Semantics and Logic of Object Calculi

Bernhard Reus*
COGS, University of Sussex
Falmer, Brighton, UK
bernhard@cogs.susx.ac.uk

Thomas Streicher
Fachbereich Mathematik
TU Darmstadt, GERMANY
streicher@mathematik.tu-darmstadt.de

Abstract

The main contribution of this paper is a formal characterization of recursive object specifications based on a denotational untyped semantics of the object calculus and the discussion of existence of those (recursive) specifications. The semantics is then applied to prove soundness of a programming logic [2] for the object calculus [1] and to suggest possible extensions. For the purposes of this discussion we use an informal *logic of predomains in order to avoid any commitment to a particular syntax of specification logic.*

1 Introduction and motivation

Programming logics have been suggested for object calculi [2, 8] as well as object-oriented programming languages [3, 6, 16, 7, 11, 12]. Since objects are inherently recursive due to self-application of objects in method calls, their specifications are recursive too. Soundness proofs of programming logics for objects w.r.t. operational semantics can thus become rather contrived. Moreover, existence of object specifications is generally neglected although a subtle point. In general, the meaning of a specification is fully described by its introduction rule w.r.t. object formation. Therefore, the existence of the specification is equivalent to the validity of its introduction rule. The resulting implicit definition of a specification $S = \Phi(S)$ neither guarantees existence nor uniqueness unless Φ is of a certain form[1]. It is Domain Theory that provides sufficient machinery to guarantee existence *and* uniqueness. Therefore, working with a denotational (domain theoretic) semantics puts us into a position to precisely account for this problem.

As far as the authors are aware, such a systematic denotational analysis of object logics has not been carried through yet although there is a successful role model, the LCF (*Logic of Computable Functions*) logic, for the functional paradigm.

The outline of this paper is as follows. Firstly, a denotational semantics of the functional and imperative untyped object calculus of [1] is given. On top of that a notion of specification inspired by the Abadi & Leino logic [2] is defined on the resulting object domains. We prove existence of these specifications under mild assumptions employing Andy Pitts' machinery for relational properties of domains [10].

The existence theorem is not only interesting in its own right, it will be also applied to

- prove soundness of the object formation rule in [2] (in an untyped way but types can be encoded as specifications)
- exemplify via counterexamples that certain specification can not exist
- suggest extensions of an existing programming logic [2, 8] introducing method invariants and logic of higher order store.

Moreover, the proposed technique is expected to be applicable to various other object-oriented languages and programming logics.

Some basic ideas date back to a draft [13] an abstract of which has been published in [14].

2 Denotational model of the object calculus

In this section we describe a most simple denotational semantics for functional and imperative object calculi within the category PreDom of predomains and partial continuous functions. Let $\rightharpoonup$ denote the partial continuous function space between predomains. By $f(a)\uparrow$ we denote that function f applied to a is undefined whereas $f(a)\downarrow$ denotes definedness.

*First author partially supported by the EPSRC under grant GR/R65190/01 and the Nuffield Foundation under grant NAL/00244/A.

[1]Usually, if Φ is monotonic then S is recursively defined. But monotonicity is too strong a condition for object specifications.

2.1 Preliminaries

When specifying the recursive types needed for the interpretation of object calculi we often have to employ record type formation in the following sense. Let $\mathbb{L}$ be a (countable) set of labels and A a predomain then the type of records with entries from A and labels from $\mathbb{L}$ is defined as follows:

$$\mathsf{Rec}_{\mathbb{L}}(A) = \Sigma_{L \in \mathcal{P}_{\mathsf{fin}}(\mathbb{L})} A^L$$

where A^L is the set of all total functions from L to A. It is easily seen that $\mathsf{Rec}_{\mathbb{L}}$ is a locally continuous functor on PreDom. A record with labels l_i and corresponding entries v_i $(1 \leq i \leq k)$ is written $\{\!|l_1 = v_1, \dots l_k = v_k|\!\}$. Notice that $\mathsf{Rec}_{\mathbb{L}}(A)$ is always non–empty as it contains the element $\langle \emptyset, \emptyset \rangle$.

Definition 2.1 *The update (and extension) operation for records is defined as in Table 1 (page 3). Selection of a label $a \in \mathbb{L}$ in a record $r \in \mathsf{Rec}_{\mathbb{L}}(A)$, short $r.a$, is defined if $r = \langle D, f \in A^D \rangle$ and $a \in D$ yielding $f(a) \in A$.*

2.2. Functional object calculus

Let us first recall the syntax of the functional object calculus of [1]. For the sake of simplicity, in the *functional* case we do not distinguish between field and method update so $\mathsf{m} \in \mathcal{M} \cup \mathcal{F}$ where $\mathcal{M}$ and $\mathcal{F}$ are sets of method names and field names, respectively.

$$\begin{array}{rcll} a, b & ::= & x & \text{variable} \\ & | & [\mathsf{m}_i{=}\varsigma(x_i)b_i]^{i=1..n} & \text{object creation} \\ & | & a.\mathsf{m}() & \text{method call} \\ & | & a.\mathsf{m}{\Leftarrow}\varsigma(x)b & \text{method update} \end{array}$$

Again, for simplicity, methods do not have additional arguments. This is not a real restriction as arguments can be encoded by fields. Let BVal denote the flat predomain of basic values like numbers or booleans. The functional object calculus most naturally finds its interpretation within the recursively specified predomain

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

which is nonempty as record types are always non–empty. If we choose BVal to be empty then we get the recursive type of [2]:

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

and if we consider a variant of object calculus without fields then we get the most simple recursive type

$$\mathcal{O} = \mathsf{Rec}_{\mathcal{M}}(\mathcal{O}{\rightharpoonup}\mathcal{O})$$

which strongly reminds one of call–by–value lambda calculus as given by the type equation $L = L{\rightharpoonup}L$. The difference essentially is that an object is not just a partial continuous function from objects to objects but a whole record of such. For an "object" $o{\in}\mathcal{O}$ and a "message" $\mathsf{m} \in \mathcal{M}$ the result of "sending message m to object o" is given by $o.\mathsf{m}(o)$ which is understood as divergent if m does not occur as a label in the record o. It makes sense to conceive methods as partial continuous functions from $\mathcal{O}$ to $\mathcal{O}$, i.e. as *strict* functions, because if $o.\mathsf{m}$ is defined then the argument o has to be defined as well.

For the sake of simplicity, in this paper we will use the last domain equation to interpret the functional object-calculus which fits with the convention that in the calculus we do not distinguish between method and field update.[2]

We write $[\![a]\!]\rho$ for the interpretation of object expression a in the environment $\rho \in \mathsf{Env} = \mathcal{O}^{\mathsf{Var}}$. This interpretation is defined by structural recursion on object expressions via the following clauses:

$$\begin{array}{lcl} [\![x]\!]\rho & = & \rho(x) \\ [\![[\mathsf{m}_i{=}\varsigma(x_i)b_i]^{i=1..n}]\!]\rho & = & \{\!|\mathsf{m}_i{=}\lambda o.[\![b_i]\!]\rho[o/x_i]|\!\}^{i=1..n} \\ [\![a.\mathsf{m}()]\!]\rho & = & ([\![a]\!]\rho).\mathsf{m}([\![a]\!]\rho) \\ [\![a.\mathsf{m} \Leftarrow \varsigma(x)b]\!]\rho & = & [\![a]\!]\rho[\mathsf{m}{:=}\lambda o.[\![b]\!]\rho[o/x]] \ . \end{array}$$

2.3. Imperative object calculus

Recall that the syntax of the imperative untyped object calculus of [1] is as follows if we distinguish between fields and methods:

$$\begin{array}{rcll} a, b & ::= & x & \text{variable} \\ & | & [\mathsf{m}_i{=}\varsigma(x_i)b_i]^{i=1..n} & \text{object creation} \\ & | & a.\mathsf{f} & \text{field selection} \\ & | & a.\mathsf{f}{\Leftarrow}b & \text{field update} \\ & | & a.\mathsf{m}() & \text{method call} \\ & | & a.\mathsf{m}{\Leftarrow}\varsigma(x)b & \text{method update} \\ & | & \mathtt{clone}(a) & \text{shallow copy} \\ & | & \mathtt{let}\,x{=}a\,\mathtt{in}\,b & \text{local def.} \end{array}$$

The imperative calculus finds its interpretation within the following slightly more complicated system of recursive types:

(1) $\mathsf{Val} = \mathsf{BVal}{+}\mathsf{Loc}$

(2) $\mathsf{St} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob})$

(3) $\mathsf{Ob} = \mathsf{Rec}_{\mathcal{F}}(\mathsf{Val}){\times}\mathsf{Rec}_{\mathcal{M}}(\mathsf{Cl})$

(4) $\mathsf{Cl} = \mathsf{Loc}{\times}\mathsf{St}{\rightharpoonup}\mathsf{Val}{\times}\mathsf{St}$

where Loc is some countable set of locations (considered as a predomain). Some notation will come in handy in later sections. We write $\mathsf{St}_{\mathsf{Val}}$ for $\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathcal{F}}(\mathsf{Val}))$. There is an obvious projection $\pi_{\mathsf{Val}} : \mathsf{St}{\to}\mathsf{St}_{\mathsf{Val}}$ with $\pi_{\mathsf{Val}}(\sigma).\ell \simeq \pi_1(\sigma.\ell)$ where π_1 projects on the first component.

[2] Note that we will distinguish, however, in the object logic.

$$\{\!|\mathsf{m}_i{=}f_i|\!\}^{i=1\ldots n}[\mathsf{m}{:=}f] \quad = \quad \begin{cases} \{\!|\mathsf{m}_1{=}f_1,\ldots,\mathsf{m}_n{=}f_n,\mathsf{m}{=}f|\!\} & \text{if } \mathsf{m} \text{ is different from all } \mathsf{m}_i \\ \{\!|\mathsf{m}_1{=}f_1,\ldots,\mathsf{m}_i{=}f,\ldots,\mathsf{m}_n{=}f_n|\!\} & \text{if } \mathsf{m} = \mathsf{m}_i \end{cases}$$

Table 1. Definition of record update (extension)

Notice that the definition of St as $\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob})$ faithfully reflects the idea of a state as an assignment of objects to a finite set of locations. We think that this modelling of states as records should also be employed when modelling e.g. simple imperative languages where only basic values can be stored to locations. Besides conceptual adequacy a technical advantage of such a modelling is that $\mathsf{Rec}_{\mathsf{Loc}}(\mathbb{N})$ is a countable flat predomain whereas the traditional choice $\mathsf{Loc}{\to}\mathbb{N}_\perp$ is uncountable and highly non–flat.

Definition 2.2 *Given an environment $\rho \in \mathsf{Env} = \mathsf{Loc}^{\mathsf{Var}}$ and an object expression a its interpretation $[\![a]\!]\rho : \mathsf{St}{\rightharpoonup}\mathsf{Loc}{\times}\mathsf{St}$ is defined in Table 2.2 (page 4). Note that for an $o \in \mathcal{O}$ we write $o.\mathsf{f}$ and $o.\mathsf{m}$ instead of $\pi_1(o).\mathsf{f}$ and $\pi_2(o).\mathsf{m}$, resp. to reduce syntactic clutter.*

If one does not distinguish between methods and fields and ignores basic values, as Abadi & Cardelli do in [1], the above system of mutual recursive type definitions simplifies as follows

(1) $\mathsf{St} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob})$

(2) $\mathsf{Ob} = \mathsf{Rec}_{\mathcal{M}}(\mathsf{Cl})$

(3) $\mathsf{Cl} = \mathsf{Loc}{\times}\mathsf{St}{\rightharpoonup}\mathsf{Loc}{\times}\mathsf{St}$

The fact that fields and methods are not distinguished implies that method overriding is allowed.

Notice that equivalently Ob can be defined by the single recursive equation

$$\mathsf{Ob} = \mathsf{Rec}_{\mathcal{M}}(\mathsf{Loc}{\times}\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob}){\rightharpoonup}\mathsf{Loc}{\times}\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob}))$$

which, obviously, is obtained from $\mathcal{O} = \mathsf{Rec}_{\mathcal{M}}(\mathcal{O}{\rightharpoonup}\mathcal{O})$ by simply replacing $\mathcal{O}$ by $\mathsf{Loc}{\times}\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob})$ on the right hand side.

2.3.1 Variation à la Abadi & Cardelli

Notice that the denotational semantics presented above is not quite in accordance with the operational semantics for the imperative object calculus given in the book [1] of Abadi and Cardelli (pp. 136-137) which implies the following domain equations

(1) $\mathsf{Val} = \mathsf{Rec}_{\mathcal{M}}(\mathsf{Loc})$

(2) $\mathsf{St} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Cl})$

(3) $\mathsf{Cl} = \mathsf{Val}{\times}\mathsf{St}{\rightharpoonup}\mathsf{Val}{\times}\mathsf{St}$

where method closures are saved in the store and meanings of object expressions a in an environment $\rho \in \mathsf{Env} = \mathsf{Val}^{\mathsf{Var}}$ are functions of type $\mathsf{St}{\rightharpoonup}\mathsf{Val}{\times}\mathsf{St}$ and not of type $\mathsf{St}{\rightharpoonup}\mathsf{Loc}{\times}\mathsf{St}$. We leave it as an exercise for the inclined reader to give the semantic equations for the object calculus according to the domain equations above such that the interpretation is adequate w.r.t. operational semantics in [1].

3 Object specifications

Having identified the meaning of the (functional and imperative resp.) object calculus (within the recursively defined predomain $\mathcal{O}$ and $\mathsf{Loc} \times \mathsf{St}$) we are in the position to use any logic of predomains for reasoning about objects. In the end one might find it useful to identify a special purpose calculus for reasoning about objects which finds its meaning by translation into some logic of predomains.[3] However, before embarking on such a project we rather discuss what is the shape of predicates expressing interesting properties of objects.

3.1 A first notion of specification

We consider the functional case first in order to understand what a specification of an object should be. A possible definition could be the following.

Definition 3.1 *Let $A \in \mathcal{P}(\mathcal{O}){\to}\mathcal{P}(\mathcal{O})$ monotonic, $\vec{B} = (B_{\mathsf{m}} \in \mathcal{P}(\mathcal{O}){\to}\mathcal{P}(\mathcal{O})$* monotonic*$)_{\mathsf{m}\in\mathcal{M}}$, and $\vec{T} = (T_{\mathsf{m}} \subseteq \mathcal{O}{\times}\mathcal{O})_{\mathsf{m}\in\mathcal{M}}$. Then these data induce a monotonic operator $\Phi_{A,\vec{B},\vec{T}} : \mathcal{P}(\mathcal{O}){\to}\mathcal{P}(\mathcal{O})$ which is defined as*

$$o \in \Phi_{A,\vec{B},\vec{T}}(S) \quad \equiv \quad o{\in}A(S) \wedge \forall \mathsf{m}{\in}\mathcal{M}.\, o.\mathsf{m}(o){\downarrow} \Rightarrow o.\mathsf{m}(o){\in}B_{\mathsf{m}}(S) \wedge T_{\mathsf{m}}(o, o.\mathsf{m}(o))$$

for $S \in \mathcal{P}(\mathcal{O})$. The B_{m} stand for result specifications and the T_{m} represent transition specifications for each method m. Finally, A specifies the remaining properties of the object, i.e. the fields. We write $\mathsf{Inv}(A, \vec{B}, \vec{T})$ for the greatest *fixpoint of $\Phi_{A,\vec{B},\vec{T}}$.*

[3] as e.g. Hoare logic which provides sort of a useful "macro-language" for reasoning about partial functions on states

$$
\begin{array}{lcl}
[\![x]\!]\rho\,\sigma & = & \langle \rho(x), \sigma\rangle \\
[\![[\mathsf{m}_i{=}\varsigma(x_i)b_i]^{i=1..n}]\!]\rho\,\sigma & = & \langle \ell, \sigma[\ell := \{\!|\mathsf{m}_i{=}\lambda\langle\ell',\sigma'\rangle.[\![b_i]\!]\rho[\ell'/x_i]\,\sigma'|\!\}^{i=1...n}]\rangle \\
 & & \text{where } \ell \text{ is a fresh location not in the domain of } \sigma \\
[\![a\,.\,\mathsf{f}]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } \sigma'.\ell.\mathsf{f} \\
[\![a\,.\,\mathsf{f}{\Leftarrow}b]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } \langle\ell, \sigma'[\ell{:=}\sigma'.\ell[\mathsf{f}{:=}[\![b]\!]\rho\,\sigma']]\rangle \\
[\![a.\mathsf{m}()]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } \sigma'.\ell.\mathsf{m}(\ell,\sigma') \\
[\![a\,.\,\mathsf{m}{\Leftarrow}\varsigma(x)b]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } \langle\ell, \sigma'[\ell{:=}\sigma'.\ell[\mathsf{m}{:=}\lambda\langle\ell',\sigma''\rangle.[\![b]\!]\rho[\ell'/x]\,\sigma'']]\rangle \\
[\![\texttt{clone}(a)]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } \langle\ell', \sigma'[\ell'{:=}\sigma'.\ell]\rangle \\
 & & \text{where } \ell' \text{ is a fresh location not in the domain of } \sigma' \\
[\![\texttt{let}\,x{=}a\,\texttt{in}\,b]\!]\rho\,\sigma & = & \textbf{let } \langle\ell,\sigma'\rangle{=}[\![a]\!]\rho\,\sigma \textbf{ in } [\![b]\!]\rho[\ell/x]\,\sigma'\ .
\end{array}
$$

Table 2. Denotational semantics for the imperative object calculus

If I is a post-fixpoint of $\Phi_{A,\vec{B},\vec{T}}$, i.e. $I \subseteq \Phi_{A,\vec{B},\vec{T}}(I)$, then every $o{\in}I$ satisfies the predicate A and whenever $o.\mathsf{m}(o)$ is defined then it satisfies $B_{\mathsf{m}}(I)$ and is related to o via T_{m}. In particular, this holds for the greatest fixpoint $\mathsf{Inv}(A,\vec{B},\vec{T})$ of $\Phi_{A,\vec{B},\vec{T}}$ as given by $\bigcup\{I{\in}\mathcal{P}(\mathcal{O}) | I{\subseteq}\Phi_{A,\vec{B},\vec{T}}(I)\}$, the union of all post-fixpoints of $\Phi_{A,\vec{B},\vec{T}}$. Thus, in order to prove that $o \in \mathsf{Inv}(A,\vec{B},\vec{T})$ one 'simply' has to exhibit a predicate P with $P{\subseteq}\Phi_{A,\vec{B},\vec{T}}(P)$ and $o{\in}P$.

Such a notion of "invariant" specification seems to be quite in accordance with the "coalgebraic view" of the object–oriented world and, therefore, is probably quite useful. However, it seems to have its limitations as exemplified by the following example.

Example 3.1 *Consider the object expression* $a \equiv [\mathsf{m}{=}\varsigma(x)x.\mathsf{m}()]$. *Operational intuition tells us that* $a.\mathsf{m}()$ *diverges and, therefore, it would be most desirable to prove this employing an appropriate notion of invariant. What comes to ones mind immediately is the invariant* $I = \mathsf{Inv}(A,\vec{B},\vec{T})$ *with* $A(S)(x) \equiv$ *True,* $T_{\mathsf{m}}(x,y) \equiv$ *False and* $B_{\mathsf{m}}(S) \equiv S$. *Then for I we have*

$$o \in I \;\Leftrightarrow\; \neg o.\mathsf{m}(o)\!\downarrow$$

Coinduction does not help in proving that $[\![a]\!] \in I$ *for an object a since one has to find a predicate P such that* $o \in P \Rightarrow o.\mathsf{m}(o)\!\downarrow\, \Rightarrow$ *false. But the only canonical choice for P is I again so nothing is achieved.*

In [2] Abadi and Leino introduced a sort of Hoare logic for a variant of the imperative object calculus which allows one to prove divergence of $a.\mathsf{m}()$ quite easily. For sake of simplicity we first discuss the following adaption of their account to the purely functional case.

3.2 Functional object specifications

A notion of specification for functional objects is suggested. The existence of such specifications is discussed in Section 4.

Definition 3.2 *For every* $A \in \mathcal{P}(\mathcal{O}) \to \mathcal{P}(\mathcal{O})$, $\vec{B} = (B_{\mathsf{m}} \in \mathcal{P}(\mathcal{O}) \to \mathcal{P}(\mathcal{O}))_{\mathsf{m}\in\mathcal{M}}$ *and for every* $\vec{T} = (T_{\mathsf{m}} \in \mathcal{P}(\mathcal{O}{\times}\mathcal{O}))_{\mathsf{m}\in\mathcal{M}}$ *let* $\mathsf{Spec}(A,\vec{B},\vec{T})$ *be the predicate* $S{\subseteq}\mathcal{O}$ *with*

$$
\begin{array}{lcl}
o \in S & \equiv & o \in A(S) \,\wedge\, \forall\mathsf{m}{\in}\mathcal{M}.\,\forall o'{\in}S.\; o.\mathsf{m}(o')\!\downarrow\; \Rightarrow \\
 & & o.\mathsf{m}(o') \in B_{\mathsf{m}}(S) \,\wedge\, T_{\mathsf{m}}(o', o.\mathsf{m}(o'))
\end{array}
$$

provided S is unique *with this property. We call* $\mathsf{Spec}(A,\vec{B},\vec{T})$ *the* specification induced by A, $\vec{B}$ and $\vec{T}$.

This is different from $\mathsf{Inv}(A,\vec{B},\vec{T})$ since one requires for methods $\mathsf{m}{\in}\mathcal{M}$ the condition $\forall o'{\in}S.\; o.\mathsf{m}(o')\!\downarrow\; \Rightarrow o.\mathsf{m}(o'){\in}B_{\mathsf{m}}(S) \,\wedge\, T_{\mathsf{m}}(o', o.\mathsf{m}(o'))$ to hold and not just $o.\mathsf{m}(o)\!\downarrow\; \Rightarrow\; o.\mathsf{m}(o){\in}B_{\mathsf{m}}(S) \,\wedge\, T_{\mathsf{m}}(o, o.\mathsf{m}(o))$. Note that S is implicitly ("recursively") specified even if the $\vec{B}$ and A do not depend on S.

Example 3.2 *To illustrate the power of this new notion we will employ the specification* $S = \mathsf{Spec}(\mathrm{True},\mathrm{True},\mathrm{False})$ *satisfying*

$$o \in S \;\Leftrightarrow\; (\forall o'{\in}S.\; o.\mathsf{m}(o')\!\downarrow\; \Rightarrow \mathrm{False})$$

(i.e. $\forall o'{\in}S.\;\; o.\mathsf{m}(o')\!\uparrow$*) for showing that* $[\![a.\mathsf{m}()]\!]\!\uparrow$. *Of course, from* $o{\in}S$ *it follows that* $o.\mathsf{m}(o)\!\uparrow$. *Thus, it remains to show that* $[\![a]\!] \in S$ *which, however, is easily seen to be the case as for* $o'{\in}S$ *we have* $[\![a]\!].\mathsf{m}(o') = o'.\mathsf{m}(o')$ *which diverges by the previous consideration.*

3.3 Imperative object specifications

For the imperative setting the corresponding notion of specification is obtained analogously to the functional case yet accounting for the underlying store (the different "implementation" of $\mathcal{O}$). Again, existence of such specifications is discussed in Section 4.

Definition 3.3 *For any predicates or families of predicates, resp.,*

$A \in \mathcal{P}(\mathsf{Loc} \times \mathsf{St}) \to \mathcal{P}(\mathsf{Loc} \times \mathsf{St})$,
$\vec{B} = (B_\mathsf{m} \in \mathcal{P}(\mathsf{Loc} \times \mathsf{St}) \to \mathcal{P}(\mathsf{Val} \times \mathsf{St}))_{\mathsf{m}\in\mathcal{M}}$
$\vec{T} = (T_\mathsf{m} \in \mathcal{P}(\mathsf{Loc} \times \mathsf{St} \times \mathsf{Val} \times \mathsf{St}))_{\mathsf{m}\in\mathcal{M}}$

let $\mathsf{Spec}(A, \vec{B}, \vec{T})$ *be the predicate* $S \subseteq \mathsf{Loc} \times \mathsf{St}$ *with*

$$\begin{array}{l}\langle \ell,\sigma\rangle \in S \ \equiv\ A(S)(\ell,\sigma) \wedge \\ \quad \forall \mathsf{m} \in \mathcal{M}.\, \forall \ell' \in \mathsf{Loc}.\, \forall \sigma' \in \mathsf{St}.\, \langle \ell',\sigma'\rangle \in S \Rightarrow \\ \quad\quad \forall v \in \mathsf{Val}.\, \forall \sigma'' \in \mathsf{St}. \\ \quad\quad \sigma.\ell.\mathsf{m}(\ell',\sigma') = \langle v,\sigma''\rangle \Rightarrow \\ \quad\quad\quad B_\mathsf{m}(S)(v,\sigma'') \ \wedge\ T_\mathsf{m}(\ell',\sigma',v,\sigma'')\end{array}$$

provided S *is* unique *with the above property.*
If $\vec{T} = (T_\mathsf{m} \in \mathcal{P}(\mathsf{Loc} \times \mathsf{St}_\mathsf{Val} \times \mathsf{Val} \times \mathsf{St}_\mathsf{Val}))_{\mathsf{m}\in\mathcal{M}}$ *let* $\mathsf{Spec}_\mathsf{flat}(A, \vec{B}, \vec{T})$ *be the predicate* $S \subseteq \mathsf{Loc} \times \mathsf{St}$ *with*

$$\begin{array}{l}\langle \ell,\sigma\rangle \in S \ \equiv\ A(S)(\ell,\sigma) \wedge \\ \quad \forall \mathsf{m} \in \mathcal{M}.\, \forall \ell' \in \mathsf{Loc}.\, \forall \sigma' \in \mathsf{St}.\, \langle \ell',\sigma'\rangle \in S \Rightarrow \\ \quad\quad \forall v \in \mathsf{Val}.\, \forall \sigma'' \in \mathsf{St}. \\ \quad\quad \sigma.\ell.\mathsf{m}(\ell',\sigma') = \langle v,\sigma''\rangle \Rightarrow \\ \quad\quad\quad B_\mathsf{m}(S)(v,\sigma'') \ \wedge\ T_\mathsf{m}(\ell',\pi_\mathsf{Val}(\sigma'),v,\pi_\mathsf{Val}(\sigma''))\end{array}$$

provided S *is* unique *with this property.*

In Section 4 it will become clear why it is useful to restrict attention to transition specifications that just refer to the "flat" part of the store and not to the "higher-order part" of the store, i.e. the method closures.

Example 3.3 *Assume that object specification* S *is supposed to express the following invariant. Field* f *is always a positive natural number. Define* S *accordingly:*

$$\begin{array}{lcl} A(S)(\ell,\sigma) & \equiv & \sigma.\ell.\mathsf{f} \in \mathbb{Z} \wedge \sigma.\ell.\mathsf{f} \geq 0 \\ T_\mathsf{m}(\ell,\sigma,v,\sigma') & \equiv & \sigma.\ell.\mathsf{f} \in \mathbb{Z} \wedge \sigma.\ell.\mathsf{f} \geq 0 \\ & & \Rightarrow\ \sigma'.\ell.\mathsf{f} \in \mathbb{Z} \wedge \sigma'.\ell.\mathsf{f} \geq 0 \\ B_\mathsf{m}(S) & \equiv & \mathrm{True}\end{array}$$

However, despite their indisputable usefulness the problem with specifications is that there is no obvious reason why they should exist as the right hand side of the equivalence characterising $\mathsf{Spec}(A, \vec{B}, \vec{T})$ contains both positive and negative occurrences of $\mathsf{Spec}(A, \vec{B}, \vec{T})$. Though in [2] specifications are used intrinsically their existence is not verified. Instead the validity of assertions for programs is defined w.r.t. *derivability of correctness assertions* which renders the value of the Soundness Theorem of [2] as somewhat mysterious (if not dubious).

4 Existence of object specifications

In this section we will identify some mild assumptions which guarantee the existence and uniqueness of specifications introduced in the previous section.

4.1 Functional object specifications

In contrast to functional[4] or imperative kernel languages the object calculus implicitly presupposes recursive types like $\mathcal{O}$. Thus, it appears necessary to employ induction principles for the recursive type involved in order to verify programs. After recalling in fairly concrete terms the induction principle for $\mathcal{O}$ we will use it for establishing the existence of specifications under fairly mild conditions.

From well–known work of Freyd and Pitts in the early nineties we know that the *bifree* solutions of the domain equation $A = F(A, A)$ can be characterised by the requirement that id_A is the least fixpoint of $\delta_F = \lambda e.\, F(e, e)$. In case of $F(X, Y) = \mathsf{Rec}_\mathcal{M}(X \rightharpoonup Y)$ we write δ for δ_F which is defined explicitly as the endo–function on $[\mathcal{O} \rightharpoonup \mathcal{O}]$ as given by

$$\delta(e)(\{\!|\mathsf{m}_i{=}f_i|\!\}^{i=1..n}) = \{\!|\mathsf{m}_i{=}e{\circ}f_i{\circ}e|\!\}^{i=1..n}$$

or, equivalently, in a more readable form by

$$\delta(e)(a).\mathsf{m} = e \circ a.\mathsf{m} \circ e$$

for $e{:}\, \mathcal{O} \rightharpoonup \mathcal{O}$, $a{\in}\mathcal{O}$ and $\mathsf{m}{\in}\mathcal{M}$.

From $\mathsf{id} = \mu(\delta) = \bigsqcup_{n\in\mathbf{N}} \delta^n(\bot)$ it follows immediately that $P(\mathsf{id})$ holds for an admissible predicate $P{\subseteq}[\mathcal{O} \rightharpoonup \mathcal{O}]$ if $\forall e{\sqsubseteq}\mathsf{id}.\, P(e){\Rightarrow}P(\delta(e))$. This *Fixpoint Induction* principle can be used directly for verifying properties of objects.

Example 4.1 *Let* $a = [\mathsf{m}{=}\varsigma(x)x.\mathsf{m}()]$*, then using Fixpoint Induction one can prove that* $[\![a.\mathsf{m}()]\!]{\uparrow}$.
Let $o = [\![a]\!]$ *and consider the admissible predicate*

$$P(e) \equiv e(o).\mathsf{m}(e(o)){=}\bot$$

on $[\mathcal{O} \rightharpoonup \mathcal{O}]$. *Obviously,* $[\![a.\mathsf{m}()]\!]{\uparrow}$ *is equivalent to* $P(\mathsf{id})$. *Thus, by Fixpoint Induction it suffices to show that* $\forall e{\sqsubseteq}\mathsf{id}.\, P(e) \ \Rightarrow\ P(\delta(e))$. *Suppose that* $e{\sqsubseteq}\mathsf{id}$ *with* $P(e)$, *i.e.* $e(o).\mathsf{m}(e(o)){=}\bot$. *Then* $P(\delta(e))$ *as*

$$\begin{array}{rcl} \delta(e)(o).\mathsf{m}(\delta(e)(o)) & \sqsubseteq & \delta(e)(o).\mathsf{m}(o) \\ & = & e(o.\mathsf{m}(e(o))) \\ & \sqsubseteq & o.\mathsf{m}(e(o)) \\ & = & e(o).\mathsf{m}(e(o)) = \bot \end{array}$$

where the last equality is the induction hypothesis $P(e)$.

The Fixpoint Induction principle will be employed once more below for proving unique existence of specifications under rather mild assumptions.

[4] For example PCF is based on the finite type hierarchy over the base type $\mathbb{N}_\bot$ and simple imperative languages for which Hoare calculus was first introduced are based on $\mathsf{Rec}_\mathsf{Loc}(\mathsf{Val}) \rightharpoonup \mathsf{Rec}_\mathsf{Loc}(\mathsf{Val})$.

Definition 4.1 *For a flat predomain I let $\mathcal{L}_I(A)$ be the complete lattice of admissible subsets (closed under chains of directed sets) of $I \times A$ ordered by $\subseteq$.*

Note that for partial correctness it would also be sufficient to consider Scott-closed predicates.

Let I be a flat predomain. For any $X, Y \in \mathcal{L}_I$, and $e \in [\mathcal{O} \rightharpoonup \mathcal{O}]$ we define

$$e : X \subseteq Y \;\equiv\; \forall o{:}\mathcal{O}.\; o{\in}X \;\wedge\; e(o){\downarrow} \;\Rightarrow\; e(o){\in}Y$$

as in [10].

For $X, Y \in \mathcal{L}_I$ the set $\{e : \mathcal{O} \rightharpoonup \mathcal{O} \mid e : X \subseteq Y\}$ is obviously a nonempty admissible subset of the domain $[\mathcal{O} \rightharpoonup \mathcal{O}]$.

The following theorem uses the same line of arguments as *loc.cit.*.

Theorem 4.2 *Given a locally continuous bifunctor on predomains F and a predomain A such that $F(A, A) \cong A$, a predomain I, and a monotonic $\Phi : \mathcal{L}_I(A)^{op} \times \mathcal{L}_I(A) \to \mathcal{L}_I(A)$, such that*

$$(\dagger) \quad e : X \subseteq X' \wedge e : Y' \subseteq Y \Rightarrow F(e, e) : \Phi(Y, X) \subseteq \Phi(Y', X')$$

for all $X, Y, X', Y' \in \mathcal{L}_I(A)$ and $e \sqsubseteq \mathsf{id}_{\mathcal{O}}$.

Then $S = \Phi(S, S)$ for a unique *$S \in \mathcal{L}_I(A)$.*

Proof. Let $\Phi : \mathcal{L}_I(A)^{op} \times \mathcal{L}_I(A) \to \mathcal{L}_I(A)$ be monotonic and satisfy the condition $(\dagger)$. Then the mapping

$$\widehat{\Phi} : \mathcal{L}_I(A)^{op} \times \mathcal{L}_I(A) \to \mathcal{L}_I(A)^{op} \times \mathcal{L}_I(A)$$
$$\widehat{\Phi}\,(Y, X) = (\Phi(X, Y), \Phi(Y, X))$$

is a monotonic endomap on the complete lattice $\mathcal{L}_I(A)^{op} \times \mathcal{L}_I(A)$. Thus, by Knaster–Tarski $\widehat{\Phi}$ has a fixpoint $(S^-, S^+) = \widehat{\Phi}(S^-, S^+)$.

For establishing $S^- {=} S^+$ we show by fixpoint induction that for the admissible predicate

$$P(e) \;\equiv e \sqsubseteq \mathsf{id}_{\mathcal{O}} \wedge e : S^- \subseteq S^+ \wedge e : S^+ \subseteq S^-$$

we have $P(\mu e.F(e, e))$ and, therefore, $P(\mathsf{id}_{\mathcal{O}})$ as $\mathsf{id}_{\mathcal{O}} = \mu e.F(e, e)$ from which it follows that $S^- = S^+$. Obviously, we have $P(\perp)$. For the induction step assume that $P(e)$. Then $F(e, e) \sqsubseteq F(\mathsf{id}, \mathsf{id}) = \mathsf{id}$. Moreover, from $e : S^- \subseteq S^+$ it follows by $(\dagger)$ that $F(e, e) : S^- = \Phi(S^+, S^-) \subseteq \Phi(S^-, S^+) = S^+$ and, analogously, it follows from $e : S^+ \subseteq S^-$ by $(\dagger)$ that $F(e, e) : S^+ = \Phi(S^-, S^+) \subseteq \Phi(S^+, S^-) = S^-$. Thus, we have $P(F(e, e))$.

Thus, we conclude that there exists at least one $S \in \mathcal{L}_I$ with $S = \Phi(S, S)$. For showing uniqueness suppose $S' = \Phi(S', S')$ for some $S' \in \mathcal{L}_I$. For the admissible predicate

$$P(e) \;\equiv e \sqsubseteq \mathsf{id}_{\mathcal{O}} \wedge e : S \subseteq S' \wedge e : S' \subseteq S$$

it follows $P(\mu e.F(e, e))$ again by fixpoint induction. Obviously, we have $P(\perp)$. Assume that $P(e)$. Then $F(e, e) \sqsubseteq F(\mathsf{id}, \mathsf{id}) = \mathsf{id}$. Moreover, it follows by $(\dagger)$ that

$$\begin{array}{lclcl} F(e, e) & : & S = \Phi(S, S) & \subseteq & \Phi(S', S') = S' \\ F(e, e) & : & S' = \Phi(S', S') & \subseteq & \Phi(S, S) = S \end{array}$$

as the induction hypothesis $P(e)$ ensures $e : S \subseteq S'$ and $e : S' \subseteq S$. But as $\mathsf{id}_{\mathcal{O}} = \mu e.F(e, e)$ we have $P(\mathsf{id})$ from which it follows immediately that $\mathsf{id} : S \subseteq S'$ and $\mathsf{id} : S' \subseteq S$, i.e. $S \subseteq S'$ and $S' \subseteq S$, and, therefore $S = S'$ as desired. □

Theorem 4.3 *(Existence Theorem)*
Let $\mathcal{L}$ denote $\mathcal{L}_1(\mathcal{O})$ and $F(Y, X) = \mathsf{Rec}_{\mathcal{M}}(Y \rightharpoonup X)$. Moreover, let $A \in \mathcal{L} \to \mathcal{L}$, $\vec{B} = (B_{\mathsf{m}} \in \mathcal{L} \to \mathcal{L})_{\mathsf{m} \in \mathcal{M}}$ and $\vec{T} = (T_{\mathsf{m}} \in \mathcal{P}(\mathcal{O} \times \mathcal{O}))_{\mathsf{m} \in \mathcal{M}}$ be families such that for all $\mathsf{m} \in \mathcal{M}$

1. *$e : X \subseteq Y$ implies $F(e, e) : A(X) \subseteq A(Y)$ for $e \sqsubseteq \mathsf{id}_{\mathcal{O}}$ and $X, Y \in \mathcal{L}$*
2. *$e : X \subseteq Y$ implies $e : B_{\mathsf{m}}(X) \subseteq B_{\mathsf{m}}(Y)$ for $e \sqsubseteq \mathsf{id}_{\mathcal{O}}$ and $X, Y \in \mathcal{L}$*
3. *$T_{\mathsf{m}}(o, -) := \{o' \in \mathcal{O} \mid T_{\mathsf{m}}(o, o')\}$ is admissible for all $o \in \mathcal{O}$ and $T_{\mathsf{m}}(o, -) \subseteq T_{\mathsf{m}}(o', -)$ whenever $o \sqsubseteq o'$.*

Then there exists a unique $S \in \mathcal{L}$ satisfying for all $o \in \mathcal{O}$

$$(*) \;\; o \in S \;\equiv\; o \in A(S) \wedge \forall \mathsf{m}{\in}\mathcal{M}.\, \forall o'{\in}S.\; o.\mathsf{m}(o'){\downarrow} \Rightarrow o.\mathsf{m}(o'){\in}B_{\mathsf{m}}(S) \;\wedge\; T_{\mathsf{m}}(o', o.\mathsf{m}(o'))\,.$$

Proof. For $Y, X \in \mathcal{L}$ consider the predicate

$$o \in \Phi(Y, X) \equiv\; o \in A(X) \wedge \forall \mathsf{m}{\in}\mathcal{M}.\, \forall o'{\in}Y.\; o.\mathsf{m}(o'){\downarrow} \Rightarrow o.\mathsf{m}(o'){\in}B_{\mathsf{m}}(X) \;\wedge\; T_{\mathsf{m}}(o', o.\mathsf{m}(o'))$$

which is admissible due to assumption (3) and since the precondition is Scott-open. Clearly, the operator $\Phi : \mathcal{L}^{op} \times \mathcal{L} \to \mathcal{L}$ is monotonic.

Obviously, the requirement $S = \Phi(S, S)$ is equivalent to $(*)$ for all $o \in \mathcal{O}$. Thus, we have to show that there exists a unique $S \in \mathcal{L}$ with $S = \Phi(S, S)$ which is guaranteed by Theorem 4.2 provided we can show that our Φ satisfies the condition $(\dagger)$ of Theorem 4.2 which we verify next.

Suppose $e \sqsubseteq \mathsf{id}_{\mathcal{O}}$ with $e : X \subseteq X'$ and $e : Y' \subseteq Y$.

For showing $F(e, e) : \Phi(Y, X) \subseteq \Phi(Y', X')$ suppose $o \in \Phi(Y, X)$ and show that $F(e, e)(o) \in \Phi(Y', X')$.

First we show that $F(e, e)(o){\in}A(X')$. But $F(e, e)(o) \sqsubseteq o{\in}A(X)$ and, therefore, also $F(e, e)(o){\in}A(X')$ due to assumption (1).

Next, let $\mathsf{m}\in\mathcal{M}$ and $o'\in Y'$ with $F(e,e)(o).\mathsf{m}(o')\downarrow$. Then $e(o')\in Y$ as $e : Y' \subseteq Y$ and, therefore, as by induction hypothesis $o \in \Phi(X,Y)$ it follows that

$$o.\mathsf{m}(e(o')) \in B_{\mathsf{m}}(X) \wedge T_{\mathsf{m}}(e(o'), o.\mathsf{m}(e(o'))) \quad .$$

But then we have

$$e(o.\mathsf{m}(e(o')) \in B_{\mathsf{m}}(X')$$

by (2) and the assumption $e : X \subseteq X'$. Moreover, we obtain

$$T_{\mathsf{m}}(o', e(o.\mathsf{m}(e(o'))))$$

by assumption (3) as $e(o.\mathsf{m}(e(o'))) \sqsubseteq o.\mathsf{m}(e(o'))$ and $e(o') \sqsubseteq o'$ since (3) implies that $x \sqsubseteq x' \wedge y' \sqsubseteq y$ implies $T(y,x)\Rightarrow T(y',x')$. Thus, it follows that

$$F(e,e)(o).\mathsf{m}(o') \in B_{\mathsf{m}}(X') \wedge T_{\mathsf{m}}(o', F(e,e)(o).\mathsf{m}(o'))$$

as $F(e,e)(o).\mathsf{m}(o') = e(o.\mathsf{m}(e(o')))$. □

4.2 Imperative object specifications

Recall from Section 2 that the imperative object calculus of [2] finds its denotational interpretation within the recursively defined predomain $\mathsf{St} = F_{\mathsf{St}}(\mathsf{St},\mathsf{St})$ if the latter is defined to be

$$\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathcal{F}}(\mathsf{Val})\times\mathsf{Rec}_{\mathcal{M}}(\mathsf{Loc}\times\mathsf{St}\rightharpoonup\mathsf{Val}\times\mathsf{St}))$$

where $\mathsf{Val} = \mathsf{BVal}+\mathsf{Loc}$.

Next we prove a variant of Theorem 4.3 for the imperative object calculus.

Theorem 4.4 *For any predicates and families of predicates, resp.,*

$A \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})\to\mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$,
$\vec{B} = (B_{\mathsf{m}} \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})\to\mathcal{L}_{\mathsf{Val}}(\mathsf{St}))_{\mathsf{m}\in\mathcal{M}}$,
$\vec{T} = (T_{\mathsf{m}} \in \mathcal{L}(\mathsf{Loc}\times\mathsf{St}\times\mathsf{Val}\times\mathsf{St}))_{\mathsf{m}\in\mathcal{M}}$

such that

(i) $e : X \subseteq X'$ *implies* $F_{\mathsf{St}}(e,e) : A(X) \subseteq A(X')$ *for all* $e \sqsubseteq \mathsf{id}_{\mathsf{St}}$.

(ii) *for all* $\mathsf{m} \in \mathcal{M}$, $e : X \subseteq X'$ *implies* $e : B_{\mathsf{m}}(X) \subseteq B_{\mathsf{m}}(X')$ *for all* $e \sqsubseteq \mathsf{id}_{\mathsf{St}}$.

(iii) *for all* $\ell \in \mathsf{Loc}$, $\mathsf{m} \in \mathcal{M}$, $\sigma,\sigma' \in \mathsf{St}$, $v \in \mathsf{Val}$ *if* $e : X \subseteq X'$ *and* $e(\sigma')\downarrow \wedge e(\sigma'')\downarrow$ *then* $T_{\mathsf{m}}(\ell, e(\sigma), v, \sigma')$ *implies* $T_{\mathsf{m}}(\ell,\sigma,v,e(\sigma'))$.

Then for $\Phi : \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})^{op}\times\mathcal{L}_{\mathsf{Loc}}(\mathsf{St})\to\mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$ *with*

$$\begin{array}{l}\Phi(Y,X)(\ell,\sigma) \equiv\ A(X)(\ell,\sigma) \wedge \\ \quad \forall \mathsf{m} \in \mathcal{M}.\,\forall \ell' \in \mathsf{Loc}.\,\forall \sigma' \in \mathsf{St}.\,\langle \ell',\sigma'\rangle \in Y \Rightarrow \\ \quad\quad \forall v \in \mathsf{Val}.\,\forall \sigma'' \in \mathsf{St}. \\ \quad\quad \sigma.\ell.\mathsf{m}(\ell',\sigma') = \langle v,\sigma''\rangle \Rightarrow \\ \quad\quad\quad B_{\mathsf{m}}(X)(v,\sigma'') \ \wedge\ T_{\mathsf{m}}(\ell',\sigma',v,\sigma'')\end{array}$$

there exists a unique $S \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$ *with* $S = \Phi(S,S)$.

Proof. Instantiating Theorem 4.2 by F_{St} and Loc guarantees the existence of a unique fixpoint for Φ provided we can verify that Φ satisfies the condition (†) of Theorem 4.2.

For subsequent use it is helpful to recall that $F_{\mathsf{St}}(e,e)(\sigma)\downarrow$ for all $\sigma \in \mathsf{St}$, and that

(a) $F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{f} = \sigma.\ell.\mathsf{f}$ for all $\mathsf{f} \in \mathcal{F}$ and

(b) $F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{m} = (\mathsf{id}_{\mathsf{Val}}\times e) \circ (\sigma.\ell.\mathsf{m}) \circ (\mathsf{id}_{\mathsf{Loc}}\times e)$ for all $\mathsf{m} \in \mathcal{M}$.

Now we show that Φ satisfies condition (†). Suppose $e \sqsubseteq \mathsf{id}_{\mathsf{St}}$ with

(1) $e : X \subseteq X'$
(2) $e : Y' \subseteq Y$

for some $X, X', Y, Y' \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$.

We have to show that $F_{\mathsf{St}}(e,e) : \Phi(Y,X) \subseteq \Phi(Y',X')$. For that purpose we suppose that

(3) $\langle \ell,\sigma\rangle \in \Phi(Y,X)$ and $F_{\mathsf{St}}(e,e)(\sigma)\downarrow$

and show that $\langle \ell, F_{\mathsf{St}}(e,e)(\sigma)\rangle \in \Phi(Y',X')$.

From (3) we get $A(X)(\ell,\sigma)$. Thus by (i) we get that $A(X')(\ell, F_{\mathsf{St}}(e,e)(\sigma))$, i.e the first part of the conjunction $\langle \ell, F_{\mathsf{St}}(e,e)(\sigma)\rangle \in \Phi(Y',X')$.

For the second part suppose that

(4) $\langle \ell',\sigma'\rangle \in Y'$ with $F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{m}(\ell',\sigma')\downarrow$.

From (b) and $F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{m}(\ell',\sigma')\downarrow$ we know that

(5) $F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{m}(\ell',\sigma') = \langle v, e(\sigma'')\rangle$ with
(6) $\langle v,\sigma''\rangle = \sigma.\ell.\mathsf{m}(\ell', e(\sigma'))$

for some value $v\in\mathsf{Val}$ and some store $\sigma''\in\mathsf{St}$. We have to show that $F_{\mathsf{St}}(e,e)(o).\ell.\mathsf{m}(\ell',\sigma') = (v, e(\sigma''))$ implies $B_{\mathsf{m}}(X')(v,e(\sigma'')) \wedge T_{\mathsf{m}}(\ell',\sigma',v,e(\sigma''))$. From (6) it follows that $e(\sigma')\downarrow$. Thus, from (4) we get by (2) that

(7) $\langle \ell', e(\sigma')\rangle \in Y$.

Thus, by (3) it follows that $B_{\mathsf{m}}(X)(\sigma.\ell.\mathsf{m}(\ell',e(\sigma')))$, i.e. $B_{\mathsf{m}}(X)\langle v,\sigma''\rangle$ by (6). By (ii) it follows now that $B_{\mathsf{m}}(X')(v,e(\sigma''))$ and, therefore, by (5) that

(8) $B_{\mathsf{m}}(X')(F_{\mathsf{St}}(e,e)(\sigma).\ell.\mathsf{m}(\ell',\sigma'))$.

It follows by the second part of the conjunction $\langle \ell,\sigma\rangle \in \Phi(Y,X)$ as ensured by (3) that

(9) $T_{\mathsf{m}}(\ell', e(\sigma'), v, \sigma'')$

as $\langle \ell', e(\sigma')\rangle \in Y$ by (7) and $\sigma.\ell.\mathsf{m}(\ell', e(\sigma'))\downarrow$ by (6). From (5) it follows that $e(\sigma'')\downarrow$ and from (6) that $e(\sigma')\downarrow$. Therefore, by assumption (iii) it follows that

(10) $T_{\mathsf{m}}(\ell',\sigma',v,e(\sigma''))$

i.e. $\langle \ell', F(e,e)(\sigma') \rangle \in \Phi(Y', X')$. $\square$

This proves that under certain conditions the specification $\mathsf{Spec}(A, \vec{B}, \vec{T})$ does really exist. But condition (iii) of Theorem 4.4 is awkward to prove and can be simplified if the method specifications only refer to the flat part of the store.

Corollary 4.5 *Given any predicates and families of predicates, resp.,*

$$A \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St}) \rightarrow \mathcal{L}_{\mathsf{Loc}}(\mathsf{St}),$$
$$\vec{B} = (B_{\mathsf{m}} \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St}) \rightarrow \mathcal{L}_{\mathsf{Val}}(\mathsf{St}))_{\mathsf{m} \in \mathcal{M}},$$
$$\vec{T} = (T_{\mathsf{m}} \in \mathcal{P}(\mathsf{Loc} \times \mathsf{St}_{\mathsf{Val}} \times \mathsf{Val} \times \mathsf{St}_{\mathsf{Val}}))_{\mathsf{m} \in \mathcal{M}}$$

such that

(i) $e : X \subseteq X'$ *implies* $F_{\mathsf{St}}(e,e) : A(X) \subseteq A(X')$ *for all* $e \sqsubseteq \mathsf{id}_{\mathsf{St}}$.

(ii) *for all* $\mathsf{m} \in \mathcal{M}$, $e : X \subseteq X'$ *implies* $F_{\mathsf{St}}(e,e) : B_{\mathsf{m}}(X) \subseteq B_{\mathsf{m}}(X')$ *for all* $e \sqsubseteq \mathsf{id}_{\mathsf{St}}$.

Then for $\Phi : \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})^{op} \times \mathcal{L}_{\mathsf{Loc}}(\mathsf{St}) \rightarrow \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$ *with*

$$\begin{array}{l} \Phi(Y,X)(\ell,\sigma) \;\equiv\; A(X)(\ell,\sigma) \wedge \\ \quad \forall \mathsf{m} \in \mathcal{M}.\, \forall \ell' \in \mathsf{Loc}.\, \forall \sigma' \in \mathsf{St}.\, \langle \ell', \sigma' \rangle \in Y \Rightarrow \\ \qquad \forall v \in \mathsf{Val}.\, \forall \sigma'' \in \mathsf{St}. \\ \qquad \sigma.\ell.\mathsf{m}(\ell', \sigma') = \langle v, \sigma'' \rangle \Rightarrow \\ \qquad\quad B_{\mathsf{m}}(X)(v, \sigma'') \;\wedge\; T_{\mathsf{m}}(\ell', \pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \end{array}$$

there exists a unique $S \in \mathcal{L}_{\mathsf{Loc}}(\mathsf{St})$ *with* $S = \Phi(S,S)$.

Proof. Applying Corollary 4.4 it only remains to show that T_{m} is admissible (which is trivial as it only ranges over flat predomains) and that condition (iii) of *loc.cit.* holds. Therefore, let a location $\ell \in \mathsf{Loc}$, a method $\mathsf{m} \in \mathcal{M}$, a map $e : X \subseteq X'$, stores $\sigma, \sigma' \in \mathsf{St}$, and a value $v \in \mathsf{Val}$ be given such that $e(\sigma)\!\downarrow$ and $e(\sigma')\!\downarrow$. We have to prove that $T_{\mathsf{m}}(\ell, \pi_{\mathsf{Val}}(e(\sigma)), v, \pi_{\mathsf{Val}}(\sigma'))$ implies $T_{\mathsf{m}}(\ell, \pi_{\mathsf{Val}}(\sigma), v, \pi_{\mathsf{Val}}(e(\sigma')))$ which simply follows from the fact that $\pi_{\mathsf{Val}}(e(\sigma)) = \pi_{\mathsf{Val}}(\sigma)$ if $e \sqsubseteq \mathsf{id}$. $\square$

5 Soundness of the Abadi & Leino logic denotationally

Theorem 5.1 *The object creation rule of the Abadi & Leino logic is correct w.r.t. our semantics.*

Proof. An object specification S in [2] reads as follows

$$S = [\mathsf{f}_i : A_i^{i \in 1,\ldots,n}, \mathsf{m}_j : \varsigma(y)\, B_j :: T_j^{j \in 1,\ldots,m}]$$

where, again, the A_i are field specifications, B_j are the result specifications for method m_j and T_j is the transitions specification for m_j. If we interpret all these predicates denotationally it follows from Theorem 4.4 that they give rise to a unique predicate that is as in Definition 3.3 and serves as the denotational interpretation of S.

It is important that in the Abadi & Leino logic the transition specifications T_j can only refer to the *flat part of the store* otherwise existence of S could not be guaranteed.

The object formation rule of *loc.cit.* says about the newly created object

$$\frac{E \vdash x_i : A_i :: Res(x_i)^{\,i \in 1..n} \quad E, y : S \vdash b_j : B_j :: T_j^{\,j \in 1..m}}{E \vdash [f_i = x_i^{\,i \in 1..n}, m_j = \varsigma(y) b_j^{\,j \in 1..m}] : S :: T_\mu}$$

where T_μ specifies the actual memory allocation and $Res(x_i)$ the absence of side effects. Ignoring the side effect issue T_μ of the creation the correctness of the rule above in our semantics follows simply from the Definition 3.3 of specification (Spec). $\square$

It should be relatively straightforward how to devise a denotational semantics for the whole logic of Abadi & Leino and to prove correctness of their calculus w.r.t. such a denotational semantics. Such a proof should be easier than the one presented in [2].

6 Possible extensions of the Abadi & Leino logic

Dealing with object specifications denotationally does not only yield a concise explanation of the Abadi & Leino logic but also suggests extensions.

6.1 Invariants

By contrast to Abadi & Leino the predicates A and B_{m} may contain recursive occurrences of the specification itself. If a field is required to fulfill the same specification as the ambient object then one needs recursion in A. If a result of method m is required to contain (or be itself) an object fulfilling the same specification as the ambient object, one needs recursion in B_{m}. In [8] Leino gives a variant of the logic [2] that allows for recursive object descriptions but requires that methods are declared in advance.

The presented approach even allows for more. In a definition of a specification $S(\ell, \sigma) = \ldots$, we may allow the B_{m} to refer to the self reference ℓ to express that the whole specification S is an invariant. By setting

$$B_{\mathsf{m}}(S)(v, \sigma) = S(\ell, \sigma)$$

it is required that a method invocation of m does preserve the specification S of the self object. This means that the specifications Spec can not only describe denotationally properties of objects at a certain point in a program (like in Abadi& Leino) but a kind of invariant that can be seen as a definition of a "class specification" in the object calculus.

6.2 Method update

In its generality the presented approach allows one to deal with method update in a limited way. Consider the following example:

Example 6.1 *A sensible requirement in presence of method update is that a method is not altered by another method, i.e. a method* n *can at most become less or equally defined by execution of* m *(less since we are dealing with partial correctness). More formally, this translates to the following transition predicate:*

$$T_{\mathsf{m}}(\ell, \sigma, v, \sigma') \equiv \sigma.\ell.\mathsf{n} \sqsupseteq \sigma'.\ell.\mathsf{n}$$

This definition satisfies condition (iii) of Theorem 4.4 since if $e \sqsubseteq \mathsf{id}$ *and* $e(\sigma).\ell.\mathsf{n} \sqsupseteq \sigma'.\ell.\mathsf{n}$ *then by transitivity from* $\sigma.\ell.\mathsf{n} \sqsupseteq e(\sigma).\ell.\mathsf{n} \sqsupseteq \sigma'.\ell.\mathsf{n} \sqsupseteq e(\sigma').\ell.\mathsf{n}$ *it follows that* $\sigma.\ell.\mathsf{n} \sqsupseteq e(\sigma').\ell.\mathsf{n}$.

To which forms of transition specifications our existence theorem applies remains to be clarified. That specifications of general form need not exist is demonstrated by a counterexample in the next section.

7 Non existing specifications

Before showing that particular object specifications do not exist we prove the following auxiliary lemma that deals with transition specifications that may also refer to the non flat part of the store, i.e. to some method closures.

Lemma 7.1 *Let* $A \subseteq \mathsf{Loc}\times\mathsf{St}\times[\mathsf{Loc}\times\mathsf{St}\rightharpoonup\mathsf{Val}\times\mathsf{St}]$ *and* $S \subseteq \mathsf{Loc}\times\mathsf{St}$ *with*

(0) $\forall \ell, \sigma.\ \langle \ell, \sigma\rangle \in S \Leftrightarrow [\forall(\ell', \sigma')\in S.\ A(\ell', \sigma', \sigma.\ell.\mathsf{m})]$.

If ℓ *is a location and* σ *a state satisfying*

(1) $\forall \ell', \sigma'.\ A(\ell', \sigma', \sigma'.\ell'.\mathsf{m}) \Rightarrow A(\ell', \sigma', \sigma.\ell.\mathsf{m})$

then

(2) $A(\ell, \sigma, \sigma.\ell.\mathsf{m})$.

Proof. From (0) it follows that

$$(\dagger) \qquad \forall \ell', \sigma'.\ \langle \ell', \sigma'\rangle \in S \Rightarrow A(\ell', \sigma', \sigma'.\ell'.\mathsf{m})$$

and, therefore, by (1) that

$$\forall \ell', \sigma'.\ \langle \ell', \sigma'\rangle\in S \Rightarrow A(\ell', \sigma', \sigma.\ell.\mathsf{m})$$

i.e. that $(\ell, \sigma) \in S$. Thus, by (†) we have $A(\ell, \sigma, \sigma.\ell.\mathsf{m})$. □

To give an example of a non–existing object specification, we will exhibit a transition specification T_{m}, a location ℓ, and a store σ such that there does not exist $S \subseteq \mathsf{Loc}\times\mathsf{St}$ satisfying

$$\langle \ell, \sigma\rangle\in S \Leftrightarrow \forall\langle \ell', \sigma'\rangle\in S.\ \sigma.\ell.\mathsf{m}(\ell', \sigma')\downarrow \Rightarrow T_{\mathsf{m}}(\ell', \sigma', \sigma.\ell.\mathsf{m}(\ell', \sigma'))$$

For such a specification the restrictive assumption of Theorem 4.5 – that T_{m} must only refer to the flat part of the store, i.e. no specifications of past or future behaviour of methods can be made – must necessarily be violated, but also condition (iii) from Theorem 4.4 cannot hold.

Example 7.2 *Consider the following object specification*

$$\langle \ell, \sigma\rangle \in S \equiv \forall\langle \ell', \sigma'\rangle\in S.\ \sigma.\ell.\mathsf{m}(\ell', \sigma')\downarrow \Rightarrow \sigma'.\ell'.\mathsf{f}\downarrow \Rightarrow T(\ell', \sigma', \sigma.\ell.\mathsf{m}(\ell', \sigma'))$$

where

$$T(\ell', \sigma', \ell'', \sigma'') \equiv \exists n\in\mathbb{N}.\ \sigma'.\ell'.\mathsf{m}(\ell', \sigma'[\ell'.\mathsf{f}{:=}n])\uparrow \quad .$$

Note that T *is not admissible. Let* $A(\ell, \sigma, h)$ *denote the property*

$$h(\ell, \sigma)\downarrow \Rightarrow \sigma.\ell.\mathsf{f}\downarrow \Rightarrow \exists n\in\mathbb{N}.\ h(\ell, \sigma[\ell.\mathsf{f}{:=}n])\uparrow$$

then the specification S *above can be reformulated as*

$$\langle \ell, \sigma\rangle \in S \equiv \forall\langle \ell', \sigma'\rangle\in S.\ A(\ell', \sigma', \sigma.\ell.\mathsf{m})$$

i.e. condition (0) of Lemma 7.1 holds for A*. Now for* (ℓ, σ) *with* $\sigma.\ell.\mathsf{f} = 0$ *and*

$$\sigma.\ell.\mathsf{m} = [\![\varsigma(x)\ \mathbf{if}\ x.\mathsf{f}{=}0\ \mathbf{then}\ x\ \mathbf{else}\ x.\mathsf{f}{:=}x.\mathsf{f}{-}1; x.\mathsf{m}()]\!]$$

one easily verifies that (1) holds but (2) is false, contradicting Lemma 7.1(2).

More natural counterexamples are expected by semantic modelling of logics for object calculi using Hoare triples like the tentative one suggested in [4].

8 Conclusions

We have shown that a denotational approach to programming logics for object calculi leads to a better understanding of the implicit recursion of object specifications and their reasoning principles. Since the notion of specification encodes the object introduction rule of the logic, the soundness of this rule is equivalent to the existence of the specification. To guarantee existence one has to be careful with reasoning on the non-flat method part of the store.

It should be possible to deal with other, similar, object calculi and logics in the same *denotational* way. The analysis of further languages should be fruitful in the quest for more (natural) counterexamples. A comparison with class-based languages has been attempted in [15].

Recursive methods can be programmed in the object calculus without explicit recursion due to the recursive higher-order definition of the underlying store. This is a particular instance of *"recursion through the store"* a more general variant of which allows unrestricted execution of code stored in memory. Such a rather liberal usage of higher-order store needs to be modelled by different domain equations and thus needs different treatment altogether. Results analogous to those in this paper are expected to be hard to establish.

In this paper we used classical logic as a meta logic but did not commit ourselves to any particular object logic. Spatial or separation logic [9, 5] is a prospective candidate for such an object logic as it simplifies handling of aliases. It remains to be seen whether predicates in such a logic pose any problems to the presented approach.

Other issues to be tackled include a soundness proof for the Abadi & Leino logic possibly extended by invariants and reasoning principles on higher-order store and the development of a logic over a typed semantics of the object calculus (with subtyping including method parameters).

Acknowledgment Thanks to Cristiano Calcagno and Peter O'Hearn for discussions on "recursion through the store" and other related matters.

References

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

[2] M. Abadi and K. Leino. A logic of object-oriented programs. In M. Bidoit and M. Dauchet, editors, *Theory and Practice of Software Development: Proceedings / TAPSOFT '97, 7th International Joint Conference CAAP/FASE*, volume 1214 of *Lecture Notes in Computer Science*, pages 682–696. Springer-Verlag, 1997.

[3] P. America and F. de Boer. A proof theory for a sequential version of POOL. Technical Report available also from URL http://www.cs.uu.nl/people/frankb/Available-papers/spool.dvi, University of Utrecht, 1999.

[4] C. Calcagno and P. W. O'Hearn. A logic for objects. Talk given in March 2001, 2001.

[5] C. Calcagno and P. W. O'Hearn. On garbage and program logic. In *FoSSaCS*, volume 2030 of *LNCS*, pages 137–151, Berlin, 2001. Springer.

[6] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about Java classes. *ACM SIGPLAN Notices*, 33(10):329–340, October 1998.

[7] G. Leavens. Modular specification and verification of object-oriented programs. *IEEE Software*, 8(4):72–80, 1991.

[8] K. R. M. Leino. Recursive object types in a logic of object-oriented programs. *Nordic Journal of Computing*, 5(4):330–360, 1998.

[9] P. W. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, volume 2142 of *LNCS*, pages 1–19, Berlin, 2001. Springer.

[10] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996. (A preliminary version of this work appeared as Cambridge Univ. Computer Laboratory Tech. Rept. No. 321, December 1993.).

[11] A. Poetzsch-Heffter and P. Müller. Logical foundations for typed object-oriented languages. In D. Gries and W. De Roever, editors, *Programming Concepts and Methods*, 1998.

[12] U. Reddy. Objects and classes in Algol-like languages. In *FOOL 5*, 1998. This paper is also available at URL http://pauillac.inria.fr/∼remy/fool/proceedings.html.

[13] B. Reus. A logic of recursive objects. In *Formal Techniques for Java Programs*, volume 251 - 5/1999 of *Informatik Berichte*, pages 58–64. FernUniversität Hagen, 1999.

[14] B. Reus. A logic of recursive objects (abstract). In S. D. A. Moreira, editor, *Object-oriented Technology, ECOOP '99 Workshop Reader*, volume 1743 of *Lecture Notes in Computer Science*, page 107, Berlin, 1999. Springer.

[15] B. Reus. Class based vs. object based: A denotational comparison. 2002. To appear in Algebraic Methodology And Software Technology.

[16] B. Reus, M. Wirsing, and R. Hennicker. A Hoare-Calculus for Verifying Java Realizations of OCL-Constrained Design Models. In *FASE 2001*, volume 2029 of *Lecture Notes in Computer Science*, pages 300–317, Berlin, 2001. Springer.