Math. Struct. in Comp. Science (2006), *vol.* 16, *pp.* 313–358. © 2006 Cambridge University Press
doi:10.1017/S0960129506005214 Printed in the United Kingdom

Denotational semantics for a program logic of objects[†]

BERNHARD REUS[‡] and JAN SCHWINGHAMMER[§]

[‡]*Department of Informatics, University of Sussex, Brighton BN1 9QH, U.K.*
[§]*Programming Systems Lab, Saarland University, 66041 Saarbrücken, Germany*

Received 17 January 2005; revised 28 November 2005

Dedicated to Klaus Keimel on the occasion of his 65th birthday

The object-calculus is an imperative and object-based programming language in which every object comes equipped with its own method suite. Consequently, methods need to reside in the store ('higher-order store'), which complicates the semantics. Abadi and Leino defined a program logic for this language enriching object types by method specifications. We present a new soundness proof for their logic using denotational semantics. It turns out that denotations of store specifications are predicates defined by mixed-variant recursion. A benefit of our approach is that derivability and validity can be kept distinct. Moreover, it reveals which of the limitations of Abadi and Leino's logic are incidental design decisions and which follow inherently from the use of a higher-order store. We discuss the implications for the development of other, more expressive, program logics.

1. Introduction and motivation

When Hoare presented his seminal work on an *axiomatic basis of computer programming* (Hoare 1969), high-level languages had just begun to gain broader acceptance. Since then programming languages have been evolving ever more rapidly, whereas verification techniques seem to be struggling to keep up. For object-oriented languages several formal systems have been proposed: see, for example, Abadi and Leino (2004), Hensel *et al.* (1998), Jacobs and Poll (2001), Reddy (2002), Poetzsch-Heffter and Müller (1999), de Boer (1999), von Oheimb (2001) and Reus *et al.* (2001). A 'standard' approach comparable to the Hoare-calculus for imperative WHILE-languages (Apt 1981) has not yet emerged. Nearly all the approaches listed above are designed for class-based languages (usually a sub-language of sequential Java), where method code is known statically.

One notable exception is the work of Abadi and Leino (Abadi and Leino 1997; 2004) where a logic for an object-based language is introduced that is derived from the imperative object calculus with first-order types, **imp**ς, of Abadi and Cardelli (1996). In object-based languages, every object contains its own suite of methods. Operationally speaking, the store for such a language contains code and is thus often called a *higher-order store*.

[†] This work was supported by the EPSRC under grant GR/R65190/01, 'Programming Logics for Denotations of Recursive Objects'.

The fact that methods are stored like any other data inside objects means that new code can be added at any time, yielding compositionality of components (objects in this case) for free. By contrast, classical fixpoint-based semantics for classes (or modules) is 'closed' in the sense that it cannot model the addition of previously unknown classes in a compositional way. As classes can be compiled into objects (Abadi and Cardelli, 1996), and object-based languages provide this kind of compositionality, higher-order store semantics can deal naturally with classes defined on-the-fly, like inner classes and classes loaded at run-time (*cf.* Reus (2002; 2003)).

Abadi and Leino's logic is a Hoare-style system, dealing with partial correctness of object expressions. Their idea was to enrich object types by method specifications, also called *transition relations*, relating pre- and post-execution states of program statements, and *result specifications* describing the result if the program terminates. Informally, an object satisfies a specification

$$A \;\equiv\; [\mathsf{f}_i : A_i{}^{i=1\ldots n},\, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}]$$

if it has fields f_i satisfying A_i and methods m_j that satisfy the transition relation T_j and, if the method invocation terminates, their result satisfies B_j. However, just as a method can use the *self*-parameter, we can assume that an object a itself satisfies A in both B_j and T_j when establishing that A holds for a. This yields a powerful and convenient proof principle for objects[†].

We are going to present a new soundness proof for this logic using an untyped denotational semantics of the language, and the logic to define validity. Every program and every specification has a meaning, a *denotation*. The denotations of specifications are simply predicates on (the domain of) objects. The properties of these predicates provide a description of inherent limitations of the logic. Such an approach is not new, for instance, it has been used in LCF, a logic for functional programs (Paulson 1987).

The difficulty in this case is to establish predicates that provide a powerful reasoning principle for objects. Reus and Streicher (2004) outlined how to use some classic domain theory (Pitts 1996) to guarantee existence and uniqueness of appropriate predicates on isolated objects. In an object-calculus program, however, an object may depend on other objects and their respective methods in the store. So object specifications must depend on specifications of other objects in the store, which gives rise to 'store specifications'. Indeed store specifications were already present in the operationally-based work of Abadi and Leino.

This paper is, therefore, not merely an application of the ideas in Reus and Streicher (2004). Much care is needed to establish the important invariance property of Abadi–Leino logic, namely that proved programs preserve store specifications. Our main achievement, in a nutshell, is that we have successfully applied the ideas of Reus and Streicher (2004) to the logic of Abadi and Leino (2004). We have obtained a more useful and more

[†] In class-based languages one can provide an analogous proof principle when using a higher-order store (see Kamin and Reddy (1994)). In a closed-world scenario, however, where (mutually recursive) classes are all known at verification time, fixpoint induction suffices as a proof principle.

instructive soundness proof, which is complementary to that of Abadi and Leino (2004) for the following reasons:

1 Abadi and Leino employ an operational semantics in which stores contain method *syntax* that can be reused in derivations. This allows them to get away with having no semantics for store specifications at all. Validity for judgements is with respect to initial stores whose methods are assumed to have been *derived correct* with respect to some store specification. Consequently, the validity (of judgements) depends on derivability (for store specifications). This approach can be justified under the assumption that only verified methods reside in initial stores. Alas, it prevents them from using induction on derivations of judgements, so they are forced to use induction on derivations of the (small-step) semantics of the judgement's program instead. As the subsumption rule is not triggered by program syntax, it has to be considered in all cases and clutters the soundness proof.
   On the other hand, by using a denotational semantics, we *can* provide a (necessarily recursive) semantics for store specifications, and keep validity and derivability distinct. This has the additional advantage of a canonical treatment of the subsumption rule. We can also define a semantics of object specifications and show how it relates to store specifications. Note that our approach would also work for stores containing method syntax as it can be interpreted in a big-step operational semantics.

2 We can easily extend the logic, for instance, by recursive specifications. In our approach, fold and unfold can be handled by subsumption rules, which are problematic in the original proof (see 1). A similar extension has been done for the Abadi–Leino logic in Leino (1998), but for a slightly different language, which has *nominal* rather than structural subtyping.

3 The essential restrictions of object logics in general are revealed and distinguished from the idiosyncratic shortcomings of the Abadi–Leino logic. For example, in Abadi and Leino (2004), transition specifications cannot talk about methods at all (see also Section 7). Our semantics shows that this is not necessary, although certain conditions must be met by the specification language in order to show the existence of the recursive store specifications.
   That specifications are preserved by verified programs is a consequence of the idea of enriching types and disallowing method update. It relieves the verifier from carrying around specifications of (parts of) the store. Unfortunately, it enforces a global verification regime and prohibits method and specification updates. Making assumptions about the store explicit is a remedy. This may sound tedious, but employing local reasoning principles like those of *Separation Logic* (for an overview see, for example, O'Hearn *et al.* (2001) and Reynolds (2002)), it should not be. Our denotational semantics directly supports such an explicit handling of store specifications.
   Problems that are inherent to object logics (or logics for higher-order stores) are the need for (recursive) store specifications, and the invariance of field types in the definition of subspecification. Our proof still refers to syntactic store specifications since programs will not preserve arbitrary predicates on stores. We do not know how to describe

Table 1. *Syntax*

a, b	$::=$	x	variable
	$\mid$	`true` $\mid$ `false`	booleans
	$\mid$	`if` x `then` a `else` b	conditional
	$\mid$	`let` $x = a$ `in` b	let
	$\mid$	$[\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j = \varsigma(y_j)b_j{}^{j=1\ldots m}]$	object construction
	$\mid$	$x.\mathsf{f}$	field selection
	$\mid$	$x.\mathsf{f} := y$	field update
	$\mid$	$x.\mathsf{m}$	method invocation

semantically those programs of a specific programming language that are verifiable in a specific logic.

4 Our work provides an alternative description of the difficulties inherent in programming logics for objects. Therefore, it may be appealing to semanticists and domain theorists. It may be useful to *develop and analyse similar, but more powerful, program logics* as well.

The structure of this paper is as follows. In the next section, the syntax and semantics of the object calculus are presented. Section 3 introduces the Abadi–Leino logic and the denotational semantics of its object specifications. There is then a discussion on store specifications and their semantics (Section 4). The main result is in Section 5, where the logic is proved to be sound. Finally, we sketch how recursive specifications can be introduced (Section 6) and discuss further extensions (Section 7). Section 8 provides a summary to conclude the paper.

An extended abstract of this article has appeared as Reus and Schwinghammer (2005).

2. The object calculus

We begin by recalling the language used in Abadi and Leino (2004), which is based on the imperative object calculus of Abadi and Cardelli (1996). Following Reus and Streicher (2004), we give a denotational semantics to this language in Section 2.2.

2.1. *Syntax*

Let $\mathcal{V}$, $\mathcal{M}$ and $\mathcal{F}$ be pairwise disjoint, countably infinite sets of *variables*, *method names* and *field names*, respectively. Let x, y range over $\mathcal{V}$, let $\mathsf{m} \in \mathcal{M}$ and $\mathsf{f} \in \mathcal{F}$. The language is defined by the grammar in Table 1.

Variables are (immutable) identifiers, and booleans and conditional have the usual semantics. The object expression `let` $x = a$ `in` b first evaluates a and then evaluates b with the result of a bound to x.

Object construction $[\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j = \varsigma(y_j)b_j{}^{j=1\ldots m}]$ allocates new storage and returns a reference to an object containing fields f_i (with initial value the value of x_i) and methods m_j. In a method $\mathsf{m}_j = \varsigma(y_j)b_j$, ς is a binder that binds the explicit self parameter y_j in the method body b_j. During method invocation, the method body is evaluated with the self parameter bound to the host object. We identify objects that differ only in the names of bound variables and the order of components.

The result of field selection $x.\mathsf{f}$ is the value of the field, and $x.\mathsf{f} := y$ is field update. A formal semantics is given in the next subsection.

Note that in contrast to the calculus of Abadi and Cardelli (1996), this language distinguishes between fields and methods, and method update is disallowed. Also note that we restrict the cases for field selection, field update, method invocation and statements to contain only variables as subterms (instead of arbitrary object terms). This is no real limitation because of the let construct, but it simplifies the statement of the rules of the logic (Abadi and Leino 2004). In the examples we use a more generous syntax (for instance, we allow the inclusion of natural numbers).

Example 2.1. We extend the syntax with integer constants and operations, and consider an object-based modelling of a bank account as an example:

```
acc(x) ≡ [balance = 0,
          deposit10 = ς(y) let z = y.balance+10 in y.balance:=z,
          interest = ς(y) let r = x.manager.rate in
                              let z = y.balance*r/100 in y.balance:=z]
```

Note how the self parameter y is used in both methods to access the `balance` field. Object *acc* depends on a 'managing' object x in the context that provides the interest rate, through a field `manager`, for the `interest` method.

2.2. *Semantics of objects*

2.2.1. *Preliminaries* We work in the category **pCpo** of ω-complete partial orders (not necessarily containing a least element) and partial continuous functions[†]. Let $A \rightharpoonup B$ denote the partial continuous function space between cpos A and B. For $f \in A \rightharpoonup B$ and $a \in A$ we write $f(a)\downarrow$ if f applied to a is defined, and $f(a)\uparrow$ otherwise.

If L is a set, then $\mathscr{P}(L)$ is its powerset, $\mathscr{P}_{\mathsf{fin}}(L)$ denotes the set of its finite subsets, and A^L is the set of all *total* functions from L to A. For a countable set $\mathbb{L}$ and a cpo A we write

$$\mathsf{Rec}_{\mathbb{L}}(A) = \sum\nolimits_{L \in \mathscr{P}_{\mathsf{fin}}(\mathbb{L})} A^L$$

for the cpo of *records* with *entries* from A and *labels* from $\mathbb{L}$. Note that $\mathsf{Rec}_{\mathbb{L}}$ extends to a locally continuous endofunctor on **pCpo**. Further note that, in the natural partial order on records defined in this way, only records with equal domain are comparable; in particular, a record and its extensions are *incomparable*.

A record $(L, f \in A^L)$, with labels $L = \{l_1, \ldots, l_n\}$ and corresponding entries $f(l_i) = a_i$, is written as $\{\!|l_1 = a_1, \ldots, l_n = a_n|\!\}$. Update (and extension) of records is defined as the

[†] Other categories of domains could be used; our results only rely on the existence of minimal invariant solutions to recursive domain equations (Pitts 1996).

corresponding operation on functions, that is,

$$\{\!|l_i = a_i{}^{i=1\ldots n}|\!\}[l := a] = \begin{cases} \{\!|l_1 = a_1, \ldots, l_k = a, \ldots, l_n = a_n|\!\} & \text{if } l = l_k \text{ for some } k \\ \{\!|l_i = a_i{}^{i=1\ldots n}, l = a|\!\} & \text{otherwise.} \end{cases}$$

The selection of a label $l \in \mathbb{L}$ of a record $r \in \mathsf{Rec}_{\mathbb{L}}(A)$ is written $r.l$. It is defined and yields $f(l)$ if r is $(D, f \in A^D)$ and $l \in D$.

2.2.2. *Interpretation* The language of the previous section finds its interpretation within the following system of recursively defined cpos in **pCpo**:

$$\begin{aligned} \mathsf{Val} &= \mathsf{BVal} + \mathsf{Loc} \\ \mathsf{St} &= \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Ob}) \\ \mathsf{Ob} &= \mathsf{Rec}_{\mathscr{F}}(\mathsf{Val}) \times \mathsf{Rec}_{\mathscr{M}}(\mathsf{Cl}) \\ \mathsf{Cl} &= \mathsf{St} \rightharpoonup (\mathsf{Val} + \{\mathsf{error}\}) \times \mathsf{St}. \end{aligned} \tag{1}$$

Here, Loc is some countably infinite set of *locations* ranged over by l, and BVal is the set of truth values *true* and *false*. Both are discrete partial orders and thus complete. Objects in Ob are pairs, consisting of a record that assigns values to the fields of the object, and a record associating *closures* to method names. Each closure is modelled as a partial map in Cl. In case of termination, the result value and resulting store are returned by the closure; exceptional termination is indicated by returning error. Finally, the cpo St models stores as finite records of objects, indexed by locations.

Consider the functor $F_{Store} : \mathbf{pCpo}^{op} \times \mathbf{pCpo} \to \mathbf{pCpo}$ obtained from (1) by solving the system of equations for St, and separating positive and negative occurrences of St on the right-hand side:

$$F_{Store}(S, T) = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{F}}(\mathsf{Val}) \times \mathsf{Rec}_{\mathscr{M}}(S \rightharpoonup (\mathsf{Val} + \{\mathsf{error}\}) \times T)).$$

This is a locally continuous bifunctor. So there exists a minimal invariant solution St to this system of equations, that is, $F_{Store}(\mathsf{St}, \mathsf{St}) = \mathsf{St}$ and, moreover, the identity on St is the least fixed point of the map $\delta(e) = F(e, e)$ (for instance, see Pitts (1996) and Smyth and Plotkin (1982)).

Let $\mathsf{Env} = \mathscr{V} \to_{\mathsf{fin}} \mathsf{Val}$ be the set of *environments*, that is, maps between $\mathscr{V}$ and Val with finite domain. Given an environment $\rho \in \mathsf{Env}$, the interpretation $[\![a]\!]\rho$ of an object expression a in $\mathsf{St} \rightharpoonup (\mathsf{Val} + \{\mathsf{error}\}) \times \mathsf{St}$ is given in Table 2. Here we use a (semantic) strict **let** that is also 'strict' with respect to error:

$$\textbf{let } (v, \sigma) = s \textbf{ in } s' \equiv \begin{cases} \text{undefined} & \text{if } s \text{ is undefined} \\ (\mathsf{error}, \sigma') & \text{if } s = (\mathsf{error}, \sigma') \\ (\lambda(v, \sigma).s')\, s & \text{otherwise.} \end{cases}$$

Note that for $o \in \mathsf{Ob}$ we just write $o.\mathsf{f}$ and $o.\mathsf{m}$ instead of $\pi_1(o).\mathsf{f}$ and $\pi_2(o).\mathsf{m}$, respectively. Similarly, we omit the injections for elements of $\mathsf{Val} + \{\mathsf{error}\}$, writing simply l instead of $\mathsf{in}_{\mathsf{Loc}}(l)$, and so on. Observe that, in contrast to Reus and Streicher (2004), we distinguish between non-termination (undefinedness) and exceptional termination (error); the latter represents dynamic type errors and null-pointer dereferencing. Finally, because Loc is

Table 2. *Denotational semantics*

$[\![x]\!]\rho\sigma$	$=$	$\begin{cases}(\rho(x),\sigma) & \text{if } x\in\mathsf{dom}(\rho)\\ (\mathsf{error},\sigma) & \text{otherwise}\end{cases}$
$[\![\mathtt{true}]\!]\rho\sigma$	$=$	(true,σ)
$[\![\mathtt{false}]\!]\rho\sigma$	$=$	(false,σ)
$[\![\mathtt{if}\ x\ \mathtt{then}\ b_1\ \mathtt{else}\ b_2]\!]\rho\sigma$	$=$	$\begin{cases}[\![b_1]\!]\rho\sigma' & \text{if } [\![x]\!]\rho\sigma=(\mathit{true},\sigma')\\ [\![b_2]\!]\rho\sigma' & \text{if } [\![x]\!]\rho\sigma=(\mathit{false},\sigma')\\ (\mathsf{error},\sigma') & \text{if } [\![x]\!]\rho\sigma=(v,\sigma') \text{ for } v\notin\mathsf{BVal}\end{cases}$
$[\![\mathtt{let}\ x=a\ \mathtt{in}\ b]\!]\rho\sigma$	$=$	$\mathbf{let}\ (v,\sigma')=[\![a]\!]\rho\sigma\ \mathbf{in}\ [\![b]\!]\rho[x:=v]\sigma'$
$[\![[\mathsf{f}_i=x_i{}^{i=1\ldots n},\mathsf{m}_j=\varsigma(y_j)b_j{}^{j=1\ldots m}]]\!]\rho\sigma$	$=$	$\begin{cases}(l,\sigma[l:=(o_1,o_2)]) & \text{if } x_i\in\mathsf{dom}(\rho), 1\leqslant i\leqslant n\\ (\mathsf{error},\sigma) & \text{otherwise}\end{cases}$ where $\begin{cases}l\notin\mathsf{dom}(\sigma)\\ o_1=\{\!\mid \mathsf{f}_i=\rho(x_i)^{i=1\ldots n}\mid\!\}\\ o_2=\{\!\mid \mathsf{m}_j=\lambda\sigma.[\![b_j]\!]\rho[y_j:=l]\sigma^{j=1\ldots m}\mid\!\}\end{cases}$
$[\![x.\mathsf{f}]\!]\rho\sigma$	$=$	$\mathbf{let}\ (l,\sigma')=[\![x]\!]\rho\sigma$ $\mathbf{in}\ \begin{cases}(\sigma'.l.\mathsf{f},\sigma') & \text{if } l\in\mathsf{dom}(\sigma') \text{ and } \mathsf{f}\in\mathsf{dom}(\sigma'.l)\\ (\mathsf{error},\sigma') & \text{otherwise}\end{cases}$
$[\![x.\mathsf{f}:=y]\!]\rho\sigma$	$=$	$\mathbf{let}\ (l,\sigma')=[\![x]\!]\rho\sigma\ \mathbf{in}\ \mathbf{let}\ (v,\sigma'')=[\![y]\!]\rho\sigma'$ $\mathbf{in}\ \begin{cases}(l,\sigma''[l:=\sigma''.l[\mathsf{f}:=v]]) & \text{if } l\in\mathsf{dom}(\sigma'') \text{ and } \mathsf{f}\in\mathsf{dom}(\sigma''.l)\\ (\mathsf{error},\sigma'') & \text{otherwise}\end{cases}$
$[\![x.\mathsf{m}]\!]\rho\sigma$	$=$	$\mathbf{let}\ (l,\sigma')=[\![x]\!]\rho\sigma$ $\mathbf{in}\ \begin{cases}\sigma'.l.\mathsf{m}(\sigma') & \text{if } l\in\mathsf{dom}(\sigma') \text{ and } \mathsf{m}\in\mathsf{dom}(\sigma'.l)\\ (\mathsf{error},\sigma') & \text{otherwise}\end{cases}$

assumed to be infinite, the condition $l\notin\mathsf{dom}(\sigma)$ in the case of object creation can always be satisfied. Therefore object creation will never raise error.

A more subtle point is the choice of the location itself: in order for the interpretation $[\![a]\!]$ to be well defined, a deterministic or parametric allocator has to be assumed (Banerjee and Naumann 2005, Section 5). For example, a deterministic allocator is one that always chooses the minimal location (in an appropriate sense) amongst those available for allocation. A more sophisticated solution may be possible in the setting of FM-sets (Shinwell and Pitts 2005; Benton and Leperchey 2005). For a language with a higher-order store this is an interesting research topic that needs further investigation.

2.2.3. *Flat stores* We will make use of a projection to the part of the store that contains just the data in Val, thus 'forgetting' all closures of objects residing in the store: let $\mathsf{St}_{\mathsf{Val}}=\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{F}}(\mathsf{Val}))$, and define the projection $\pi_{\mathsf{Val}}:\mathsf{St}\to\mathsf{St}_{\mathsf{Val}}$ by

$$(\pi_{\mathsf{Val}}\,\sigma).l.\mathsf{f}\ =\ \sigma.l.\mathsf{f}$$

for all $l\in\mathsf{Loc}$ and $\mathsf{f}\in\mathscr{F}$. We refer to $\pi_{\mathsf{Val}}(\sigma)$ as the *flat part* of σ. Note that for all $\sigma,\sigma'\in\mathsf{St}$,

$$\sigma\sqsubseteq\sigma'\quad\Longrightarrow\quad\pi_{\mathsf{Val}}(\sigma)=\pi_{\mathsf{Val}}(\sigma')$$

since $\mathsf{St}_{\mathsf{Val}}=\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{F}}(\mathsf{Val}))$ inherits the discrete order from Val.

3. Abadi–Leino logic

In this section we recall the logic of Abadi and Leino (2004). A slightly different presentation can be found in Tang and Hofmann (2002), where the proof system is given in a syntax-directed way.

3.1. *Transition relations and specifications*

Transition relations T correspond to the pre- and post-conditions of Hoare logic and allow us to express state changes caused by computations. The syntax of transition relations is defined by the following grammar:

$$\begin{aligned} T &::= e_0 = e_1 \mid \mathsf{alloc}_{pre}(e) \mid \mathsf{alloc}_{post}(e) \mid \neg T \mid T_0 \wedge T_1 \mid \forall x.T \\ e &::= x \mid \mathsf{f} \mid \mathsf{result} \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{sel}_{pre}(e_0, e_1) \mid \mathsf{sel}_{post}(e_0, e_1). \end{aligned}$$

Expressions e range over variables $x \in \mathscr{V}$, field names $\mathsf{f} \in \mathscr{F}$, constants true, false and result (which stands for the result value of a computation), and function symbol applications: intuitively, the application $\mathsf{sel}_{pre}(x, y)$ yields the value of field y of the object at location x before execution, provided this exists in the store, and is undefined otherwise. Correspondingly, $\mathsf{sel}_{post}(x, y)$ gives the value of field y after execution. The predicates $\mathsf{alloc}_{pre}(x)$ and $\mathsf{alloc}_{post}(x)$ are true if the location x is allocated before and after the execution, respectively, and false otherwise. The notions of free and bound variables of a transition relation T carry over directly from first-order logic. As usual, further logical constants and connectives such as *True*, *False*, disjunction and implication can be defined as abbreviations.

Specifications combine transition relations for each method together with the result types into a single specification for the whole object. They generalise the first-order types from Abadi and Cardelli (1996), and are

$$A, B \in Spec \ ::= \ Bool \mid [\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}].$$

In the case of an object specification, ς in $\varsigma(y_j)B_j::A_j$ binds the variable y_j in B_j and T_j. Specifications are identified up to renaming of bound variables and reordering of components, which will be justified by our semantics.

Intuitively, `true` and `false` satisfy *Bool*, and an object satisfies the specification $A \equiv [\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}]$ if it has fields f_i satisfying A_i and methods m_j that satisfy the transition relation T_j and, if the method invocation terminates, the result satisfies B_j. Corresponding to the fact that a method m_j can use the *self*-parameter y_j, in both T_j and B_j one can refer to the ambient object y_j.

Let Γ range over *specification contexts* $x_1{:}A_1, \ldots, x_n{:}A_n$. A specification context is *well formed* if no variable x_i occurs more than once, and the free variables of A_k are contained in the set $\{x_1, \ldots, x_{k-1}\}$. In writing $\Gamma, x{:}A$, we will always assume that x does not appear in Γ. Sometimes we will write $\varnothing$ for the empty context. Given Γ, we write $[\Gamma]$ for the list of variables occurring in Γ:

$$[x_1{:}A_1, \ldots, x_n{:}A_n] \ = \ x_1, \ldots, x_n.$$

Table 3. *Well-formed specifications and contexts*

WFSPEC1

$$\frac{}{\overline{x} \vdash \mathit{Bool}}$$

WFSPEC2

$$\frac{\overline{x} \vdash A_i{}^{i=1\ldots n} \qquad \overline{x}, y_j \vdash B_j{}^{j=1\ldots m} \qquad \overline{x}, y_j \vdash T_j{}^{j=1\ldots m}}{\overline{x} \vdash [\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j : \varsigma(y_j) B_j :: T_j{}^{j=1\ldots m}]}$$

WFCTXT1

$$\frac{}{\varnothing \vdash \mathsf{ok}}$$

WFCTXT2

$$\frac{\Gamma \vdash \mathsf{ok} \qquad [\Gamma] \vdash A \qquad x \notin \mathsf{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathsf{ok}}$$

Table 4. *Transition relations* T_{res}, T_{obj} *and* T_{upd}

$T_{\mathsf{res}}(e)$	$\equiv$	$\mathsf{result} = e \wedge \forall x \forall f.\ (\mathsf{alloc}_{pre}(x) \leftrightarrow \mathsf{alloc}_{post}(x) \wedge \mathsf{sel}_{pre}(x,f) = \mathsf{sel}_{post}(x,f))$
$T_{\mathsf{obj}}(\mathsf{f}_i{=}x_i)^{i=1\ldots n}$	$\equiv$	$\neg\mathsf{alloc}_{pre}(\mathsf{result}) \wedge \mathsf{alloc}_{post}(\mathsf{result}) \wedge \bigwedge_{i=1\ldots n} \mathsf{sel}_{post}(\mathsf{result}, \mathsf{f}_i) = x_i$ $\wedge \forall x \forall f.\ x {\neq} \mathsf{result} \rightarrow (\mathsf{alloc}_{pre}(x) {\leftrightarrow} \mathsf{alloc}_{post}(x) \wedge \mathsf{sel}_{pre}(x,f) = \mathsf{sel}_{post}(x,f))$
$T_{\mathsf{upd}}(x,f,e)$	$\equiv$	$\forall x'.\ \mathsf{alloc}_{pre}(x') \leftrightarrow \mathsf{alloc}_{post}(x') \wedge \mathsf{sel}_{post}(x,f) = e \wedge \mathsf{result} = x$ $\wedge \forall x' \forall f'.\ (x' \neq x \vee f' \neq f) \rightarrow \mathsf{sel}_{pre}(x',f') = \mathsf{sel}_{post}(x',f')$

If it is clear from the context, we use the notation $\overline{x}$ for a sequence $x_1, \ldots, x_n$, and, similarly, $\overline{x} : \overline{A}$ for $x_1{:}A_1, \ldots, x_n{:}A_n$. To make the notions of well-formed specifications and well-formed specification contexts formal, there are judgements for:

— well-formed transition relations: $\overline{x} \vdash T$

— well-formed specifications: $\overline{x} \vdash A$

— well-formed specification contexts: $\Gamma \vdash \mathsf{ok}$.

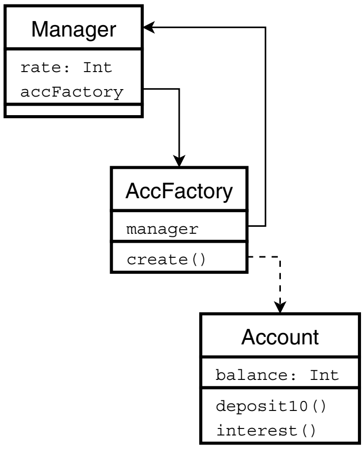
For transition relations, $x_1, \ldots, x_n \vdash T$ holds if all the free variables of T appear in $x_1, \ldots, x_n$; we will omit the simple rules. Table 3 contains the rules for specifications and specification contexts. When A is closed we may simply write A instead of $\vdash A$, and similarly for closed T.

Table 4 defines several transition relations that are used in the statement of the rules in the following subsection. The relation $T_{\mathsf{res}}(e)$ states that the result of a computation is e and that the flat part of the store remains unchanged. While transition relations do not talk about the non-flat part of the store, the stored methods remain necessarily unchanged since the variant of the object calculus considered here has no method update. $T_{\mathsf{obj}}(\mathsf{f}_i = x_i)$ describes the allocation of a new object in memory, which is initialised with field f_i set to x_i, and whose location is returned as the result. $T_{\mathsf{upd}}(x,f,e)$ describes the effect on the store when updating field $x.f$. Note that in Abadi and Leino (2004) the relation T_{res} is called *Res* and T_{upd} is called *Update*. There is no abbreviation corresponding to T_{obj}.

Example 3.1. Table 5 shows a specification for the bank accounts of Example 2.1. Although we are using UML-like notation, these diagrams actually stand for individual objects, not classes – in fact there are no classes in the language. Observe how the

Table 5. *An example of transition and result specifications*

$$\begin{aligned}
T_{\texttt{deposit10}}(y) &\equiv \exists z.\, z = \mathsf{sel}_{pre}(y, \texttt{balance}) \\
&\quad \wedge T_{\mathsf{upd}}(y, \texttt{balance}, z + 10) \\
T_{\texttt{interest}}(x, y) &\equiv \exists z.\, z = \mathsf{sel}_{pre}(y, \texttt{balance}) \\
&\quad \wedge \exists m.\, m = \mathsf{sel}_{pre}(x, \texttt{manager}) \\
&\quad \wedge \exists r.\, r = \mathsf{sel}_{pre}(m, \texttt{rate}) \\
&\quad \wedge T_{\mathsf{upd}}(y, \texttt{balance}, z * r/100) \\
T_{\texttt{create}}(x) &\equiv T_{\mathsf{obj}}(\texttt{balance} = 0) \\
A_{\texttt{Account}}(x) &\equiv [\texttt{balance} : \mathit{Int}, \\
&\quad\ \texttt{deposit10} : \varsigma(y)[] :: T_{\texttt{deposit10}}(y), \\
&\quad\ \texttt{interest} : \varsigma(y)[] :: T_{\texttt{interest}}(x, y)] \\
A_{\texttt{AccFactory}} &\equiv [\texttt{manager} : [\texttt{rate} : \mathit{Int}], \\
&\quad\ \texttt{create} : \varsigma(x) A_{\texttt{Account}}(x) :: T_{\texttt{create}}(x)] \\
A_{\texttt{Manager}} &\equiv [\texttt{rate} : \mathit{Int}, \\
&\quad\ \texttt{accFactory} : A_{\texttt{AccFactory}}]
\end{aligned}$$



specification $T_{\texttt{interest}}$ depends not only on the self parameter y of the host object but also on the statically enclosing object x.

The result specifications [] for the methods `deposit10` and `interest` in $A_{\texttt{Account}}$ stand for the 'empty' object. However, by the subspecification relation, defined below in Section 3.2, this specification is satisfied by any object. In particular, this is the case for both method implementations in Example 2.1 where the result is the object itself, according to the semantics of field update.

3.2. *Abadi–Leino logic*

Abadi and Leino generalised the notion of subtypes to a form of *subspecification*, $\overline{x} \vdash A <: A'$, that is defined inductively by $\overline{x} \vdash \mathit{Bool} <: \mathit{Bool}$ and the rule

$$\frac{\begin{array}{cccc} \overline{x} \vdash A_i^{\,i=1\ldots n+p} & \overline{x}, y_j \vdash B_j^{\,j=m+1\ldots m+q} & \overline{x}, y_j \vdash T_j^{\,j=1\ldots m+q} & \overline{x}, y_j \vdash T_j'^{\,j=1\ldots m} \\ & \overline{x}, y_j \vdash B_j <: B_j'^{\,j=1\ldots m} & \vdash_{\mathsf{fo}} T_j \to T_j'^{\,j=1\ldots m} & \end{array}}{\overline{x} \vdash [\mathsf{f}_i : A_i^{\,i=1\ldots n+p}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j^{\,j=1\ldots m+q}] <: [\mathsf{f}_i : A_i^{\,i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j' :: T_j'^{\,j=1\ldots m}]}$$

where $\vdash_{\mathsf{fo}} \varphi$ denotes provability in first-order logic with equality (in the theory with axioms stating that `true`, `false` and all $\mathsf{f} \in \mathscr{F}$ are distinct). Just as with subtyping in the corresponding type system (Abadi and Cardelli 1996), the subspecification relation is covariant along method specifications and transition relations, and invariant in field specifications. Observe that $\overline{x} \vdash A_1 <: A_2$ implies $\overline{x} \vdash A_i$ for $i = 1, 2$.

In the logic, judgements of the form $\Gamma \vdash a{:}A{::}T$ can be derived, where Γ is a well-formed specification context, a is an object expression, A is a specification and T is a transition relation. The rules guarantee that all the free variables of a, A and T appear in $[\Gamma]$. There is one rule for each syntactic form of the language, and, additionally, a subsumption rule that generalises the consequence rule of classical Hoare logic. The rules are given in Table 6.

Table 6. *Inference rules of Abadi–Leino logic*

Subsumption	$\dfrac{[\Gamma]\vdash A <: A' \quad \Gamma\vdash a{:}A{::}T \quad [\Gamma]\vdash A' \quad [\Gamma]\vdash T' \quad \vdash_{\mathsf{fo}} T\to T'}{\Gamma\vdash a{:}A'{::}T'}$
Variable	$\dfrac{\Gamma\vdash \mathsf{ok} \quad x{:}A \text{ in } \Gamma}{\Gamma\vdash x{:}A{::}T_{\mathsf{res}}(x)}$
Booleans	$\dfrac{\Gamma\vdash \mathsf{ok}}{\Gamma\vdash \mathtt{false}{:}\mathit{Bool}{::}T_{\mathsf{res}}(\mathsf{false})} \qquad \dfrac{\Gamma\vdash \mathsf{ok}}{\Gamma\vdash \mathtt{true}{:}\mathit{Bool}{::}T_{\mathsf{res}}(\mathsf{true})}$
Conditional	$\dfrac{\begin{array}{c} A[\mathsf{true}/x]\equiv A_t[\mathsf{true}/x] \text{ and } A[\mathsf{false}/x]\equiv A_f[\mathsf{false}/x] \\ T[\mathsf{true}/x]\equiv T_t[\mathsf{true}/x] \text{ and } T[\mathsf{false}/x]\equiv T_f[\mathsf{false}/x] \\ \Gamma\vdash x{:}\mathit{Bool}{::}T_{\mathsf{res}}(x) \quad \Gamma\vdash a{:}A_t{::}T_t \quad \Gamma\vdash b{:}A_f{::}T_f \end{array}}{\Gamma\vdash \mathtt{if}\ x\ \mathtt{then}\ a\ \mathtt{else}\ b{:}A{::}T}$
Let	$\dfrac{\begin{array}{c} \Gamma\vdash a{:}A'{::}T' \quad \Gamma, x{:}A'\vdash b{:}B{::}T'' \quad [\Gamma]\vdash B \quad [\Gamma]\vdash T \\ \vdash_{\mathsf{fo}} T'[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}] \\ \wedge\ T''[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)] \to T \end{array}}{\Gamma\vdash \mathtt{let}\ x = a\ \mathtt{in}\ b{:}B{::}T}$
Object Construction	$\dfrac{\begin{array}{c} A\equiv[\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}] \\ \Gamma\vdash x_i{:}A_i{::}T_{\mathsf{res}}(x_i)^{i=1\ldots n} \qquad \Gamma, y_j{:}A\vdash b_j{:}B_j{::}T_j{}^{j=1\ldots m} \end{array}}{\Gamma\vdash[\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j=\varsigma(y_j)b_j{}^{j=1\ldots m}]{:}A{::}T_{\mathsf{obj}}(\mathsf{f}_1{=}x_1\ldots\mathsf{f}_n{=}x_n)}$
Field Selection	$\dfrac{\Gamma\vdash x{:}[\mathsf{f}{:}A]{::}T_{\mathsf{res}}(x)}{\Gamma\vdash x.\mathsf{f}{:}A{::}T_{\mathsf{res}}(\mathsf{sel}_{pre}(x,\mathsf{f}))}$
Field Update	$\dfrac{\begin{array}{c} A\equiv[\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}] \\ \Gamma\vdash x{:}A{::}T_{\mathsf{res}}(x) \quad \Gamma\vdash y{:}A_k{::}T_{\mathsf{res}}(y) \quad 1\leqslant k\leqslant n \end{array}}{\Gamma\vdash x.\mathsf{f}_k := y{:}A{::}T_{\mathsf{upd}}(x,\mathsf{f}_k,y)}$
Method Invocation	$\dfrac{\Gamma\vdash x{:}[\mathsf{m}{:}\varsigma(y)A{::}T]{::}T_{\mathsf{res}}(x)}{\Gamma\vdash x.\mathsf{m}{:}A[x/y]{::}T[x/y]}$

As indicated earlier, one of the most interesting and powerful rules of the logic is the Object Construction rule,

$$\dfrac{\begin{array}{c} A\equiv[\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}] \\ \Gamma\vdash x_i{:}A_i{::}T_{\mathsf{res}}(x_i)^{i=1\ldots n} \qquad \Gamma, y_j{:}A\vdash b_j{:}B_j{::}T_j{}^{j=1\ldots m} \end{array}}{\Gamma\vdash[\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j=\varsigma(y_j)b_j{}^{j=1\ldots m}]{:}A{::}T_{\mathsf{obj}}(\mathsf{f}_i = x_i)^{i=1\ldots n}}$$

In order to establish that the object satisfies specification A, when verifying the methods b_j, we can *assume* that the self parameter y_j also satisfies A. Essentially, this causes the semantics of store specifications, introduced in the next section, to be defined by a *mixed-variant* recursion.

The rule for the Let case is somewhat unusual in that it introduces additional function and relation symbols to the signature, $\mathsf{sel}_{int}(\cdot,\cdot)$ and $\mathsf{alloc}_{int}(\cdot)$, to capture the intermediate state of the store in first-order logic. In the hypothesis, textual substitution of *function* and *predicate* symbols, respectively, is used to compose the first and second transition relation: for instance,

$$\mathsf{sel}_{post}(e_1, e_2)[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot)] \;\equiv\; \mathsf{sel}_{int}(e_1, e_2).$$

We omit the obvious general definition of this substitution operation. The side condition $[\Gamma] \vdash T$ ensures that the transition relation in the conclusion does not export this intermediate state.

Example 3.2. The proof rules of Abadi and Leino's logic can be used to derive the judgement

$$x{:}A_{\texttt{AccFactory}} \vdash acc(x) : A_{\texttt{Account}}(x) :: T_{\mathsf{obj}}(\texttt{balance} = 0) \tag{2}$$

for the *acc* object (*cf.* Examples 2.1 and 3.1). Using the Object Construction rule, (2) can be reduced to a trivial proof obligation for the field `balance`, a judgement for the method `deposit10`,

$$\Gamma \vdash \texttt{let } z\texttt{=(}y\texttt{.balance)+10 in } y\texttt{.balance:=}z : [] :: T_{\texttt{deposit10}}(y) \tag{3}$$

where Γ is the context $x{:}A_{\texttt{AccFactory}}, y{:}A_{\texttt{Account}}(x)$, and a similar judgement for the method `interest`. In turn, a proof of (3) involves showing

$$\Gamma \vdash (y\texttt{.balance)+10} : Int :: T_{\mathsf{res}}(\mathsf{sel}_{pre}(y, \texttt{balance}) + 10) \tag{4}$$

$$\Gamma, z{:}Int \vdash y\texttt{.balance:=}z : [] :: T_{\mathsf{upd}}(y, \texttt{balance}, z) \tag{5}$$

for the constituents of the let expression, by an application of Let. These can be proved from the Field Selection and Field Update rules, respectively.

As another example, the structural subspecification $\vdash A_{\texttt{Manager}} <: [\texttt{rate} : Int]$ and Subsumption can be used to prove

$$m{:}A_{\texttt{Manager}}, x{:}A_{\texttt{AccFactory}} \vdash x\texttt{.manager:=}m : A_{\texttt{AccFactory}} :: T_{\mathsf{upd}}(x, \texttt{manager}, m)$$

when creating a reference to the manager object in the `manager` field of the factory object, as indicated by the diagram in Table 5.

3.3. *Semantics of specifications*

Having recalled Abadi and Leino's logic, we next give a denotational semantics of specifications. In transition relations it is possible to quantify over field names, for examples of this see the transition relations in Table 4. We write $\mathsf{Env}^* = \mathscr{V} \to_{\mathsf{fin}} (\mathsf{Val} \uplus \mathscr{F})$ when interpreting transition relations:

$$[\![\overline{x} \vdash T]\!] : \mathsf{Env}^* \to \mathscr{P}(\mathsf{St}_{\mathsf{Val}} \times \mathsf{Val} \times \mathsf{St}_{\mathsf{Val}}).$$

This can be defined in a straightforward way, a few typical cases are given in Tables 7 and 8. Note that even though expressions may be undefined (for instance, because they refer to non-existent fields), the interpretation of transition relations is two-valued. Also observe that the meaning of a transition relation $\overline{x} \vdash T$ without free variables does not depend on the environment, so we may omit the environment and simply write $[\![T]\!]$ for closed T.

Table 7. *Semantics of expressions*

$[\![\overline{x} \vdash e]\!] : \mathsf{Env}^* \to \mathsf{St}_{\mathsf{Val}} \to \mathsf{Val} \to \mathsf{St}_{\mathsf{Val}} \rightharpoonup (\mathsf{Val} \uplus \mathscr{F})$			
$[\![\overline{x} \vdash x]\!]\rho\sigma v\sigma'$	$=$	$\rho(x)$	if $x \in \mathsf{dom}(\rho)$
		undefined	otherwise
$[\![\overline{x} \vdash \mathsf{f}]\!]\rho\sigma v\sigma'$	$=$	f	
$[\![\overline{x} \vdash \mathsf{result}]\!]\rho\sigma v\sigma'$	$=$	v	
$[\![\overline{x} \vdash \mathsf{true}]\!]\rho\sigma v\sigma'$	$=$	*true*	
$[\![\overline{x} \vdash \mathsf{false}]\!]\rho\sigma v\sigma'$	$=$	*false*	
$[\![\overline{x} \vdash \mathsf{sel}_{pre}(e_0, e_1)]\!]\rho\sigma v\sigma'$	$=$	$\sigma.l.\mathsf{f}$	if $[\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma' = l \in \mathsf{Loc}$ defined and $[\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma' = \mathsf{f} \in \mathscr{F}$ defined and $l \in \mathsf{dom}(\sigma)$ and $\mathsf{f} \in \mathsf{dom}(\sigma.l)$
		undefined	otherwise
$[\![\overline{x} \vdash \mathsf{sel}_{post}(e_0, e_1)]\!]\rho\sigma v\sigma'$	$=$	$\sigma'.l.\mathsf{f}$	if $[\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma' = l \in \mathsf{Loc}$ defined and $[\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma' = \mathsf{f} \in \mathscr{F}$ defined and $l \in \mathsf{dom}(\sigma')$ and $\mathsf{f} \in \mathsf{dom}(\sigma'.l)$
		undefined	otherwise

Table 8. *Semantics of transition relations*

$[\![\overline{x} \vdash T]\!] : \mathsf{Env}^* \to \mathscr{P}(\mathsf{St}_{\mathsf{Val}} \times \mathsf{Val} \times \mathsf{St}_{\mathsf{Val}})$		
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash e_0 = e_1]\!]\rho$	$\Longleftrightarrow$	both $[\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma'$ and $[\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma'$ are defined and equal, or both undefined
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash \mathsf{alloc}_{pre}(e)]\!]\rho$	$\Longleftrightarrow$	$[\![\overline{x} \vdash e]\!]\rho\sigma v\sigma'\downarrow \ \wedge\ [\![\overline{x} \vdash e]\!]\rho\sigma v\sigma' \in \mathsf{dom}(\sigma)$
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash \mathsf{alloc}_{post}(e)]\!]\rho$	$\Longleftrightarrow$	$[\![\overline{x} \vdash e]\!]\rho\sigma v\sigma'\downarrow \ \wedge\ [\![\overline{x} \vdash e]\!]\rho\sigma v\sigma' \in \mathsf{dom}(\sigma')$
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash \neg T]\!]\rho$	$\Longleftrightarrow$	$(\sigma, v, \sigma') \notin [\![\overline{x} \vdash T]\!]\rho$
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash T_0 \wedge T_1]\!]\rho$	$\Longleftrightarrow$	$(\sigma, v, \sigma') \in [\![\overline{x} \vdash T_0]\!]\rho \cap [\![\overline{x} \vdash T_1]\!]\rho$
$(\sigma, v, \sigma') \in [\![\overline{x} \vdash \forall x.T]\!]\rho$	$\Longleftrightarrow$	for all $u \in \mathsf{Val} \uplus \mathscr{F}.\ (\sigma, v, \sigma') \in [\![\overline{x}, x \vdash T]\!]\rho[x := u]$

Table 9. *Semantics of specifications*

$[\![\overline{x} \vdash A]\!] : \mathsf{Env} \to \mathscr{P}(\mathsf{Val} \times \mathsf{St})$
$[\![\overline{x} \vdash Bool]\!]\rho \ = \ \mathsf{BVal} \times \mathsf{St}$
$[\![\overline{x} \vdash [\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]]\!]\rho$
$= \left\{ (l, \sigma) \in \mathsf{Loc} \times \mathsf{St} \ \middle\vert \ \begin{array}{l} \text{(F)}\ \ \forall 1 \leqslant i \leqslant n.\ (\sigma.l.\mathsf{f}_i, \sigma) \in [\![\overline{x} \vdash A_i]\!]\rho \\ \text{(M)}\ \ \forall 1 \leqslant j \leqslant m.\ \text{ if } \sigma.l.\mathsf{m}_j(l, \sigma) = (v, \sigma')\downarrow \\ \qquad \text{then } (v, \sigma') \in [\![\overline{x}, y_j \vdash B_j]\!]\rho[y_j := l] \\ \qquad \text{and } (\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![\overline{x}, y_j \vdash T_j]\!]\rho[y_j := l] \end{array} \right\}$

Remark 3.3. We think it would have been clearer to use a multi-sorted logic, with different quantifiers ranging over locations, basic values and field names, respectively, but decided to keep close to the original presentation of Abadi and Leino's logic.

An object specification $\overline{x} \vdash A$ gives rise to a predicate that depends on values for the free variables $\overline{x}$. However, since the underlying logic in the transition relations is untyped, the types of the free variables are not relevant. The interpretation of object specifications $\overline{x} \vdash A$,

$$[\![\overline{x} \vdash A]\!] : \mathsf{Env} \to \mathscr{P}(\mathsf{Val} \times \mathsf{St}),$$

is given in Table 9.

We begin with two simple observations about the interpretation.

Lemma 3.4. For all specifications $\overline{x} \vdash A$, store $\sigma \in \mathsf{St}$ and environments ρ we have $(\mathsf{error}, \sigma) \notin [\![\overline{x} \vdash A]\!]\rho$.

Proof. The statement is immediate from the definition of $[\![\overline{x} \vdash A]\!]\rho$. □

This observation will be used to obtain soundness of the proof system, in the sense of 'well-typed (or rather, *verified*) programs do not raise errors'.

Lemma 3.5 (Soundness of subspecification). Suppose $\overline{x} \vdash A <: B$. Then, for all environments ρ, $[\![\overline{x} \vdash A]\!]\rho \subseteq [\![\overline{x} \vdash B]\!]\rho$ for values $\overline{v}$.

Proof. We use induction on the derivation of $\overline{x} \vdash A <: B$. The cases for reflexivity and transitivity are immediate. For the case where both A and B are object specifications we need a similar lemma for transition relations:

If $\overline{x} \vdash T$ and $\overline{x} \vdash T'$, then

$$\vdash_{\mathsf{fo}} T \to T' \quad\Longrightarrow\quad [\![\overline{x} \vdash T]\!]\rho \subseteq [\![\overline{x} \vdash T']\!]\rho \tag{6}$$

for all $\rho \in \mathsf{Env}^*$. However, (6) follows immediately since $\vdash_{\mathsf{fo}}$ is sound for all models of first-order logic. □

4. Store specifications

Object specifications are not sufficient. This is a phenomenon of languages with a higher-order store and is well known from subject reduction and type soundness proofs in both operational and denotational settings (for instance, see Abadi and Cardelli (1996, Chapter 11) and the references therein, and Levy (2002), respectively). Since statements may call subprograms residing in the store, the store has to be checked as well. However, it may contain loops, so induction on the reachable part of the store is unavailable.

The standard remedy – also used in Abadi and Leino (2004) – is to relativise the typing judgement such that it only needs to hold for 'verified' stores. In other words, judgements are interpreted with respect to *store specifications*. A store specification Σ assigns a specification to each location in a store:

$$\Sigma \;\equiv\; l_1{:}A_1, \ldots, l_n{:}A_n.$$

When an object is created, the specification assigned to it at the time of creation is included in the store specification. This leads to a natural notion of *store specification extension*.

In this section we will interpret such store specifications using techniques from Reus and Streicher (2004). Since the denotations of a store specification will rely on mixed-variant recursion, we were unable to define a semantic notion of subspecification for stores. However, the logic of Abadi and Leino makes essential use of subspecifications.

We get around this problem by only using a subset relationship on denotations of object specifications. In object specifications there is no contravariant occurrence of store as the semantics of objects is with respect to one fixed store (*cf.* Table 9).

Unfortunately, we are restricted by the logic's requirement that verified statements never break the validity of store specifications. Suppose y denotes an object in σ satisfying a specification B. For a field update $\sigma.l.\mathsf{f} := y$ to preserve a specification $l : A$ where $A <: [\mathsf{f} : B]$, the location l must be in the set

$$\{l \in \mathsf{Loc} \mid \Sigma.l \text{ is } A' \text{ and } \vdash A' <: A\}, \tag{7}$$

which ensures that A and A' both have a component named f, of type B. Since the semantic interpretation of the subspecification relation as set containment cannot reflect this invariance, preservation *cannot* be guaranteed for locations in the (semantically more appealing) set

$$\{l \in \mathsf{dom}(\sigma) \mid (l, \sigma) \in [\![A]\!]\}.$$

Therefore we were forced to use the former set (7) for the interpretation of A in the semantics of store specifications.

4.1. *Result specifications, store specifications and a tentative semantics*

A store specification Σ assigns *closed* specifications $\vdash A$ to (a finite set of) locations.

Definition 4.1 (Store specification). A record $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$ is a *store specification* if for all $l \in \mathsf{dom}(\Sigma)$, $\Sigma.l$ is a closed object specification. Let *StSpec* denote the set of store specifications.

Because we focus on closed specifications in the following, we need a way to turn the components B_j of a specification $[\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}]$ (which will, in general, depend on the self parameter y_j) into closed specifications. We do this by extending the syntax of expressions with locations: there is one symbol $\underline{l}$ for each $l \in \mathsf{Loc}$, and we define $[\![\overline{x} \vdash \underline{l}]\!]\rho = l$ (*cf.* Table 7). Similarly, we set $\underline{\mathit{true}} = \mathtt{true}$ and $\underline{\mathit{false}} = \mathtt{false}$. When it is clear from the context, we will simply write v in place of $\underline{v}$.

Furthermore, we write $A[\rho/\overline{x}]$ (respectively, $A[\rho/\Gamma]$) for the simultaneous substitution of all $x \in \overline{x}$ (respectively, $x \in [\Gamma]$) in A by $\underline{\rho(x)}$. Then we can prove the following substitution lemma.

Lemma 4.2 (Substitution lemma). Suppose ρ is an environment, $\overline{x} \vdash T$ is a transition relation and $\overline{y} \vdash A$ and $\overline{y} \vdash A'$ are specifications. Then:

— $T[\rho/\overline{x}]$ is well formed, and $[\![T[\rho/\overline{x}]]\!] = [\![\overline{x} \vdash T]\!]\rho$.
— $A[\rho/\overline{y}]$ is well formed, and $[\![A[\rho/\overline{y}]]\!] = [\![\overline{y} \vdash A]\!]\rho$.
— if $\overline{y} \vdash A <: A'$, then $\vdash A[\rho/\overline{y}] <: A'[\rho/\overline{y}]$.

Proof. The first part is by induction on T, the second by induction on A and the last by induction on the derivation of $\overline{y} \vdash A <: A'$. □

Definition 4.3 (Store specification extension). Let $\Sigma, \Sigma' \in \mathit{StSpec}$ be store specifications. Σ' *extends* Σ, written $\Sigma' \succcurlyeq \Sigma$, if $\mathsf{dom}(\Sigma) \subseteq \mathsf{dom}(\Sigma')$ and $\Sigma.l = \Sigma'.l$ for all $l \in \mathsf{dom}(\Sigma)$.

Table 10. *Store specifications, first (and incorrect) attempt*

$$
\begin{array}{lll}
\sigma \in [\![\Sigma]\!] & \iff & \forall l \in \mathsf{dom}(\Sigma) \text{ where } \Sigma.l \equiv [\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}] : \\
 & & \text{(F)} \quad \forall 1 \leqslant i \leqslant n.\ \ \sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma; \quad \text{and} \\
 & & \text{(M)} \quad \forall 1 \leqslant j \leqslant m \ \ \forall \Sigma' \succcurlyeq \Sigma\ \forall \sigma', \sigma'' \in \mathsf{St}\ \forall l' \in \mathsf{Loc}\ \forall v \in \mathsf{Val}. \\
 & & \qquad\quad \text{if } l' \in \|\Sigma.l\|_{\Sigma'} \ \wedge\ \sigma' \in [\![\Sigma']\!] \ \wedge\ \sigma.l.\mathsf{m}_j(l', \sigma') = (v, \sigma'') \text{ then} \\
 & & \text{(M1)} \qquad (\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![T_j[l'/y_j]]\!] \quad \text{and} \\
 & & \text{(M2)} \qquad \exists \Sigma'' \in \mathit{StSpec}.\ \Sigma'' \succcurlyeq \Sigma' \ \wedge\ \sigma'' \in [\![\Sigma'']\!] \quad \text{and} \\
 & & \text{(M3)} \qquad v \in \|B_j[l'/y_j]\|_{\Sigma''}
\end{array}
$$

Note that $\succcurlyeq$ is reflexive and transitive. We can then abstract away from particular stores $\sigma \in \mathsf{St}$, and interpret closed result specifications $\vdash A$ with respect to such store specifications.

Definition 4.4 (Object specifications). Suppose $\Sigma \in \mathit{StSpec}$ is a store specification. For closed $\vdash A$, let $\|A\|_\Sigma \subseteq \mathsf{Val}$ be defined by

$$
\begin{aligned}
\|\mathit{Bool}\|_\Sigma &= \mathsf{BVal} \\
\|B\|_\Sigma &= \{ l \in \mathsf{Loc} \mid \ \vdash \Sigma.l <: B \}
\end{aligned}
$$

where $B \equiv [\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}]$ and $\vdash B$. We extend this definition to specification contexts $\|\Gamma\|_\Sigma \subseteq \mathsf{Env}$ in the natural way:

$$
\begin{aligned}
\rho \in \|\varnothing\|_\Sigma &\iff \text{always} \\
\rho \in \|\Gamma, x{:}A\|_\Sigma &\iff \rho \in \|\Gamma\|_\Sigma \quad \wedge \quad \rho(x) \in \|A[\rho/\Gamma]\|_\Sigma .
\end{aligned}
$$

Observe that for all A, if $\Sigma' \succcurlyeq \Sigma$, then $\|A\|_\Sigma \subseteq \|A\|_{\Sigma'}$. We obtain the following lemma for *context extensions.*

Lemma 4.5 (Context extension). If $\rho \in \|\Gamma\|_\Sigma$ and $\Gamma, x{:}A \vdash \mathsf{ok}$ and $v \in \|A[\rho/\Gamma]\|_\Sigma$, then $\rho[x := v] \in \|\Gamma, x{:}A\|_\Sigma$.

Proof. The result follows immediately from the definition once we show $\rho[x := v] \in \|\Gamma\|_\Sigma$. This can be seen to hold since $x \notin \mathsf{dom}(\Gamma)$, hence, for all $y{:}B$ in Γ we know that x is not free in B and we must have $B[\rho[x := v]/\Gamma] \equiv B[\rho/\Gamma]$. □

The semantics of a store specification $\Sigma = l_1{:}A_1, \ldots, l_n{:}A_n$ as a predicate over stores in St must be more sophisticated than one might think at first sight. In particular, it is not enough to stipulate that $\sigma \in [\![\Sigma]\!]$ iff $(\sigma.l_i, \sigma) \in [\![A_i]\!]$ for all $1 \leqslant i \leqslant n$, for then we could not infer anything about stores that are derived from σ by *updating* or *allocating additional objects*. To show that Σ is invariant throughout the execution of a program, a key lemma for the soundness theorem, the assumption $\sigma \in [\![\Sigma]\!]$ is not sufficiently strong to conclude that, for example, $\sigma[l := \sigma.l[\mathsf{f} := 0]]$ or $\sigma[l_{new} := o]$ still fulfil $[\![\Sigma]\!]$.

This deficiency explains the need for a mixed variant recursive definition of the semantics of $[\![\Sigma]\!]$, with a universal quantification over *arbitrary argument stores* σ' assumed to already fulfil the store specification we are defining. Thus, if the equivalence in Table 10 is taken as the defining property of $[\![\Sigma]\!]$, this quantification in (M) provides a handle to the update problem. Furthermore, using the extension relation of store specifications allows us to

Table 11. *Functional for store specifications, first (and incorrect) attempt*

$$
\begin{array}{lll}
\sigma \in \Phi(Y,X)_\Sigma & \iff & \forall l \in \mathsf{dom}(\Sigma) \text{ where } \Sigma.l \equiv [\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}] : \\
 & & \text{(F)} \quad \forall 1 \leqslant i \leqslant n.\ \ \sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma ; \quad \text{and} \\
 & & \text{(M)} \quad \forall 1 \leqslant j \leqslant m\ \ \forall \Sigma' \succcurlyeq \Sigma\ \forall \sigma', \sigma'' \in \mathsf{St}\ \forall l' \in \mathsf{Loc}\ \forall v \in \mathsf{Val}. \\
 & & \qquad\quad \text{if } l' \in \|\Sigma.l\|_{\Sigma'}\ \wedge\ \sigma' \in Y_{\Sigma'}\ \wedge\ \sigma.l.\mathsf{m}_j(l',\sigma') = (v,\sigma'') \text{ then} \\
 & & \text{(M1)} \qquad (\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T_j[l'/y_j]]\!] \ \text{ and} \\
 & & \text{(M2)} \qquad \exists \Sigma'' \in \mathit{StSpec}.\ \Sigma'' \succcurlyeq \Sigma'\ \wedge\ \sigma'' \in X_{\Sigma''} \ \text{ and} \\
 & & \text{(M3)} \qquad v \in \|B_j[l'/y_j]\|_{\Sigma''}
\end{array}
$$

address the issues with allocation: the universal quantification over extensions Σ' of Σ in (M) accounts for the specifications of objects allocated between the time they are defined and the time the methods are called. The existential quantification over extensions Σ'' of Σ in (M2) and (M3) provides for objects allocated by the method. In particular, since the result of a method call may be a freshly allocated object, it is not sufficient simply to use Σ' in (M2) and (M3).

This semantic structure also appears in models for many other languages with dynamic allocation (Levy 2002; Levy 2004; Reddy and Yang 2004; Stark 1998; Banerjee and Naumann 2005; Moggi 1990). It is particularly obvious in those models explicitly constructed in terms of possible worlds. Indeed, we may view $\|A\|$ as a functor from the partial order category $(\mathit{StSpec}, \succcurlyeq)$ to the category of subsets of Val ordered by set inclusion, for every specification A.

The standard approach to obtain predicates defined by a mixed-variant recursion proceeds as follows. Let $\mathscr{R} = \mathscr{P}(\mathsf{St})^{\mathit{StSpec}}$ denote the collection of predicates on St, indexed by store specifications, and define a functional $\Phi : \mathscr{R}^{op} \times \mathscr{R} \to \mathscr{R}$ according to Table 11. We would then like to write $[\![\Sigma]\!]$ for $\mathrm{fix}(\Phi)_\Sigma$. Unfortunately, there is a problem with the definition of $[\![\Sigma]\!]$ as $\mathrm{fix}(\Phi)_\Sigma$, which we discuss next.

4.2. *On the existence of store specifications*

The contravariant occurrence of Y in case (M) of the 'definition' of Φ above is forced by the premise of the object construction rule in the Abadi–Leino logic. It states that, in order to prove that specification A holds for a new object, one can assume that the self object in methods already fulfils the specification A. It is this contravariance, in turn, that calls for some advanced domain theory to show that the fixpoint of Φ does actually exist.

Unfortunately, the usual techniques (Pitts 1996; Reus and Streicher 2004) for establishing the existence of such predicates involving a mixed-variance recursion (suitably extended to *families* of predicates) do not apply: they require the functional Φ of Table 11 to map *admissible predicates* to *admissible predicates*. However, because of the existential quantification in cases (M2) and (M3), ranging over extensions of Σ', this property fails here. To see this failure, a counterexample is sketched in the remainder of this subsection, which the reader may wish to skip.

Essentially, the counterexample relies on the fact that we are dealing with *families* $X = (X_\Sigma)_{\Sigma \in \mathit{StSpec}}$ of predicates. Due to the existential quantification over the indices $\Sigma \in \mathit{StSpec}$ it is possible to pick *different* Σ_i for each element of an ω-chain $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ so that

$f_i(x) \in X_{\Sigma_i}$, that is, the chain need not be in line with the family $(X_\Sigma)_\Sigma$. Thus, in general, there need not be any $\Sigma^\circ \in \mathit{StSpec}$ such that $f(x) \in X_{\Sigma^\circ}$ holds for the lub $f = \sqcup_i f_i$, even under the assumption that each X_Σ is closed under taking least upper bounds.

In more detail, let

$$\Sigma = l_0 : [\mathsf{m}_0 : \varsigma(x)[\mathsf{m}_1 : \varsigma(y)[]::\mathit{True}]::\mathit{True}],$$

which, informally, describes a store with a single object at location l_0 containing a method m_0. When a call of this method converges, it returns an object satisfying $[\mathsf{m}_1 : \varsigma(y)[]::\mathit{True}]$ (which is not much of a restriction). However, this resulting object has to be allocated in the store, so a proper extension of the original store specification Σ has to be found.

Next, for $i \in \mathbb{N}$ let A_i be the object specification defined inductively by

$$\begin{aligned} A_0 &= [\mathsf{m}_1 : \varsigma(y)[]::\mathit{False}] \\ A_{i+1} &= [\mathsf{m}_1 : \varsigma(y)A_i::\mathit{True}]. \end{aligned}$$

In particular, this means that the method m_1 of objects satisfying A_0 *must* diverge. The method m_1 of an object satisfying A_i returns an object satisfying A_{i-1}. Hence, for such objects x, it is possible to have method calls $x.\mathsf{m}_1.\mathsf{m}_1 \ldots \mathsf{m}_1$ at most i times, of which the i-th call must necessarily diverge (the others may or may not terminate).

The example below uses the fact that we can construct an ascending chain of objects for which the first $i-1$ calls indeed terminate, and which therefore do *not* satisfy A_{i-1}. Then, the limit of this chain is an object x for which an arbitrary number of calls $x.\mathsf{m}_1.\mathsf{m}_1 \ldots \mathsf{m}_1$ terminates, and which therefore does not satisfy *any* of the A_i: set $\Sigma_i'' = \Sigma, l : A_i$ and let $\underline{\sigma} \in [\![\Sigma]\!]$ denote some store satisfying Σ according to the tentative definition above. Moreover, define

$$\sigma_i \quad = \quad \{\!|l_0 = \{\!|\mathsf{m}_0 = \lambda\_.(l, \underline{\sigma} + \sigma_i'')|\!\}|\!\}$$

where $\sigma_0'' = \{\!|l = \{\!|\mathsf{m}_1 = \lambda\_.\bot|\!\}|\!\}$ and $\sigma_{i+1}'' = \{\!|l = \{\!|\mathsf{m}_1 = \lambda\_.(l, \underline{\sigma} + \sigma_i'')|\!\}|\!\}$, and let $\sigma = \sqcup_i \sigma_i$. Finally, define (indexed) relations $X, Y \in \mathscr{R}$ by

$$X_{\hat{\Sigma}} \quad = \quad \begin{cases} \{\underline{\sigma} + \sigma_i''\} & \text{if } \exists i \in \mathbb{N}.\, \hat{\Sigma} \equiv \Sigma_i'' \\ \varnothing & \text{otherwise} \end{cases}$$

$$Y_{\hat{\Sigma}} \quad = \quad \begin{cases} \{\underline{\sigma}\} & \text{if } \hat{\Sigma} \equiv \Sigma \\ \varnothing & \text{otherwise.} \end{cases}$$

By construction, both X and Y are admissible in every component $\hat{\Sigma}$. By induction, we get $\sigma_0'' \sqsubseteq \sigma_1'' \sqsubseteq \ldots$, and therefore $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \ldots$ in $\Phi(Y, X)_\Sigma \subseteq \mathsf{St}$. Hence we must show $\sigma \in \Phi(Y, X)_\Sigma$. But this is not the case, since it would entail, by (M2) and

$$\sigma.l_0.\mathsf{m}_0(l, \underline{\sigma}) = \bigsqcup_i \sigma_i.l_0.\mathsf{m}_0(l, \underline{\sigma}) = (l, \underline{\sigma} + \bigsqcup_i \sigma_i''),$$

that there exists $\Sigma'' \succcurlyeq \Sigma$ such that $\underline{\sigma} + \sqcup_i \sigma_i'' \in X_{\Sigma''}$. Clearly this does not hold: $\underline{\sigma} + \sqcup_i \sigma_i''$ is *strictly* above every $\underline{\sigma} + \sigma_i''$ and therefore not in any of the $X_{\Sigma_i''}$. Hence, by choice of X, there is no store specification $\Sigma'' \succcurlyeq \Sigma$ such that $\underline{\sigma} + \sigma_i'' \in X_{\Sigma''}$. This shows that $\Phi(Y, X)_\Sigma$ is not necessarily admissible, even if X (and also Y) is.

Table 12. *Store specifications*

$(\sigma,\phi)\in\Phi(Y,X)_\Sigma$	$\iff$	(Dom1)	$\mathsf{dom}(\Sigma)=\mathsf{dom}(\phi)$; and
		(Dom2)	$\forall l\in\mathsf{dom}(\Sigma).$
			if $\mathsf{dom}(\Sigma.l)\equiv[\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j:\varsigma(y_j)B_j::T_j{}^{j=1\ldots m}]$
			then $\mathsf{dom}(\phi.l)=\{\mathsf{m}_j\}_{j=1\ldots m}$; and
		$\forall l\in\mathsf{dom}(\Sigma)$ where	$\Sigma.l\equiv[\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j:\varsigma(y_j)B_j::T_j{}^{j=1\ldots m}]$:
		(F)	$\forall 1\leqslant i\leqslant n.\ \sigma.l.\mathsf{f}_i\in\|A_i\|_\Sigma$; and
		(M)	$\forall 1\leqslant j\leqslant m\ \forall\Sigma'\geqslant\Sigma\ \forall\sigma',\sigma''\in\mathsf{St}\ \forall\phi'\in\mathsf{RSF}\ \forall l'\in\mathsf{Loc}\ \forall v\in\mathsf{Val}.$
			if $l'\in\|\Sigma.l\|_{\Sigma'}\ \wedge\ (\sigma',\phi')\in Y_{\Sigma'}\ \wedge\ \sigma.l.\mathsf{m}_j(l',\sigma')=(v,\sigma'')$
			then $\exists\Sigma''\geqslant\Sigma'\ \exists\phi''\in\mathsf{RSF}.\ \phi.l.\mathsf{m}_j(l',\sigma',\phi',\Sigma')=(\Sigma'',\phi'')$ and
		(M1)	$(\pi_{\mathsf{Val}}(\sigma'),v,\pi_{\mathsf{Val}}(\sigma''))\in[\![T_j[l'/y_j]]\!]$; and
		(M2)	$(\sigma'',\phi'')\in X_{\Sigma''}$; and
		(M3)	$v\in\|B_j[l'/y_j]\|_{\Sigma''}$

4.3. *A refined semantics of store specifications*

We refine the definition of store predicates by replacing the existential quantifier in condition (M2) of the functional Φ from Table 11 by a choice function, as follows: we call the elements of the (recursively defined) domain

$$\phi\in\mathsf{RSF}=\mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{M}}(\mathsf{St}\times\mathsf{RSF}\times \mathit{StSpec}\rightharpoonup \mathit{StSpec}\times\mathsf{RSF})) \tag{8}$$

(records of) *choice functions*. The intuition is that, given a store $\sigma\in[\![\Sigma]\!]$ and some extension $\Sigma'\geqslant\Sigma$, if $\sigma'\in[\![\Sigma']\!]$ with choice function ϕ' and the method invocation $\sigma.l.\mathsf{m}(\sigma')$ terminates, then $\phi.l.\mathsf{m}(\sigma',\phi',\Sigma')=(\Sigma'',\phi'')$ yields a store specification $\Sigma''\geqslant\Sigma'$ such that $\sigma''\in[\![\Sigma'']\!]$ (and ϕ'' is a choice function for the extension Σ'' of Σ). This is again an abstraction of the actual store σ, this time abstracting the *dynamic effects* of methods with respect to allocation, on the level of store specifications. Note that the argument store σ' is needed in general to determine the resulting extension of the specification, since allocation behaviour may depend on the actual values of fields.

We use the domain RSF of choice functions explicitly in the interpretation of store specifications below. This has the effect of constraining the existential quantifier to work *uniformly* on the elements of increasing chains, hence precluding the counterexample to admissibility of the previous subsection.

Definition 4.6 (Store predicate, revisited). Let $\mathscr{S}=\mathscr{P}(\mathsf{St}\times\mathsf{RSF})^{\mathit{StSpec}}$ denote the collection of families of subsets of $\mathsf{St}\times\mathsf{RSF}$, indexed by store specifications. Table 12 defines a functional $\Phi:\mathscr{S}^{op}\times\mathscr{S}\to\mathscr{S}$; we write $\sigma\in[\![\Sigma]\!]$ if there is some $\phi\in\mathsf{RSF}$ such that $(\sigma,\phi)\in\mathrm{fix}(\Phi)_\Sigma$.

Lemma 4.7 (Existence). Functional Φ, defined in Definition 4.6, does have a unique fixed point.

Proof. First, we show that Φ is monotonic and maps admissible predicates to admissible predicates, in the sense that for all X and Y,

$$\begin{aligned}&\forall\Sigma\in\mathit{StSpec}.\ X_\Sigma\subseteq\mathsf{St}\times\mathsf{RSF}\text{ admissible}\\ &\quad\Longrightarrow\quad\forall\Sigma\in\mathit{StSpec}.\ \Phi(Y,X)_\Sigma\subseteq\mathsf{St}\times\mathsf{RSF}\text{ admissible.}\end{aligned} \tag{9}$$

Indeed, if $(\sigma_0, \phi_0) \sqsubseteq (\sigma_1, \phi_1) \sqsubseteq \dots$ is a chain in $\Phi(Y, X)_\Sigma$, then $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \dots$ in St and $\phi_0 \sqsubseteq \phi_1 \sqsubseteq \dots$ in RSF. Let $\sigma = \sqcup_k \sigma_k$ and $\phi = \sqcup_k \phi_k$ (so $(\sigma, \phi) = \sqcup_k(\sigma_k, \phi_k)$). We show $(\sigma, \phi) \in \Phi(Y, X)_\Sigma$ under the assumption that $X_{\Sigma'}$ is admissible for all $\Sigma' \in \mathit{StSpec}$.

Clearly, conditions (Dom1) and (Dom2) of Definition 4.6 are satisfied. As for (F) and (M), suppose $l \in \mathsf{dom}(\Sigma)$ with $\Sigma.l = [\mathsf{f}_i : A_i{}^{i=1\dots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\dots m}]$. Since, for all $1 \leqslant i \leqslant n$,

$$\sigma_0.l.\mathsf{f}_i = \sigma_1.l.\mathsf{f}_i = \dots = \sigma.l.\mathsf{f}_i,$$

and we obtain $\sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma$ by the assumption that $(\sigma_0, \phi_0) \in \Phi(Y, X)_\Sigma$, showing (F). To prove (M), suppose $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in Y_{\Sigma'}$ and $\sigma.l.\mathsf{m}_j(\sigma') = (v, \sigma'')\downarrow$ for some $v \in \mathsf{Val}$ and $\sigma'' \in \mathsf{St}$. By definition of σ as $\sqcup_k \sigma_k$ and continuity of $\sigma.l.\mathsf{m}_j$, there must be $\sigma''_k \in \mathsf{St}$ and $\sigma_k.l.\mathsf{m}_j(\sigma') = (v, \sigma''_k)\downarrow$ for sufficiently large k, and

$$(v, \sigma'') = \bigsqcup_k \sigma_k.l.\mathsf{m}_j(\sigma') = \bigsqcup_k (v, \sigma''_k).$$

By assumption, $(\sigma_k, \phi_k) \in \Phi(Y, X)_\Sigma$, so for all sufficiently large k, $\phi_k.l.\mathsf{m}_j(\sigma', \phi', \Sigma') = (\Sigma''_k, \phi''_k)$ with $\Sigma''_k \succcurlyeq \Sigma'$ and:

— $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma''_k)) \in [\![T_j[l/y_j]]\!]$;
— $(\sigma''_k, \phi''_k) \in X_{\Sigma''_k}$; and
— $v \in \|B_j[l/y_j]\|_{\Sigma''_k}$.

Since $\pi_{\mathrm{Val}}(\sigma''_k) = \pi_{\mathrm{Val}}(\sigma'')$, condition (M1) follows. The discrete order on *Spec* entails $\Sigma''_k \equiv \Sigma''_{k+1} \equiv \dots$, so, $\phi(\sigma', \phi', \Sigma') = \sqcup(\Sigma''_k, \phi''_k) = (\Sigma'', \sqcup_k \phi''_k)$ with $\Sigma'' \equiv \Sigma''_k \equiv \Sigma''_{k+1} \equiv \dots$, and (M3) is clearly satisfied. By assumption, $X_{\Sigma''}$ is admissible, so condition (M2) also holds, as required, that is, $(\sigma'', \phi'') = \sqcup(\sigma''_k, \phi''_k) \in X_{\Sigma''}$. This proves claim (9).

Next, define, for all admissible $X, X' \in \mathscr{S}$ and $e = (e_1, e_2) \in [\mathsf{St} \rightharpoonup \mathsf{St}] \times [\mathsf{RSF} \rightharpoonup \mathsf{RSF}]$,

$$\begin{aligned} e : X \subset X' \quad \iff \quad & \forall \Sigma \in \mathit{StSpec}\ \ \forall \sigma \in \mathsf{St}\ \ \forall \phi \in \mathsf{RSF}. \\ & (\sigma, \phi) \in X_\Sigma \ \wedge\ (e_1(\sigma)\downarrow\ \vee e_2(\phi)\downarrow) \\ & \quad \implies e_1(\sigma)\downarrow\ \wedge e_2(\phi)\downarrow\ \wedge (e_1(\sigma), e_2(\phi)) \in X'_\Sigma \end{aligned}$$

such that $e : X \subset X'$ states that e maps pairs of stores and choice functions that are in X_Σ to pairs of stores and choice functions that are in the corresponding component X'_Σ of X'. Let F_{Store} be the locally continuous, mixed-variant functor associated with the domain equations (1), for which $F_{\mathit{Store}}(\mathsf{St}, \mathsf{St}) = \mathsf{St}$, and consider the locally continuous functor $F_{\mathsf{St},\mathsf{RSF}}(R, S) : (\mathbf{pCpo} \times \mathbf{pCpo})^{op} \times \mathbf{pCpo} \times \mathbf{pCpo} \to \mathbf{pCpo} \times \mathbf{pCpo}$

$$\begin{aligned} F_{\mathsf{St},\mathsf{RSF}}(R, S) = & \langle F_{\mathit{Store}}(\Pi_1(R), \Pi_1(S)), \\ & \qquad \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{M}}(\Pi_1(R) \times \Pi_2(R) \times \mathit{StSpec} \rightharpoonup \mathit{StSpec} \times \Pi_2(S)))\rangle \end{aligned}$$

where Π_i is the projection to the i-th component. Hence $(\mathsf{St}, \mathsf{RSF})$ is the minimal invariant for $F_{\mathsf{St},\mathsf{RSF}}$. In the following, we write F_{St} for the functor $\Pi_1 \circ F_{\mathsf{St},\mathsf{RSF}}$ and F_{RSF} for $\Pi_2 \circ F_{\mathsf{St},\mathsf{RSF}}$. By the results of Pitts (1996), all that remains is to show that

$$e : X \subset X' \ \wedge\ e : Y' \subset Y \quad \implies \quad F_{\mathsf{St},\mathsf{RSF}}(e, e) : \Phi(Y, X) \subset \Phi(Y', X') \qquad (\dagger)$$

for all $X, Y, X', Y' \in \mathscr{S}$ and $e = (e_1, e_2)$ with $e_1 \sqsubseteq id_{\mathsf{St}}$ and $e_2 \sqsubseteq id_{\mathsf{RSF}}$, which follows from a similar line of reasoning to that in Reus and Streicher (2004). Suppose $e = (e_1, e_2)$ such that $e_1 \sqsubseteq id_{\mathsf{St}}$, $e_2 \sqsubseteq id_{\mathsf{RSF}}$ and

$$e : X \subset X' \quad \wedge \quad e : Y' \subset Y \tag{10}$$

for some $X, Y, X', Y' \in \mathscr{S}$. Assume $(\sigma, \phi) \in \Phi(Y, X)_\Sigma$. We must show $F_{\mathsf{St,RSF}}(e, e)(\sigma, \phi) \in \Phi(Y', X')_\Sigma$ to prove (†). Recall that

$$F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{f} = \sigma.l.\mathsf{f} \tag{11a}$$

$$F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{m}(\sigma') = (id_{\mathsf{Val}} \times e_1)(\sigma.l.\mathsf{m}(e_1(\sigma'))) \tag{11b}$$

$$F_{\mathsf{RSF}}(e,e)(\sigma,\phi).l.\mathsf{m}(\sigma',\phi',\Sigma') = (id_{StSpec} \times e_2)(\phi.l.\mathsf{m}(e_1(\sigma'), e_2(\phi'), \Sigma')) \tag{11c}$$

for all $\mathsf{f} \in \mathscr{F}$ and $\mathsf{m} \in \mathscr{M}$. In particular, conditions (DOM1) and (DOM2) of Definition 4.6 are immediately seen to be satisfied for $F_{\mathsf{St,RSF}}(e,e)(\sigma,\phi)$.

To show (F) and (M), let $l \in \mathsf{dom}(\Sigma)$ and $\Sigma.l \equiv [\mathsf{f}_i\!: A_i{}^{i=1\ldots n}, \mathsf{m}_j\!: \varsigma(y_j)B_j\!::\!T_j{}^{j=1\ldots m}]$. From $(\sigma,\phi) \in \Phi(Y,X)_\Sigma$ and (11a), we obtain for all $1 \leqslant i \leqslant n$

(F) $\quad F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{f}_i \in \|A_i\|_\Sigma$

We still need to check conditions (M1)–(M3) of Definition 4.6. Let $1 \leqslant j \leqslant m$. Suppose $\Sigma' \succcurlyeq \Sigma$, $\phi' \in \mathsf{RSF}$ and $\sigma' \in \mathsf{St}$ with $(\sigma', \phi') \in Y'_{\Sigma'}$ and such that $F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{m}_j(\sigma')\downarrow$. Thus, by (11b), we know that

$$F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{m}_j(\sigma') = (v, e_1(\sigma''))$$
$$\text{where } (v, \sigma'') = \sigma.l.\mathsf{m}_j(e_1(\sigma'))$$

for some $v \in \mathsf{Val}$ and $\sigma'' \in \mathsf{St}$. By (10), assumption $(\sigma', \phi') \in Y'_{\Sigma'}$ shows $e(\sigma', \phi') = (e_1(\sigma'), e_2(\phi')) \in Y_{\Sigma'}$. Together with the assumption $(\sigma, \phi) \in \Phi(Y, X)_\Sigma$ and (11c), this entails

$$F_{\mathsf{RSF}}(e,e)(\phi).l.\mathsf{m}_j(\sigma',\phi',\Sigma') = (\Sigma'', e_2(\phi''))$$
$$\text{where } (\Sigma'', \phi'') = \phi.l.\mathsf{m}_j(e_1(\sigma'), e_2(\phi'), \Sigma')$$

for $\phi'' \in \mathsf{RSF}$ and $\Sigma'' \succcurlyeq \Sigma'$ such that:

(M1′) $\quad (\pi_{\mathrm{Val}}(e_1(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T[l/y_j]]\!]$.
(M2′) $\quad (\sigma'', \phi'') \in X_{\Sigma''}$.
(M3) $\quad v \in \|B_j[l/y_j]\|_{\Sigma''}$.

Since, by assumption, $e_1 \sqsubseteq id_{\mathsf{St}}$, we know $e_1(\sigma'') \sqsubseteq \sigma''$, and, in particular, $\pi_{\mathrm{Val}}(e_1(\sigma'')) = \pi_{\mathrm{Val}}(\sigma'')$. Similarly, for σ', $\pi_{\mathrm{Val}}(e_1(\sigma')) = \pi_{\mathrm{Val}}(\sigma')$ since $\sigma' \sqsupseteq e_1(\sigma')$. Hence, (M1′) entails $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(e_1(\sigma''))) \in [\![T[l/y_j]]\!]$, that is, (M1) holds. Finally, assumption (10) and (M2′) give $(e_1(\sigma''), e_2(\phi'')) \in X'_{\Sigma''}$, which shows (M2), and we have proved (†). □

Note that our proof of property (†) relies on the fact that the predicates denoting transition specifications are upward-closed in the pre-execution store and downward-closed in the post-execution store. This holds in Abadi–Leino logic as transition specifications are only defined on the flat part of the store. If they referred to the method part, (†) could not necessarily be shown, unless one finds an appropriate way to restrict the reference to methods in transition specifications. See Reus and Streicher (2004) for more discussion and some suggestions for how to lift this restriction.

5. Soundness

In this section we present a new soundness proof of Abadi and Leino's logic using denotational semantics. Before we can embark on this endeavour, we need to define the semantics of judgements $\Gamma \vdash a : A :: T$, which provides a notion of *validity* for those judgements. We write $\Gamma \models a : A :: T$ for the semantics of judgement $\Gamma \vdash a : A :: T$. *Soundness* means that every judgement that is derivable is indeed valid.

The definition of validity has to be assembled in a compositional way from the semantics of the constituents of the judgement, which were discussed in the previous sections. The definition must also take into consideration the fact that the initial store in which the program a is executed may contain methods that are called by a. Store specifications have been introduced exactly for this purpose. The context Γ specifies objects that are referred to by variables of the environment, but which actually lie in the store. This means context specification and store specification are intertwined. Because of Definition 4.4, context specifications can be interpreted with respect to store specifications Σ, which resolves this entanglement, so we obtain the following definition of validity.

Definition 5.1 (Validity). $\Gamma \models a : A :: T$ if and only if for all store specifications $\Sigma \in \mathit{StSpec}$, for all $\rho \in \|\Gamma\|_\Sigma$ and all $\sigma \in [\![\Sigma]\!]$, if $[\![a]\!]\rho\sigma = (v, \sigma')$, then $(v, \sigma') \in [\![[\Gamma] \vdash A]\!]\rho$ and $(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![[\Gamma] \vdash T]\!]\rho$.

Thus $\Gamma \models a : A :: T$ states the following: suppose program $[\![a]\!]$ returns (v, σ) when run in a store σ that satisfies some store specification Σ (and therefore does not contain any code that violates the restrictions needed for the logic, see also the discussion in Section 4) and in an environment ρ that satisfies Γ with respect to σ (expressed via reference to Σ). Then (v, σ) satisfies the specification $[\![[\Gamma] \vdash A]\!]\rho$ and the state transformation provoked satisfies $[\![[\Gamma] \vdash T]\!]\rho$.

The rest of this section is dedicated to a proof of the soundness theorem (Theorem 5.5), which relies on a number of properties that are derived in the following.

Recall from the previous section that the semantics of store specifications is defined in terms of the semantics $\|A\|_\Sigma$ for result specifications A without any reference to the object specification $[\![A]\!]$. The following key lemma establishes the relation between the store specifications of Section 4 and the object specifications $[\![A]\!]$ as defined in Section 3.3.

Lemma 5.2. *For all object specifications A, store specifications Σ, stores σ and locations l, if $\sigma \in [\![\Sigma]\!]$ and $l \in \mathsf{dom}(\Sigma)$ such that $\vdash \Sigma.l <: A$, then $(l, \sigma) \in [\![A]\!]$.*

Proof. We use induction on the structure of A. Because A is an object specification, it is necessarily of the form

$$A \;\equiv\; [\mathsf{f}_i : A_i{}^{i=1\ldots n},\ \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}].$$

We have to show that $(l, \sigma) \in [\![A]\!]$, that is, that

— $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![A_i]\!]$ for all $1 \leqslant i \leqslant n$; and

— if $\sigma.l.\mathsf{m}_j(\sigma) = (v, \sigma')$, then $(v, \sigma') \in [\![y_j \vdash B_j]\!](y_j \mapsto l)$ and $(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![y_j \vdash T_j]\!](y_j \mapsto l)$ for all $1 \leqslant j \leqslant m$.

From the subtyping relation and $\Sigma.l <: A$, we find

$$\Sigma.l \quad \equiv \quad [\mathsf{f}_i{:}A_i'^{\,i=1\ldots n+p}, \mathsf{m}_j{:}\varsigma(y_j)B_j'{::}T_j'^{\,j=1\ldots m+p}]$$

where $y_j \vdash B_j' <: B_j$ and $y_j \vdash_{\mathsf{fo}} T_j' \to T_j$.

For the first part, by Definition 4.6 (F) and $\sigma \in [\![\Sigma]\!]$, we have $\sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma$. If A_i is *Bool*, then $\|A_i\|_\Sigma = \mathsf{BVal}$, hence, $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![Bool]\!] = [\![A_i]\!]$. Otherwise, A_i is an object specification and the definition of $\|A_i\|_\Sigma$ implies $\vdash \Sigma.(\sigma.l.\mathsf{f}_i) <: A_i$, again by Definition 4.6 (F). Hence, by the induction hypothesis, we obtain $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![A_i]\!]$, as required.

For the second part, suppose that $\sigma.l.\mathsf{m}_j(\sigma) = (v, \sigma'')$. From Definition 4.6 (M2) and (M3), and the assumption $\sigma \in [\![\Sigma]\!]$, we find $v \in \left\|B_j'[l/y_j]\right\|_{\Sigma''}$ and $\sigma'' \in [\![\Sigma'']\!]$ for some $\Sigma'' \succcurlyeq \Sigma$. In the case where B_j is *Bool*, we therefore have $v \in \mathsf{BVal}$ and obtain

$$(v, \sigma'') \in [\![Bool]\!] = [\![y_j \vdash Bool]\!](y_j \mapsto l).$$

Next, if B_j is an object specification, by the definition of $\left\|B_j'[l/y_j]\right\|_{\Sigma''}$, we have

$$\vdash \Sigma''.v <: B_j'[l/y_j].$$

By the induction hypothesis (applied to $B_j'[l/y_j]$, Σ'', σ'' and v), $(v, \sigma'') \in [\![B_j'[l/y_j]]\!]$. Thus,

$$\begin{aligned}(v, \sigma'') \in [\![B_j'[l/y_j]]\!] &= [\![y_j \vdash B_j']\!](y_j \mapsto l) && \text{by substitution lemma (Lemma 4.2)}\\ &\subseteq [\![y_j \vdash B_j]\!](y_j \mapsto l) && \text{by subtype soundness (Lemma 3.5),}\end{aligned}$$

as required.

Finally, by Definition 4.6 (M1), we obtain

$$\begin{aligned}(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T_j'[l/y_j]]\!] &= [\![y_j \vdash T_j']\!](y_j \mapsto l) && \text{by substitution (Lemma 4.2)}\\ &\subseteq [\![y_j \vdash T_j]\!](y_j \mapsto l) && \text{by soundness of } \vdash_{\mathsf{fo}}.\end{aligned}$$

This concludes the proof. □

Before proving the main technical result in Lemma 5.4, we state the following fact about the transition relation that appears in the let rule.

Lemma 5.3. Suppose that for $\sigma, \sigma', \sigma'' \in \mathsf{St}$ and $v, v' \in \mathsf{Val}$, we have

$$(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![\overline{x} \vdash T']\!]\rho$$

and

$$(\pi_{\mathrm{Val}}(\sigma'), v', \pi_{\mathrm{Val}}(\sigma'')) \in [\![\overline{x}, x \vdash T'']\!]\rho[x := v]$$

Then, if $\overline{x} \vdash T$ and

$$\begin{aligned}&\vdash_{\mathsf{fo}} T'[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}]\\ &\quad \wedge\, T''[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)] \to T\end{aligned} \tag{12}$$

we have $(\pi_{\mathrm{Val}}(\sigma), v', \pi_{\mathrm{Val}}(\sigma'')) \in [\![\overline{x} \vdash T]\!]\rho$.

Proof. Consider an extended signature of transition relations, with additional function and predicate symbols $\mathsf{sel}_{int}(\cdot,\cdot)$ and $\mathsf{alloc}_{int}(\cdot)$, respectively. We extend the interpretation of transition relations (and expressions) in the natural way to operate on three stores,

$$[\![x_1,\ldots,x_k \vdash T]\!]\rho : \mathscr{P}(\mathsf{St}_{\mathsf{Val}} \times \mathsf{Val} \times \mathsf{St}_{\mathsf{Val}} \times \mathsf{St}_{\mathsf{Val}})$$

where the second store argument is used to interpret $\mathsf{sel}_{int}(\cdot,\cdot)$ and $\mathsf{alloc}_{int}(\cdot)$:

$$(\sigma,v,\hat{\sigma},\sigma') \in [\![\overline{x} \vdash \mathsf{alloc}_{int}(e)]\!]\rho \quad\Longleftrightarrow\quad [\![\overline{x} \vdash e]\!]\rho\sigma v\hat{\sigma}\sigma' \downarrow \;\wedge\; [\![\overline{x} \vdash e]\!]\rho\sigma v\hat{\sigma}\sigma' \in \mathsf{dom}(\hat{\sigma})$$

and

$$[\![\overline{x} \vdash \mathsf{sel}_{int}(e_0,e_1)]\!]\rho\sigma v\hat{\sigma}\sigma' \;=\; \begin{cases} \hat{\sigma}.l.\mathsf{f} & \text{if } [\![\overline{x} \vdash e_0]\!]\rho\sigma v\hat{\sigma}\sigma' = l \in \mathsf{Loc} \\ & \text{and } [\![\overline{x} \vdash e_1]\!]\rho\sigma v\hat{\sigma}\sigma' = \mathsf{f} \in \mathscr{F} \\ & \text{and } l \in \mathsf{dom}(\hat{\sigma}) \text{ and } \mathsf{f} \in \mathsf{dom}(\hat{\sigma}.l) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

By assumption and using the fact that neither T' nor T'' contains the new predicates, we also have

$$(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma'), \pi_{\mathrm{Val}}(\sigma')) \in [\![\overline{x}, x \vdash T']\!]\rho[x := v], \quad \text{and}$$
$$(\pi_{\mathrm{Val}}(\sigma'), v', \pi_{\mathrm{Val}}(\sigma'), \pi_{\mathrm{Val}}(\sigma'')) \in [\![\overline{x}, x \vdash T'']\!]\rho[x := v].$$

Thus,

$$(\pi_{\mathrm{Val}}(\sigma), v', \pi_{\mathrm{Val}}(\sigma'), \pi_{\mathrm{Val}}(\sigma'')) \in$$
$$[\![\overline{x}, x \vdash T'[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}]]\!]\rho[x := v]$$

since there are no occurrences of $\mathsf{sel}_{post}(\cdot,\cdot)$, $\mathsf{alloc}_{post}(\cdot)$ and result, and

$$(\pi_{\mathrm{Val}}(\sigma), v', \pi_{\mathrm{Val}}(\sigma'), \pi_{\mathrm{Val}}(\sigma'')) \in$$
$$[\![\overline{x}, x \vdash T''[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)]]\!]\rho[x := v]$$

since there are no occurrences of $\mathsf{sel}_{pre}(\cdot,\cdot)$ and $\mathsf{alloc}_{pre}(\cdot)$. From soundness of first-order provability and assumption (12), we obtain

$$(\pi_{\mathrm{Val}}(\sigma), v', \pi_{\mathrm{Val}}(\sigma'), \pi_{\mathrm{Val}}(\sigma'')) \in [\![\overline{x}, x \vdash T]\!]\rho[x := v],$$

and the result follows since T does not depend on x or the new predicates, by $\overline{x} \vdash T$. □

5.1. *The invariance lemma*

In this subsection we state and prove the main lemma of the soundness proof. Intuitively, it shows that store specifications Σ are 'invariant' under proved programs,

$$\sigma \in [\![\Sigma]\!] \;\;\wedge\;\; [\![a]\!]\rho\sigma = (v,\sigma') \quad\Longrightarrow\quad \exists\,\Sigma' \succcurlyeq \Sigma.\ \sigma' \in [\![\Sigma']\!]. \tag{13}$$

Note that the program a will in general allocate further objects, so the resulting store only satisfies an extension of the original store specification. The precise conditions of when (13) holds are given in the statement of the following lemma, and take the choice functions $\phi \in \mathsf{RSF}$ introduced in Section 4 into account. We write SF for the domain of 'individual' choice functions,

$$\mathsf{SF} = [\mathsf{St} \times \mathsf{RSF} \times \mathit{StSpec} \rightharpoonup \mathit{StSpec} \times \mathsf{RSF}]$$

for which $\mathsf{RSF} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathscr{M}}(\mathsf{SF}))$.

Lemma 5.4. Suppose:

(H1) $\Gamma \vdash a : A :: T$.

(H2) $\Sigma \in \mathit{StSpec}$ is a store specification.

(H3) $\rho \in \|\Gamma\|_{\Sigma}$.

Then there exists $\phi \in \mathsf{SF}$ such that, for all $\Sigma' \succcurlyeq \Sigma$ and for all $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$, if $[\![a]\!]\rho\sigma' = (v, \sigma'')\downarrow$, the following hold:

(S1) $\exists \Sigma'' \succcurlyeq \Sigma' \; \exists \phi'' \in \mathsf{RSF}.\;\; \phi(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$.

(S2) $(\sigma'', \phi'') \in \mathit{fix}(\Phi)_{\Sigma''}$.

(S3) $v \in \|A[\rho/\Gamma]\|_{\Sigma''}$.

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma] \vdash T]\!]\rho$.

Proof. The proof is by induction on the derivation of $\Gamma \vdash a : A :: T$.

— Lemma 4.5 is applied in the cases Let and Object Construction, where an extended specification context is used in the induction hypothesis.

— Invariance of subspecifications in field specifications is needed in the case for Field Update.

— In the cases where the store changes, that is, Object Construction and Field Update, we must show explicitly that the resulting store satisfies the store specification, according to Definition 4.6. In this case one makes use of the fact that methods in the semantics of store specifications are specified using arbitrary store arguments that recursively fulfil the same store specification. This provides a sufficiently strong hypothesis to deal with the changed store.

We consider cases, according to the last rule applied in the derivation of the judgement $\Gamma \vdash a : A :: T$.

— Subsumption

Suppose that $\Gamma \vdash a : A :: T$ has been obtained by an application of the subsumption rule, and that:

(H2) Σ is a store specification.

(H3) $\rho \in \|\Gamma\|_{\Sigma}$.

We have to show that there is $\phi \in \mathsf{SF}$ such that whenever $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and $[\![a]\!]\rho\sigma' = (v, \sigma')$, we have that (S1)–(S4) hold.

Recalling the subsumption rule,

$$\frac{[\Gamma] \vdash A' <: A \quad \Gamma \vdash a{:}A'{::}T' \quad [\Gamma] \vdash A \quad [\Gamma] \vdash T \quad \vdash_{\mathsf{fo}} T' \to T}{\Gamma \vdash a{:}A{::}T},$$

we must have $\Gamma \vdash a : A' :: T'$ for some specification A' and transition relation T' with $\vdash_{\mathsf{fo}} T' \to T$ and $[\Gamma] \vdash A' <: A$. By the induction hypothesis, there exists $\phi \in \mathsf{SF}$ such that for all $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ with $[\![a]\!]\rho\sigma' = (v, \sigma')$:

(S1) There exists $\Sigma'' \succcurlyeq \Sigma'$ and $\phi'' \in \mathsf{RSF}$ such that $\phi(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$.

(S2) $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$.

(S3$'$) $v \in \|A'[\rho/\Gamma]\|_{\Sigma'}$.

(S4$'$) $(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![[\Gamma] \vdash T']\!]\rho$.

Because $\vdash_{\mathsf{fo}} T' \to T$, we know $[\![\Gamma \vdash T']\!]\rho \subseteq [\![\Gamma \vdash T]\!]\rho$, and therefore (S4′) implies

$$(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![\Gamma \vdash T]\!]\rho. \tag{S4}$$

We still need to show

$$v \in \|A[\rho/\Gamma]\|_{\Sigma'}. \tag{S3}$$

Note that by the subtyping rules, $A \equiv \mathit{Bool}$ if and only if $A' \equiv \mathit{Bool}$. In this case, (S3) follows directly from (S3′). In the case where A' is an object specification, assumption $[\Gamma] \vdash A' <: A$ and Lemma 4.2 entail $\vdash A'[\rho/\Gamma] <: A[\rho/\Gamma]$. Transitivity of $<:$ and (S3′) then prove (S3), by the definition of $\|A'[\rho/\Gamma]\|_{\Sigma'}$.

— Var

Suppose $\Gamma \vdash a : A :: T$ has been derived by an application of the Var rule

$$\frac{\Gamma \vdash \mathsf{ok} \quad x{:}A \text{ in } \Gamma}{\Gamma \vdash x{:}A{::}T_{\mathsf{res}}(x)},$$

for which we see that $a \equiv x$, $x{:}A$ in Γ, and $T \equiv T_{\mathsf{res}}(x)$. Further, assume:

(H2) $\Sigma \in \mathit{StSpec}$.

(H3) $\rho \in \|\Gamma\|_{\Sigma}$.

Define the (partial continuous) map $\phi \in \mathsf{SF}$ by

$$\phi(\sigma', \phi', \Sigma') = (\Sigma', \phi').$$

Now suppose $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and $[\![a]\!]\rho\sigma' = (v, \sigma'')$. By the assumptions and the semantics of variables, we obtain $(v, \sigma'') = (\rho(x), \sigma')$, that is,

$$v = \rho(x) \quad \wedge \quad \sigma'' = \sigma'. \tag{14}$$

By the definition of ϕ above, we can choose Σ'' as Σ', since then, as required:

(S1) $\phi(\sigma', \phi', \Sigma') = (\Sigma', \phi')$.

(S2) $(\sigma'', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$, by $\sigma'' = \sigma'$ and assumption $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$.

(S3) $v \in \|A[\rho/\Gamma]\|_{\Sigma'}$, by $v = \rho(x)$, by $x{:}A \in \Gamma$ and (H3).

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma] \vdash T_{\mathsf{res}}(x)]\!]\rho$, by the definition of $[\![[\Gamma] \vdash T]\!]$ in Table 8 and (14).

— Const

This is similar to the previous case.

— Conditional

We use case distinction, depending on whether the value of the guard x is *true* or *false*.

— Let

Suppose (H1) $\Gamma \vdash a : A :: T$ has been derived by an application of the Let rule. Hence, a is `let` $x = a_1$ `in` a_2. Assume that:

(H2) $\Sigma \in \mathit{StSpec}$ is a store specification.

(H3) $\rho \in \|\Gamma\|_{\Sigma}$.

Now recall the rule for this case,

$$\frac{\begin{array}{c}\Gamma \vdash a_1{:}A_1{::}T_1 \quad \Gamma, x{:}A_1 \vdash a_2{:}A{::}T_2 \quad [\Gamma] \vdash A \quad [\Gamma] \vdash T \\ \vdash_{\mathsf{fo}}\ T_1[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}] \\ \wedge\ T_2[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)] \rightarrow T\end{array}}{\Gamma \vdash \mathtt{let}\ x = a_1\ \mathtt{in}\ a_2{:}A{::}T}$$

By the premiss of this rule, we must have:

(H1′) $\Gamma \vdash a_1 : A_1 :: T_1$.

(H1″) $\Gamma, x{:}A_1 \vdash a_2 : A :: T_2$.

By the induction hypothesis applied to (H1′), there is $\phi_1 \in \mathsf{SF}$ such that, for all $\Sigma' \succcurlyeq \Sigma$ and $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ with $[\![a_1]\!]\rho\sigma' = (\hat{v}, \hat{\sigma})$, the conclusions of the lemma hold:

(S1′) there exists $\hat{\Sigma} \succcurlyeq \Sigma'$ and $\hat{\phi} \in \mathsf{RSF}$ such that $\phi_1(\sigma', \phi', \Sigma') = (\hat{\Sigma}, \hat{\phi})$.

(S2′) $(\hat{\sigma}, \hat{\phi}) \in \mathrm{fix}(\Phi)_{\hat{\Sigma}}$.

(S3′) $\hat{v} \in \|A_1[\rho/\Gamma]\|_{\hat{\Sigma}}$.

(S4′) $(\pi_{\mathrm{Val}}(\sigma'), \hat{v}, \pi_{\mathrm{Val}}(\hat{\sigma})) \in [\![[\Gamma] \vdash T_1]\!]\rho$.

In particular, by conclusion (S3′) and context extension (Lemma 4.5),

$$\rho[x := \hat{v}] \in \|\Gamma, x{:}A_1\|_{\hat{\Sigma}}\,.$$

Therefore, by the induction hypothesis applied to (H1″), there is $\phi_{\hat{\Sigma}\hat{v}} \in \mathsf{SF}$ such that, for all $\Sigma' \succcurlyeq \hat{\Sigma}$ and all $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ with $[\![a_2]\!]\rho[x := \hat{v}]\sigma' = (v, \sigma'')$, the following hold:

(S1″) There exists $\Sigma'' \succcurlyeq \Sigma'$ and $\phi'' \in \mathsf{RSF}$ such that $\phi_{\hat{\Sigma}\hat{v}}(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$.

(S2″) $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$.

(S3″) $v \in \|A[\rho[x := \hat{v}]/\Gamma, x{:}A_1]\|_{\Sigma''}$.

(S4″) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma, x{:}A_1] \vdash T_2]\!]\rho$.

Now define $\phi \in \mathsf{SF}$ for all σ', ϕ' and Σ' by

$$\phi(\sigma', \phi', \Sigma') = \begin{cases} \phi_{\hat{\Sigma}\hat{v}}(\hat{\sigma}, \hat{\phi}, \hat{\Sigma}) & \text{if } [\![a_1]\!]\rho\sigma' = (\hat{v}, \hat{\sigma})\ \wedge\ \phi_1(\sigma', \phi', \Sigma') = (\hat{\Sigma}, \hat{\phi}) \\ \text{undefined} & \text{otherwise,} \end{cases}$$

which is continuous as all its constituents are, and since Val and *StSpec* are flat.

We show that the conclusion of the lemma holds. Let $\Sigma' \succcurlyeq \Sigma$ and $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$, and suppose $[\![a]\!]\rho\sigma' = (v, \sigma'')$. From the definition of the semantics,

$$(v, \sigma'') \quad = \quad \mathbf{let}\ \ (\hat{v}, \hat{\sigma}) = [\![a_1]\!]\rho\sigma'\ \ \mathbf{in}\ \ [\![a_2]\!]\rho[x := \hat{v}]\hat{\sigma},$$

which shows:

– $[\![a_1]\!]\rho\sigma' = (\hat{v}, \hat{\sigma})$.

– $[\![a_2]\!]\rho[x := \hat{v}]\hat{\sigma} = (v, \sigma'')$.

We now show that ϕ defined above does fulfill the necessary requirements:

(S1) There is $\Sigma'' \in \mathit{StSpec}$ such that $\Sigma'' \succcurlyeq \hat{\Sigma} \succcurlyeq \Sigma'$ and $\phi(\sigma', \phi', \Sigma') = \phi_{\hat{\Sigma}\hat{v}}(\hat{\sigma}, \hat{\phi}, \hat{\Sigma}) = (\Sigma'', \phi'')$, where $\phi_1(\sigma', \phi', \Sigma') = (\hat{\Sigma}, \hat{\phi})$, by (S1′) and (S1″).

(S2) $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$, by (S2′) and (S2″).

(C3) $v \in \|A[\rho[x := \hat{v}]/\Gamma, x{:}A_1]\|_{\Sigma''}$, by (S3′) and (S3″).

(C4′) $(\pi_{\mathrm{Val}}(\sigma'), \hat{v}, \pi_{\mathrm{Val}}(\hat{\sigma})) \in [\![[\Gamma] \vdash T_1]\!]\rho$, by (S4′).

(C4″) $(\pi_{\mathrm{Val}}(\hat{\sigma}), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma, x{:}A_1] \vdash T_2]\!]\rho[x := \hat{v}]$, by (S4″).

Since $[\Gamma] \vdash A$, that is, x is not free in A, we have

$$A[\rho[x := v]/(\Gamma, x{:}A_1)] \quad \equiv \quad A[\rho/\Gamma]. \tag{15}$$

Moreover, (C4′), (C4″), Lemma 5.3 and

$$\begin{aligned}\vdash_{\mathsf{fo}} T_1[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}]\\ \wedge\ T_2[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)] \to T\end{aligned}$$

prove

$$(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![\Gamma \vdash T]\!]\rho. \tag{16}$$

We therefore obtain, as required:

(S3) $v \in \|A[\rho/\Gamma]\|_{\Sigma'}$, by (C3) and (15) as Σ'' extends Σ'.

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![\Gamma \vdash T]\!]\rho$ by (16).

— Object Construction

Suppose (H1): $\Gamma \vdash a : A :: T$ has been derived by an application of the Object Construction rule. Necessarily, $a \equiv [\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j = \varsigma(y_j)b_j{}^{j=1\ldots m}]$. Suppose that:

(H2) $\Sigma \in \mathit{StSpec}$.

(H3) $\rho \in \|\Gamma\|_\Sigma$.

Recalling the object introduction rule

$$\frac{\begin{array}{c}A \equiv [\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]\\ \Gamma \vdash x_i{:}A_i :: T_{\mathsf{res}}(x_i)^{i=1\ldots n} \quad \Gamma, y_j{:}A \vdash b_j{:}B_j{::}T_j{}^{j=1\ldots m}\end{array}}{\Gamma \vdash [\mathsf{f}_i = x_i{}^{i=1\ldots n}, \mathsf{m}_j = \varsigma(y_j)b_j{}^{j=1\ldots m}]{:}A{::}T_{\mathsf{obj}}(\mathsf{f}_1 = x_1 \ldots \mathsf{f}_n = x_n)},$$

we see that A is $[\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}B_j{::}T_j]$, that T is $T_{\mathsf{obj}}(\mathsf{f}_1 = x_1 \ldots \mathsf{f}_n = x_n)$ and that:

(H1′) $\Gamma \vdash x_i : A_i :: T_{\mathsf{res}}(x_i)$ for $1 \leqslant i \leqslant n$.

(H1″) $\Gamma, y_j{:}A \vdash b_j : B_j :: T_j$ for $1 \leqslant j \leqslant m$.

We have to show that there is $\phi \in \mathsf{SF}$ such that, for all $\Sigma' \succcurlyeq \Sigma$ and all $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ with $[\![a]\!]\rho\sigma' = (v, \sigma'')$, conditions (S1)–(S4) hold.

From (H3) and context extension (Lemma 4.5), we know that for all $\hat{\Sigma} \succcurlyeq \Sigma$ and $l_0 \in \|A[\rho/\Gamma]\|_{\hat{\Sigma}}$,

$$\rho[y_j := l_0] \in \|\Gamma, y_j{:}A\|_{\hat{\Sigma}}\,.$$

Hence, by the induction hypothesis on (H1″), for all $1 \leqslant j \leqslant m$ there is $\phi^j_{\hat{\Sigma}\, l_0} \in \mathsf{SF}$ such that, for all $\Sigma_1 \succcurlyeq \hat{\Sigma}$ and all $(\sigma_1, \phi_1) \in \mathrm{fix}(\Phi)_{\hat{\Sigma}}$ with $[\![b_j]\!]\rho[y_j := l_0]\sigma_1 = (v_2, \sigma_2)\!\downarrow$, we obtain the conclusions (S1)–(S4) of the lemma, that is:

(S1′) There exists $\Sigma_2 \succcurlyeq \Sigma_1 \succcurlyeq \hat{\Sigma}$ and $\phi_2 \in \mathsf{RSF}$ such that $\phi^j_{\hat{\Sigma}\, l_0}(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$.

(S2′) $(\sigma_2, \phi_2) \in \mathrm{fix}(\Phi)_{\Sigma_2}$.

(S3′) $v_2 \in \|B_j[\rho[y_j := l_0]/\Gamma, y_j{:}A]\|_{\Sigma_2}$.

(S4′) $(\pi_{\mathrm{Val}}(\sigma_1), v_2, \pi_{\mathrm{Val}}(\sigma_2)) \in [\![[\Gamma, y_j{:}A] \vdash T_j]\!]\rho[y_j := l_0]$.

Now we can define $\phi \in \mathsf{SF}$ as follows:

$$\phi(\sigma', \phi', \Sigma') = \begin{cases} ((\Sigma', l_0{:}A[\rho/\Gamma]), \phi' + \{\!|l_0 = \{\!|\mathsf{m}_j = \phi^j_{(\Sigma', l_0:A[\rho/\Gamma])\,l_0}|\!\}|\!\}) \\ \qquad\qquad \text{if } \Sigma' \succcurlyeq \Sigma \text{ and } [\![a]\!]\rho\sigma' = (l_0, \sigma'') \\ \text{undefined} \quad \text{otherwise,} \end{cases} \tag{17}$$

which is continuous as all constituents are and *StSpec* is a flat cpo. We now show that (S1)–(S4) hold. Let $\Sigma' \succcurlyeq \Sigma$ and $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\hat{\Sigma}}$, and suppose $[\![a]\!]\rho\sigma' = (v, \sigma'')$. By the definition of the semantics, and the fact that (H1′) entails $\rho(x_i)\downarrow$ for $1 \leqslant i \leqslant n$, for $[\![a]\!]\rho\sigma' = (v, \sigma'') \in \mathsf{Loc} \times \mathsf{St}$, we obtain $v = l_0$ where $l_0 \notin \mathsf{dom}(\sigma')$ (and so $l_0 \notin \mathsf{dom}(\Sigma')$) and

$$\sigma'' = \sigma' + \{\!|l_0 = \{\!|\mathsf{f}_i = \rho(x_i), \mathsf{m}_j = \lambda\sigma.[\![b_j]\!]\rho[y_j := l_0]\sigma|\!\}|\!\}. \tag{18}$$

We then get that there exists $\phi'' \in \mathsf{RSF}$ such that:

(S1) $\phi(\sigma', \phi', \Sigma') = ((\Sigma', l_0{:}A[\rho/\Gamma]), \phi'')$, by construction of ϕ in (17).

(S3) $v = l_0 \in \|A[\rho/\Gamma]\|_{(\Sigma', l_0:A[\rho/\Gamma])}$, by definition of $\|\cdot\|$.

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![\Gamma \vdash T_{\mathsf{obj}}(\mathsf{f}_1 = x_1 \ldots \mathsf{f}_n = x_n)]\!]\rho$, which is easily checked from the definition of $T_{\mathsf{obj}}(\ldots)$, the semantics in Table 8 and equation (18).

Now all we need to show is (S2): $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$, where Σ'' is $\Sigma', l_0{:}A[\rho/\Gamma]$. By the construction of ϕ in (17),

$$\phi'' = \phi' + \{\!|l_0 = \{\!|\mathsf{m}_j = \phi^j_{(\Sigma', l_0:A[\rho/\Gamma])\,l_0}|\!\}|\!\}.$$

We show (S2) according to Definition 4.6.

First, by assumption, the domains of ϕ' and Σ' agree, so (Dom1) holds. By construction of ϕ, we also have $\mathsf{dom}(\phi''.l_0) = \{\mathsf{m}_1, \ldots, \mathsf{m}_m\}$, from which (Dom2) follows. For the remaining conditions, suppose $l \in \mathsf{dom}(\Sigma'')$. We distinguish two cases:

– $l \neq l_0$: Then

$$\Sigma''.l = \Sigma'.l = [\mathsf{g}_i{:}A_i'^{\,i=1\ldots p}, \mathsf{n}_j{:}\varsigma(y_j)B_j' :: T_j'^{\,1\ldots q}].$$

(F) For all $1 \leqslant i \leqslant p$, $\sigma''.l.\mathsf{g}_i = \sigma'.l.\mathsf{g}_i$, so from $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$,

$$\sigma''.l.\mathsf{g}_i \in \|A_i'\|_{\Sigma'} \subseteq \|A_i'\|_{\Sigma''}.$$

(M) Let $1 \leqslant j \leqslant q$, $\Sigma_1 \succcurlyeq \Sigma''$ and $(\sigma_1, \phi_1) \in \mathrm{fix}(\Phi)_{\Sigma_1}$, and suppose $\sigma''.l.\mathsf{n}_j(\sigma_1) = (v_2, \sigma_2)$. Since $\sigma''.l.\mathsf{n}_j = \sigma'.l.\mathsf{n}_j$ and $\Sigma_1 \succcurlyeq \Sigma'$, the assumption that $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and the construction of ϕ'' entails that there exists $\Sigma_2 \succcurlyeq \Sigma_1$ and $\phi_2 \in \mathsf{RSF}$ such that $\phi''.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma_1) = \phi'.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$ and:

(M1) $(\pi_{\mathrm{Val}}(\sigma_1), v_2, \pi_{\mathrm{Val}}(\sigma_2)) \in [\![T_j'[l/y_j]]\!]$.

(M2) $(\sigma_2, \phi_2) \in \mathrm{fix}(\Phi)_{\Sigma_2}$.

(M3) $v_2 \in \left\|B_j'[l/y_j]\right\|_{\Sigma_2}$.

– $l = l_0$:

(F) By the assumption, (H1$'$) and $\rho \in \|\Gamma\|_\Sigma$, we know that there is $\vdash A_i{}' <: A_i$ for all $1 \leqslant i \leqslant n$ such that $x_i{:}A_i{}'$ in Γ. Hence,

$$\sigma''.l_0.\mathsf{f}_i = \rho(x_i) \in \|A_i{}'\|_\Sigma \subseteq \|A_i\|_\Sigma \subseteq \|A_i\|_{\Sigma''}\,.$$

(M) Let $1 \leqslant j \leqslant m$. Suppose $\Sigma_1 \succcurlyeq \Sigma''$, let $(\sigma_1, \phi_1) \in \mathrm{fix}(\Phi)_{\Sigma_1}$ and suppose $\sigma''.l_0.\mathsf{m}_j(\sigma_1) = (v_2, \sigma_2)$. Since $\sigma''.l_0.\mathsf{m}_j = [\![b_j]\!]\rho[y_j := l_0]\sigma_1$, $\Sigma_1 \succcurlyeq \Sigma''$ and $(\sigma_1, \phi_1) \in \mathrm{fix}(\Phi)_{\Sigma_1}$, by setting $\hat{\Sigma} := \Sigma''$ from (S1$'$), we get $\phi^j_{\Sigma''\, l_0}(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$, so we obtain the required Σ_2 and ϕ_2 such that

$$\phi''.l_0.\mathsf{m}(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$$

by definition of ϕ''. Thus, we can prove the remaining goals:

(M1) $(\pi_{\mathrm{Val}}(\sigma_1), v_2, \pi_{\mathrm{Val}}(\sigma_2)) \in [\![[\Gamma, y_j{:}A] \vdash T_j]\!]\rho[y_j := l_0] = [\![T_j[\rho/\Gamma][l_0/y_j]]\!]$, by (S4$'$);

(M2) $(\sigma_2, \phi_2) \in \mathrm{fix}(\Phi)_{\Sigma_2}$, by (S2$'$);

(M3) $v_2 \in \|B_j[\rho[y_j := l_0]/\Gamma, y_j{:}A]\|_{\Sigma_2} = \|B_j[\rho/\Gamma][l_0/y_j]\|_{\Sigma_2}$, by (S3$'$);

where the equations in (M1) and (M2) are by applications of the substitution lemma, Lemma 4.2.

Thus, we have shown $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$, that is, (S2) holds.

— Method Invocation

Suppose $\Gamma \vdash a : A :: T$ is derived by an application of the Method Invocation rule:

$$\frac{\Gamma \vdash x{:}[\mathsf{m}{:}\varsigma(y)A'{::}T']{::}T_{\mathsf{res}}(x)}{\Gamma \vdash x.\mathsf{m}{:}A'[x/y]{::}T'[x/y]}.$$

Necessarily, a is of the form $x.\mathsf{m}$ and there are A' and T' such that $A \equiv A'[x/y]$ and $T \equiv T'[x/y]$. So, suppose:

(H1) $\Gamma \vdash a : A'[x/y] :: T'[x/y]$.

(H2) Σ is a store specification.

(H3) $\rho \in \|\Gamma\|_\Sigma$.

Define $\phi \in \mathsf{SF}$ using 'self-application' of the argument,

$$\phi(\sigma', \phi', \Sigma') \quad = \quad \phi'.\rho(x).\mathsf{m}(\sigma', \phi', \Sigma'). \tag{19}$$

Now let $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and suppose $[\![a]\!]\rho\sigma' = \sigma'.\rho(x).\mathsf{m}(\sigma') = (v, \sigma'')$ terminates. We show that (S1)–(S4) hold.

By the hypothesis of the method invocation rule, we can assume

$$\Gamma \vdash x{:}[\mathsf{m}{:}\varsigma(y)A'{::}T']{::}T_{\mathsf{res}}(x). \tag{H1$'$}$$

Since this implies $x{:}B \in \Gamma$ for some $[\Gamma] \vdash B <: [\mathsf{m} : \varsigma(y)A' :: T']$, by assumption (H3) this entails

$$\vdash \Sigma'.(\rho(x)) <: [\mathsf{m} : \varsigma(y)A' :: T']\,[\rho/\Gamma].$$

That is, there are A_i, A'', B_j and T_j, T'' such that

$$\vdash \Sigma'.\rho(x) \equiv [\mathsf{f}_i{:}A_i, \mathsf{m}_j{:}\varsigma(y_j)B_j :: T_j, \mathsf{m}{:}\varsigma(y)A''::T'']$$

where

$$y \vdash A'' <: A'[\rho/\Gamma] \quad \wedge \quad \vdash_{\mathsf{fo}} T'' \rightarrow T'[\rho/\Gamma]. \tag{20}$$

Now, the assumption that $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ together with equation (19) implies that there are Σ'', ϕ'' such that:

(S1) $\phi(\sigma', \phi', \Sigma') = \phi'.(\rho(x)).\mathsf{m}(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$.

(S2) $(\sigma'', \phi'') \in \mathrm{fix}(\Phi)_{\Sigma''}$.

(S3′) $v \in \|A''[\rho(x)/y]\|_{\Sigma''}$.

(S4′) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![\vdash T''[\rho(x)/y]]\!]$.

By the transitivity of $<:$, equation (20), Lemma 4.2 and (S3′)

$$v \in \|A'[\rho/\Gamma][\rho(x)/y]\|_{\Sigma''}.$$

Since $A'[\rho/\Gamma, \rho(x)/y] \equiv A'[x/y][\rho/\Gamma]$, we also have:

(S3) $v \in \|A'[x/y][\rho/\Gamma]\|_{\Sigma''} = \|A[\rho/\Gamma]\|_{\Sigma''}$.

Similarly, by (20) and (S4′),

$$\begin{aligned}(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T''[\rho(x)/y]]\!] &\subseteq [\![T'[\rho/\Gamma][\rho(x)/y]]\!] \\ &= [\![[\Gamma] \vdash T[x/y]]\!]\rho,\end{aligned} \tag{S4}$$

which is what we had to show.

— Field Selection

This is similar to the previous case. ϕ can be chosen as $\phi(\sigma', \phi', \Sigma') = (\phi', \Sigma')$.

— Field Update

Suppose:

(H1) $\Gamma \vdash a{:}A::T$ has been derived by an application of Field Update.

(H2) Σ is a store specification.

(H3) $\rho \in \|\Gamma\|_{\Sigma}$.

Let $\Sigma' \succcurlyeq \Sigma$ and $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$, and suppose $[\![a]\!]\rho\sigma' = (v, \sigma'')$ terminates. Recall the rule for field update,

$$\frac{\begin{array}{c}A \equiv [\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}] \\ \Gamma \vdash x{:}A::T_{\mathsf{res}}(x) \quad \Gamma \vdash y{:}A_k::T_{\mathsf{res}}(y)\end{array}}{\Gamma \vdash x.\mathsf{f}_k := y{:}A::T_{\mathsf{upd}}(x, \mathsf{f}_k, y)} \ (1 \leqslant k \leqslant n).$$

In particular, a is of the form $x.\mathsf{f}_k := y$ and T is $T_{\mathsf{upd}}(x, \mathsf{f}_k, y)$. From the semantics of $[\![a]\!]\rho\sigma'$, this means $v = \rho(x) \in \mathsf{Loc}$ and

$$\sigma'' = \sigma'[v := \sigma'.v[\mathsf{f}_k := \rho(y)]]. \tag{21}$$

Let us define the required $\phi \in \mathsf{SF}$ by $\phi(\sigma', \phi', \Sigma') = (\Sigma', \phi')$, which reflects the fact that no new objects are created. We have to show that (S1)–(S4) hold.

By (H3), $\rho(x) \in \|A[\rho/\Gamma]\|_{\Sigma} \subseteq \|A[\rho/\Gamma]\|_{\Sigma'}$. Then, by the construction of ϕ and (21):

(S1) $\phi(\sigma', \phi', \Sigma') = (\Sigma', \phi')$.

(S3) $v = \rho(x) \in \|A[\rho/\Gamma]\|_{\Sigma'}$.

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma] \vdash T]\!]\rho$, from the semantics given in Table 8.

We still need to show (S2): $(\sigma'', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$.

By the assumption that $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and $\mathsf{dom}(\sigma'') = \mathsf{dom}(\sigma')$, the conditions (Dom1) and (Dom2) of Definition 4.6 are satisfied. As for (F) and (M), let $l \in \mathsf{dom}(\Sigma')$ and $\Sigma'.l \equiv [\mathsf{g}_i{:}A_i'^{\,i=1...p}, \mathsf{n}_j{:}\varsigma(y_j)B_j' :: T_j'^{\,1...q}]$.

(F) We distinguish two cases:

- Case $l = \rho(x)$ and $\mathsf{g}_i = \mathsf{f}_k$. Then, by (21), $\sigma''.l.\mathsf{g}_i = \rho(y)$. By (H3), $\rho(x) \in \|A[\rho/\Gamma]\|_{\Sigma} \subseteq \|A[\rho/\Gamma]\|_{\Sigma'}$, which entails
$$\vdash \Sigma'.l <: A[\rho/\Gamma],$$
and, in particular, by the definition of the subspecification relation,
$$A_k' = \Sigma'.l.\mathsf{f}_k \equiv A_k[\rho/\Gamma].$$
Note that *invariance of subspecification* in the field components is needed to conclude this. Now, again by (H3),
$$\rho(y) \in \|A_k[\rho/\Gamma]\|_{\Sigma} \subseteq \|A_k[\rho/\Gamma]\|_{\Sigma'} = \|A_k'\|_{\Sigma'}\,.$$
Hence, $\sigma''.l.\mathsf{g}_i \in \|A_i'\|_{\Sigma'}$, as required.
- Case $l \neq \rho(x)$ or $\mathsf{g}_i \neq \mathsf{f}_k$. Then $\sigma''.l.\mathsf{g}_i = \sigma'.l.\mathsf{g}_i$, by (21). Hence, by the assumption that $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$, we have $\sigma''.l.\mathsf{g}_i \in \|A_i'\|_{\Sigma'}$.

(M) Let $\Sigma'' \succcurlyeq \Sigma'$ and $(\sigma_1, \phi_1) \in \mathrm{fix}(\Phi)_{\Sigma''}$, and suppose $\sigma''.l.\mathsf{n}_j(\sigma_1) = (v_2, \sigma_2)$. Then, by the assumption that $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$ and the fact that $\sigma''.l.\mathsf{n}_j = \sigma'.l.\mathsf{n}_j$ by (21), we get $\phi'.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma'') = (\Sigma_2, \phi_2)$ where $\Sigma_2 \succcurlyeq \Sigma''$, and, as required:

(M1) $(\pi_{\mathrm{Val}}(\sigma_1), v_2, \pi_{\mathrm{Val}}(\sigma_2)) \in [\![T_j'[l/y_j]]\!]$.

(M2) $(\sigma_2, \phi_2) \in \mathrm{fix}(\Phi)_{\Sigma_2}$.

(M3) $v_2 \in \left\|B_j'[l/y_j]\right\|_{\Sigma_2}$.

This concludes the proof. □

5.2. *Soundness theorem*

With Lemma 5.2 and Lemma 5.4 proved above, it is now easy to establish our main result.

Theorem 5.5 (Soundness). If $\Gamma \vdash a : A :: T$, then $\Gamma \models a : A :: T$.

Proof. Suppose $\Gamma \vdash a : A :: T$, let $\Sigma \in \mathit{StSpec}$ be a store specification and suppose $\rho \in \mathsf{Env}$ such that $\rho \in \|\Gamma\|_{\Sigma}$. Let $\sigma \in [\![\Sigma]\!]$, so, by definition, there exists $\phi \in \mathsf{RSF}$ such that $(\sigma, \phi) \in \mathit{fix}(\Phi)_{\Sigma}$. Next suppose
$$[\![a]\!]\rho\sigma = (v, \sigma').$$
By Lemma 5.4, there exists $\phi_a \in \mathsf{RSF}$ such that:

(S1) $\phi_a(\sigma, \phi, \Sigma) = (\Sigma', \phi')$ where $\Sigma' \succcurlyeq \Sigma$ and

(S2) $(\sigma', \phi') \in \mathrm{fix}(\Phi)_{\Sigma'}$
(S3) $v \in \|A[\rho/\Gamma]\|_{\Sigma'}$
(S4) $(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![[\Gamma] \vdash T]\!]\rho$.

In particular, $\sigma' \in [\![\Sigma']\!]$ follows.

Now, in the case where A is *Bool* we obtain $(v, \sigma') \in [\![[\Gamma] \vdash A]\!]\rho$ from $\|Bool\|_{\Sigma'} = \mathsf{BVal}$. Otherwise, A is an object specification, and we must have $\vdash \Sigma'.v <: A[\rho/\Gamma]$ by the definition of $\|A[\rho/\Gamma]\|_{\Sigma'}$. Hence, by Lemma 5.2,

$$(v, \sigma') \in [\![A[\rho/\Gamma]]\!] = [\![[\Gamma] \vdash A]\!]\rho,$$

where the last equality is by the the substitution lemma, Lemma 4.2. □

In particular, if $\vdash a : A :: T$ and $[\![a]\!]\sigma = (v, \sigma')$, then $(v, \sigma') \in [\![A]\!]$, so $v \neq \mathsf{error}$ by Lemma 3.4.

6. Recursive specifications

In this section we investigate an extension of the logic with recursive specifications. These are required when a field of an object or a result of one of the object's methods are supposed to satisfy the same specification as the object itself. In particular, they are needed for the specification of any recursive datatype: referring back to the example of the account manager in Table 5, if $A_{\texttt{Manager}}$ included a list of accounts, we would need a recursive specification $\mu(X)[\mathsf{head} : A_{\texttt{Account}}, \mathsf{tail} : X]$.

In this section we discuss in more detail how recursive specifications can be dealt with in the logic.

6.1. *Syntax and proof rules*

To accommodate reasoning about elements of recursive types such as lists, we introduce recursive specifications $\mu(X)A$. To prevent meaningless specifications such as $\mu(X)X$, we only allow recursion through object specifications, thereby enforcing 'formal contractiveness'.

$$\begin{aligned} \underline{A} &::= \top \mid Bool \mid [\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}] \mid \mu(X)\underline{A} \\ A, B &::= \underline{A} \mid X \end{aligned}$$

where X ranges over an infinite set $\mathscr{SV}$ of specification variables. X is bound in $\mu(X)A$, and, as usual, we identify specifications up to the names of bound variables.

In addition to specification contexts Γ, we introduce contexts Δ that contain specification variables with an upper bound, $X <: A$, where A is either another variable or $\top$. In the rules of the logic, we replace each judgement $\Gamma \vdash a{:}A{::}T$ by $\Gamma; \Delta \vdash a{:}A{::}T$, and the definitions of well-formed specifications and well-formed specification contexts are extended in a similar way to the case of recursive types for the object calculus (Abadi and Cardelli 1996),

$$\frac{\Gamma; \Delta \vdash Y \quad X \notin \Delta}{\Gamma; \Delta, X <: Y \vdash \mathsf{ok}} \qquad \frac{\Gamma; \Delta \vdash \mathsf{ok} \quad X \notin \Delta}{\Gamma; \Delta, X <: \top \vdash \mathsf{ok}}$$

and

$$\frac{\Gamma;\Delta, X <: A, \Delta' \vdash \mathsf{ok}}{\Gamma;\Delta, X <: A, \Delta' \vdash X} \qquad \frac{\Gamma;\Delta, X <: \top \vdash A}{\Gamma;\Delta \vdash \mu(X)A} \qquad \frac{\Gamma;\Delta \vdash \mathsf{ok}}{\Gamma;\Delta \vdash \top}.$$

We will simply write Δ, X, Δ' in place of $\Delta, X <: \top, \Delta'$.

Subspecifications for recursive specifications are obtained by the 'usual' recursive subtyping rule (Amadio and Cardelli 1993), and $\top$ is the greatest specification,

$$\textsc{RecSub} \quad \frac{\Gamma;\Delta, Y <: \top, X <: Y \vdash A <: B}{\Gamma;\Delta \vdash \mu(X)A <: \mu(Y)B}$$

$$\textsc{Top} \quad \frac{\Gamma;\Delta \vdash A}{\Gamma;\Delta \vdash A <: \top}.$$

As can be seen from the semantics below, in our model, a recursive specification and its unfolding are not just isomorphic but equal, that is, $[\![\mu(X)A]\!] = [\![A[(\mu(X)A)/X]]\!]$. Because of this, we do not need to introduce *fold* and *unfold* terms: we can deal with (un)folding of recursive specifications through the subsumption rule once we add the following subspecifications:

$$\textsc{Fold} \quad \frac{\Gamma;\Delta \vdash \mu(X)A}{\Gamma;\Delta \vdash A[(\mu(X)A)/X] <: \mu(X)A}$$

$$\textsc{Unfold} \quad \frac{\Gamma;\Delta \vdash \mu(X)A}{\Gamma;\Delta \vdash \mu(X)A <: A[(\mu(X)A)/X]}.$$

We will prove their soundness below.

6.2. *Existence of store specifications*

In this section, we adapt our notion of store specification to recursive specifications. This is very similar to Definition 4.6; for completeness we will spell it out in detail.

Definition 6.1. A store specification, for the purposes of the present section, is a record $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$ such that for each $l \in \mathsf{dom}(\Sigma)$, $\Sigma.l$ is a closed (recursive) object specification of the form $\mu(X)[\mathsf{f}_i\!:\!A_i{}^{i=1\ldots n},\ \mathsf{m}_j\!:\!\varsigma(y_j)B_j\!::\!T_j{}^{j=1\ldots m}]$. We continue to write *StSpec* for the set of store specifications.

Note that because of the Fold and Unfold rules of recursive types, the requirement that only object specifications with a μ-binder in head position occur in Σ is no real restriction. The definition of the functional Φ of Definition 4.6 remains virtually the same, apart from an unfolding of the recursive specification in the cases for field and method result specifications.

Definition 6.2. Let $\mathscr{S} = \mathscr{P}(\mathsf{St} \times \mathsf{RSF})^{\mathit{StSpec}}$ denote the collection of families of subsets of $\mathsf{St} \times \mathsf{RSF}$, indexed by store specifications (in the sense of Definition 6.1). Table 13 defines a functional $\Phi : \mathscr{S}^{op} \times \mathscr{S} \to \mathscr{S}$; as before, we write $\sigma \in [\![\Sigma]\!]$ if there is some $\phi \in \mathsf{RSF}$ such that $(\sigma, \phi) \in \mathrm{fix}(\Phi)_\Sigma$.

The proof of Lemma 4.7 can be easily adapted to show that this functional also has a unique fixed point.

Table 13. *Store specifications*

$(\sigma,\phi) \in \Phi(Y,X)_\Sigma$	$\iff$	(Dom1) $\mathsf{dom}(\Sigma) = \mathsf{dom}(\phi)$; and
		(Dom2) $\forall l \in \mathsf{dom}(\Sigma)$.
		if $\mathsf{dom}(\Sigma.l) \equiv \mu(X)[\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$
		then $\mathsf{dom}(\phi.l) = \{\mathsf{m}_j\}_{j=1\ldots m}$; and
		$\forall l \in \mathsf{dom}(\Sigma)$ where $\Sigma.l \equiv \mu(X)[\mathsf{f}_i{:}A_i{}^{i=1\ldots n}, \mathsf{m}_j{:}\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$:
		(F) $\forall 1 \leqslant i \leqslant n.\ \sigma.l.\mathsf{f}_i \in \Vert A_i[\Sigma.l/X]\Vert_\Sigma$; and
		(M) $\forall 1 \leqslant j \leqslant m.\ \forall\Sigma' \succcurlyeq \Sigma\ \forall\sigma',\sigma'' \in \mathsf{St}\ \forall\phi' \in \mathsf{RSF}\ \forall l' \in \mathsf{Loc}\ \forall v \in \mathsf{Val}$.
		if $l' \in \Vert\Sigma.l\Vert_{\Sigma'} \ \wedge\ (\sigma',\phi') \in Y_{\Sigma'} \ \wedge\ \sigma.l.\mathsf{m}_j(l',\sigma') = (v,\sigma'')$
		then $\exists\Sigma'' \succcurlyeq \Sigma'\ \exists\phi'' \in \mathsf{RSF}.\ \phi.l.\mathsf{m}_j(l',\sigma',\phi',\Sigma') = (\Sigma'',\phi'')$ and
		(M1) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T_j[l'/y_j]]\!]$; and
		(M2) $(\sigma'',\phi'') \in X_{\Sigma''}$; and
		(M3) $v \in \Vert B_j[\Sigma.l/X][l'/y_j]\Vert_{\Sigma''}$

Lemma 6.3 (Existence). Functional Φ in Definition 6.2 has a unique fixpoint $\mathrm{fix}(\Phi)$.

6.3. *Semantics of recursive specifications*

Definition 6.4. We extend the interpretation of specifications to the new cases, where η maps specification variables to admissible subsets of $\mathsf{Val} \times \mathsf{St}$:

$$[\![\Gamma;\Delta \vdash \top]\!]\rho\eta = \mathsf{Val} \times \mathsf{St}$$
$$[\![\Gamma;\Delta \vdash X]\!]\rho\eta = \eta(X)$$
$$[\![\Gamma;\Delta \vdash \mu(X)A]\!]\rho\eta = \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X{<:}\top \vdash A]\!]\rho\eta[X = \chi])$$

We write $\eta \vDash \Delta$ if, for all $X <: Y$ in Δ, we have $\eta(X) \subseteq \eta(Y)$.

We briefly observe the following facts, (the duals of) which are standard (Davey and Priestley 2002):

- By Tarski's Fixed Point Theorem, every monotonic map $f : L \to L$ on a complete lattice $(L, \leqslant)$ has a greatest fixed-point $\mathsf{gfp}(f)$ (which is in fact the greatest *post*-fixed point).
- If $f : L \to L$ also preserves meets of decreasing chains $x_0 \geqslant x_1 \geqslant \ldots$, that is, $f(\bigwedge_i x_i) = \bigwedge_i f(x_i)$, then the greatest fixed point can be obtained as $\mathsf{gfp}(f) = \bigwedge\{f^n(\top) \mid n \in \mathbb{N}\}$ where $\top$ is the greatest element of L.
- For a complete lattice $(L, \leqslant)$ and any set A, the set of maps $A \to L$ forms a complete lattice when ordered pointwise, with the meet of $\{f_i \mid i \in I\}$ given by $\lambda a. \bigwedge_i f_i(a)$.
- The greatest fixed point operator is monotonic: if $f \leqslant g$ are monotonic maps in the lattice $L \to L$, then $\mathsf{gfp}(f) \leqslant \mathsf{gfp}(g)$.
- Composition preserves meets of descending chains: if $f_0 \geqslant f_1 \geqslant \ldots$ and $g_0 \geqslant g_1 \geqslant \ldots$ are maps in $L \to L$ such that every f_i and g_j is monotonic and preserves meets of descending chains, then $\bigwedge_i f_i \circ \bigwedge_j g_j = \bigwedge_n (f_n \circ g_n)$. It follows that $\mathsf{gfp}(\bigwedge_i f_i) = \bigwedge_i \mathsf{gfp}(f_i)$ that is, gfp also preserves meets of chains.

The set of admissible subsets of $\mathsf{Val} \times \mathsf{St}$, $\mathscr{A}dm(\mathsf{Val} \times \mathsf{St})$, is closed under arbitrary intersections, and hence forms a complete lattice when ordered by set inclusion. Therefore, specification environments $\eta : \mathscr{S}\mathscr{V} \to \mathscr{A}dm(\mathsf{Val} \times \mathsf{St})$ with the pointwise ordering form a complete lattice.

In the following, we show that the interpretation of specifications given above is well defined. More specifically, we show that meets of descending chains of environments are preserved.

Lemma 6.5 (Well-definedness). $[\![\Gamma;\Delta \vdash A]\!]$ preserves meets of descending chains:

$$\eta_0 \geqslant \eta_1 \geqslant \dots \quad \Longrightarrow \quad [\![\Gamma;\Delta \vdash A]\!]\rho(\textstyle\bigwedge_i \eta_i) = \bigcap_i [\![\Gamma;\Delta \vdash A]\!]\rho\eta_i.$$

In particular, this lemma shows that the greatest fixed point used in Definition 6.4 exists by the observations made above.

Proof. We use induction on the structure of A. The only interesting case is where A is of the form $\mu(X)B$.

Suppose $\eta_0 \geqslant \eta_1 \geqslant \dots$. If we let $f_i : \mathscr{A}dm(\mathsf{Val} \times \mathsf{St}) \to \mathscr{A}dm(\mathsf{Val} \times \mathsf{St})$,

$$f_i(\chi) = [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta_i[X=\chi], \qquad i \in \mathbb{N},$$

then the induction hypothesis entails that each f_i is monotonic, and $f_0 \geqslant f_1 \geqslant \dots$ is a descending chain of environments. Moreover, since for each $i \in \mathbb{N}$ and descending chain $\chi_0 \supseteq \chi_1 \supseteq \dots$ in $\mathscr{A}dm(\mathsf{Val} \times \mathsf{St})$,

$$\textstyle\bigwedge_j \eta_i[X=\chi_j] = \eta_i[X = \bigcap_j \chi_j]$$

the induction hypothesis shows that each f_i preserves meets:

$$\begin{aligned} f_i(\textstyle\bigcap_j \chi_j) &= [\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_j \eta_i[X=\chi_j]) \\ &= \textstyle\bigcap_j [\![\Gamma;\Delta, X \vdash B]\!]\rho(\eta_i[X=\chi_j]) \\ &= \textstyle\bigcap_j f_i(\chi_j). \end{aligned}$$

We obtain

$$\begin{aligned} [\![\Gamma;\Delta \vdash A]\!]\rho(\textstyle\bigwedge_i \eta_i) &= \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_i \eta_i)[X=\chi]) && \text{by definition} \\ &= \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_i \eta_i[X=\chi])) && \text{pointwise meet} \\ &= \mathsf{gfp}(\lambda\chi.\textstyle\bigcap_i [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta_i[X=\chi]) && \text{by induction} \\ &= \mathsf{gfp}(\textstyle\bigwedge_i f_i) && \text{pointwise meet} \\ &= \textstyle\bigcap_i \mathsf{gfp}(f_i) && \text{gfp preserves meet} \\ &= \textstyle\bigcap_i [\![\Gamma;\Delta \vdash A]\!]\rho\eta_i && \text{by definition,} \end{aligned}$$

which concludes the proof □

Lemma 6.6 (Substitution). For all $\Gamma;\Delta, X \vdash A$, $\Gamma;\Delta \vdash B$, ρ and η,

$$[\![\Gamma;\Delta, X \vdash A]\!]\rho(\eta[X{=}[\![\Gamma;\Delta \vdash B]\!]\rho\eta]) = [\![\Gamma;\Delta \vdash A[B/X]]\!]\rho\eta.$$

Proof. The proof is by induction on A. □

Table 14. *Approximations*

$X\vert^{k+1}$	$=$	X
$\top\vert^{k+1}$	$=$	$\top$
$Bool\vert^{k+1}$	$=$	$Bool$
$\mu(X)A\vert^{k+1}$	$=$	$A[\mu(X)A/X]\vert^{k+1}$
$B\vert^{k+1}$	$=$	$[\mathsf{f}_i : A_i\vert^k,\ \mathsf{m}_j : \varsigma(y_j)B_j\vert^k :: T_j]_{i\in I, j\in J}$

where $B \equiv [\mathsf{f}_i\!:\!A_i{}^{i=1\ldots n},\ \mathsf{m}_j\!:\!\varsigma(y_j)B_j\!::\!T_j{}^{j=1\ldots m}]$

6.4. *Syntactic approximations*

Recall the statement of Lemma 5.2, one of the key lemmas in the proof of the soundness theorem:

$$\forall\sigma,\Sigma,l,A.\ A \text{ closed } \wedge\ \sigma \in [\![\Sigma]\!]\ \wedge\ \vdash \Sigma.l <: A \implies (l,\sigma) \in [\![A]\!]. \tag{22}$$

In Section 5 this was proved by induction on the structure of A. This inductive proof cannot be extended directly to prove a corresponding result for recursive specifications: the recursive unfolding in cases (F) and (M3) of Definition 6.2 would force a similar unfolding of A in the inductive step, thus not necessarily decreasing the size of A.

The remainder of this section is therefore concerned with the derivation of (22). Instead of using induction on A, we consider finite approximations in the sense of Amadio and Cardelli (1993), where we get rid of recursion by unfolding a finite number of times and then replacing all remaining occurrences of recursion by $\top$. We call a specification *non-recursive* if it does not contain any occurrences of specifications of the form $\mu(X)B$.

Definition 6.7 (Approximations). For each A and each $k \in \mathbb{N}$, we define $A|^0 = \top$ and $A|^{k+1}$ by the cases given in Table 14.

Note that, as in Amadio and Cardelli (1993), well definedness of approximation can be shown by a well-founded induction on the lexicographic order on k and the number of μ in head position. In particular, observe that our definition of recursive specifications has already ruled out troublesome cases such as $\mu(X)X$.

6.4.1. *Properties of approximations*

Unfortunately, approximations $A|^k$ as defined above are not in fact approximating to A (from above) with respect to the subspecification relation $<:$, the reason being the invariance in field specifications. For example, if we consider an object specification $A \equiv [\mathsf{f}_1 : X, \mathsf{f}_2 : Bool]$, we can observe the following:

$$\begin{aligned}
\mu(X)\mu(Y)A|^2 &= [\mathsf{f}_1 : \mu(X)\mu(Y)A, \mathsf{f}_2 : Bool]|^2\\
&= [\mathsf{f}_1 : \mu(X)\mu(Y)A|^1, \mathsf{f}_2 : Bool\,|^1]\\
&= [\mathsf{f}_1 : [\mathsf{f}_1 : \mu(X)\mu(Y)A, \mathsf{f}_2 : Bool]|^1, \mathsf{f}_2 : Bool]\\
&= [\mathsf{f}_1 : [\mathsf{f}_1 : \mu(X)\mu(Y)A|^0, \mathsf{f}_2 : Bool\,|^0], \mathsf{f}_2 : Bool]\\
&= [\mathsf{f}_1 : [\mathsf{f}_1 : \top, \mathsf{f}_2 : \top], \mathsf{f}_2 : Bool].
\end{aligned}$$

Table 15. *The generalised object subspecification rule*

$$\frac{\begin{array}{lll} & & \Gamma, y_j \vdash T_j{}^{j=1\ldots m+q} \\ \Gamma \quad ;\Delta \vdash A_i{}^{i=1\ldots n+p} & \Gamma \quad ;\Delta \vdash A_i <: A_i'{}^{i=1\ldots n} & \Gamma, y_j \vdash T_j'{}^{j=1\ldots m} \\ \Gamma, y_j ;\Delta \vdash B_j{}^{j=m+1\ldots m+q} & \Gamma, y_j ;\Delta \vdash B_j <: B_j'{}^{j=1\ldots m} & \vdash_{\mathsf{fo}} T_j \to T_j'{}^{j=1\ldots m} \end{array}}{\Gamma;\Delta \vdash [\mathsf{f}_i : A_i{}^{i=1\ldots n+p}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m+q}] <: [\mathsf{f}_i : A_i'{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j' :: T_j'{}^{j=1\ldots m}]}$$

By inspection of the rules, $\vdash \mu(X)\mu(Y)A <: \mu(X)\mu(Y)A|^2$ requires us to show

$$\Gamma;\Delta \vdash [\mathsf{f}_1 : [\mathsf{f}_1 : \mu(X)\mu(Y)A, \mathsf{f}_2 : \mathit{Bool}], \mathsf{f}_2 : \mathit{Bool}] <: [\mathsf{f}_1 : [\mathsf{f}_1 : \top, \mathsf{f}_2 : \top], \mathsf{f}_2 : \mathit{Bool}]$$

for appropriate Γ and Δ. But subspecifications of object specifications can only be derived for equal components f_1 with the rules of Section 3.

Therefore, we consider the more generous subspecification relation that also allows subspecifications in field components by replacing the rule for object specifications with the one given in Table 15.

We write $<:^*$ for this relation, and observe that $\vdash A <: B$ implies $\vdash A <:^* B$. It is still sufficient to guarantee soundness in our case as will be shown below. First, we obtain the following approximation lemma for the $<:^*$ relation.

Lemma 6.8 (Approximation). For all specifications $\Gamma;\Delta \vdash A$, the following hold:

1 For all $k \in \mathbb{N}$, $\Gamma;\Delta \vdash A <:^* A|^k$.
2 For all $k, l \in \mathbb{N}$, $\Gamma;\Delta \vdash A|^{k+l} <:^* A|^k$.
3 If A is non-recursive, there exists $n \in \mathbb{N}$ such that for all $k \geqslant n$, $A \equiv A|^k$.

Proof. The proofs are by induction on the lexicographic order on k and the number of μ in head position, and then by considering cases for the specification A. □

6.4.2. *Soundness of the subspecification*

Lemma 6.9 (Soundness of $<:^*$). If $\Gamma;\Delta \vdash A <:^* B$, $\rho \in \mathsf{Env}$ and $\eta \models \Delta$, then $[\![\Gamma;\Delta \vdash A]\!]\rho\eta \subseteq [\![\Gamma;\Delta \vdash B]\!]\rho\eta$.

Proof. We use induction on the derivation of $\Gamma;\Delta \vdash A <:^* B$:

— The cases for Reflexivity and Transitivity are immediate, as is Top.
— Fold and Unfold follow from the fact that the denotation of $\mu(X)A$ is indeed a fixed point,

$$\begin{aligned} [\![\Gamma;\Delta \vdash \mu(X)A]\!]\rho\eta &= \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X \vdash A]\!]\rho\eta[X = \chi]) && \text{by definition} \\ &= [\![\Gamma;\Delta, X \vdash A]\!]\rho(\eta[X = [\![\Gamma;\Delta \vdash \mu(X)A]\!]\rho\eta]) && \text{fixed point} \\ &= [\![\Gamma;\Delta \vdash A[\mu(X)A/X]]\!]\rho\eta && \text{by Lemma 6.6.} \end{aligned}$$

— The case of the (generalised) object subspecification rule SubObject follows by induction and is rather straightforward.
— Finally, in the case of the SubRec rule, suppose that $\Gamma;\Delta \vdash \mu(X)A <:^* \mu(Y)B$ has been derived from $\Gamma;\Delta, Y <: \top, X <: Y \vdash A <:^* B$. We use the fact that $[\![\Gamma;\Delta \vdash \mu(Y)B]\!]\rho\eta$

is the greatest post-fixed point of the map

$$f(\chi) = [\![\Gamma;\Delta, Y \vdash B]\!]\rho\eta[X = \chi],$$

which is monotonic as shown in Lemma 6.5. Since $\alpha = [\![\Gamma;\Delta \vdash \mu(X)A]\!]\rho\eta$ is a fixed point of $\lambda\chi.[\![\Gamma;\Delta \vdash A]\!]\rho\eta[X = \chi]$, we calculate

$$\begin{array}{rll} \alpha & = [\![\Gamma;\Delta, X, Y \vdash A]\!]\rho\eta[X = \alpha, Y = \alpha] & \Gamma;\Delta, X \vdash A \text{ independent of } \eta(Y) \\ & \subseteq [\![\Gamma;\Delta, X, Y \vdash B]\!]\rho\eta[X = \alpha, Y = \alpha] & \text{by induction} \\ & = f(\alpha) & \Gamma;\Delta, Y \vdash B \text{ independent of } \eta(X), \end{array}$$

which shows that α is a post-fixed point of f. Hence, $[\![\Gamma;\Delta \vdash \mu(X)A]\!]\rho\eta = \alpha \subseteq \mathsf{gfp}(f) = [\![\Gamma;\Delta \vdash \mu(Y)B]\!]\rho\eta$, as required. □

6.4.3. *Relating semantics and syntactic approximations* Taken together, Lemma 6.9 and Lemma 6.8(1) show $[\![\Gamma;\Delta \vdash A]\!]\rho\eta \subseteq [\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta$ for all $\eta \models \Delta$ and $k \in \mathbb{N}$; in particular,

$$[\![A]\!]\eta \subseteq \textstyle\bigcap_{k\in\mathbb{N}} [\![A|^k]\!]\eta \tag{23}$$

for closed specifications A. For the reverse inclusion, we use the characterisation of greatest fixed points as the meet of a descending chain, which is in close correspondence with the syntactic approximations.

Lemma 6.10 (Combining substitution and approximation). For all specifications A, B, all X such that $\Gamma;\Delta \vdash B$ and $\Gamma;\Delta, X \vdash A$, and for all $k, l \in \mathbb{N}$,

$$\Gamma;\Delta \vdash A[B/X]|^l <:^* A[B|^k/X]|^l.$$

In particular, by Lemma 6.9, $[\![\Gamma;\Delta \vdash A[B/X]|^l]\!]\rho\eta \subseteq [\![\Gamma;\Delta \vdash A[B|^k/X]|^l]\!]\rho\eta$, for all $\eta \models \Delta$.

Proof. We use induction on the lexicographic order on l and the number of μ in head position:

— $l = 0$. Clearly, $\Gamma;\Delta \vdash A[B/X]|^0 <:^* \top = A[B|^k/X]|^0$.

— $l > 0$. We consider possible cases for A:

- A is X. In this case $\Gamma;\Delta \vdash A[B/X]|^l = B|^l <:^* B|^k|^l = A[B|^k/X]|^l$.
- A is $\top, Bool$ or $Y \neq X$. In this case $\Gamma;\Delta \vdash A[B/X]|^l = A|^l <:^* A|^l = A[B|^k/X]|^l$.
- A is $[\mathsf{f}_i : A_i{}^{i=1\ldots n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1\ldots m}]$. This case again follows easily by the induction hypothesis.
- A is $\mu(Y)C$, without loss of generality Y not free in B. In this case, by the induction hypothesis, we find $\Gamma;\Delta \vdash C[A/Y][B/X]|^l <:^* C[A/Y][B|^k/X]|^l$. Using the properties of syntactic substitutions, we calculate

$$\begin{array}{rcll} A[B/X]|^l & \equiv & \mu(Y)(C[B/X])|^l & \text{substitution} \\ & \equiv & C[B/X][(\mu(Y)(C[B/X]))/Y]|^l & \text{definition of } (-)|^l \\ & \equiv & C[B/X][(A[B/X])/Y]|^l & \text{substitution} \\ & \equiv & C[A/Y][B/X]|^l & Y \text{ not free in } B, \end{array}$$

and, analogously, $C[A/Y][B|^k/X]|^l = A[B|^k/X]|^l$, which entails the result. □

Lemma 6.11 (Approximation of specifications). For all $\Gamma;\Delta \vdash A$, $\rho \in \mathsf{Env}$ and environments $\eta \models \Delta$,

$$[\![\Gamma;\Delta \vdash A]\!]\rho\eta = \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta.$$

Proof. By (23), all that remains to be shown is $[\![\Gamma;\Delta \vdash A]\!]\rho\eta \supseteq \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta$. We proceed by induction on the lexicographic order on pairs (M, A) where M is an upper bound on the number of μ-binders in A. For the base case, $M = 0$, by Lemma 6.8(3), there exists $n \in \mathbb{N}$ such that for all $k \geqslant n$, $A|^k = A$, and thus, in fact,

$$[\![\Gamma;\Delta \vdash A]\!]\rho\eta = [\![\Gamma;\Delta \vdash A|^n]\!]\rho\eta \supseteq \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta.$$

Now suppose that A contains at most $M+1$ μ binders. We consider cases for A:

— A of the form $\top$, X or *Bool*. Then, as above, there exists $n \in \mathbb{N}$ such that for all $k \geqslant n$, $A|^k = A$, and we are done.

— A is $[\mathsf{f}_i\!:\!A_i{}^{i=1\ldots n}, \mathsf{m}_j\!:\!\varsigma(y_j)B_j\!::\!T_j{}^{j=1\ldots m}]$. Then, by the induction hypothesis,

$$[\![\Gamma;\Delta \vdash A_i]\!]\rho\eta \supseteq \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A_i|^k]\!]\rho\eta$$

and

$$[\![\Gamma, y_j;\Delta \vdash B_j]\!](\rho[y_j := l])\eta \supseteq \bigcap_{k\in\mathbb{N}}[\![\Gamma, y_j;\Delta \vdash B_j|^k]\!](\rho[y_j := l])\eta$$

for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Hence, if $(l, \sigma) \in [\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta$ for all $k \in \mathbb{N}$, then

$$(\sigma.l.\mathsf{f}_i, \sigma) \in \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A_i|^k]\!]\rho\eta \subseteq [\![\Gamma;\Delta \vdash A_i]\!]\rho\eta$$

and $\sigma.l.\mathsf{m}_j(\sigma) = (v, \sigma')$ implies

$$(v, \sigma') \in \bigcap_{k\in\mathbb{N}}[\![\Gamma, y_j;\Delta \vdash B_j|^k]\!](\rho[y_j := l])\eta \subseteq [\![\Gamma, y_j;\Delta \vdash B_j]\!](\rho[y_j := l])\eta$$

by the definition of $A|^k$. This shows $(l, \sigma) \in [\![\Gamma;\Delta \vdash A]\!]\rho\eta$, as required.

— A is $\mu(X)B$. Recall that

$$[\![\Gamma;\Delta \vdash A]\!]\rho\eta = \mathsf{gfp}(f_A)$$

is the greatest *post*-fixed point of $f_A(\chi) = [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta[X = \chi]$. We show that $\alpha := \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta$ is a post-fixed point of f_A, from which

$$[\![\Gamma;\Delta \vdash A]\!]\rho\eta \supseteq \bigcap_{k\in\mathbb{N}}[\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta$$

then follows. First note that, by Lemma 6.8(2) and Lemma 6.9,

$$\eta[X = [\![\Gamma;\Delta \vdash A|^0]\!]\rho\eta] \geqslant \eta[X = [\![\Gamma;\Delta \vdash A|^1]\!]\rho\eta] \geqslant \ldots$$

forms a descending chain of environments. Hence we can calculate

$$\begin{aligned}
f_A(\alpha) &= [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta[X = \alpha] && \text{definition of } f_A \\
&= [\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_k \eta[X = [\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta]) && \text{definition of } \alpha \text{ and meets} \\
&= \textstyle\bigcap_{k\in\mathbb{N}} [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta[X = [\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta] && \text{Lemma 6.5, meets} \\
&= \textstyle\bigcap_{k\in\mathbb{N}} [\![\Gamma;\Delta \vdash B[A|^k/X]]\!]\rho\eta && \text{Lemma 6.6, substitution} \\
&\supseteq \textstyle\bigcap_{k\in\mathbb{N}} \bigcap_{l\in\mathbb{N}} [\![\Gamma;\Delta \vdash B[A|^k/X]|^l]\!]\rho\eta && \text{induction hypothesis} \\
&\supseteq \textstyle\bigcap_{l\in\mathbb{N}} [\![\Gamma;\Delta \vdash B[A/X]|^l]\!]\rho\eta && \text{Lemma 6.10} \\
&= \textstyle\bigcap_{k\in\mathbb{N}} [\![\Gamma;\Delta \vdash A|^k]\!]\rho\eta && \text{definition of } \mu(X)A|^k,
\end{aligned}$$

that is, $\alpha \subseteq f_A(\alpha)$. Note that we can apply induction in the fourth line since $A|^k$ does not contain any μ and thus $B[A|^k/X]$ contains at most M μ-binders.

This concludes the proof. □

6.5. *Soundness*

The technical development in the preceding subsection means we can now prove (22). The soundness proof of the logic extended with recursive specifications then follows from this result following the lines of the proof presented in Section 5 for non-recursive specifications.

Lemma 6.12. For all $\sigma \in \mathsf{St}$, $\Sigma \in \mathit{StSpec}$, $l \in \mathsf{Loc}$ and closed A, if $\sigma \in [\![\Sigma]\!]$ and $\vdash \Sigma.l <:^* A$, then $(l, \sigma) \in [\![A]\!]$.

Proof. The proof proceeds by considering finite specifications first. This can be proved by induction on A, as in Lemma 5.2. When applying the induction hypothesis we use the fact that $\vdash A <: B$ implies $\vdash A <:^* B$.

To extend the proof to all (possibly recursive) specifications, note that by Lemma 6.8, $\vdash A <:^* A|^k$ for all $k \in \mathbb{N}$, which entails $\vdash \Sigma.l <:^* A|^k$ for all k by transitivity. Every $A|^k$ is non-recursive, so, by the above considerations, $(l, \sigma) \in [\![A|^k]\!]$ for all k. Thus

$$(l, \sigma) \in \textstyle\bigcap_{k\in\mathbb{N}} [\![A|^k]\!] = [\![A]\!]$$

by Lemma 6.11. □

7. Outlook

The study of Abadi–Leino logic from a denotational viewpoint was not just carried out in order to advertise denotational techniques, or in the belief that it is the best logic one can devise. However, it was the first (and, to the best of our knowledge, so far the only) logic for the object-calculus, and is thus an ideal playground and starting point for our research programme.

Our long-term objective is to design a better, more powerful and more complete logic, building on the lessons learnt from analysing Abadi–Leino logic. To that end, we plan to carry out the following extensions or changes to Abadi and Leino's calculus. We strongly believe that the denotational approach will guide us in finding the right rules (and a modular correctness proof).

Local store In this paper we have worked with a global store model. Every object, its fields and methods, are visible to any other object. For Abadi–Leino logic this was sufficient, but one significant feature of object-oriented programs is *encapsulation*. Encapsulation is modelled only by a refined notion of store – and, accordingly, more refined store specifications. Reddy and Yang (2004), and Benton and Leperchey (2005) have presented such models for higher-order languages with storable references (but no higher-order store). Their models give rise to a large number of correct program equivalences, and the authors expressed the need to extend their respective models to a full object-oriented language and to specifications. Coming from the other end, we have a logic for a simple object-oriented language, but need to incorporate locality and encapsulation.

A complementary approach to information hiding is to restrict the language by imposing a *confinement* condition on programs. Banerjee and Naumann (2005) used this approach to prove representation independence for a class-based Java-like language. It should be interesting to try similar techniques for a language with higher-order store, such as the object calculus, in order to prove a restricted form of encapsulation and representation independence.

Invariants of fields Abadi and Leino's logic is peculiar in that verified programs need to preserve store specifications. Put another way, only properties that are in fact preserved can be expressed by object specifications. In particular, specifying fields in object specifications is limited. For example, we cannot formulate invariants like $\texttt{balance} \geqslant 0$, stating that an account cannot be overdrawn. Note that such an axiom in a transition specification only guarantees that the *current* balance is positive. Using a store with local fields (as described above), means such invariants can be accommodated. Invariance of such a field then has to be established for just those methods that can see it.

Method parameters Formal method parameters of the form $x{:}A$ can be attached to method specifications – for example, $\texttt{deposit}(x{:}\texttt{Int}) : \varsigma(y)[] :: T_{\texttt{deposit}}$ – by adding an extra assumption to the definition of store specifications. When $\sigma' \in [\![\Sigma']\!]$, the conditions (M1)–(M3) have to be shown for all $v \in \|A\|_{\Sigma'}$ where v is the actual parameter replacing the formal parameter x in the method call. There are limitations, however, as the resulting object specification must still allow for subspecifications. In particular, its semantics should not be defined by a recursion with negative occurrences of store.

Dynamic loading Dynamic loading of objects is, in a way, already available in the object calculus (and this is one of its advantages over class-based languages). Loading an object for which one only knows the specification $A \equiv [\mathsf{f}_i : A_i; \mathsf{m}_j : \varsigma(x_j)B_j :: T_j]$ corresponds to using a command for which one only knows the result specification A. Thus, $x : [\mathsf{load} : \varsigma(y)A :: \exists \bar{z}.\ T_{\mathsf{obj}}(\bar{\mathsf{f}} = \bar{z})] \vdash x.\mathsf{load} : A :: \exists \bar{z}.\ T_{\mathsf{obj}}(\bar{\mathsf{f}} = \bar{z})$ simulates dynamic loading where x may be thought of as a class loader, and where the load command is $x.\mathsf{load}$. If A is simply $[]$, nothing is known about the loaded object. In this case one has to assume another command to check at runtime whether a given object fulfils a given specification. This implies that specifications have to be representable in the progamming language, and thus a form of *reflection*.

Recursive specifications Recursive specifications are necessary when a field of an object or a result of one of the object's methods are supposed to satisfy the same specification as the object itself. As outlined at the beginning of Section 6, they are required for the implementation of recursive datatypes such as lists and trees. In the preceding section we described how the logic can be safely enriched by recursive specifications.

Parametric method specifications Transition specifications in Abadi–Leino logic cannot refer to methods. This is unnecessary, as method specifications are fixed at object introduction and assumed to be invariant afterwards. But for programs that, for instance, use delegations (similar to the Command pattern of Gamma *et al.* (1995)), this is not adequate: the specification in use is not known at the time of object creation but only at update (and it may change with further updates). As a remedy, one could allow placeholders for specifications (B and T) that can be shared inside objects. For example, with X and Y acting as such placeholders, $[\mathsf{f} : [\mathsf{n} : \varsigma(x)X::Y], \mathsf{m} : \varsigma(x)X::Y]$ states that m satisfies the same method specification that n satisfies. Note that only n can be updated via f. The invariance of specification still holds, it is just that every object providing a method n will meet specification $[\mathsf{n} : \varsigma(x)X::Y]$, and m will still satisfy $\mathsf{m} : \varsigma(x)X::Y$ if it is implemented as $\mathsf{m} = \varsigma(y)x.\mathsf{f}.\mathsf{n}$. One can conceive of more general transition specifications for m that assume that Y holds only for certain calls of n. The restrictions revealed by the existence theorem may turn out to be useful in finding the correct semantics for such specifications.

Method update Although method update is not allowed in Abadi–Leino logic, fields, and thus the methods in a field object, can be updated in a way similar to the Decorator pattern (Gamma *et al.* 1995) as seen in the example above. By the invariance of field components of object specifications, the object used for the update must satisfy the specification of the field to be updated. Any extra conditions that the new object may fulfil are not recorded in the logic and cannot be used later. More useful would be a 'behavioural' update in which the result and transition specifications of the overriding method are subspecifications of the original method. But this would require a relaxation of the idea of invariance of store specifications.

Invariance of store specifications The previous discussion shows the need for a logic in which object specifications are not always preserved or in which one may refine the object specifications. It seems it would be worthwhile developing a calculus that makes invariance properties explicit in the logic. Even though this may clutter proofs (for *users* of such a logic), it may reveal limitations of logics with higher-order *dynamic* store. Moreover, it is expected that Separation Logic (O'Hearn *et al.* 2001; Reynolds 2002), will be able to provide support for local reasoning. However, this first requires more research to marry Separation Logic to higher-order stores.

As a consequence, one should be able to devise more practical and more expressive program logics. It may also be instructive to derive a class-based logic by translating classes into objects (Abadi and Cardelli 1996) using the object-calculus. With this calculus it is easy to model dynamic (class) loading.

8. Conclusion

Using a denotational semantics, we have given a soundness proof for Abadi and Leino's program logic of an object-based language. Compared to the original proof, which was carried out with respect to an operational semantics, our techniques allowed us to distinguish the notions of derivability and validity. Furthermore, we have used the denotational framework to extend the logic to recursive object specifications. In comparison to a similar logic presented in Leino (1998), our notion of subspecification is structural rather than nominal.

Although our proof is very different from the original one, the nature of the logic forces us to work with store specifications too. Information for locations referenced from the environment Γ will be needed for derivations. Since the context Γ cannot reflect the dynamic aspect of the store (which is growing), one uses store specifications Σ. They do not show up in the rules of Abadi–Leino logic as they are automatically preserved by programs. This is shown as part of the soundness proof rather than by being a proof obligation on the level of derivations. In contrast with Abadi and Leino (2004), we can view store specifications as predicates on stores that need to be defined by mixed-variant recursion due to the form of the object introduction rule. Unfortunately, such recursively defined predicates do not directly admit an interpretation of either subsumption or weakening. This led us to a *positive recursive* semantics of individual objects, for which the set containment models the syntactic subspecification relation (*cf.* Lemma 3.5).

Conditions (M1)–(M3) in the semantics of store specifications ensure that methods in the store preserve not only the current store specification but also arbitrary extensions $\Sigma' \succcurlyeq \Sigma$. This accounts for the (specifications of) objects allocated between definition time and call time.

Clearly, not every predicate on stores is preserved. As we lack a semantic characterisation of those specifications that are syntactically definable (as Σ), the specification syntax appears in the definition of $\sigma \in [\![\Sigma]\!]$ (Definition 4.6). More annoyingly, field update requires subspecifications to be invariant in the field components, otherwise even type soundness is invalidated. We do not know how to express this property of object specifications semantically (on the level of predicates), and need to use the inductively defined syntactic subspecification relation instead.

The proof of Theorem 4.7, establishing the existence of store predicates, explains why transition relations of the Abadi–Leino logic express properties of the flat part of stores only: semantically, a (sufficient) condition is that transition relations are upwards and downwards closed in their first and second store argument, respectively.

In Section 7 we have described some of the limitations of Abadi–Leino logic and sketched possible improvements. The results established in this paper pave the way for this line of research.

Acknowledgements

We wish to thank Thomas Streicher for discussions and comments. The DOMAINS Workshops have been a constant source of inspiration to us, and we are grateful to Professor Klaus Keimel for having kicked off this series in Darmstadt in 1994.

References

Abadi M. and Cardelli, L. (1996) *A Theory of Objects*, Springer.

Abadi, M. and Leino, K. R. M. (1997) A logic of object-oriented programs. In: Bidoit, M. and Dauchet, M. (eds.) Proceedings of Theory and Practice of Software Development. *Springer-Verlag Lecture Notes in Computer Science* **1214** 682–696.

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

Amadio, R. M. and Cardelli, L. (1993) Subtyping recursive types. *ACM Transactions on Programming Languages and Systems* **15** (4) 575–631.

Apt, K. R. (1981) Ten years of Hoare's logic: A survey – part I. *ACM Transactions on Programming Languages and Systems* **3** (4) 431–483.

Banerjee, A. and Naumann, D. A. (2005) Ownership confinement ensures representation independence for object-oriented programs. *Journal of the ACM* **52** (6).

Benton, N. and Leperchey, B. (2005) Relational reasoning in a nominal semantics for storage. In: Urzyczyn, P. (ed.) Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA'05). *Springer-Verlag Lecture Notes in Computer Science* **3461** 86–101.

Davey, B. A. and Priestley, H. A. (2002) *Introduction to Lattices and Order*, Cambridge University Press, second edition.

de Boer, F. S. (1999) A WP-calculus for OO. In: Thomas, W. (ed.) Foundations of Software Science and Computation Structures. *Springer-Verlag Lecture Notes in Computer Science* **1578** 135–149.

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

Hensel, U., Huisman, M., Jacobs, B. and Tews, H. (1998) Reasoning about classes in object-oriented languages: Logical models and tools. In: Hankin, C. (ed.) Proceedings of the 7th European Symposium on Programming (ESOP'98). *Springer-Verlag Lecture Notes in Computer Science* **1381** 105–121.

Hoare, C. A. R. (1969) An Axiomatic Basis of Computer Programming. *Communications of the ACM* **12** 576–580.

Jacobs, B. and Poll, E. (2001) A logic for the Java modeling language JML. In: Hussmann, H. (ed.) Fundamental Approaches to Software Engineering (FASE'01). *Springer-Verlag Lecture Notes in Computer Science* **2029** 284–299.

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

Leino, K. R. M. (1998) Recursive object types in a logic of object-oriented programs. In: Hankin, C. (ed.) Proceedings of the 7th European Symposium on Programming (ESOP'98). *Springer-Verlag Lecture Notes in Computer Science* **1381** 170–184.

Levy, P. B. (2002) Possible world semantics for general storage in call-by-value. In: Bradfield, J. (ed.) Proceedings of the 16th Workshop on Computer Science Logic (CSL'02). *Springer-Verlag Lecture Notes in Computer Science* **2471**.

Levy, P. B. (2004) Call-By-Push-Value. A Functional/Imperative Synthesis. *Semantic Structures in Computation* **2**, Kluwer.

Moggi, E. (1990) An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh.

O'Hearn, P. W., Reynolds, J. C. and Yang, H. (2001) Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) Computer Science Logic (CSL '01). *Springer-Verlag Lecture Notes in Computer Science* **2142** 1–19.

Paulson, L. C. (1987) *Logic and Computation: Interactive proof with Cambridge LCF*, Cambridge Tracts in Theoretical Computer Science **2**, Cambridge University Press.

Pitts, A. M. (1996) Relational properties of domains. *Information and Computation* **127** 66–90.

Poetzsch-Heffter, A. and Müller, P. (1999) A programming logic for sequential Java. In: Swierstra, S. D. (ed.) Proceedings of the 8th European Symposium on Programming (ESOP'99). *Springer-Verlag Lecture Notes in Computer Science* **1576** 162–176.

Reddy, U. S. (2002) Objects and classes in Algol-like languages. *Information and Computation* **172** (1) 63–97.

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

Reus, B. (2002) Class-based versus object-based: A denotational comparison. In: Kirchner, H. and Ringeissen, C. (eds.) Proceedings of the 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02). *Springer-Verlag Lecture Notes in Computer Science* **2422** 473–488.

Reus, B. (2003) Modular semantics and logics of classes. In: Baatz, M. and Makowsky, J. A. (eds.) 17th Workshop on Computer Science Logic (CSL'03). *Springer-Verlag Lecture Notes in Computer Science* **2803** 456–469.

Reus, B. and Schwinghammer, J. (2005) Denotational semantics for Abadi and Leino's logic of objects. In: Sagiv, M. (ed.) Proceedings of the European Symposium on Programming (ESOP'05). *Springer-Verlag Lecture Notes in Computer Science* **3444** 264–279.

Reus, B. and Streicher, T. (2004) Semantics and logic of object calculi. *Theoretical Computer Science* **316** 191–213.

Reus, B., Wirsing, M. and Hennicker, R. (2001) A Hoare-Calculus for Verifying Java Realizations of OCL-Constrained Design Models. In: Hussmann, H. (ed.) Fundamental Approaches to Software Engineering (FASE'01). *Springer-Verlag Lecture Notes in Computer Science* **2029** 300–317.

Reynolds, J.C. (2002) Separation Logic: A Logic for Shared Mutable Data Structures. In: *LICS*, IEEE Computer Society 55–74.

Shinwell, M. R. and Pitts, A. M. (2005) On a monadic semantics for freshness. *Theoretical Computer Science* **342** (1) 28–55.

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

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

Tang, F. and Hofmann, M. (2002) Generation of verification conditions for Abadi and Leino's logic of objects. Presented at 9th International Workshop on Foundations of Object-Oriented Languages.

von Oheimb, D. (2001) Hoare logic for Java in Isabelle/HOL. *Concurrency and Computation: Practice and Experience* **13** (13) 1173–1214.