A Typed Semantics of 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 intrinsic denotational model for a similar language based on a possible-world semantics, recently given by Levy [14], and relate it to an untyped model by a logical relation. Following the methodology of Reynolds [22], this relation is used to establish coherence of the typed semantics, with a coercion interpretation of subtyping. We obtain a typed denotational semantics of (imperative) object-based languages.

1 Introduction

Languages such as Standard ML and Scheme allow to store values of arbitrary types, including function types. Essentially the same effect is pervasive in object-based languages (see [1, 21]), 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: Besides introducing recursion to the language [13], 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 [4, 5, 14], and most of the work done on semantics of storage does not readily apply to languages with higher-order store.

In a recent paper, Paul Levy proposed a typed semantics for a language with higher-order functions and higher-order store [14]. 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 [22], 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., [7]).

In contrast to intrinsic semantics, an *extrinsic* semantics gives meaning to *all* terms. Types (and typing judgements) are interpreted as, e.g., predicates or partial equivalence relations over an untyped model. Usually, the interpretation of subtyping is straightforward in such models. In [22], 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. 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* [16] 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 logical relation requires a non-trivial proof — we use the framework of Pitts [18] 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, 11]. Some evidence that the model of this paper can indeed serve as basis for such logics is provided, by giving a semantics to the object calculus [1]. This is done using a typed variant of Kamin and Reddy's "closure model" [11]. To the best of our knowledge this is the first (intrinsically typed) domain-theoretic model of the imperative object calculus.

In summary, our technical contributions here are (1) we present a model of a language that includes general references and subtyping, (2) we successfully apply the ideas of Reynolds [22] to prove coherence, and (3) we provide the first (typed) model of the imperative object calculus, based on cpos.

Structure of the paper. In the next section, language and type system are introduced. Then, typed and untyped models are presented (Sects. 3 and 4). 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 Sect. 6. In Sect. 7 both a derived per semantics and an interpretation of objects in the model are discussed.

Table 1. Typing

$$\frac{\Gamma \rhd e : A \quad A \preceq B}{\Gamma \rhd e : B} \qquad \frac{x{:}A \in \Gamma}{\Gamma \rhd x : A}$$

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

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

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

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

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

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

Complete proofs, further examples and discussions can be found in the technical report [23].

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 $\mathbf{1} \overset{def}{=} \{\}$ to be the empty record type. Finally, we have a type $\mathsf{ref}\ A$ 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:

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

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

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

Note that there is no rule for reference types as these need to be invariant, i.e., $\mathsf{ref}\ A \preceq \mathsf{ref}\ B$ 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 for subtyping of terms.

3 Intrinsic Semantics

In this section we recall the possible worlds model of [14]. 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. 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 discrete 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, $[\![\{\mathsf{m}_i : A_i\}_{i \in I}]\!]_w = \{\!|\mathsf{m}_i : [\![A_i]\!]_w\,|\!\}_{i \in I}$ 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 $[\![\mathsf{ref}\ A]\!]_w \subseteq [\![\mathsf{ref}\ A]\!]_{w'}$. Records act pointwise on the components, $[\![\{\mathsf{m}_i : A_i\}]\!]_w^{w'} = \lambda r.\{\!|\mathsf{m}_i = [\![A_i]\!]_w^{w'}(r.\mathsf{m}_i)|\!\}$. The type of functions $A \Rightarrow B$ is the most interesting since it involves the store S,

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

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 [14] 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; and
– in the category $\mathcal{C}^E$ of embedding-projection pairs of $\mathcal{C}$, every ω-chain $\Delta = D_0 \to D_1 \to \dots$ has an O-colimit [24], 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 $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ 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 $\delta(e) \overset{def}{=} F(e, e)$ [18].

Following [14] the semantics of types can now be obtained as minimal invariant of the locally continuous functor $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ (derived from the domain equations for types by separating positive and negative occurrences of the store) over the bilimit-compact category

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

Here, $[\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}]$ denotes the category where objects are functors $A, B : \mathcal{W} \to \mathbf{Cpo}$ and morphisms are partial natural transformations $\mu : A \overset{\cdot}{\rightharpoonup} B$, 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 define $S_w \overset{def}{=} D_{Sw}$ 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, D gives 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 $A \preceq B$ determines a *coercion*, which is in fact a (total) natural transformation from $[\![A]\!]$ to $[\![B]\!]$, defined in Table 2. We follow the notation of [22] and write $\mathcal{P}(J)$ to distinguish a *derivation* of judgement J from the judgement itself.

Writing $[\![\Gamma]\!]_w$ for the set of maps from variables to $\bigcup_A [\![A]\!]_w$ s.t. $\rho(x) \in [\![A]\!]_w$ for all $x{:}A \in \Gamma$, we define the semantics of (derivations of) typing judgments

$$[\![\Gamma \rhd e : A]\!]_w : [\![\Gamma]\!]_w \to S_w \rightharpoonup \textstyle\sum_{w'\geq w}(S_{w'} \times [\![A]\!]_{w'}) \enspace .$$

As observed in Levy's paper, each *value* $\Gamma \rhd v : A$ 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.

Table 2. Coercion maps

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

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

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

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

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

The semantics of subtyping judgements is used for the subsumption rule,

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

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

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

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 $\langle w, \langle s, f\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd x : A \Rightarrow B)]\!]_w\, \rho s$ and $\langle w, \langle s, a\rangle\rangle = [\![\mathcal{P}(\Gamma \rhd y : A)]\!]_w\, \rho s$. The remaining cases are similarly straightforward (see [14, 23]).

4 An Untyped Semantics

We give an untyped semantics of the language in **pCpo**. Let Val satisfy

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

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:

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

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

Table 3. Kripke logical relation

$$
\begin{array}{lcl}
\langle x, y\rangle \in R_w^{\mathsf{bool}} & \overset{def}{\Longleftrightarrow} & y \in \mathsf{BVal} \ \wedge \ x = y \\
\langle r, s\rangle \in R_w^{\{\mathsf{m}_i : A_i\}} & \overset{def}{\Longleftrightarrow} & s \in \mathsf{Rec}_{\mathbb{L}}(\mathsf{Val}) \ \wedge \ \forall i.\ (s.\mathsf{m}_i\!\downarrow \ \wedge \langle r.\mathsf{m}_i, s.\mathsf{m}_i\rangle \in R_w^{A_i}) \\
\langle f, g\rangle \in R_w^{A \Rightarrow B} & \overset{def}{\Longleftrightarrow} & g \in [\mathsf{St} \times \mathsf{Val} \rightharpoonup \mathsf{St} \times \mathsf{Val}] \ \wedge \\
& & \forall w' \geq w\ \forall \langle s, \sigma\rangle \in R_{w'}^{St}\ \forall \langle x, y\rangle \in R_{w'}^{A} \\
& & \quad (f_{w'}(s, x)\!\uparrow \ \wedge g(\sigma, y)\!\uparrow) \\
& & \quad \vee\ \exists w'' \geq w'\ \exists s' \in S_{w'}\ \exists x' \in [\![B]\!]_{w'}\ \exists \sigma' \in \mathsf{St}\ \exists y' \in \mathsf{Val}. \\
& & \qquad (f_{w'}(s, x) = \langle w'', \langle s', x'\rangle\rangle \ \wedge \ g(\sigma, y) = \langle \sigma', y'\rangle \\
& & \qquad \wedge \langle s', \sigma'\rangle \in R_{w''}^{St} \ \wedge \ \langle x', y'\rangle \in R_{w''}^{B}) \\
\langle x, y\rangle \in R_w^{\mathsf{ref}\ A} & \overset{def}{\Longleftrightarrow} & y \in w \cap \mathsf{Loc}_A \ \wedge \ x = y
\end{array}
$$

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

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

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

The semantics of new_A may be slightly surprising as there is still some 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 Sect. 5 below. See also the discussion in Sect. 7.1.

5 A Kripke Logical Relation

While in [22] a logical relation between typed and untyped models was used to establish coherence, here this must be slightly generalised to a Kripke logical relation. Kripke logical relations are not only indexed by types but also by possible worlds, subject to a monotonicity condition (Lemma 2 below).

In Table 3 such a family of *Type*- and $\mathcal{W}$-indexed relations $R_w^A \subseteq [\![A]\!]_w \times \mathsf{Val}$ is defined. The existence of this family R has to be established: There are both positive and negative occurrences of R_w^{St} in the case of function types $A \Rightarrow B$. Thus 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).

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 $G : \mathbf{pCpo}^{op} \times \mathbf{pCpo} \to \mathbf{pCpo}$

be the locally continuous functor for which (4) is the minimal invariant, so that $\langle D, \mathsf{Val}\rangle$ is the minimal invariant of $F \times G$. A relational structure $\mathcal{R}$ on the category $\mathcal{C} \times \mathbf{pCpo}$, in the sense of [18], 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 $R^A_w \subseteq X_{Aw} \times Y$ and $R^{St}_w \subseteq X_{Sw} \times \mathsf{Rec}_{\mathsf{Loc}}(Y)$.
- 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 $f : R \subset S$ iff, for all $w \in \mathcal{W}$, $A \in \mathit{Type}$, for all $x \in X_{Aw}$, $y \in Y$, $s \in X_{Sw}$ and $\sigma \in \mathsf{Rec}_{\mathsf{Loc}}(Y)$,

$$\langle x, y\rangle \in R^A_w \implies \begin{cases} f_{1\,Aw}(x)\uparrow \wedge f_2(y)\uparrow \quad \text{or} \\ f_{1\,Aw}(x)\downarrow \wedge f_2(y)\downarrow \wedge \langle f_{1\,Aw}(x), f_2(y)\rangle \in S^A_w \end{cases}$$

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

We define a functional $\Phi(R^-, R^+)$ on $\mathcal{R}$ corresponding to the equations for the Kripke logical relation R above (by separating positive and negative occurrences of R in the right-hand sides) such that for $S \in \mathcal{R}(X, Y)$ and $S' \in \mathcal{R}(X', Y')$ we have $\Phi(S, S') \in \mathcal{R}((F \times G)(\langle X, Y\rangle\langle X', Y'\rangle))$. The map Φ is an admissible action of the functor $F \times G$ on $\mathcal{R}$, in the following sense:

Lemma 1. *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 $(F \times G)(e, f) : \Phi(R, S) \subset \Phi(R', S')$.*

According to [18], Lemma 1 guarantees that Φ has a unique fixed point $\mathit{fix}(\Phi)$ in $\mathcal{R}(D, \mathsf{Val})$, and we obtain the Kripke logical relation $R = \mathit{fix}(\Phi)$ satisfying $R = \Phi(R, R)$ as required.

Theorem 1 (Existence, [18]). *The functional Φ has a unique fixed point.*

5.2 The Basic Lemma

By induction on A and the derivation of $A \preceq B$, resp., the following monotonicity properties are established:

Lemma 2 (Kripke Monotonicity). *Suppose $\langle a, u\rangle \in R^A_w$ and $w' \geq w$. Then $\langle [\![A]\!]^{w'}_w (a), u\rangle \in R^A_{w'}$.*

Lemma 3 (Subtype Monotonicity). *Let $w \in \mathcal{W}$, $A \preceq B$ and $\langle a, u\rangle \in R^A_w$. Then $\langle [\![A \preceq B]\!]_w (a), u\rangle \in R^B_w$.*

Lemmas 2 and 3 show a key property of the relation R, which is at the heart of the coherence proof: For $\langle a, u\rangle \in R^A_w$ 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. It is not hard to prove the fundamental property of logical relations which says that the (typed and untyped) denotations of well-typed terms compute related results.

Table 4. Bracketing maps

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

Lemma 4 (Basic Lemma). *Suppose $\Gamma \triangleright e : A$, $w \in \mathcal{W}$, $\langle\rho, \eta\rangle \in R_w^{\Gamma}$ and $\langle s, \sigma\rangle \in R_w^{St}$. Then*

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

Proof. By induction on the derivation of $\Gamma \triangleright e : A$, using Lemmas 2 and 3. $\square$

5.3 Bracketing

Next, in Table 4, 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.

As in [22], the retraction property follows from a more general result which justifies the term "bracketing", $\phi_w^A \subseteq R_w^A \subseteq (\psi_w^A)^{op}$, relating the (graphs of the) bracketing maps and the Kripke logical relation of the previous section.

Theorem 2 (Bracketing). *For all $w \in \mathcal{W}$ and $A \in Type$,*

- $\forall x \in [\![A]\!]_w\,.\ \langle x, \phi_w^A(x)\rangle \in R_w^A$;

– $\forall s \in S_w.\ \langle s, \phi^{St}_w(s)\rangle \in R^{St}_w$;
– $\forall \langle x, y\rangle \in R^A_w.\ x = \psi^A_w(y)$; *and*
– $\forall \langle s, \sigma\rangle \in R^{St}_w.\ s = \psi^{St}_w(\sigma)$.

Proof (Sketch). Compared to Reynolds work, the proof of the theorem is more involved, again due to the (mixed-variant) type recursion caused by higher-order store. By a simultaneous induction on n we first prove the properties

– $\forall x \in [\![A]\!]_w.\ \pi^{Aw}_n(x)\downarrow \implies \langle \pi^{Aw}_n(x), \phi^A_w(\pi^{Aw}_n(x))\rangle \in R^A_w$
– $\forall \langle x, y\rangle \in R^A_w.\ \pi^{Aw}_n(x)\downarrow \implies \pi^{Aw}_n(x) = \pi^{Aw}_n(\psi^A_w(y))$

for all $n \in \mathbb{N}$, using 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 $\pi^{Sw}_n \overset{def}{=} \delta^n(\bot)_{Sw}$. 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. Also, $\bigsqcup_n \pi^{Sw}_n = \mathsf{id}_{Sw}$.

Now for the first part of the theorem let $x \in [\![A]\!]_w$. Thus, $x = \bigsqcup_n \pi^{Aw}_n(x)$ entails $\pi^{Aw}_n(x)\downarrow$ for n sufficiently large. By the above, $\langle \pi^{Aw}_n(x), \phi^A_w(\pi^{Aw}_n(x))\rangle \in R^A_w$ for all sufficiently large $n \in \mathbb{N}$. This is a countable chain in $[\![A]\!]_w \times \mathsf{Val}$, and admissibility of R^A_w and continuity of ϕ^A_w prove the result. 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 [22]): Suppose $\mathcal{P}_1(\Gamma \rhd e : A)$ and $\mathcal{P}_2(\Gamma \rhd e : A)$ are derivations of the judgement $\Gamma \rhd e : A$. We show that their semantics agree. Let $w \in \mathcal{W}$, $\rho \in [\![\Gamma]\!]_w$ and $s \in S_w$. By Theorem 2 parts (1) and (2), $\langle \rho, \phi^\Gamma_w(\rho)\rangle \in R^\Gamma_w$ and $\langle s, \phi^{St}_w(s)\rangle \in R^{St}_w$. Hence, by two applications of the Basic Lemma, either

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

or else there exist $w_i \geq w$, $s_i \in S_{w_i}$, $v_i \in [\![A]\!]_{w_i}$ and $\sigma \in \mathsf{St}$, $v \in \mathsf{Val}$ such that

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

where $\langle s_i, \sigma\rangle \in R^{St}_{w_i}$ and $\langle v_i, v\rangle \in R^A_{w_i}$, for $i = 1, 2$. The definition of the relation $R^{St}_{w_i}$ entails $w_1 = \mathsf{dom}(\sigma) = w_2$, and by Theorem 2 parts (3) and (4), $s_1 = \psi^{St}_{w_1}(\sigma) = \psi^{St}_{w_2}(\sigma) = s_2$ and $v_1 = \psi^A_{w_1}(v) = \psi^A_{w_2}(v) = v_2$. Thus we have shown

Theorem 3 (Coherence). *All derivations of a judgement $\Gamma \rhd e : A$ 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

$$x{:}\{\mathsf{m} : \mathsf{bool}\} \rhd \mathsf{new}\ x; \mathsf{true} : \mathsf{bool} \tag{5}$$

one without use of subsumption, and one where x is coerced to type **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 (5) 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 Discussion

We consider some aspects in more detail. Firstly, the technical development so far can be used to obtain an (extrinsic) semantics over the untyped model, based on partial equivalence relations. Secondly, we show that our simple notion of subtyping is useful in obtaining a pleasingly straightforward semantics of the object calculus [1]. Finally, we demonstrate 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.

7.1 Extrinsic PER Semantics

Apart from proving coherence, Reynolds used (his analogue of) Theorem 2 to develop an *extrinsic* semantics of types in the language [22]. Besides Theorem 2 this only depends on the Basic Lemma, and we can do exactly the same here. More precisely, the binary relation $||A||_w$, defined as $(R^A_w)^{op} \circ R^A_w$, is a partial equivalence relation (per) on $\mathsf{Val} \times \mathsf{Val}$. We observe that a direct proof of transitivity is non-trivial, but it follows easily with part (3) of Theorem 2.

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 $\Gamma \rhd e : A$ then for all $w \in \mathcal{W}$, for all $\langle \eta, \eta' \rangle \in ||\Gamma||_w$ and all $\langle \sigma, \sigma' \rangle \in ||w||$,

$$[\![e]\!]\,\eta\sigma\!\downarrow\ \vee\ [\![e]\!]\,\eta'\sigma'\!\downarrow \quad\Longrightarrow\quad \begin{cases} [\![e]\!]\,\eta\sigma = \langle \sigma_1, u \rangle \ \wedge\ [\![e]\!]\,\eta'\sigma' = \langle \sigma_1', u' \rangle\ \wedge \\ \exists w' \geq w.\ \langle \sigma_1, \sigma_1' \rangle \in ||w'||\ \wedge\ \langle u, u' \rangle \in ||A||_{w'} \end{cases} \tag{6}$$

The resulting per model satisfies some of the expected typed equations: For instance, $\{\!|\mathsf{m} = \mathit{true}, \mathsf{m}' = \mathit{true}|\!\}$ and $\{\!|\mathsf{m} = \mathit{true}, \mathsf{m}' = \mathit{false}|\!\}$ are equal at $\{\mathsf{m} : \mathsf{bool}\}$. Unfortunately, no non-trivial equations involving store are valid 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 [14].

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 [19], and Benton and Leperchey [6], shows how more useful equivalences can be built in into typed models of languages with storable references. It would be interesting to investigate if 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 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.

Further, it is interesting to observe the role 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.

Previously we have given a denotational semantics for a logic of objects [3], where an untyped cpo model was used [20]. This logic has a built-in notion of invariance which makes it very similar to a type system, and the semantic structure of function types used in [20] closely resembles (6). In fact, in [20] an ad-hoc construction was necessary to "determinise" the existential quantification over world extensions of (6) in order to preserve admissibility of predicates (corresponding to types and specifications of the logic). 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.

7.2 A Semantics of Objects

Next, we sketch how to give a semantics to Abadi and Cardelli's imperative object calculus with first-order types [1], 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

$$\frac{\begin{array}{c} A \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{\Rightarrow}C_j]_{i,j} \\ \Gamma \vdash x_i : A_i \ \forall i \quad \Gamma, y_j{:}A, z_j{:}B_j \vdash b_j : C_j \ \forall j \end{array}}{\Gamma \vdash [\mathsf{f}_i = x_i, \mathsf{m}_j = \varsigma(y_j)\lambda z_j.\, b_j]_{i,j} : A} \tag{7}$$

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

$$\frac{B_j{\Rightarrow}C_j \preceq B'_j{\Rightarrow}C'_j \ \forall j \in J' \quad I' \subseteq I \quad J' \subseteq J}{[\mathsf{f}_i : A_i, \mathsf{m}_j : B_j \Rightarrow C_j]_{i\in I, j\in J} \preceq [\mathsf{f}_i : A_i, \mathsf{m}_j : B'_j \Rightarrow C'_j]_{i \in I', j\in J'}} \tag{8}$$

The following is essentially a (syntactic) presentation of the fixed-point (or *closure*) model of objects [11], 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, cf. [8]), and soundness of (8) follows directly from the rules of Sect. 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

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

Here $\mu f(x).e$ is a recursively defined function f; note that this is meaningful: Using the fixed-point of the map $h \mapsto [\![\lambda x.e]\!]_w\, \rho[f := h]$ in **Cpo** recursive functions can be interpreted in the model. The Basic Lemma holds also for the language extended with recursive functions [23]. Soundness of (7) follows immediately from this interpretation of objects and object types.

7.3 Reasoning about Higher-order Store and Objects

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 $B \preceq A$). The program computes the factorial, making the recursive calls through the store.

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

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 higher-order store, following the general ideas of [21]: 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 $P_w \subseteq [\![\mathsf{int} \Rightarrow \mathsf{int}]\!]_w$,

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

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.* [9] (where, rather different to here, the proof relies on well-founded induction using a termination order).

Existence of P is established along the lines of Theorem 1. Then, assuming that l is the location allocated for field f, a simple fixed-point induction shows

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

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

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

for some $w'' \in \mathcal{W}$ and $s'' \in S_{w''}$.

8 Related Work

Possible worlds models of programming languages were first considered in the work of Reynolds and Oles on the semantics of local stack-allocated variables [17]. The current work is closer in spirit to the various possible worlds models for languages with dynamic allocation of *heap* storage [14, 19, 25, 6].

Apart from Levy's work [14, 15] which we built upon here, we are aware of only few other semantic models of higher-order store in the literature. The models [4, 12] 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 Sect. 7.3. Ahmed, Appel and Virga [5] 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 [10] 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.

The proof principles applied in Sect. 7.3 are direct adaptations of those presented in [21] in the context of an untyped model of the object calculus.

9 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 [22], the overall structure of the coherence proof is almost identical to *loc.cit.* It could be interesting to work out the general conditions needed for the construction.

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 [8]). In [15] 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. In fact, also the extension with ML-like (prenex) polymorphism is straightforward – essentially because there is no interaction with the store.

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

Acknowledgement I wish to thank Bernhard Reus for many helpful discussions. Paul Levy pointed out a flaw in an earlier version. Financial support was provided by EPSRC 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 *Proc. POPL'96*, pages 396–409. 1996.
3. M. Abadi and K. R. M. Leino. A logic of object-oriented programs. In *Verification: Theory and Practice. Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, LNCS, pages 11–41. Springer, 2004.
4. S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc. LICS'98*, pages 334–344. 1998.
5. A. J. Ahmed, A. W. Appel, and R. Virga. A stratified semantics of general references embeddable in higher-order logic. In *Proc. LICS'02*, pages 75–86. 2002.
6. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. TLCA'05*, volume 3461 of LNCS, pages 86–101. 2005.
7. V. Breazu-Tannen, T. Coquand, G. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.
8. K. B. Bruce, L. Cardelli, and B. C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, Nov. 1999.
9. K. Honda, M. Berger, and N. Yoshida. An observationally complete program logic for imperative higher-order functions. To appear in *Proc. LICS'05*, 2005.
10. A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. In *Proc. LICS'02*, pages 101–112. 2002.
11. S. N. Kamin and U. S. Reddy. Two semantic models of object-oriented languages. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, pages 464–495. MIT Press, 1994.
12. J. Laird. A categorical semantics of higher-order store. In *Proc. CTCS'02*, volume 69 of *ENTCS*, pages 1–18. 2003.
13. P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4):308–320, Jan. 1964.
14. P. B. Levy. Possible world semantics for general storage in call-by-value. In *Proc. CSL'02*, volume 2471 of *LNCS*. 2002.
15. P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*, volume 2 of *Semantic Structures in Computation*. Kluwer, 2004.
16. 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.
17. F. J. Oles. *A Category-theoretic approach to the semantics of programming languages*. PhD thesis, Syracuse University, 1982.
18. A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
19. 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.
20. B. Reus and J. Schwinghammer. Denotational semantics for Abadi and Leino's logic of objects. In *Proc. ESOP'05*, volume 3444 of *LNCS*, pages 264–279. 2005.
21. B. Reus and T. Streicher. Semantics and logic of object calculi. *Theoretical Computer Science*, 316:191–213, 2004.
22. J. C. Reynolds. What do types mean? — From intrinsic to extrinsic semantics. In *Essays on Programming Methodology*. Springer, 2002.
23. J. Schwinghammer. A typed semantics for languages with higher-order store and subtyping. Technical Report 2005:05, Informatics, University of Sussex, 2005.
24. 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.
25. I. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998.