secondly, we consider a simple call-back mechanism [21]: the method cb we wish to specify simply sends requests on to a method m of an object ac-

TABLE 10. Translation of object calculus

Types $[\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]^*_{i\in I, j\in J} \ \equiv\ \{\mathsf{f}_i{:}\mathsf{ref}\ A_i^*, \mathsf{m}_j{:}B_j^*{\Rightarrow}C_j^*\}_{i,j}$

Terms $(a.\mathsf{m}(b))^* \ \equiv a^*.\mathsf{m}(b^*)$

$(a.\mathsf{f})^* \ \equiv\ \mathsf{deref}(a^*.\mathsf{f})$

$(a.\mathsf{f} := b)^* \ \equiv\ (a^*.\mathsf{f}){:=}b^*$

$[\mathsf{f}_i{=}x_i, \mathsf{m}_j{=}\varsigma(y_j)\lambda z_j.\, b_j]^*_{i\in I, j\in J}$

$\quad\equiv\ \mathsf{let}\ s = \{\mathsf{f}_i = \mathsf{new}_{A_i}(x_i)\}_{i\in I}\ \mathsf{in}\ \mathit{Meth}_A(s)(\{\mathsf{m}_j = \lambda y_j \lambda z_j.\, b_j^*\}_{j\in J})$

where $A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i\in I, j\in J}$

$\mathit{Meth}_A \equiv \mu f(s).\lambda m.\, \{\mathsf{f}_i = s.\mathsf{f}_i, \mathsf{m}_j = \lambda z_j.\, (m.\mathsf{m}_j(f(s)(m)))(z_j)\}_{i\in I, j\in J}$

cessible via one of its fields f. As such, this method may be changed at run-time. To reflect this, a sensible specification of the call-back would be of the form *if method* m *satisfies a specification S, then S holds of* cb *too*, where S ranges over a suitable class of specifications.

RECURSION THROUGH THE STORE: THE FACTORIAL. In the following program let $A \equiv [\mathsf{fac} : \mathsf{int} \Rightarrow \mathsf{int}]$, and $B \equiv [\mathsf{f} : A, \mathsf{fac} : \mathsf{int} \Rightarrow \mathsf{int}]$ (so $B \prec: A$). The program computes the factorial, making the recursive calls through the store. Suppose x is declared as integer variable, and consider the program

$\mathsf{let}\ a : A = [\mathsf{fac} = \varsigma(x)\lambda n.\, n]$
$\mathsf{let}\ b : B = [\mathsf{f} = a, \mathsf{fac} = \varsigma(x)\lambda n.\ \mathsf{if}\ n < 1\ \mathsf{then}\ 1\ \mathsf{else}\ n \times (x.\mathsf{f}.\mathsf{fac}(n-1))]$
$\mathsf{in}\ b.\mathsf{f} := b;\ b.\mathsf{fac}(x)$

While we certainly do not claim that this is a particularly realistic example, it does show how higher-order store complicates reasoning. We illustrate a pattern for dealing with the self-application, arising from the use of higher-order store, following the general ideas of [44]: To prove that the call in the last line indeed computes the factorial of x, consider the family of predicates $P = (P_w)_w$, where w ranges over worlds $\geq \{l{:}A\}$ and $P_w \subseteq [\![\mathsf{int} \Rightarrow \mathsf{int}]\!]_w$,

$$h \in P_w \overset{\mathit{def}}{\iff} \forall w' \geq w\, \forall s \in S_{w'}\, \forall n \in [\![\mathsf{int}]\!]_{w'}.\, (s.l.\mathsf{fac} \in P_{w'} \wedge n \geq 0 \wedge h_{w'}(s,n)\!\downarrow)$$
$$\implies \exists w'' \geq w'\, \exists s' \in S_{w''}.\, h_{w'}(s,n) = \langle w'', \langle s', n!\rangle\rangle$$

Note that P_w corresponds to a partial correctness assertion, i.e., *if* the result is defined, then it is indeed $n!$. This example has also been considered in the context of total correctness, in recent work of Honda *et al.* [24] (where, rather different to here, the proof relies on well-founded induction using a termination order).

Due to the (negative) occurrence of $P_{w'}$ in the definition of P_w existence of such a family P has to be established. This can be done along the lines of Theorem 5.3: A relational structure $\mathcal{R}$ on the category $\mathcal{C}$ is given by defining $\mathcal{R}(X)$ to be the type- and world-indexed admissible relations on X, and defining

$$f : R \subset T \quad \text{iff} \quad \begin{cases} \forall w \in \mathcal{W}\, \forall A \in \mathit{Type}\, \forall x \in R^A_w.\, f_{Aw}(x)\downarrow \implies f_{Aw}(x) \in T^A_w \\ \forall w \in \mathcal{W}\, \forall s \in R^{\mathit{St}}_w.\, f_{Sw}(s)\downarrow \implies f_{Sw}(s) \in T^{\mathit{St}}_w \end{cases}$$

for all $R \in \mathcal{R}(X)$, $T \in \mathcal{R}(Y)$ and $\mathcal{C}$-morphisms $f : X \to Y$. A functional Φ is defined corresponding to the predicate $\mathcal{P}$ above,

$$\begin{aligned} f \in \Phi(R)^{\mathsf{int}\Rightarrow\mathsf{int}}_w \iff\ & \forall w'' \geq w' \geq w\, \forall n \geq 0\, \forall s \in S_{w'}\, \forall m \in [\![\mathsf{int}]\!]_{w''}\, \forall s' \in S_{w''}. \\ & (s.l \in R^{\mathsf{int}\Rightarrow\mathsf{int}}_{w'} \wedge f_{w'}(s,n) = \langle w'', \langle s', m\rangle\rangle \implies m = n!) \end{aligned}$$

at worlds $w \geq \{l{:}\mathsf{int} \Rightarrow \mathsf{int}\}$ (the value of Φ at other types, as well as on worlds not extending $\{l{:}\mathsf{int} \Rightarrow \mathsf{int}\}$, does not really matter and could be chosen as the empty relation, for instance). This definition forms an admissible action of the functor $F : \mathcal{C} \to \mathcal{C}$ used to construct the model:

$$e^- : R' \subset R \wedge e^+ : T \subset T' \quad \implies \quad F(e^-, e^+) : \Phi(R) \subset \Phi(R') \qquad (15)$$

As in Section 5, property (15) suffices to establish well-definedness of the predicates $\mathcal{P}$ (see [40]).

Assuming that l is the location allocated for field f, a simple fixed-point induction shows

$$[\![x{:}\mathsf{int}, a{:}A \rhd [\mathsf{f} = a, \mathsf{fac} = \varsigma(x)\lambda n.\, \mathsf{if}\ \ldots\,] : B]\!]_w\, \rho s = \langle w', \langle s', o\rangle\rangle$$

such that w' is $w \cup \{l{:}A\}$, and $o.\mathsf{fac} \in P_{w'}$.

Now let $\hat{s} = s'[l := [\![B \prec: A]\!]_{w'}(o)]$. Thus, $\hat{s}.l.\mathsf{fac} = o.\mathsf{fac} \in P_{w'}$; and if $\rho(x) \geq 0$ we conclude

$$\begin{aligned} & [\![x{:}\mathsf{int}, a{:}A, b{:}[\mathsf{f}{:}A,\ \mathsf{fac}{:}\mathsf{int}{\Rightarrow}\mathsf{int}] \rhd b.\mathsf{f} := b; b.\mathsf{fac}(x) : \mathsf{int}]\!]_{w'}\, \rho[b := o]\hat{s} \\ &= \hat{s}.l.\mathsf{fac}_{w'}(\hat{s}, \rho(x)) \\ &= \langle w'', \langle s'', \rho(x)!\rangle\rangle \end{aligned}$$

for some w'' and s''.

CALL-BACKS. As a second example, we treat the call-back example considered in [44]. Call-backs are used in object-oriented programming to decouple the dependency between caller and callee objects. A typical example is that of generic buttons in user interface libraries, described in [21] by the *command pattern*: As the implementor of the button class cannot have any knowledge about the functionality associated with a particular window button instance, it is assumed that there will be an object supplied (at run-time) that encapsulates the desired behaviour for the *button* *pressed* event, by providing a method execute. Apart from implementing this interface, there are no further requirements on the supplied object. In particular, no assumptions about its execute method are made. The buttonPressed method of the button class will then react to events by forwarding to the execute method. In terms of specifications, buttonPressed would thus satisfy any specification that execute satisfies.

The techniques developed in [44] to express such parametric specifications carry over to the present semantics. However, in contrast to *loc. cit.*, and highlighting the fact that our model is typed, we are able to give a more concise specification of such call-back methods (see (16) below): The existence of all the required methods in the participating objects is already ensured by the intrinsic typing. In the untyped semantics of [44], existence of a method named execute has to be required explicitly in the specification of the method buttonPressed.

We proceed analogous to [44]. Before considering the button example in detail, a notation for (semantic) Hoare triples is defined: Firstly, for $X \in [\mathcal{W}, \mathbf{pCpo}]$ let $\mathit{Adm}(X)$ denote the complete lattice of families $\mathcal{P} = (P_w)_w$ of admissible predicates $P_w \subseteq S_w \times X_w$, ordered pointwise by inclusion. Next, suppose $\mathcal{P} \in \mathit{Adm}([\![A]\!])$ and $\mathcal{Q} \in \mathit{Adm}([\![B]\!])$ are such families of predicates. For $w \in \mathcal{W}$ and $h \in [\![A \Rightarrow B]\!]_w$ we define

$$\begin{aligned}\{\mathcal{P}\}h\{\mathcal{Q}\} \iff \forall w' \geq w\, \forall \langle s, a\rangle \in S_{w'} \times [\![A]\!]_{w'}\,. \\ (\langle s, a\rangle \in P_{w'} \;\wedge\; h_{w'}(s,a) = \langle w'', \langle s', b\rangle\rangle \implies \langle s', b\rangle \in Q_{w''})\end{aligned}$$

expressing a partial correctness assertion of the function h. It can be verified that if each Q_w is an admissible subset of $S_w \times [\![B]\!]_w$ then $\{\mathcal{P}\} - \{\mathcal{Q}\}$ denotes an admissible subset of the function space $[\![A \Rightarrow B]\!]_w$.

Next we consider an implementation of the generic buttons described above. Let B describe the interface $B \equiv [\mathsf{execute} : \mathbf{1} \Rightarrow \mathbf{1}]$ and

$$A \equiv [\mathsf{evHdl} : B, \mathsf{buttonPressed} : \mathbf{1} \Rightarrow \mathbf{1}]$$

be the type of such button objects. Let a be the object

$$x{:}B \;\rhd\; \begin{bmatrix} \mathsf{evHdl} = x, \\ \mathsf{buttonPressed} = \varsigma(y)\lambda z.\; y.\mathsf{evHdl}.\mathsf{execute}(\{\}) \end{bmatrix} \;:\; A$$

Let $l \in \mathsf{Loc}_B$, let $w' \geq w \geq \{l{:}B\}$ and set $T^l_{w,w'} \subseteq S_w \times S_{w'}$ to

$$\begin{aligned}\langle s, s'\rangle \in T^l_{w,w'} \iff \forall \mathcal{P}, \mathcal{Q} \in \mathit{Adm}([\![\mathbf{1}]\!]). \\ \{\mathcal{P}\} s.l.\mathsf{execute}\{\mathcal{Q}\} \;\wedge\; \langle s, \{\![\,]\!\}\rangle \in P_w \implies \langle s', \{\![\,]\!\}\rangle \in Q_{w'}\end{aligned}$$

Since each $Q_{w'}$ is admissible and admissible predicates are closed under universal quantification, $T^l_{w,w'}(s, -)$ determines an admissible predicate over $S_{w'}$ for fixed $s \in S_w$. Thus, the set

$$\Big\{ h \in [\![\mathbf{1} \Rightarrow \mathbf{1}]\!]_w \;\Big|\; \forall s, s'.\, h_w(s, \{\}) = \langle w', \langle s', \{\}\rangle\rangle \implies \langle s, s'\rangle \in T^l_{w,w'} \Big\} \qquad (16)$$

is admissible in $[\![\mathbf{1} \Rightarrow \mathbf{1}]\!]_w$. Now assume that l is the location allocated for the field evHdl of a. A fixed-point induction shows that for any w and $w' = w \cup \{l{:}B\}$, there is an $o \in [\![A]\!]_{w'}$ such that

$$\langle w', \langle s', o\rangle\rangle = [\![x{:}B \rhd a : A]\!]_w\, \rho s$$

and o.buttonPressed $\in [\![\mathbf{1} \Rightarrow \mathbf{1}]\!]_{w'}$ satisfies the specification (16).

8.4 Classes

We take the view, by now often reiterated, of *classes as generators* and *objects as recursive records*. More precisely, we will interpret

- a class as a record of *pre-methods* [18, 17, 26, 13, 1, 9, 10], i.e., functions abstracted on the self-parameter;
- inheritance as extension of this record, possibly replacing some of the pre-methods in the modelling of method redefinition ("override");
- object creation ("new") as the creation of a recursively defined record, obtained from the record of pre-methods by binding the self-parameter (using the function *Meth*$_A$ defined in Section 8.2 above).

The syntax of *class expressions* is defined by the grammar

$$c \;::=\; \mathsf{Root} \;\mid\; \mathsf{class}\ (\overline{x}\,\overline{y})\{\overline{A}\ \bar{\mathsf{f}} = \overline{y}, B'_j\ \mathsf{m}_j = \lambda(y_j{:}B_j).e_i\}_j\ \mathsf{extends}\ c(\overline{x})$$

The class Root is the root in the class hierarchy, and we do not consider multiple inheritance here. The intended meaning of a class expression

$$\mathsf{class}\ (\overline{x}\,\overline{y})\{\overline{A}\ \bar{\mathsf{f}} = \overline{y}, B'_j\ \mathsf{m}_j = \lambda(y_j{:}B_j).e_j\}_j\ \mathsf{extends}\ c(\overline{x})$$

is to declare a *subclass* of another class, the *superclass* c. Instances (objects) of this class have instance variables (fields) $\bar{\mathsf{f}}$ and methods m_j, as well as the instance variables and (a subset of) the methods of the superclass. For simplicity we disallow the shadowing of instance variables, i.e. none of the field names $\mathsf{f} \in \bar{\mathsf{f}}$ is assumed to appear in c. However, method redefinition is allowed: only those methods m of the superclass with $\mathsf{m} \neq \mathsf{m}_j$ for all j appear in instance objects. The self parameter this is taken to be a distinguished variable which is visible in all the method bodies e_j.

In order to create instances, all the fields of the class must be initialised: At object creation time, values $\overline{y}$ have to be supplied for the fields $\bar{\mathsf{f}}$, and similarly values $\overline{x}$ for the fields of the superclass c. Our notation class $(\overline{x}\,\overline{y})\{\dots\}_j$ is supposed to provide a (simplistic) approximation of the constructor methods of class-based programming languages.

TABLE 11. Typing of classes

$$B \equiv [\overline{\mathsf{ff'}}{:}\overline{AA'},\ \mathsf{m}_k{:}B_k{\Rightarrow}B'_k,\ \mathsf{m}_j{:}C_j{\Rightarrow}C'_j]_{k\in K-J, j\in J}$$

$$\frac{\begin{array}{ll} \triangleright\, c : \mathsf{class}(\bar{\mathsf{f}}{:}\overline{A},\ \mathsf{m}_k{:}B_k{\Rightarrow}B'_k)_{k\in K} & B_j \prec: C_j \quad \forall j \in J\cap K \\ \mathsf{this}{:}B^*, y_j{:}C_j \triangleright e_j : C'_j \quad \forall j\in J & C'_j \prec: B'_j \quad \forall j \in J\cap K \end{array}}{\begin{array}{r} \triangleright\, \mathsf{class}\ (\overline{x}\,\overline{y})\{\overline{A'}\ \overline{\mathsf{f'}} = \overline{y}, C'_j\ \mathsf{m}_j = \lambda(y_j{:}C_j).e_j\}_{j\in J}\ \mathsf{extends}\ c(\overline{x}) : \\ \mathsf{class}(\overline{\mathsf{ff'}}{:}\overline{AA'},\ \mathsf{m}_k{:}B_k{\Rightarrow}B'_k,\ \mathsf{m}_j{:}C_j{\Rightarrow}C'_j)_{k\in K-J, j\in J} \end{array}}$$

We introduce *class types* in order to express the well-formedness of class tables constructed from the these class expressions,

$$\mathsf{class}(\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}B'_j)_{i,j}$$

The intended meaning is that instances of a class of this type are objects with type $[\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}B'_j]_{i,j}$. For the root class there is the obvious introduction rule,

$$\frac{}{\triangleright\, \mathsf{Root} : \mathsf{class}()}$$

and we have a type inference rule for subclassing as given in Table 11. Here the object type B is the type of instances of this class; it is used as type of the self parameter this when typing the method bodies e_j. More precisely, the record type B^* is used for this purpose (recall that object types are interpreted as record types, replacing each field declaration f:A by f:ref A). Finally, note that refinement of argument and result type of methods during method redefinition is allowed ("specialisation").

Arising from the informal interpretation of classes and objects outlined at the beginning of this subsection, the semantics of these class types is already forced upon us:

$$\begin{aligned} &\mathsf{class}(\bar{\mathsf{f}}{:}\overline{A},\ \mathsf{m}_j{:}B_j{\Rightarrow}B'_j)^*_j \\ &\qquad\equiv\ (\overline{A}{\Rightarrow}\{\mathsf{m}_j : B \Rightarrow B_j \Rightarrow B'_j\}_j{\Rightarrow}B) \times \{\mathsf{m}_j : B \Rightarrow B_j \Rightarrow B'_j\}_j \end{aligned}$$

where $B \equiv [\bar{\mathsf{f}}{:}\overline{A}, \mathsf{m}_j{:}B_j{\Rightarrow}B'_j]_j$ stands for the type of instances. The first component of this pair will contain the function instantiating objects from the record of pre-methods, i.e., the second component. We reuse the recursive functions *Meth*$_B$ of Section 8.2 for this purpose. Formally, the semantics of class expressions is obtained by providing a translation of *derivations* into the procedural language of Section 2. For simplicity, we omit the types here, since the class type of a class expression is in fact uniquely determined. Thus,

$$\mathsf{Root}^* \ \equiv\ \langle \lambda\_.\,\mathit{Meth}_{[]}\{\}, \{\}\rangle$$

for the root class, and

$$(\mathsf{class}(\overline{x}\,\overline{y})\{\overline{A'}\,\overline{\mathsf{f}'} = \overline{y},\ B'_j\mathsf{m}_j = \lambda(y_j{:}B'_j).e_j\}\ \mathsf{extends}\ c)^* \ \equiv$$

$$\mathsf{let}\ \langle m, r\rangle = c\ \mathsf{in}\ \begin{array}{l}\langle\lambda\overline{xy}.\mathsf{let}\ s = \{\bar{\mathsf{f}} = \mathsf{new}_{\overline{A}}(\overline{x}), \overline{\mathsf{f}'} = \mathsf{new}_{\overline{A'}}(\overline{y})\}\ \mathsf{in}\ \mathit{Meth}_B(s),\\ \{\mathsf{m}_k = r.\mathsf{m}_k,\ \mathsf{m}_j = \lambda\mathsf{this}\lambda y_j.e_j\}_{j,k}\rangle\end{array}$$

where k ranges over the method names inherited from the superclass c. If c is a class expression, then instantiation of a class is

$$\mathsf{new}\,c(\overline{v})^* \ \equiv\ \mathsf{let}\ \langle m, r\rangle = c\ \mathsf{in}\ m(\overline{v})(r)$$

This semantics validates the expected typings: Instances of subclasses have types which are subtypes of those of instances of superclasses.

8.5 Remarks

Our account of classes and inheritance closely follows the one considered in Kamin and Reddy's article [26], termed *closure semantics*, or *fixed point semantics*. However, in *loc. cit.* this was done in an untyped setting, thus side-stepping two of the central issues we have had to deal with in our semantics: modelling types in the presence of *dynamic allocation of heap storage* and *subtyping*.

The interpretation of objects and classes sketched above also highlights another point which we believe deserves some discussion. The following brief remark at the end of [26] compared the closure semantics to their alternative *self-application model* of objects, correctly foreseeing the advantage of the former with respect to typed languages:

"Because self-application uses universal reflexive domains, the fixed point approach is better suited for typed languages."

Self-application here refers to the self-parameter of methods, which is then bound to the host object *at method invocation time*. It is well-known that, in a typed language, this contra-variant occurrence of the self-type in the type signature of methods blocks desirable subtypings. Finding solutions to this problem triggered a long line of work in applying advanced type-theoretic concepts to modelling objects (see [15, 16, 20, 39, 23, 14, 8] amongst others). A similar problem had been encountered in the work on Abadi and Leino's logic of objects, where a self-application semantics was used [3, 44, 43]. For instance, in [43] the problem was avoided by not admitting subtyping on values residing in the store.

We want to conclude this section by pointing out that the above quotation of [26] may be misleading: Of course there is self-application also in the *imperative* fixed-point model of objects and classes. This is in fact due to the higher-order store; as explained in Section 3 the domains used to interpret the language of Section 2 are reflexive domains, too. That this use of reflexive domains seems unavoidable is witnessed by programs using recursion through the store, such as the factorial example of Section 8.3. However, the store parameter remains implicit in the semantics; in particular, it does not appear in the source-level type of the methods of an object and thus does not interfere with subtyping.

9 Polymorphism

We extend the language and the type system with (explicit) predicative, prenex- (or "let"-) polymorphism, similar to the (implicit) polymorphism found in Standard ML [32] and Haskell [38]. Essentially, the type system is stratified into simple types and *type schemes*, with universally quantified type variables ranging over simple (non-polymorphic) types only; moreover, the quantification occurs only on top-level. In particular, function arguments must have simple types. In contrast to ML, and in line with subtyping on simple types considered in previous sections, we actually consider *bounded* universal quantification. The universal quantification of ML can be recovered by using a trivial upper bound, $\top$, of which every type is a subtype.

While this form of polymorphic typing may seem fairly restricted, it has proved very popular and useful in practice: It provides a good compromise between expressiveness and type inference that is tractable in many relevant cases, witnessed by the ML and Haskell languages.

Our theory goes through without any unexpected complications: After presenting the syntax and type inference rules, the semantics of bounded quantification is given using coercion maps (following [12]). Coherence of the extended system is proved by a logical relations theorem and introducing bracketing maps, as in Sects. 5 and 6. In the last part of this Section we introduce a polymorphic allocation operator. It is used in another short case study where *generic classes* are considered.

9.1 Syntax and Typing

We assume a countably infinite set of *type variables*, ranged over by identifiers $\alpha, \beta, \dots$, and a type $\top$ in order to denote trivial upper bounds on type variables.

$$A, B \in \mathit{Type} ::= \ \dots \ \mid \ \alpha \ \mid \ \top$$

Following Milner's terminology [31], a *monotype* is a type A without occurrences of type variables. *Type schemes* σ are

$$\sigma, \tau ::= A \ \mid \ \forall \alpha {\prec:} A.\, \sigma$$

where α is bound in σ by the universal quantifier. Semantically, the type variable α in $\forall \alpha {\prec:} A.\, \sigma$ ranges over monotypes $B \prec: A$ only. A *mono-*

type substitution is an assignment θ of monotypes for type variables. By a *monotype instance* of a type scheme σ we mean a substitution instance $\sigma\theta$ without free type variables.

Contexts Γ may now contain subtype constraints of the form $\alpha \prec: A$, with at most one of these occurring for every α. Hence the derivations of subtypings may depend on the context, and there is the obvious rule to derive the subtyping $\Gamma \rhd \alpha \prec: A$ from an assumption $\alpha \prec: A$ in Γ. The syntax of terms is extended with cases for type abstraction and type application,

$$\begin{aligned} v \in \mathit{Val} &::= \ \dots \ \mid \Lambda\alpha{\prec:}A.\, e \\ e \in \mathit{Exp} &::= \ \dots \ \mid x_A \end{aligned}$$

where Λ binds α in e. In the type system we now derive typings $\Gamma \rhd e : \sigma$. There are two new rules corresponding to these cases for type abstraction and application,

$$\frac{\Gamma, \alpha{\prec:}A \rhd e : \sigma}{\Gamma \rhd \Lambda\alpha{\prec:}A.\, e : \forall\alpha{\prec:}A.\, \sigma} \qquad \frac{\Gamma \rhd B \prec: A \quad \Gamma \rhd x : \forall\alpha{\prec:}A.\, \sigma}{\Gamma \rhd x_B : \sigma[B/\alpha]}$$

with the assumption that α does not occur in Γ or A in the former rule.

9.2 Semantics

The idea of the semantic interpretation follows those of [12] for an applicative language: Recall that subtyping $A \prec: B$ is interpreted by coercion maps $[\![A]\!] \dot{\to} [\![B]\!]$. Informally, a polymorphic value of type $\forall\alpha{\prec:}A.\sigma$ denotes a function that takes a coercion from $[\![B]\!]$ to $[\![A]\!]$ as argument, for any $B \prec: A$.

Therefore, we introduce a type constructor $A \multimap B$ for the space of coercions; in particular, for monotypes A, B, we let $[\![A \multimap B]\!]$ be the cpo of coercions. To make this precise, consider the property of coercions γ from A to B, stated in Lemma 5.5, i.e.,

$$\forall w.\ \langle a, u\rangle \in R^A_w \quad \Longrightarrow \quad \langle \gamma_w(a), u\rangle \in R^B_w \tag{17}$$

We make this the defining property of our semantic notion of coercion. To this end we define a relation $R^{A\multimap B}_w \subseteq ([\![A]\!]_w \to [\![B]\!]_w)) \times (\mathsf{Val} \to \mathsf{Val})$ by

$$\begin{aligned} \langle \phi, \psi\rangle \in R^{A\multimap B}_w \quad \overset{def}{\iff} \quad & \forall \langle a, u\rangle \in [\![A]\!]_w \times \mathsf{Val}. \\ & (\langle a, u\rangle \in R^A_w \implies \langle \phi(a), \psi(u)\rangle \in R^B_w) \end{aligned}$$

Clearly requirement (17) can be restated as $\langle \gamma_w, \mathsf{id}\rangle \in R^{A\multimap B}_w$. With this definition we let $[\![A \multimap B]\!]$ denote the functor $\mathcal{W} \to \mathbf{pCpo}$ where for $w \in \mathcal{W}$

$$[\![A \multimap B]\!]_w = \{\langle \gamma_{w'}\rangle_{w' \geq w} \mid \forall w' \geq w.\, \langle \gamma_{w'}, \mathsf{id}\rangle \in R^{A\multimap B}_w\}$$

ordered pointwise, and with the action on morphisms given by restriction.

The type $\top$ is interpreted as the one-element cpo, $[\![\top]\!]_w = \{*\}$. Further let $R^{\top}_w = [\![\top]\!]_w \times \mathsf{Val}$ be the universal relation, and define bracketing maps in the obvious way: For all $a \in [\![\top]\!]_w$ and $u \in \mathsf{Val}$ let

$$\begin{array}{ll} \phi^{\top}_w : [\![\top]\!]_w \rightharpoonup \mathsf{Val} & \qquad \psi^{\top}_w : \mathsf{Val} \rightharpoonup [\![\top]\!]_w \\ \phi^{\top}_w(a) = \{\!|\,|\!\} & \qquad \psi^{\top}_w(u) = * \end{array}$$

The coercion maps interpreting the subtyping $A \prec: \top$ are given by the unique total maps into $\{*\}$. It is easily verified that the basic lemma as well as the bracketing theorem of Section 5 continue to hold. The semantics of closed type schemes $\forall\alpha{\prec:}A.\,\sigma$ is given at $w \in \mathcal{W}$ by

$$\textstyle\prod_{w' \geq w} \prod \left\{ S_{w'} \times [\![B \multimap A]\!]_{w'} \rightharpoonup \sum_{w'' \geq w'} (S_{w''} \times [\![\sigma[B/\alpha]]\!]_{w''}) \;\middle|\; \begin{array}{l} B \prec: A \\ B \text{ monotype} \end{array} \right\}$$

We assume there are unused *term* variables c_α for every *type* variable α, and write Γ_{c} for the environment obtained from Γ by replacing every type constraint $\alpha \prec: A$ by the assumption $\mathsf{c}_\alpha : \alpha \multimap A$. The semantics of environments is defined via the semantics of monotype instances:

$$[\![\Gamma]\!] = \langle [\![\Gamma_{\mathsf{c}}\,\theta]\!] \mid \theta \text{ a monotype substitution} \rangle$$

As mentioned above, because of the subtype constraints subtyping gives rise to coercions that may depend on the context. In particular, $\Gamma \triangleright B \prec: A$ denotes a total natural transformation $\mathcal{W} \to \mathbf{pCpo}$, for every monotype instance of the judgement:

$$[\![\Gamma \triangleright A \prec: B]\!] = \left\langle [\![\Gamma]\!]_\theta \stackrel{\cdot}{\to} [\![A\theta \multimap B\theta]\!] \;\middle|\; \theta \text{ a monotype substitution} \right\rangle$$

The coercion corresponding to a subtyping derivation from an assumption is determined from the context,

$$\left[\!\!\left[\Gamma, \alpha{\prec:}A, \Gamma' \triangleright \alpha \prec: A\right]\!\!\right]_{\theta\, w} \rho = \rho(\mathsf{c}_\alpha)$$

and the remaining cases are as in Table 3 on page 9, passing the environment around appropriately. The semantics $[\![\Gamma \triangleright e : \sigma]\!]$ of judgements is of type

$$\textstyle\prod \left\{ \prod_w [\![\Gamma_{\mathsf{c}}\,\theta]\!] \to S_w \rightharpoonup \sum_{w' \geq w} S_{w'} \times [\![\sigma\theta]\!]_{w'} \;\middle|\; \theta \text{ a monotype substitution} \right\}$$

In particular, for the derivations ending with an application of the new inference rules the semantics is given in Table 12.

Because of the type of coercions $[\![A \multimap B]\!]$, the semantics of the subsumption rule also changes slightly, by using the environment ρ to construct the coercion and then projecting at the result world w' (see Table 13).

TABLE 12. Semantics of type abstraction and application

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma, \alpha{\prec}{:}A \rhd e : \sigma)}{\Gamma \rhd \Lambda\alpha{\prec}{:}A.\, e : \forall\alpha{\prec}{:}A.\, \sigma}\right]\!\!\right]_{\theta,w} \rho s$$

$$= \langle w, \langle s, \lambda_{w' \geq w}.\, \lambda_{B \prec : A\theta}.\, \lambda\langle s, \phi\rangle.\; [\![\mathcal{P}(\Gamma, \alpha{\prec}{:}A \rhd e : \sigma)]\!]_{(\theta[\alpha:=B]),\, w'} \\ (([\![\Gamma]\!]_{\theta[\alpha:=B]})_{w}^{w'}(\rho)[\mathsf{c}_\alpha := \phi])s\rangle\rangle$$

$$\left[\!\!\left[\frac{\begin{array}{c}\mathcal{P}(\Gamma \rhd B \prec: A)\\ \mathcal{P}(\Gamma \rhd x : \forall\alpha{\prec}{:}A.\, \sigma)\end{array}}{\Gamma \rhd x_B : \sigma[B/\alpha]}\right]\!\!\right]_{\theta,w} \rho s = f_{w,B\theta}(s, [\![\mathcal{P}(\Gamma \rhd B \prec: A)]\!]_{\theta,w}\, \rho)$$

where $\langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : \forall\alpha{\prec}{:}A.\, \sigma)]\!]_{\theta,w}\ \rho s$

TABLE 13. Semantics of subsumption

$$\left[\!\!\left[\frac{\mathcal{P}(\Gamma \rhd e : A)\ \ \mathcal{P}(\Gamma \rhd A \prec: B)}{\Gamma \rhd e : B}\right]\!\!\right]_{\theta w} \rho s$$

$$= \begin{cases} \langle w', \langle s', ([\![\mathcal{P}(\Gamma \rhd A \prec: B)]\!]_{\theta w}\ \rho)_{w'}(a)\rangle\rangle \\ \qquad\qquad \text{if } [\![\mathcal{P}(\Gamma \rhd e : A)]\!]_{\theta w}\, \rho s = \langle w', \langle s', a\rangle\rangle\downarrow \\ \text{undefined} \quad \text{otherwise} \end{cases}$$

TABLE 14. Semantics of terms

$$[\![\Lambda\alpha{\prec}{:}A.\,e]\!]_\theta\,\eta\sigma = \langle\sigma, \lambda_B.\lambda\langle\sigma', v\rangle.\,[\![e]\!]_{\theta[\alpha:=B]}\,\eta\sigma'\rangle$$

$$[\![x_B]\!]_\theta\,\eta\sigma = \begin{cases} p_{(B\theta)}(\sigma', \{\!|\,|\!\}) \text{ if } [\![x]\!]_\theta\,\eta\sigma = \langle\sigma', p\rangle \\ \qquad\qquad\qquad \in \mathsf{St} \times \prod_B(\mathsf{St}\times\mathsf{Val} \rightharpoonup \mathsf{St}\times\mathsf{Val}) \\ \text{undefined} \quad \text{otherwise} \end{cases}$$

9.3 Coherence of the Polymorphic System

We extend the coherence proof to the enriched language. For the untyped[2] semantics we introduce a new summand in order to interpret polymorphic values,

$$\mathsf{Val} = \cdots + \prod\{\mathsf{Val} \mid B \text{ a monotype}\}$$

As in the typed case, the semantics of terms is relative to a monotype substitution θ. Type abstraction (and application, resp.) are interpreted as construction of a dummy abstraction $\lambda\_.e$ in this product (and projection and application of the corresponding component, resp.). This is detailed in Table 14.

Next, we consider the logical relation.

Suppose $\forall\alpha{\prec}{:}A.\,\tau$ is closed, i.e., only α occurs free in τ. We define the logical relation

$$R_w^{\forall\alpha\prec:A.\,\tau} \subseteq [\![\forall\alpha{\prec}{:}A.\,\tau]\!]_w \times \mathsf{Val}$$

by $\langle a, u\rangle \in R_w^{\forall\alpha\prec:A.\,\tau}$ iff

$$\begin{aligned} & u_B \in \textstyle\prod_B(\mathsf{St}\times\mathsf{Val} \rightharpoonup \mathsf{St}\times\mathsf{Val}) \text{ and} \\ & \forall w' \geq w\ \forall B{\prec}{:}A\ \forall\gamma \in [\![B \multimap A]\!]_{w'}\ \forall\langle s,\sigma\rangle \in R^{\mathit{St}}_{w'}. \\ & \qquad\qquad (a_{w'\,B}(s,\gamma)\uparrow \wedge u_B(\sigma, \{\!|\,|\!\})\uparrow) \text{ or} \\ & \qquad\qquad \exists w'' \geq w'.\ (a_{w'\,B}(s,\gamma) = \langle w'', \langle s', b\rangle\rangle \\ & \qquad\qquad \wedge u_B(\sigma, \{\!|\,|\!\}) = \langle\sigma', v\rangle \wedge \langle s',\sigma'\rangle \in R^{\mathit{St}}_{w''} \wedge \langle b, v\rangle \in R^{\tau[B/\alpha]}_{w''}) \end{aligned}$$

Note that the existence this time is immediate from the existence proof in Section 5.1. This is due to the stratification of types into simple types and type schemes: The store only contains values of monotypes but no polymorphic values. It is not hard to see that Kripke monotonicity, Lemma 5.4, still holds.

[2]"untyped" seems even more inappropriate than in Section 4 since the semantics does no longer work uniformly on all types

We prove the analogue of Lemma 5.5 with respect to environments.

Lemma 9.1 (Subtype Monotonicity). *Let θ be a monotype substitution. Suppose that $\rho \in [\![\Gamma]\!]_{\theta w}$ and $w' \geq w$. If $\langle a, u\rangle \in R^{A\theta}_{w'}$ and $\mathcal{P}(\Gamma \rhd A \prec: B)$ then $\langle (([\![\mathcal{P}(\Gamma \rhd A \prec: B)]\!]_{\theta w}\, \rho)_{w'}(a), u\rangle \in R^{B\theta}_{w'}$.*

Proof. We consider the new case, where the derivation $\mathcal{P}(\Gamma \rhd A \prec: B)$ ends with an application of the rule for type variables. Thus, $A \equiv \alpha$ is a type variable and

$$\left[\!\!\left[\frac{}{\Gamma, \alpha{\prec:}B, \Gamma' \rhd \alpha \prec: B}\right]\!\!\right]_{\theta,w} \rho = \rho(\mathsf{c}_\alpha)$$

The assumption $\rho \in [\![\Gamma]\!]_{\theta w}$ implies $\langle \rho(\mathsf{c}_\alpha)_{w'}, \mathsf{id}\rangle \in R^{\theta(\alpha) \multimap B\theta}_{w'}$. Therefore $\langle \rho(\mathsf{c}_\alpha)_{w'}(a), u\rangle = \langle \rho(\mathsf{c}_\alpha)_{w'}(a), \mathsf{id}(u)\rangle \in R^{B\theta}_{w'}$ follows immediately from the hypothesis $\langle a, u\rangle \in R^{\theta(\alpha)}_{w'}$ of the Lemma.

The remaining cases are proved as in Lemma 5.5. $\square$

We obtain an alternative, useful characterisation of the derivable coercion maps as the *unique* maps in relation with the identity. This justifies our definition of $[\![A \multimap B]\!]$.

Corollary 9.2 (Uniqueness). *Let the notation be as in Lemma 9.1.*

- $\langle ([\![\mathcal{P}(\Gamma \rhd A \prec: B)]\!]_{\theta w}\, \rho)_{w'}, \mathsf{id}\rangle \in R^{A\theta \multimap B\theta}_{w'}$
- $\langle \gamma, \mathsf{id}\rangle \in R^{A\theta \multimap B\theta}_{w'} \implies \gamma = ([\![\mathcal{P}(\Gamma \rhd A \prec: B)]\!]_{\theta w}\, \rho)_{w'}$

In particular, $[\![\mathcal{P}(\Gamma \rhd A \prec: B)]\!]_{\theta w}\, \rho = \lambda w' \geq w. \psi^{B\theta}_{w'} \circ \phi^{A\theta}_{w'} \in [\![A\theta \multimap B\theta]\!]_w$ is uniquely determined.

Proof. The first part follows immediately from Lemma 9.1 and the definition of $R^{A\theta \multimap B\theta}_{w}$. For the second part, let $\gamma \in [\![A\theta]\!]_{w'} \to [\![B\theta]\!]_{w'}$ be any map such that $\langle \gamma, \mathsf{id}\rangle \in R^{A\theta \multimap B\theta}_{w'}$. We show that necessarily

$$\gamma = \psi^{B\theta}_{w'} \circ \phi^{A\theta}_{w'}$$

Let $a \in [\![A\theta]\!]_{w'}$ be arbitrary. By Theorem 5.7, $\langle a, \phi^{A\theta}_{w'}(a)\rangle \in R^{A\theta}_{w'}$. Thus, $\langle \gamma(a), \phi^{A\theta}_{w'}(a)\rangle \in R^{A\theta}_{w'}$ by definition of $R^{A\theta \multimap B\theta}_{w'}$. By another application of Theorem 5.7, $\gamma(a) = (\psi^{B\theta}_{w'} \circ \phi^{A\theta}_{w'})(a)$. $\square$

We extend the logical relation to environments as before, with the requirement that condition (17) holds as outlined above:

$$\langle \rho, \eta\rangle \in R^{\Gamma\,\theta}_{w} \iff \begin{cases} \langle \rho(x), \eta(x)\rangle \in R^{\sigma\theta}_{w} & \text{for all } x{:}\sigma \in \Gamma \\ \rho(\mathsf{c}_\alpha) \in [\![\theta(\alpha) \multimap B\theta]\!]_w & \text{for all } \alpha \prec: B \in \Gamma \end{cases}$$

The basic lemma now takes the form

Lemma 9.3 (Basic Lemma). *Suppose $\Gamma \rhd e : \tau$ and let θ be a monotype substitution. Let $w \in \mathcal{W}$, $\langle \rho, \eta\rangle \in R^{\Gamma\,\theta}_{w}$ and $\langle s, \sigma\rangle \in R^{St}_{w}$. Then*

- *either* $[\![\Gamma \triangleright e : A]\!]_{\theta\, w}\ \rho s \uparrow$ *and* $[\![e]\!]_{\theta}\, \eta\sigma \uparrow$*, or*
- *there are* $w' \geq w, s', a, \sigma', u$ *s.t.* $[\![\Gamma \triangleright e : A]\!]_{\theta\, w}\ \rho s = \langle w', \langle s', a\rangle\rangle \downarrow$ *and* $[\![e]\!]_{\theta}\, \eta\sigma = \langle \sigma', u\rangle \downarrow$ *s.t.* $\langle s', \sigma'\rangle \in R^{St}_{w'}$ *and* $\langle a, u\rangle \in R^{\tau\theta}_{w'}$.

Proof. We consider the new cases, for type abstraction and type application.

- (Type Abstraction) From the semantics it is immediate that both

  $$[\![\Gamma \triangleright e : A]\!]_{\theta, w}\ \rho s \downarrow \text{ and } [\![e]\!]_{\theta}\, \eta\sigma \downarrow$$

  and we must show $\langle a, u\rangle \in R^{(\forall\alpha\prec:A.\tau)\theta}_{w'}$, where

  $$a = \lambda_{w' \geq w}.\, \lambda_{B\prec:A\theta}.\, \lambda\langle s, \gamma\rangle.\ [\![\mathcal{P}(\Gamma, \alpha\prec:A \triangleright e : \sigma)]\!]_{(\theta[\alpha:=B]), w'} \qquad ((\![\![\Gamma]\!]_{\theta[\alpha:=B]})^{w'}_{w}(\rho)[\mathsf{c}_\alpha := \gamma])s$$

  $$u = \lambda_B.\lambda\langle\sigma', v\rangle.\, [\![e]\!]_{\theta[\alpha:=B]}\, \eta\sigma'$$

  Let $w' \geq w$, let $B \prec: A\theta$ and $\langle s', \sigma'\rangle \in R^{St}_{w}$. Suppose $\gamma \in [\![B \multimap A\theta]\!]_{w'}$, and observe that in this case,

  $$\langle ([\![\Gamma]\!]_{\theta[\alpha:=B]})^{w'}_{w}(\rho), \eta\rangle \in R^{\Gamma\,(\theta[\alpha:=B])}_{w'}$$

  by Kripke monotonicity. For $\rho' = (([\![\Gamma]\!]_{\theta[\alpha:=B]})^{w'}_{w}(\rho))[\mathsf{c}_\alpha := \gamma]$ this entails

  $$\langle \rho', \eta\rangle \in R^{(\Gamma, \alpha\prec:A)(\theta[\alpha:=B])}_{w'}$$

  By induction hypothesis, either both

  $$a_{w'\, B}(s', \gamma) = [\![\Gamma, \alpha\prec:A \triangleright e : \tau]\!]_{(\theta[\alpha:=B])\, w'}\ \rho' s' \uparrow$$

  and

  $$u_B(\sigma', \{\!|\,|\!\}) = [\![e]\!]_{\theta[\alpha:=B]}\, \eta\sigma' \uparrow$$

  or else there exist $w'' \geq w', s'', b, \sigma'', v$ such that

  $$a_{w'\, B}(s', \gamma) = \langle w'', \langle s'', b\rangle\rangle \text{ and } u_B(\sigma', \{\!|\,|\!\}) = \langle \sigma'', v\rangle$$

  where $\langle s'', \sigma''\rangle \in R^{St}_{w''}$ and $\langle b, v\rangle \in R^{\tau(\theta[\alpha:=B])}_{w''} = R^{(\tau(\theta-\alpha))[B/\alpha]}_{w''}$. Thus we have shown $\langle a, u\rangle \in R^{(\forall\alpha\prec:A.\tau)\theta}_{w'}$.

- (Type Application) By induction hypothesis,

  $$\langle f, \eta(x)\rangle \in R^{(\forall\alpha\prec:A.\tau)\theta}_{w}$$

  where $\langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \triangleright x : \forall\alpha\prec:A.\tau)]\!]_{\theta\, w}\, \rho s$.

  Let $\gamma = [\![\mathcal{P}(\Gamma \triangleright B\prec:A)]\!]_{\theta\, w}\, \rho$. Then, by Corollary 9.2,

  $$\gamma \in [\![B\theta \multimap A\theta]\!]_{w}$$

  From the definition of the logical relation $R^{(\forall\alpha\prec:A.\tau)\theta}_{w}$ it now follows

TABLE 15. Bracketing maps

$$\phi_w^{\forall\alpha\prec:A.\tau}(a) = \lambda_B\lambda\langle\sigma,v\rangle.\begin{cases}\langle\phi_{w''}^{St}(s),\phi_{w''}^{\tau[B/\alpha]}(b)\rangle \\ \qquad\qquad \text{if } B \prec: A, \mathsf{dom}(\sigma)=w', \psi_{w'}^{St}(\sigma)\downarrow \text{ and} \\ \qquad\qquad a_{w'B}(\psi_{w'}^{St}(\sigma),\gamma_{w'}) = \langle w'',\langle s,b\rangle\rangle \\ \text{undefined otherwise}\end{cases}$$

where $\gamma_w = \lambda_{w'\geq w}.\psi_{w'}^B \circ \phi_{w'}^A$ is the unique coercion map in $[\![A \multimap B]\!]_w$ (see Corollary 9.2)

$$\psi_w^{\forall\alpha\prec:A.\sigma}(u) = \lambda_{w'\geq w}\,\lambda_{B\prec:A}\,\lambda\langle s,\gamma\rangle.\begin{cases}\langle\psi_{w'}^{St}(\sigma'),\psi_{w'}^{\tau[B/\alpha]}(v)\rangle \\ \qquad\qquad \text{if } \phi_w^{St}(s)=\sigma, \\ \qquad\qquad u_B(\sigma,\{\!|\,|\!\})=\langle\sigma',v\rangle \\ \qquad\qquad \mathsf{dom}(\sigma)'=w', \text{ and} \\ \qquad\qquad \psi_{w'}^{St}(\sigma')\downarrow, \psi_{w'}^{\tau[B/\alpha]}(v)\downarrow \\ \text{undefined otherwise}\end{cases}$$

that either $f_{wB\theta}(s,\gamma)\uparrow$ and $\eta(x)_{B\theta}(\sigma,\{\!|\,|\!\})\uparrow$ are both undefined, or there are $w'\geq w, s', b, \sigma', v$ such that

$$f_{w\,B\theta}(s,\gamma) = \langle w',\langle s',b\rangle\rangle \text{ and } \eta(x)_{B\theta}(\sigma,\{\!|\,|\!\}) = \langle\sigma',v\rangle$$

with $\langle s',\sigma'\rangle \in R_{w'}^{St}$ and $\langle b,v\rangle \in R_{w'}^{\tau\theta}$, which was to show.

The case for subsumption follows easily with Lemma 9.1. The remaining cases are proved as in Lemma 5.6, passing the substitution θ around appropriately. This concludes the proof. □

Finally, in Table 15 we consider the bracketing maps

$$\begin{tikzcd}[\![\sigma]\!]_w \arrow[r, shift left, "\phi_w^\sigma"] & \mathsf{Val} \arrow[l, shift left, "\psi_w^\sigma"]\end{tikzcd}$$

between the semantics of closed type schemes σ and the untyped model. The cases for quantified type schemes follow a similar pattern to the bracketing maps given earlier for simple types. In order to recover a coercion for derivable subtypings $A \prec: B$ we appeal to Corollary 9.2 which ensures that the semantics of an arbitrary derivation $\mathcal{P}(\triangleright A \prec: B)$ will do. It remains to prove the bracketing theorem, i.e., the analogue of Theorem 5.7.

Theorem 9.4 (Bracketing). *For all type schemes τ without free type variables and for all $w \in \mathcal{W}$,* *1. for all $x \in [\![\tau]\!]_w \,.\, \langle x, \phi^\tau_w(x)\rangle \in R^\tau_w$*

2. for all $\langle x, y\rangle \in R^\tau_w$. $x = \psi^\tau_w(y)$

Proof. The proof is by induction on the number of universal quantifiers in the type scheme τ. For simple types the claims are proved in Theorem 5.7. Now consider the case where τ is of the form $\forall\alpha{\prec}{:}A.\tau'$.

1. Recall that

$$\phi^\tau_w(x) = \lambda_B \lambda\langle\sigma, v\rangle. \begin{cases} \langle\phi^{St}_{w''}(s), \phi^{\tau[B/\alpha]}_{w''}(b)\rangle \\ \qquad\qquad \text{if } B \prec: A, \mathsf{dom}(\sigma) = w', \psi^{St}_{w'}(\sigma)\downarrow \text{ and} \\ \qquad\qquad x_{w'B}(\psi^{St}_{w'}(\sigma), \gamma_{w'}) = \langle w'', \langle s, b\rangle\rangle \\ \text{undefined otherwise} \end{cases}$$

where $\gamma_{w'} = \lambda_{w'' \geq w'}.\psi^B_{w''} \circ \phi^A_{w''} \in [\![A \multimap B]\!]_w$. Let $w' \geq w$, $B \prec: A$, let $\delta \in [\![B \multimap A]\!]_{w'}$ and $\langle s, \sigma\rangle \in R^{St}_{w'}$. We note

$$s = \psi^{St}_{w'}(\sigma) \qquad \text{(by Theorem 5.7)}$$
$$\delta = \gamma_{w'} \qquad \text{(by Corollary 9.2)}$$

Moreover, since $\phi^{St}_{w''}$ and $\phi^{\tau'[B/\alpha]}_{w''}$ are total maps,

$$(\phi^\tau_w(x))_B(\sigma, \{\!|\,|\!\})\uparrow \iff a_{w'B}(s, \delta)\uparrow$$

It remains to consider the case where both terms are defined. Suppose there are $w'' \geq w$, $s \in S_{w''}$ and $b \in [\![\tau'[B/\alpha]]\!]_{w''}$,

$$a_{w'B}(s, \delta) = \langle w'', \langle s', b\rangle\rangle$$

$$(\phi^\tau_w(x))_B(\sigma, \{\!|\,|\!\}) = \langle\phi^{St}_{w''}(s'), \phi^{\tau'[B/\alpha]}_{w''}(b)\rangle$$

By Theorem 5.7, $\langle s', \phi^{St}_{w''}(s')\rangle \in R^{St}_{w''}$, and by induction hypothesis, $\langle b, \phi^{\tau'[B/\alpha]}_{w''}(b)\rangle \in R^{\tau'[B/\alpha]}_{w''}(b)$. Thus we have proved $\langle x, \phi^\tau_w(x)\rangle \in R^\tau_w$.

2. Suppose $\langle x, y\rangle \in R^\tau_w$. By definition,

$$\psi^\tau_w(y) = \lambda_{w' \geq w}\, \lambda_{B \prec: A}\, \lambda\langle s, \delta\rangle. \begin{cases} \langle\psi^{St}_{w'}(\sigma'), \psi^{\tau[B/\alpha]}_{w'}(v)\rangle \\ \qquad\qquad \text{if } \phi^{St}_w(s) = \sigma, \\ \qquad\qquad y_B(\sigma, \{\!|\,|\!\}) = \langle\sigma', v\rangle \\ \qquad\qquad \mathsf{dom}(\sigma)' = w', \text{ and} \\ \qquad\qquad \psi^{St}_{w'}(\sigma')\downarrow, \psi^{\tau[B/\alpha]}_{w'}(v)\downarrow \\ \text{undefined otherwise} \end{cases}$$

We have to show $x = \psi^\tau_w(y)$, so let $w' \geq w$, let $s \in S_{w'}$ and let $\delta \in [\![B \multimap A]\!]_{w'}$ be arbitrary. Note that $\phi^{St}_{w'}$ is total, and $\langle s, \phi^{St}_{w'}(s)\rangle \in R^{St}_{w'}$ by Theorem 5.7. By definition of the logical relation, either $x_{w'B}(s, \delta)$ and $(\psi^\tau_w(y))_{w'B}(s, \delta) = y_B(\phi^{St}_{w'}(s), \{\!|\,|\!\})$ are both undefined, or else there are $w'' \geq w, s', b, \sigma'$ and v such that

$$x_{w'B}(s, \delta) = \langle w'', \langle s', b \rangle \rangle$$
$$(\psi^{\tau}_{w}(y))_{w'B}(s, \delta) = \langle \sigma', v \rangle$$

where $\langle s', \sigma' \rangle \in R^{St}_{w''}$ and $\langle b, v \rangle \in R^{\tau[B/\alpha]}_{w''}$. This yields

$$s' = \psi^{St}_{w''}(\sigma) \qquad \text{(by Theorem 5.7)}$$
$$b = \psi^{\tau[B/\alpha]}_{w''}(v) \qquad \text{(by induction hypothesis)}$$

i.e., $x_{w'B} = (\psi^{\tau}_{w}(y))_{w'B}$ for all $w' \geq w$ and $B \prec: A$. This concludes the proof.

□

Coherence follows, as in Section 6, from these preliminary considerations.

Theorem 9.5 (Coherence). *All derivations of a judgement $\Gamma \triangleright e : \tau$ have the same meaning in the intrinsic semantics.*

9.4 A Polymorphic Allocation Operator and Generic Classes

Finally we can add a polymorphic allocation operator, new, with typing

$$\frac{}{\Gamma \triangleright \mathsf{new} : \forall \alpha \prec: \top. \, \alpha \Rightarrow \mathsf{ref}\ \alpha}$$

The semantics is defined using the monotype instances,

$$\left[\!\!\left[\frac{}{\Gamma \triangleright \mathsf{new} : \forall \alpha \prec: \top. \, \alpha \Rightarrow \mathsf{ref}\ \alpha} \right]\!\!\right]_{\theta w} \rho s$$
$$= \langle w, \langle s, \lambda w' \geq w \, \lambda A \, \lambda \langle s', \delta \rangle. \, [\![\triangleright \lambda x. \mathsf{new}_A \, x : A \Rightarrow \mathsf{ref}\ A]\!]_{\theta w'} \, s' \rangle \rangle \qquad (18)$$

Recall that because of the coherence theorem of Section 9.3 above, any derivation of the judgement $\triangleright \lambda x. \mathsf{new}_A \, x : A \Rightarrow \mathsf{ref}\ A$ will lead to the same semantics in this case.

In order to show coherence of the interpretation of the language extended with the polymorphic new operation it remains to show that its semantics is in relation with the corresponding element of the untyped model, $[\![\mathsf{new}]\!]$,

$$[\![\mathsf{new}]\!]_{\theta} \, \eta \sigma = \langle \sigma, \lambda A \, \lambda \langle \sigma', v \rangle. \, [\![\lambda x. \mathsf{new}_A \, x]\!]_{\theta} \, \sigma' \rangle \qquad (19)$$

Proof of Lemma 9.3, cont. Let θ be a monotype substitution, let $w \in \mathcal{W}$, $\langle \rho, \eta \rangle \in R^{\Gamma \, \theta}_{w}$ and $\langle s, \sigma \rangle \in R^{St}_{w}$. Immediately from the defining equations (18) and (19) above, we only have to show

$$\langle f, g \rangle \in R^{\forall \alpha \prec: \top. \alpha \Rightarrow \mathsf{ref}\ \alpha}_{w}$$

where

$$f = \lambda w' \geq w\, \lambda A\, \lambda\langle s', \delta\rangle.\, [\![\rhd\lambda x.\mathsf{new}_A\, x : A \Rightarrow \mathsf{ref}\ A]\!]_{\theta w'}\, s' \\ g = \lambda A\, \lambda\langle \sigma', v\rangle.\ [\![\lambda x.\mathsf{new}_A\, x]\!]_\theta\, \sigma' \tag{20}$$

To this end, suppose $w' \geq w$, A is any monotype, $\delta \in [\![A \multimap \top]\!]_{w'}$ is the unique coercion from A to $\top$, and let $\langle s', \sigma'\rangle \in R^{St}_{w'}$. By induction hypothesis and the fact that the term $\lambda x.\mathsf{new}_A\, x$ is a value it follows that

$$[\![\rhd\lambda x.\mathsf{new}_A\, x : A \Rightarrow \mathsf{ref}\ A]\!]_\theta\, s' = \langle w'', \langle s'', a\rangle\rangle \\ [\![\lambda x.\mathsf{new}_A\, x]\!]_\theta\, \sigma' = \langle \sigma'', u\rangle$$

with $\langle s'', \sigma''\rangle \in R^{St}_{w''}$ and $\langle a, u\rangle \in R^{A\Rightarrow\mathsf{ref}\ A}_{w''}$. Thus from the definition in (20), f and g are in relation as required. □

AN APPLICATION: GENERIC CLASSES. The concept of polymorphism is not only used in functional languages, but more recently also in mainstream, object-oriented languages such as Java [11, 37], leading to *parametric* or *generic classes*. Indeed, the semantics of this section is sufficient to interpret parametric container classes: We will consider the case of objects implementing memory cells [1]; such objects can be instantiated from a class that is parametric in the type of the stored elements.

The type of memory cells storing values of type α is

$$A(\alpha)\ \equiv\ [\mathsf{cont} : \alpha,\ \mathsf{get} : \mathbf{1}{\Rightarrow}\alpha,\ \mathsf{set} : \alpha{\Rightarrow}\mathbf{1}]$$

providing just a field cont to store the data, and methods get and set to access and update the contents. A simple class implementing objects of this type is

$$c\ \equiv\ \mathsf{class}\langle\alpha \prec: \top\rangle(x) \left\{ \begin{array}{l} \alpha : \mathsf{cont} = x, \\ \alpha\, \mathsf{get} = \lambda(z{:}\mathbf{1})\, \mathsf{this.cont}, \\ \mathbf{1}\, \mathsf{set} = \lambda(z{:}\alpha)\, \mathsf{this.cont}{:=}z \end{array} \right\} \text{ extends Root}$$

The semantics of c is obtained from the translation to the pair consisting of instance generator and record of pre-methods as in Section 8.4,

$$\Lambda\alpha \prec: \top.\langle \lambda x.\mathsf{let}\ s = \{\mathsf{cont} = \mathsf{new}(x)\}\ \mathsf{in}\ \mathit{Meth}_{A(\alpha)}(s), \\ \{\mathsf{set} = \lambda\mathsf{this}\lambda z.\, \mathsf{deref}(\mathsf{this.cont}), \mathsf{get} = \lambda\mathsf{this}\lambda z.\, \mathsf{this.cont}{:=}z\}\rangle$$

Instantiating this class c, with argument v of (mono-) type B

$$\mathsf{new}\, c\langle B\rangle(v)\quad\equiv\quad \mathsf{let}\ \langle m, r\rangle = c_B\ \mathsf{in}\ m(v)$$

generates a memory cell storing values of type B with initial contents v.

More realistic examples are provided by the collection classes of the Java libraries, implementing for instance generic linked lists, queues and vectors. However these examples additionally require type recursion, which we do not consider in this report.

10 Related Work

Apart from Levy's work [29, 30] which we built upon here, we are aware of only few other semantic models of higher-order store in the literature. The models [5, 27] use games semantics and are not location-based, i.e., the store is modelled only indirectly via possible program behaviours. They do not appear to give rise to reasoning principles such as those necessary to establish the existence of the logical relation, or the predicate used in Section 8.3. Ahmed, Appel and Virga [6] construct a model with a rather operational flavour: The semantics of types is obtained by approximating absence of type errors in a reduction semantics; soundness of this construction follows from an encoding into type theory. Again we do not see how strong reasoning principles can be obtained. Jeffrey and Rathke [25] provide a model of the object calculus in terms of interaction traces, very much in the spirit of games semantics. Apart from Jeffrey and Rathke's semantics, none of these models deals with subtyping.

There is a vast body of work on interpreting objects in procedural languages. Bruce, Cardelli and Pierce [14] provide a fairly comprehensive overview and comparison of the more successful encodings of typed objects. Most of the encodings in the literature consider functional objects only. Exceptions are the denotational analyses of objects and inheritance by Kamin and Reddy [26], and Cook and Palsberg [17]. However, they use untyped models, side-stepping the issues of modelling types and subtyping in the presence of updates and dynamic allocation. Many others apply similar ideas in a purely syntactic setting [9, 49, 2, 10]. This suffices for proofs of type soundness, but more powerful specifications are not usually discussed, and indeed logics of languages with higher-order store are hard to justify in such a setting (see [3] for an attempt).

The proof principles applied in Section 8.3 are direct adaptations of those presented in [44] in the context of an untyped model of the object calculus. Similar techniques have been used in [43] to prove soundness of a logic for the object calculus, due to Abadi and Leino [3]. Honda, Berger and Yoshida [24] present a program logic for higher-order procedures and general references. Soundness is proved with respect to a term model. They do not consider subtyping, and the type system is stratified to prevent aliasing. It is not clear to us to which extent their proof system can deal with properties of higher-order store: They deal with *total* correctness assertions and thus can use well-founded induction on a termination order in cases of recursion-through-the-store. In particular this entails that even very simple *non-terminating* programs cannot be proved as such.

11 Conclusions and Future Work

We have extended a model of general references with subtyping, to obtain a semantics of imperative objects. While the individual facts are much more intricate to prove than for the functional language considered in [45], the overall structure of the coherence proof is almost identical to *loc. cit.* This suggests it could be interesting to work out the general conditions needed for the construction (for example, using the setting of [35]).

In a different direction, we can extend the language with a more expressive type system: Recursive types and polymorphism feature prominently in the work on semantics of *functional* objects (see [14]). Here we have shown that the techniques to establish coherence scale well to the extension of the type system with ML-like (prenex) polymorphism [31, 50] – essentially because there is no interaction with the store. We are less optimistic about polymorphism in general; the combination of second-order lambda calculus and higher-order storage certainly appears to be challenging. In [30] it is suggested that the construction of the intrinsic model also works for a variant of recursive types. We haven't considered the combination with subtyping yet, but do not expect any difficulties.

Finally, we plan to develop (Hoare-style) logics, with pre- and post-conditions, for languages involving higher-order store. As a starting point, we are currently trying to adapt the program logic of [3] to the language considered here.

Acknowledgement I am grateful to Bernhard Reus for many helpful discussions and comments. Paul Levy pointed out a flaw in an earlier version of Section 3, and Andreas Rossberg provided useful comments on polymorphic types. Financial support was provided by the EPSRC under grant GR/R65190/01, "Programming Logics for Denotations of Recursive Objects"

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, 1996.

[2] M. Abadi, L. Cardelli, and R. Viswanathan. An interpretation of objects and object types. In *Conference record of the 23rd Symposium on Principles of Programming Languages*, pages 396–409. ACM Press, 1996.

[3] M. Abadi and K. R. M. Leino. A logic of object-oriented programs. In N. Dershowitz, editor, *Verification: Theory and Practice. Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, Lecture Notes in Computer Science, pages 11–41. Springer, 2004.

[4] H. Abelson, R. K. Dybvig, C. T. Haynes, R. Rozas, N. I. Adams IV, D. P. Friedman, K. Kohlbecker, G. L. Steele Jr., D. H. Bartley, R. Halstead, D. Oxley, G. J. Sussman, B. Brooks, C. Hanson, K. M. Pitman, and M. Wand. Revised^5 report on the algorithmic language Scheme. *Higher-Order and Symbolic Computation*, 11(1):7–105, Aug. 1998.

[5] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general

references. In *Proceedings 13th Annual IEEE Symposium on Logic in Computer Science*, pages 334–344. IEEE Computer Society Press, 1998.

[6] A. J. Ahmed, A. W. Appel, and R. Virga. A stratified semantics of general references embeddable in higher-order logic. In *Proceedings of 17th Annual IEEE Symposium Logic in Computer Science*, pages 75–86. IEEE Computer Society Press, 2002.

[7] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *To appear in Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications (TLCA '05)*, Lecture Notes in Computer Science. Springer, 2005.

[8] V. Bono and M. Bugliesi. Interpretations of extensible objects and types. In *Proceedings of the 12th Int. Symposium on Fundamentals of Computing*, volume 1684 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 1999.

[9] V. Bono, A. J. Patel, V. Shmatikov, and J. C. Mitchell. A core calculus of classes and objects. In *15th Conference on the Mathematical Foundations of Programming Semantics*, volume 20 of *Electronic Notes in Computer Science*, Apr. 1999.

[10] G. Boudol. The recursive record semantics of objects revisited. *Journal of Functional Programming*, 14(3):263–315, May 2004.

[11] G. Bracha, M. Odersky, D. Stoutamire, and P. Wadler. Making the future safe for the past: Adding genericity to the Java programming language. *ACM SIGPLAN Notices*, 33(10):183–200, Oct. 1998.

[12] V. Breazu-Tannen, T. Coquand, G. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.

[13] K. B. Bruce. A paradigmatic object-oriented programming language: Design, static typing and semantics. *Journal of Functional Programming*, 4(2):127–206, Apr. 1994.

[14] K. B. Bruce, L. Cardelli, and B. C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, Nov. 1999.

[15] P. Canning, W. Cook, W. Hill, W. Olthoff, and J. Mitchell. F-bounded polymorphism for object-oriented programming. In *Proceedings 4th International Conference on Functional Programming Languages and Computer Architecture*, pages 273–280. ACM Press, 1989.

[16] L. Cardelli, S. Martini, J. C. Mitchell, and A. Scedrov. An extension of System F with subtyping. *Information and Computation*, 109(1–2):4–56, 1994.

[17] W. Cook and J. Palsberg. A denotational semantics of iinheritance and its correctness. *Information and Computation*, 114(2):329–350, Nov. 1994.

[18] W. R. Cook. *A Denotational Semantics of Inheritance*. Ph.D. thesis, Department of Computer Science, Brown University, May 1989.

[19] L. Erkök and J. Launchbury. Recursive monadic bindings. *ACM SIGPLAN Notices*, 35(9):174–185, Sept. 2000.

[20] K. Fisher, F. Honsell, and J. C. Mitchell. A lambda calculus of objects and method specialization. *Nordic Journal of Computing*, 1:3–37, 1994.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[22] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proc. 16th Int. Workshop Computer Science Logic (CSL 2002)*, volume 2471 of *Lecture Notes in Computer Science*, pages 553–568. Springer, 2002.

[23] M. Hofmann and B. Pierce. A unifying type-theoretic framework for objects. *Journal of Functional Programming*, 5(4):593–635, Oct. 1995.

[24] K. Honda, M. Berger, and N. Yoshida. An observationally complete program logic for

imperative higher-order functions. In *Proceedings LiCS'05*, 2005. To appear.

[25] A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. In *Proc. Lics2002, 17^{th} Annual Symposium on Logic in Computer Science*, pages 101–112. IEEE Computer Society Press, 2002.

[26] S. N. Kamin and U. S. Reddy. Two semantic models of object-oriented languages. In C. A. Gunter and J. C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, pages 464–495. MIT Press, 1994.

[27] J. Laird. A categorical semantics of higher-order store. In R. Blute and P. Selinger, editors, *Proceedings of the 9th Conference on Category Theory and Computer Science, CTCS '02*, volume 69 of *Electronic notes in Theoretical Computer Science*, pages 1–18. Elsevier, 2003.

[28] P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4):308–320, Jan. 1964.

[29] P. B. Levy. Possible world semantics for general storage in call-by-value. In J. Bradfield, editor, *CSL: 16th Workshop on Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*. Springer, 2002.

[30] P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*, volume 2 of *Semantic Structures in Computation*. Kluwer, 2004.

[31] R. Milner. A theory of type polymorphism in programming languages. *Journal of Computer and System Science*, 17(3):348–375, 1978.

[32] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. The MIT Press, 1997.

[33] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[34] J. C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51(1–2):99–124, 1991.

[35] J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In E. Börger, G. Jäger, H. K. Büning, S. Martini, and M. M. Richter, editors, *Computer Science Logic '92, Selected Papers*, volume 702 of *Lecture Notes in Computer Science*, pages 352–378. Springer, 1993.

[36] E. Moggi and A. Sabry. An abstract monadic semantics for value recursion. *Theoretical Informatics and Applications*, 38(4):375–400, 2004.

[37] A. Ohori and P. Buneman. Static type inference for parametric classes. *ACM SIGPLAN Notices*, 24(10):445–456, Oct. 1989. OOPSLA '89 Conference Proceedings.

[38] S. Peyton Jones, editor. *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press, 2003.

[39] B. C. Pierce and D. N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, 1994.

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

[41] U. S. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1–3):129–160, March 2004.

[42] B. Reus. Class-based versus object-based: A denotational comparison. In H. Kirchner and C. Ringeissen, editors, *Proceedings of 9th International Conference on Algebraic Methodology And Software Technology*, volume 2422 of *Lecture Notes in Computer Science*, pages 473–488. Springer, 2002.

[43] B. Reus and J. Schwinghammer. Denotational semantics for Abadi and Leino's logic of objects. In M. Sagiv, editor, *Proceedings of the European Symposium on Programming,*

volume 3444 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2005.

[44] B. Reus and T. Streicher. Semantics and logic of object calculi. *Theoretical Computer Science*, 316:191–213, 2004.

[45] J. C. Reynolds. What do types mean? — From intrinsic to extrinsic semantics. In A. McIver and C. Morgan, editors, *Essays on Programming Methodology*. Springer, 2002.

[46] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, Nov. 1982.

[47] I. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998.

[48] R. D. Tennent and D. R. Ghica. Abstract models of storage. *Higher-Order and Symbolic Computation*, 13(1–2):119–129, Apr. 2000.

[49] L. Thorup and M. Tofte. Object-oriented programming and standard ML. In *Record of the 1994 ACM SIGPLAN Workshop on Standard ML and its Applications*, 1994.

[50] A. K. Wright. Simple imperative polymorphism. *LISP and Symbolic Computation*, 8(4):343–355, Dec. 1995.

[51] Zhang and Nowak. Logical relations for dynamic name creation. In *17th Workshop on Computer Science Logic (CSL 2003)*, volume 2803 of *Lecture Notes in Computer Science*, pages 575–588. Springer, 2003.