Denotational Semantics for Abadi and Leino's Logic of Objects

Bernhard Reus
Jan Schwinghammer

Report 2004:03 December 2004

Department of Informatics
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

Denotational Semantics for Abadi and Leino's Logic of Objects

Bernhard Reus and Jan Schwinghammer

Abstract. Abadi-Leino Logic is a Hoare-calculus style logic for a simple imperative and object-based language where every object comes with its own method suite. Consequently, methods need to reside in the store ("higher-order store"). We present a new soundness proof for this logic using a denotational semantics where object specifications are recursive predicates on the domain of objects. Our semantics reveals which of the limitations of Abadi and Leino's logic are deliberate design decisions and which follow from the use of higher-order store. We discuss the implications for the development of other, more expressive, program logics.

Contents

Supported by the EPSRC under grant GR/R65190/01, "Programming Logics for Denotations of Recursive Objects"

1 Introduction

When Hoare presented his seminal work about an *axiomatic basis of computer programming* [7], high-level languages had just started to gain broader acceptance. While programming languages are evolving ever more rapidly, verification techniques seem to be struggling to keep up. For object-oriented languages several formal systems have been proposed, e.g. [2, 6, 13, 12, 5, 20, 17]. A "standard" comparable to the Hoare-calculus for imperative While-languages [4] 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 Abadi and Leino's work [2] where a logic for an object-based language is introduced that is derived from the imperative object calculus with first-order types, **imp**ς, [1]. 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 called *higher-order store*) and modularity is for free simply by the fact that all programs can depend on the objects' code in the store. We therefore consider object-based languages ideal for studying modularity issues that occur also in class-based languages. Class-based programs can be compiled into object-based ones (see [1]), and object-based languages can naturally deal with classes defined on-the-fly, like inner classes and classes loaded at run-time (cf. [14, 15]).

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 in case of program termination. Informally, an object satisfies such a specification

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

if it has fields f_i satisfying A_i and methods m_j that satisfy the transition relation T_j and, in case of termination of the method invocation, 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.[1]

We are going to present a new proof using a (untyped) denotational semantics (of the language and the logic) to define validity. Every program and every specification have a meaning, a *denotation*. Those of specifications are simply predicates on (the domain of) objects. The properties of these predicates provide

[1]This also works for class-based languages. But an easier solution for those is to interpret class specifications as mutually defined predicates over classes (and their class names).

a description of inherent limitations of the logic. Such an approach is not new, it has been used e.g. in LCF, a logic for functional programs [10].

The difficulty in this case is to establish predicates that provide the powerful reasoning principle for objects. Reus and Streicher have outlined in [16] how to use some classic domain theory [11] 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 its methods) in the store. So object specifications must depend on specifications of other objects in the store which gives rise to "store specifications" (already present in the work of Abadi and Leino).

For the reasons given above, this paper is not "just" an application of the ideas in [16]. 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 [16] to the logic of [2] to obtain a soundness proof that can be used to *analyse this logic* and to *develop similar but more powerful program logics* as well.

Our soundness proof is not just "yet another proof" either. We consider it complementary (if not superior) to the one in [2] which relies on the operational semantics of the object calculus and does not assign proper "meaning" to specifications. Our claim is backed up by the following reasons:

- By using denotational semantics we can introduce a clear notion of validity with no reference to derivability. This helps clarifying *what the proof is actually stating* in the first place.
- We can extend the logic easily e.g. for recursive specifications. This has been done for the Abadi-Leino logic in [8] but for a slightly different language with nominal subtyping.
- Some essential and unavoidable restrictions of the logic are revealed and justified.
- Analogously, it is revealed where restrictions have been made for the sake of simplicity that could be lifted to obtain a more powerful logic. For example, in [2] transition specifications cannot talk about methods at all.
- Our proof widens the audience for Abadi and Leino's work to semanticists and domain theorists.

The outline of this report is as follows. In the next section, syntax and semantics of the object-calculus are presented. Section 3 introduces the Abadi-Leino logic and the denotational semantics of its object specifications. It follows a discussion about store specifications and their semantics (Section 4). The main result is in Section 5 where the logic is proved sound. Finally we sketch how

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

Table 1. Syntax

recursive specifications can be introduced (Section 6) and discuss the benefits of the denotational approach (Section 7).

When presenting the language and logic, we deliberately keep close to the original presentation [2].

2 The Object Calculus

Below, we review the language of [2], which is based on the imperative object calculus of Abadi and Cardelli [1]. Following [16] we give a denotational semantics in Section 2.2.

2.1 Syntax

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

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

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 m_j, ς is a binder, binding 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 below.

Note that in contrast to [1] we distinguish between fields and methods, and that method update is disallowed. Also note that we restrict the cases for field selection, field update, method invocation and if statement to contain only variables (instead of arbitrary object terms). This is no real limitation because of the let construct[2], but it simplifies the statement of the rules of the logic [2].

2.2 Semantics of Objects

Preliminaries

We work in the category PreDom of predomains and partial continuous functions. Let $A \rightharpoonup B$ denote the partial continuous function space between predomains 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 $\mathcal{P}(L)$ is its powerset, $\mathcal{P}_{\mathsf{fin}}(L)$ 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 predomain A we write

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

for the predomain of *records* with *entries* from A and *labels* from $\mathbb{L}$. Note that $\mathsf{Rec}_{\mathbb{L}}$ extends to a locally continuous endofunctor on PreDom.

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

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

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 $l \in D$.

Interpretation

The language of the previous section finds its interpretation within the following system of recursively defined predomains in PreDom

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

Here, Loc is some countably infinite set of *locations* ranged over by l, and BVal is the set of truth values *true* and *false*, considered as flat predomains. The functor $F_{Store} : \mathsf{PreDom}^{op} \times \mathsf{PreDom} \to \mathsf{PreDom}$ associated with (1),

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

is locally continuous bifunctor, and there exists indeed a minimal invariant solution St s.t. $F_{Store}(\mathsf{St}, \mathsf{St}) = \mathsf{St}$ (see, e.g., [11, 18]).

[2]we use a more generous syntax in the examples

$$
\begin{array}{ll}
[\![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 & = (\mathit{true}, \sigma) \\
[\![\mathtt{false}]\!]\rho\sigma & = (\mathit{false}, \sigma) \\
[\![\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' \\
\multicolumn{2}{l}{[\![[\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 \le i \le n \\ (\mathsf{error}, \sigma) & \text{otherwise} \end{cases} \\
 & \quad \text{where } l \notin \mathsf{dom}(\sigma) \\
 & \qquad\qquad o_1 = \{\!|\mathsf{f}_i = \rho(x_i)|\!\}^{i=1\ldots n} \\
 & \qquad\qquad o_2 = \{\!|\mathsf{m}_j = \lambda\sigma.[\![b_j]\!]\rho[y_j := l]\sigma|\!\}^{j=1\ldots m} \\
[\![x.\mathsf{f}]\!]\rho\sigma & = \ \mathbf{let}\ (l, \sigma') = [\![x]\!]\rho\sigma \\
 & \quad \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, (v, \sigma'') = [\![y]\!]\rho\sigma' \\
 & \quad \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 \\
 & \quad \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}
\end{array}
$$

TABLE 2. Denotational semantics

Let $\mathsf{Env} = \mathsf{Var} \to_{\mathsf{fin}} \mathsf{Val}$ be the set of *environments*, i.e. maps between Var 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" wrt. error:

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

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 $\mathsf{in}_{\mathsf{Loc}}(l)$ etc. Observe that, in contrast to [16], we distinguish between non-termination (undefinedness) and exceptional termination, error. Finally, because Loc is assumed to be infinite, the condition $l \notin \mathsf{dom}(\sigma)$ in the case for object creation can always be satisfied. Therefore object creation will never raise error.

We will also use a projection to the part of the store that contains data in Val only (i.e., forget about the closures that live in the store), $\pi_{\mathsf{Val}} : \mathsf{St} \to \mathsf{St}_{\mathsf{Val}}$ defined by $(\pi_{\mathsf{Val}}\,\sigma).l.\mathsf{f} = \sigma.l.\mathsf{f}$, where $\mathsf{St}_{\mathsf{Val}} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathcal{F}}(\mathsf{Val}))$. We refer to $\pi_{\mathsf{Val}}(\sigma)$ as the *flat part* of σ.

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.*

3 Abadi-Leino Logic

We recall the logic of Abadi and Leino [2] (see also [19]).

3.1 Transition Relations and Specifications

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

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

There is a constant for each field name $\mathsf{f} \in \mathcal{F}$ (which we just write f, too), and constants result, true and false. Intuitively, the function $\mathsf{sel}_{pre}(x, y)$ 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 connectives such as *False* and implication can be defined as abbreviations.

Specifications combine transition relations for each method as well as the result types into a single specification for the whole object. They generalise the first-order types of [1], and are

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

In the case of an object specification, $\varsigma(y_j)$ binds the variable y_j in B_j and T_j, and 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, in case of termination of the method invocation, their 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 it is possible to 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 write $\emptyset$ for the empty context. Given Γ, we write $[\Gamma]$ for the list of variables occurring in Γ:

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

If clear from 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:

$$x_1, \ldots, x_n \vdash T, \text{ if all the free variables of } T \text{ appear in } x_1, \ldots, x_n$$

- for well-formed specifications:

$$\overline{x} \vdash Bool \quad \text{and} \quad \frac{\begin{array}{c} A \equiv [\mathsf{f}_i{:}\, A_i{}^{i=1\ldots n},\ \mathsf{m}_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}] \\ \overline{x} \vdash A_i{}^{i=1\ldots n} \quad \overline{x}, y_j \vdash B_j{}^{j=1\ldots m} \quad \overline{x}, y_j \vdash T_j{}^{j=1\ldots m} \end{array}}{\overline{x} \vdash [\mathsf{f}_i{:}\, A_i{}^{i=1\ldots n},\ \mathsf{m}_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]}$$

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

$$\emptyset \vdash \mathsf{ok} \quad \text{and} \quad \frac{\Gamma \vdash \mathsf{ok} \quad [\Gamma] \vdash A \quad x \notin \mathsf{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathsf{ok}}$$

In case A is closed we may simply write A instead of $\vdash A$, and similarly for closed T.

3.2 *Abadi-Leino Logic*

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

$$\frac{\begin{array}{c} \overline{x} \vdash A_i{}^{i=1\ldots n+p} \quad \overline{x}, y_j \vdash B_j{}^{j=m+1\ldots m+q} \quad \overline{x}, y_j \vdash T_j{}^{j=1\ldots m+q} \quad \overline{x}, y_j \vdash T'_j{}^{j=1\ldots m} \\ \overline{x}, y_j \vdash B_j <: B'_j{}^{j=1\ldots m} \quad \vdash_{\mathsf{fo}} T_j \Rightarrow T'_j{}^{j=1\ldots m} \end{array}}{\overline{x} \vdash [\mathsf{f}_i{:}\, A_i{}^{i=1\ldots n+p},\ \mathsf{m}_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m+q}] <: [\mathsf{f}_i{:}\, A_i{}^{i=1\ldots n},\ \mathsf{m}_j{:}\, \varsigma(y_j)B'_j{::}T'_j{}^{j=1\ldots m}]}$$

where $\vdash_{\mathsf{fo}} \varphi$ denotes provability in first-order logic (in the theory with axioms for equality, and axioms stating that `true`, `false` and all $\mathsf{f} \in \mathcal{F}$ are distinct). Just as subtyping in the corresponding type system [1], the subspecification relation is covariant along method specifications and transition relations, and invariant in field specifications. Observe that $\overline{x} \vdash A_1 <: A_2$ in particular 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 $[\Gamma]$. We use the following transition relations in the rules:

$$\begin{aligned}
T_{\mathsf{res}}(e) &\equiv \mathsf{result} = e \\
&\quad \wedge \forall x, f.\mathsf{alloc}_{pre}(x) \leftrightarrow \mathsf{alloc}_{post}(x) \wedge \mathsf{sel}_{pre}(x,f) = \mathsf{sel}_{post}(x,f) \\
T_{\mathsf{obj}}(\mathsf{f}_i = x_i)^{i=1\ldots n} &\equiv \neg\mathsf{alloc}_{pre}(\mathsf{result}) \wedge \mathsf{alloc}_{post}(\mathsf{result}) \\
&\quad \wedge \forall x, f.x \neq \mathsf{result} \rightarrow \\
&\qquad (\mathsf{alloc}_{pre}(x) \leftrightarrow \mathsf{alloc}_{post}(x) \wedge \mathsf{sel}_{pre}(x,f) = \mathsf{sel}_{post}(x,f)) \qquad (2) \\
&\quad \wedge \mathsf{sel}_{post}(\mathsf{result},\mathsf{f}_1) = x_1 \wedge \cdots \wedge \mathsf{sel}_{post}(\mathsf{result},\mathsf{f}_n) = x_n \\
T_{\mathsf{upd}}(x,f,e) &\equiv \forall x'.\mathsf{alloc}_{pre}(x') \leftrightarrow \mathsf{alloc}_{post}(x') \wedge \mathsf{sel}_{post}(x,f) = e \\
&\quad \wedge \forall x', f'.(x' \neq x \wedge f' \neq f) \rightarrow \mathsf{sel}_{pre}(x',f') = \mathsf{sel}_{post}(x',f') \\
&\quad \wedge \mathsf{result} = x
\end{aligned}$$

$T_{\mathsf{res}}(e)$ states that the result of a computation is e and the flat part of the store remains unchanged. $T_{\mathsf{obj}}(\mathsf{f}_i = x_i)$ 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 result. $T_{\mathsf{upd}}(x,f,e)$ describes the effect on the store when updating field $x.f$.[3]

Example 3.1. *Fig. 1 shows a specification for bank accounts as in the previous example.*[4] *Observe how the specification* $T_{\mathtt{interest}}$ *depends not only on the self parameter y of the host object but also on the statically enclosing object x.*

There is one rule for each syntactic form of the language, plus a subsumption rule, which generalises the consequence rule of classical Hoare logic. The rules are given in Tab. 3.

As indicated before, one of the most interesting and powerful rules of the logic is the object introduction rule,

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

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,

[3]In [2] T_{res} is called *Res* and T_{upd} is called *Update*. There is no abbreviation corresponding to T_{obj}.

[4]Note that 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.

subsumption

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

variable

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

booleans

$$\frac{\Gamma \vdash \mathsf{ok}}{\Gamma \vdash \mathtt{false}{:}\mathit{Bool}{::}T_{\mathsf{res}}(\mathsf{false})} \qquad \frac{\Gamma \vdash \mathsf{ok}}{\Gamma \vdash \mathtt{true}{:}\mathit{Bool}{::}T_{\mathsf{res}}(\mathsf{true})}$$

conditional

$$\frac{\begin{array}{c} A[\mathsf{true}/x] \equiv A_t[\mathsf{true}/x] \text{ and } A[\mathsf{false}/x] \equiv A_f[\mathsf{false}/x] \\ T[\mathsf{true}/x] \equiv T_t[\mathsf{true}/x] \text{ and } T[\mathsf{false}/x] \equiv T_f[\mathsf{false}/x] \\ \Gamma \vdash x{:}\mathit{Bool}{::}T_{\mathsf{res}}(x) \quad \Gamma \vdash a{:}A_t{::}T_t \quad \Gamma \vdash b{:}A_f{::}T_f \end{array}}{\Gamma \vdash \mathtt{if}\ x\ \mathtt{then}\ a\ \mathtt{else}\ b{:}A{::}T}$$

let

$$\frac{\begin{array}{c} \Gamma \vdash a{:}A'{::}T' \quad \Gamma, x{:}A' \vdash b{:}B{::}T'' \quad [\Gamma] \vdash B \quad [\Gamma] \vdash T \\ \vdash_{\mathsf{fo}} T'[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{post}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{post}(\cdot), x/\mathsf{result}] \\ \wedge\ T''[\mathsf{sel}_{int}(\cdot,\cdot)/\mathsf{sel}_{pre}(\cdot,\cdot), \mathsf{alloc}_{int}(\cdot)/\mathsf{alloc}_{pre}(\cdot)] \to T \end{array}}{\Gamma \vdash \mathtt{let}\ x = a\ \mathtt{in}\ b{:}B{::}T}$$

object construction

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

field selection

$$\frac{\Gamma \vdash x{:}[\mathsf{f}{:}A]{::}T_{\mathsf{res}}(x)}{\Gamma \vdash x.\mathsf{f}{:}A{::}T_{\mathsf{res}}(\mathsf{sel}_{pre}(x, \mathsf{f}))}$$

field update

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

method invocation

$$\frac{\Gamma \vdash x{:}[\mathsf{m}{:}\varsigma(y)A{::}T]{::}T_{\mathsf{res}}(x)}{\Gamma \vdash x.\mathsf{m}{:}A[x/y]{::}T[x/y]}$$

TABLE 3. Inference rules of Abadi-Leino logic

$$\begin{aligned}
T_{\texttt{deposit}}(y) &\equiv \exists z.z = \mathsf{sel}_{pre}(y, \texttt{balance}) \\
&\quad \wedge T_{\mathsf{upd}}(y, \texttt{balance}, z + 10)
\end{aligned}$$

$$\begin{aligned}
T_{\texttt{interest}}(x, y) &\equiv \exists z.z = \mathsf{sel}_{pre}(y, \texttt{balance}) \\
&\quad \wedge \exists m.m = \mathsf{sel}_{pre}(x, \texttt{manager}) \\
&\quad \wedge \exists r.r = \mathsf{sel}_{pre}(m, \texttt{rate}) \\
&\quad \wedge T_{\mathsf{upd}}(y, \texttt{balance}, z * r/100)
\end{aligned}$$

$$T_{\texttt{create}}(x) \equiv T_{\mathsf{obj}}(\texttt{balance} = 0)$$

$$\begin{aligned}
A_{\texttt{Account}}(x) \equiv [&\texttt{balance} : \mathit{Int}, \\
&\texttt{deposit10} : \varsigma(y)[] :: T_{\texttt{deposit}}(y), \\
&\texttt{interest} : \varsigma(y)[] :: T_{\texttt{interest}}(x, y)]
\end{aligned}$$

$$\begin{aligned}
A_{\texttt{AccFactory}} \equiv [&\texttt{manager} : [\texttt{rate} : \mathit{Int}], \\
&\texttt{create} : \varsigma(x) A_{\texttt{Account}}(x) :: T_{\texttt{create}}(x)]
\end{aligned}$$

$$\begin{aligned}
A_{\texttt{Manager}} \equiv [&\texttt{rate} : \mathit{Int}, \\
&\texttt{accFactory} : A_{\texttt{AccFactory}}]
\end{aligned}$$

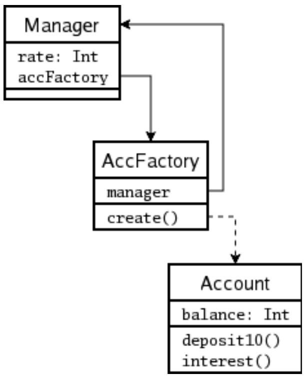

FIGURE 1. An example of transition and result specifications

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 relation symbols, $\mathsf{sel}_{int}(\cdot, \cdot)$ and $\mathsf{alloc}_{int}(\cdot)$, to capture the intermediate state of the store. It uses textual substitution of *predicate* symbols to compose the first and second transition relation. The side condition $[\Gamma] \vdash T$ ensures that the transition relation in the conclusion does not export this intermediate state.

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 an example of this see the transition relations in (2)), and we write $\mathsf{Env}^+ = \mathsf{Var} \rightarrow_{\mathsf{fin}} (\mathsf{Val} + \mathcal{F})$ when interpreting transition relations:

$$[\![\overline{x} \vdash T]\!] : \mathsf{Env}^+ \rightarrow \mathcal{P}(\mathsf{St}_{\mathsf{Val}} \times \mathsf{Val} \times \mathsf{St}_{\mathsf{Val}})$$

This can be defined in a straightforward way, a few typical cases are given in Tab. 4. Note that even though expressions may be undefined (e.g., because of referring 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. Therefore we may omit the environment and simply write $[\![T]\!]$ for closed T.

Intuitively, an object specification $\overline{x} \vdash A$ gives rise to a predicates that depends on values for the free variables. However, since the underlying logic in

$$[\![\overline{x} \vdash e]\!] : \mathsf{Env}^+ \to \mathsf{St}_{\mathsf{Val}} \to \mathsf{Val} \to \mathsf{St}_{\mathsf{Val}} \rightharpoonup (\mathsf{Val} + \mathcal{F})$$

$$\begin{array}{ll}
[\![\overline{x} \vdash x]\!]\rho\sigma v\sigma' & = \begin{cases} \rho(x) & \text{if } x \in \mathsf{dom}(\rho) \\ \text{undefined} & \text{otherwise} \end{cases} \\
[\![\overline{x} \vdash \mathsf{f}]\!]\rho\sigma v\sigma' & = \mathsf{f} \\
[\![\overline{x} \vdash \mathsf{result}]\!]\rho\sigma v\sigma' & = v \\
[\![\overline{x} \vdash \mathsf{true}]\!]\rho\sigma v\sigma' & = \mathit{true} \\
[\![\overline{x} \vdash \mathsf{false}]\!]\rho\sigma v\sigma' & = \mathit{false} \\
[\![\overline{x} \vdash \mathsf{sel}_{pre}(e_0, e_1)]\!]\rho\sigma v\sigma' & = \begin{cases} \sigma.l.\mathsf{f} & \text{if } [\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma' = l \in \mathsf{Loc} \text{ and} \\ & \quad [\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma' = \mathsf{f} \in \mathcal{F} \text{ are defined} \\ \text{undefined} & \text{otherwise} \end{cases} \\
[\![\overline{x} \vdash \mathsf{sel}_{post}(e_0, e_1)]\!]\rho\sigma v\sigma' & = \begin{cases} \sigma'.l.\mathsf{f} & \text{if } [\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma' = l \in \mathsf{Loc} \text{ and} \\ & \quad [\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma' = \mathsf{f} \in \mathcal{F} \text{ are defined} \\ \text{undefined} & \text{otherwise} \end{cases}
\end{array}$$

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

$$\begin{array}{lll}
(\sigma, v, \sigma') \in [\![\overline{x} \vdash e_0 = e_1]\!]\rho & \textit{iff} & \text{both } [\![\overline{x} \vdash e_0]\!]\rho\sigma v\sigma' \text{ and } [\![\overline{x} \vdash e_1]\!]\rho\sigma v\sigma' \text{ are defined} \\ & & \quad \text{and equal, or both undefined} \\
(\sigma, v, \sigma') \in [\![\overline{x} \vdash \mathsf{alloc}_{pre}(e)]\!]\rho & \textit{iff} & [\![\overline{x} \vdash e]\!]\rho\sigma v\sigma' \in \mathsf{dom}(\sigma) \\
(\sigma, v, \sigma') \in [\![\overline{x} \vdash \mathsf{alloc}_{post}(e)]\!]\rho & \textit{iff} & [\![\overline{x} \vdash e]\!]\rho\sigma v\sigma' \in \mathsf{dom}(\sigma') \\
(\sigma, v, \sigma') \in [\![\overline{x} \vdash \forall x.T]\!]\rho & \textit{iff} & \text{for all } u \in \mathsf{Val} + \mathcal{F}.\ (\sigma, v, \sigma') \in [\![\overline{x}, x \vdash T]\!]\rho[x := u]
\end{array}$$

Table 4. Meaning of expressions and transition relations

the transition relations is untyped, the types of the free variables are not relevant.The interpretation of object specifications $\overline{x} \vdash A$,

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

is given in Tab. 5.

We begin with a number of observations about the interpretation.

Lemma 3.2. *For all specifications $\overline{x} \vdash A$, all $\sigma \in \mathsf{St}$ and environments ρ we have* $(\mathsf{error}, \sigma) \notin [\![\overline{x} \vdash A]\!]\rho$.

Proof. Immediate from the definition of $[\![\overline{x} \vdash A]\!]\rho$. □

Lemma 3.3 (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. This follows by 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 $\vdash_{\mathsf{fo}} T \to T'$ implies

$$[\![\overline{x} \vdash T]\!]\rho \subseteq [\![\overline{x} \vdash T']\!]\rho \qquad (3)$$

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

$$[\![\overline{x} \vdash \mathit{Bool}]\!]\rho = \mathsf{BVal} \times \mathsf{St}$$

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

$$= \left\{ \begin{array}{c|l} (l,\sigma) \in & (i)\quad \text{for all } 1 \le i \le n.\ \sigma.l.\mathsf{f}_i \in [\![\overline{x} \vdash A_i]\!]\rho \\ \mathsf{Loc} \times \mathsf{St} & (ii)\quad \text{for all } 1 \le j \le m,\ \text{if } \sigma.l.\mathsf{m}_j(\sigma) = (v,\sigma')\downarrow \\ & \qquad \text{then } (v,\sigma') \in [\![\overline{x}, y_j \vdash B_j]\!]\rho[y_j := l] \\ & \qquad \text{and } (\pi_{\mathsf{Val}}(\sigma), v, \pi_{\mathsf{Val}}(\sigma')) \in [\![\overline{x}, y_j \vdash T_j]\!]\rho[y_j := l] \end{array} \right\}$$

TABLE 5. Semantics of specifications

for all $\rho \in \mathsf{Env}^+$. However, (3) follows immediately since $\vdash_{\mathsf{fo}}$ holds in *all* models. $\square$

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

4 Store Specifications

Object specifications are not sufficient. This is a phenomenon of languages with higher-order store well known from subject reduction and type soundness proofs (see [1, Ch. 11], [9]). Since statements may call subprograms residing in the store, the store has to be checked as well. However, it may contain loops and therefore induction on the reachable part of the store is unavailable.

The standard remedy – also used in [2] – is to relativise the typing judgement such that it only needs to hold for "verified" stores. In other words, judgements are interpreted w.r.t. *store specifications*. A store specification assigns a specification to each location in a store. When an object is created, the specification assigned to it at the time of creation is included in the store specification.

In this section we will interpret such store specifications using the techniques from [16]. Since their denotations will rely on mixed-variant recursion, it is impossible to define a semantic notion of subspecification for stores. Alas, the Abadi-Leino logic makes essential use of subspecifications.

We get around this problem by only using a subset relationship on (denotations of) object specifications (where there is no contravariant occurrence of store as the semantics of objects is w.r.t. one fixed store, cf. Tab. 5).

Unfortunately, we are restricted by the logic's requirement that verified statements never break the validity of store specifications. In the case of field update this implies that subspecifications[5] need to be invariant in their fields. As the semantic interpretation of the subspecification relation cannot reflect this, we were forced to sometimes use syntactic subspecifications.

[5]this also holds for subtypes

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* $l \in \mathit{dom}(\Sigma)$, $\Sigma.l = A$ *is a closed object specification.*

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 in general will 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 define $[\![\overline{x} \vdash \underline{l}]\!]\rho = l$ (cf. Tab. 4). Similarly, we set $\underline{\mathit{true}} = \mathtt{true}$ and $\underline{\mathit{false}} = \mathtt{false}$. When clear from context we will simply write v in place of $\underline{v}$.

Further we write $A[\rho/\overline{x}]$ (and $A[\rho/\Gamma]$, resp.) for the simultaneous substitution of all $x \in \overline{x}$ ($x \in [\Gamma]$, resp.) 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*

1. $\vdash T[\rho/\overline{x}]$ *and* $[\![\vdash T[\rho/\overline{x}]]\!] = [\![\overline{x} \vdash T]\!]\rho$
2. $\vdash A[\rho/\overline{y}]$ *and* $[\![\vdash A[\rho/\overline{y}]]\!] = [\![\overline{y} \vdash A]\!]\rho$
3. *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* $\Sigma, \Sigma' \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$ *be store specifications.* Σ' extends Σ, *written* $\Sigma' \geqslant \Sigma$, *if* $\Sigma.l = \Sigma'.l$ *for all* $l \in \mathit{dom}(\Sigma)$.

Note that $\geqslant$ 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 Σ is a store specification. For closed $\vdash A$ let $\|A\|_\Sigma \subseteq \mathsf{Val}$ be defined by*

$$\|\mathit{Bool}\|_\Sigma = \mathsf{BVal}$$
$$\|A\|_\Sigma = \{l \in \mathsf{Loc} \mid\ \vdash \Sigma.l <: A\}$$

where $A \equiv [f_i{:}\, A_i{}^{i=1\ldots n},\ \mathsf{m}_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$ *and* $\vdash A$. *We extend this to contexts in the natural way:*

$$\rho \in \|\emptyset\|_\Sigma \ \textit{for all}\ \rho \in \mathsf{Env},$$
$$\rho \in \|\Gamma, x{:}A\|_\Sigma \ \textit{iff}\ \rho \in \|\Gamma\|_\Sigma \ \textit{and}\ \rho(x) \in \|A[\rho/\Gamma]\|_\Sigma$$

Observe that for all A, if $\Sigma' \succcurlyeq \Sigma$ then $\|A\|_\Sigma \subseteq \|A\|_{\Sigma'}$. We obtain the following lemma about *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 $B[\rho[x := v]/\Gamma] \equiv B[\rho/\Gamma]$. □

We want to interpret store specifications as predicates over stores, as follows.

Definition 4.6 (Store Predicate, Tentative). *Let $\mathcal{P} = \mathcal{P}(\mathsf{St})^{\mathsf{Rec}_{\mathsf{Loc}}(Spec)}$ denote the collection of predicates on St, indexed by store specifications. We define a functional $\Phi : \mathcal{P}^{op} \times \mathcal{P} \to \mathcal{P}$ as follows.*

$\sigma \in \Phi(Y, X)_\Sigma :\Leftrightarrow$
  $\forall l \in \mathit{dom}(\Sigma)$ *where* $\Sigma.l = [f_i{:}\, A_i{}^{i=1\ldots n},\ m_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$:
    (F) $\sigma.l.f_i \in \|A_i\|_\Sigma$, *for all* $1 \leq i \leq n$, *and*
    $\forall \Sigma' \succcurlyeq \Sigma\ \forall \sigma' \in Y_{\Sigma'}\ \forall v \in \mathsf{Val}\ \forall \sigma'' \in \mathsf{St}$, *if* $\sigma.l.m_j(\sigma') = (v, \sigma'')\downarrow$ *then*
    (M1) $(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![T_j[l/y_j]]\!]$
    (M2) *there exists* $\Sigma'' \succcurlyeq \Sigma'$ *s.t.* $\sigma'' \in X_{\Sigma''}$
    (M3) $v \in \left\|B_j[l/y_j]\right\|_{\Sigma''}$
      for all $1 \leq j \leq m$

Then we write $[\![\Sigma]\!]$ *for* $\mathit{fix}(\Phi)_\Sigma$.

4.2 On the Existence of Store Specifications

The contravariant occurrence in this definition, Y, 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.

Indeed, we cannot show existence of $\mathit{fix}(\Phi)$ with the techniques of [11] as it stands. The problem is the existential quantification in **(M2)** which has the effect that Φ does not necessarily map *admissible* predicates on St to admissible predicates. To see this, consider the following example.

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. In case 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, and so a proper extension of the original store specification Σ has to be found.

So let $A_0 \equiv [\mathsf{m}_1 : \varsigma(y)[]::\mathit{False}]$ and $A_{i+1} \equiv [\mathsf{m}_1 : \varsigma(y)A_i::\mathit{True}]$. 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 $x.\mathsf{m}_1.\mathsf{m}_1 \dots \mathsf{m}_1$ 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 therefore do *not* satisfy A_{i-1}. Then, the limit of this chain is an object x for which an arbitrary number of calls $x.\mathsf{m}_1.\mathsf{m}_1 \dots \mathsf{m}_1$ terminates, and which therefore does not satisfy *any* of the A_i:

Set $\Sigma''_i \equiv \Sigma, l : A_i$ and let $\underline{\sigma} \in [\![\Sigma]\!]$ denote some store satisfying Σ. Moreover, define

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

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 $X, Y \in \mathcal{P}$ by

$$\begin{aligned} X_{\Sigma''_i} &= \{\underline{\sigma} + \sigma''_i\}, \text{ for } i \in \mathbb{N} \\ X_{\hat{\Sigma}} &= \emptyset, \text{ for all other } \hat{\Sigma} \\[1ex] Y_{\Sigma} &= \{\underline{\sigma}\} \\ Y_{\hat{\Sigma}} &= \emptyset, \text{ for all other } \hat{\Sigma} \end{aligned}$$

By construction, both X and Y are admissible in every component $\hat{\Sigma}$. By induction one obtains $\sigma''_0 \sqsubseteq \sigma''_1 \sqsubseteq \dots$, therefore $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \dots$ in $\Phi(Y, X)_\Sigma$. Hence we must show $\sigma \in \Phi(Y, X)_\Sigma$. But this is not the case, since it would entail, by **(M2)** and

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

that there exists $\Sigma'' \geqslant \Sigma$ such that $\underline{\sigma} + \sqcup_i \sigma''_i \in X_{\Sigma''}$. Clearly this is not the case, since $\underline{\sigma} + \sqcup_i \sigma''_i$ is *strictly* greater than every $\underline{\sigma} + \sigma''_i$ and therefore not in any of the $X_{\Sigma''_i}$.

4.3 A Refined Semantics of Store Specifications

We refine the definition of store predicates by replacing the existential quantifier in **(M2)** of Definition 4.6 by a *Skolem function*, as follows: We call the elements of the (recursively defined) domain

$$\phi \in \mathsf{RSF} = \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathcal{M}}(\mathsf{St} \times \mathsf{RSF} \times \mathit{Spec} \rightharpoonup \mathit{Spec} \times \mathsf{RSF})) \qquad (4)$$

choice functions, or *Skolem Functions*. The intuition is that, given a store $\sigma \in [\![\Sigma]\!]$, if $\sigma' \in [\![\Sigma']\!]$ with choice function ϕ', for some extension $\Sigma' \geqslant \Sigma$ and the method invocation $\sigma.l.\mathsf{m}(\sigma')$ terminates, then $\phi.l.\mathsf{m}(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$ yields a store specification $\Sigma'' \succcurlyeq \Sigma'$ 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 wrt. 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, for example.

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 counter-example to admissibility of the previous subsection.

Definition 4.7 (Store Predicate). *Let $\mathcal{P} = \mathcal{P}(\mathsf{St} \times \mathit{RSF})^{\mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})}$ denote the collection of families of subsets of $\mathsf{St} \times \mathit{RSF}$, indexed by store specifications. We define a functional $\Phi : \mathcal{P}^{op} \times \mathcal{P} \to \mathcal{P}$ as follows.*

$(\sigma, \phi) \in \Phi(Y, X)_\Sigma :\Leftrightarrow$
(1) $\mathit{dom}(\Sigma) = \mathit{dom}(\phi)$ and $\forall l \in \mathit{dom}(\Sigma).\ \mathit{dom}(\pi_2(\Sigma.l)) = \mathit{dom}(\phi.l)$, and
(2) $\forall l \in \mathit{dom}(\Sigma)$ where $\Sigma.l = [f_i{:}\, A_i{}^{i=1\ldots n},\ m_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$:
(F) *$\sigma.l.f_i \in \|A_i\|_\Sigma$, for all $1 \le i \le n$, and*
$\forall \Sigma' \succcurlyeq \Sigma\ \forall (\sigma', \phi') \in Y_{\Sigma'}$. if $\sigma.l.m_j(\sigma') = (v, \sigma'')\!\downarrow$
then $\phi.l.m_j(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$ s.t. $\Sigma'' \succcurlyeq \Sigma'$ and
(M1) $(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![T_j[l/y_j]]\!]$
(M2) $(\sigma'', \phi'') \in X_{\Sigma''}$
(M3) $v \in \left\|B_j[l/y_j]\right\|_{\Sigma''}$
for all $1 \le j \le m$

We write $\sigma \in [\![\Sigma]\!]$ if there is some $\phi \in \mathit{RSF}$ s.t. $(\sigma, \phi) \in \mathit{fix}(\Phi)_\Sigma$.

Lemma 4.8. *Functional Φ, defined in Def. 4.7, does have a unique fixed point.*

Proof. Firstly, one shows that Φ is monotonic and maps admissible predicates to admissible predicates, in the sense that for all X and Y,

$$\forall \Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec}).\ X_\Sigma \subseteq \mathsf{St} \times \mathsf{RSF} \text{ admissible } \Rightarrow \\ \forall \Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec}).\ \Phi(Y, X)_\Sigma \subseteq \mathsf{St} \times \mathsf{RSF} \text{ admissible}$$

Indeed, if $(\sigma_0, \phi_0) \sqsubseteq (\sigma_1, \phi_1) \sqsubseteq \ldots$ is a chain in $\Phi(Y, X)_\Sigma$, then $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \ldots$ in St and $\phi_0 \sqsubseteq \phi_1 \sqsubseteq \ldots$ in RSF. Let $\sigma = \sqcup_i \sigma_i$ and $\phi = \sqcup_i \phi_i$ (so $(\sigma, \phi) = \sqcup_i (\sigma_i, \phi_i)$), we show $(\sigma, \phi) \in \Phi(Y, X)_\Sigma$ under the assumption that $X_{\Sigma'}$ is admissible for all $\Sigma' \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$.

Clearly condition (1) of Definition 4.7 is satisfied. As for (2), suppose $l \in \mathsf{dom}(\Sigma)$ with $\Sigma.l = [\mathsf{f}_i{:}\, A_i{}^{i=1\ldots n},\ \mathsf{m}_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$. Since, for all $1 \le i \le n$,

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

we obtain $\sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma$ by assumption $(\sigma_j, \phi_j) \in \Phi(Y,X)_\Sigma$. Next, suppose $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in Y_{\Sigma'}$ and $\sigma.l.\mathsf{m}_j(\sigma') = (v, \sigma'')\downarrow$. By definition of σ as $\bigsqcup_k \sigma_k$ and continuity, we must have $\sigma_k.l.\mathsf{m}_j(\sigma') = (v, \sigma''_k)\downarrow$ for sufficiently large k, and

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

By assumption, for all sufficiently large k, $\phi_k.l.\mathsf{m}_j(\sigma', \phi', \Sigma') = (\Sigma''_k, \phi''_k)$ with $\Sigma''_k \succcurlyeq \Sigma'$ and

- $(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma''_k)) \in [\![T_j[l/y_j]]\!]$,
- $(\sigma''_k, \phi''_k) \in X_{\Sigma''_k}$, and
- $v \in \left\|B_j[l/y_j]\right\|_{\Sigma''_k}$

Since $\pi_{\mathsf{Val}}(\sigma''_k) = \pi_{\mathsf{Val}}(\sigma'')$, **(M1)** follows. The discrete order on *Spec* entails $\Sigma''_k \equiv \Sigma''_{k+1} \equiv \ldots$, hence, $\phi(\sigma', \phi', \Sigma') = \sqcup(\Sigma''_k, \phi''_k) = (\Sigma'', \sqcup_k \phi''_k)$ with $\Sigma'' \equiv \Sigma''_k \equiv \Sigma''_{k+1} \equiv \ldots$, and clearly **(M3)** is satisfied. By assumption $X_{\Sigma''}$ is admissible therefore also condition **(M2)** holds as required, i.e., $(\sigma'', \phi'') = \sqcup(\sigma''_k, \phi''_k) \in X_{\Sigma''}$.

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

$$e : X \subset X' \text{ iff } \forall \Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(Spec)\ \forall \sigma \in \mathsf{St}\ \forall \phi \in \mathsf{RSF}. \\ (\sigma, \phi) \in X_\Sigma \Rightarrow (e_1(\sigma), e_2(\phi)) \in X'_\Sigma$$

such that $e : X \subseteq X'$ 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 corresponding component X'_Σ of X'. Let F_{Store} be the locally continuous, mixed-variant functor associated with the domain equations (1), for which $F_{Store}(\mathsf{St}, \mathsf{St}) = \mathsf{St}$, and consider the locally continuous functor $F_{\mathsf{St},\mathsf{RSF}}(R, S) : (\mathsf{PreDom} \times \mathsf{PreDom})^{op} \times \mathsf{PreDom} \times \mathsf{PreDom} \to \mathsf{PreDom} \times \mathsf{PreDom}$

$$F_{\mathsf{St},\mathsf{RSF}}(R, S) = \langle F_{Store}(\pi_1(R), \pi_1(S)), \\ \mathsf{Rec}_{\mathsf{Loc}}(\mathsf{Rec}_{\mathcal{M}}(\pi_1(R) \times \pi_2(R) \times Spec \rightharpoonup Spec \times \pi_2(S)))\rangle$$

for which $(\mathsf{St}, \mathsf{RSF})$ is the minimal invariant. In the following, we write F_{St} for the functor $\pi_1 \circ F_{\mathsf{St},\mathsf{RSF}}$ and F_{RSF} for $\pi_2 \circ F_{\mathsf{St},\mathsf{RSF}}$. According to [11] it only remains to be shown that

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

for all $X, Y, X', Y' \in \mathcal{P}$ and $e \sqsubseteq id_{\mathsf{St}\times\mathsf{RSF}}$ which follows from a similar line of reasoning as in [16]: Suppose $e = (e_1, e_2) \sqsubseteq id_{\mathsf{St}\times\mathsf{RSF}}$ such that

$$e : X \subset X' \text{ and } e : Y' \subset Y \qquad (5)$$

for some $X, Y, X', Y' \in \mathcal{P}$, and assume $(\sigma, \phi) \in \Phi(Y, X)_\Sigma$. We must show that

$F_{\mathsf{St,RSF}}(e,e)(\sigma,\phi) \in \Phi(Y',X')_\Sigma$ which proves (†). Recall that

$$\begin{aligned} F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{f} &= \sigma.l.\mathsf{f} \\ F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{m}(\sigma') &= (\mathsf{id}_{\mathsf{Val}} \times e_1)(\sigma.l.\mathsf{m}(e_1(\sigma'))) \qquad (6) \\ F_{\mathsf{RSF}}(e,e)(\sigma,\phi).l.\mathsf{m}(\sigma',\phi',\Sigma') &= (\mathsf{id}_{Spec} \times e_2)(\phi.l.\mathsf{m}(e_1(\sigma'),e_2(\phi'),\Sigma')) \end{aligned}$$

for all $\mathsf{f} \in \mathcal{F}$ and $\mathsf{m} \in \mathcal{M}$. In particular, condition (1) of Definition 4.7 is satisfied for $F_{\mathsf{St,RSF}}(e,e)(\sigma,\phi)$.

To show (2) let $l \in \mathsf{dom}(\Sigma)$, and $\Sigma.l = [\mathsf{f}_i\colon A_i{}^{i=1\ldots n}, \mathsf{m}_j\colon \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$. From $(\sigma,\phi) \in \Phi(Y,X)_\Sigma$ and (6) we immediately obtain

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

Now suppose $\Sigma' \succcurlyeq \Sigma$, $\phi' \in \mathsf{RSF}$ and $\sigma' \in \mathsf{St}$ with $(\sigma',\phi') \in Y'_{\Sigma'}$ and such that $F_{\mathsf{St}}(e,e)(\sigma,\phi).l.\mathsf{m}_j(\sigma')\downarrow$. By (6) we thus know that, for all $1 \le j \le m$,

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

for some $v \in \mathsf{Val}$ and $\sigma'' \in \mathsf{St}$. By (5), assumption $(\sigma',\phi') \in Y'_{\Sigma'}$ shows $e(\sigma',\phi') = (e_1(\sigma'),e_2(\phi')) \in Y_{\Sigma'}$. Together with the assumption $(\sigma,\phi) \in \Phi(Y,X)$ this entails

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

for $\phi'' \in \mathsf{RSF}$ and $\Sigma'' \succcurlyeq \Sigma'$ s.t.

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

Since $e \sqsubseteq \mathsf{id}_{\mathsf{St}\times\mathsf{RSF}}$ we know $e_1(\sigma'') \sqsubseteq \sigma''$, and in particular $\pi_{\mathrm{Val}}(e_1(\sigma'')) = \pi_{\mathrm{Val}}(\sigma'')$. Similarly for σ'. Hence, **(M1')** entails $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(e_1(\sigma''))) \in [\![T[l/y_j]]\!]$, i.e., **(M1)** holds. Finally, assumption (5) and condition **(M2')** above give $(e_1(\sigma''),e_2(\phi'')) \in X'_{\Sigma''}$ which shows **(M2)**, and we have proved (†).

Note that it is necessary 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. [6] □

[6] Unless one finds an appropriate way to restrict the reference to methods in transitions specifications (see [16]).

5 Soundness

5.1 Preliminaries

Recall from the previous section that the semantics of store specifications is defined in terms of the semantics $\|A\|_\Sigma$ for result specifications A that does not mention St at all. The following key lemma establishes the relation between store specifications and object specifications $[\![\vdash A]\!]$ as defined in Section 3.3:

Lemma 5.1. *For all object specifications A, store specifications Σ, stores σ, and locations l, if $\sigma \in [\![\Sigma]\!]$ and $l \in \mathit{dom}(\Sigma)$ such that $\vdash \Sigma.l <: A$ then $(l, \sigma) \in [\![A]\!]$.*

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

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

We have to show that $(l, \sigma) \in [\![\vdash A]\!]$, i.e., that

- $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![\vdash A_i]\!]$ for all $1 \leq i \leq n$ 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 $1 \leq j \leq m$.

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

$$\Sigma.l \equiv [\mathsf{f}_i{:}A_i{}^{i=1\dots n+p}, \mathsf{m}_j{:}\varsigma(y_j)B'_j{::}T'_j{}^{j=1\dots m+p}]$$

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 4.7 **(F)** and $\sigma \in [\![\Sigma]\!]$ we have $\sigma.l.\mathsf{f}_i \in \|A_i\|_\Sigma$. If A_i is *Bool* then from $\|\mathit{Bool}\|_\Sigma = \mathsf{BVal}$, hence, $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![\vdash \mathit{Bool}]\!]$. 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 4.7 **(F)**. Hence by induction hypothesis we obtain $(\sigma.l.\mathsf{f}_i, \sigma) \in [\![\vdash A_i]\!]$ as required.

For the second part, suppose that $\sigma.l.\mathsf{m}_j(\sigma) = (v, \sigma'')$. From Definition 4.7 part **(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 $\Sigma'' \succcurlyeq \Sigma$. Therefore in the case where B_j is *Bool*, $v \in \mathsf{BVal}$ and

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

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

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

By induction hypothesis (applied to $B'_j[l/y_j]$, Σ'', σ'' and v) this yields $(v, \sigma'') \in$

$[\![\vdash B'_j[l/y_j]]\!]$. Thus,

$$\begin{aligned}(v,\sigma'') \in [\![\vdash B'_j[l/y_j]]\!] = [\![y_j \vdash B'_j]\!](y_j \mapsto l) && \text{Lemma 4.2}\\ \subseteq [\![y_j \vdash B_j]\!](y_j \mapsto l) && \text{Lemma 3.3}\end{aligned}$$

as required.

Finally, by Definition 4.7 **(M1)** we obtain

$$\begin{aligned}(\pi_{\mathrm{Val}}\,\sigma, v, \pi_{\mathrm{Val}}\,\sigma'') \in [\![\vdash T'_j[l/y_j]]\!] = [\![y_j \vdash T'_j]\!](y_j \mapsto l) && \text{Lemma 4.2}\\ \subseteq [\![y_j \vdash T_j]\!](y_j \mapsto l) && \text{soundness of } \vdash_{\mathsf{fo}}\end{aligned}$$

This concludes the proof. □

We can now define the semantics of judgements of Abadi-Leino logic and prove the key lemma.

Definition 5.2 (Validity). $\Gamma \vDash a : A :: T$ *if and only if for all store specifications* $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$*, 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* $(\pi_{\mathrm{Val}}(\sigma), v, \pi_{\mathrm{Val}}(\sigma')) \in [\![[\Gamma] \vdash T]\!]\rho$*.*

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

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

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

Proof. Consider an extended signature of transition relations with predicates $\mathsf{sel}_{int}(\cdot,\cdot)$ and $\mathsf{alloc}_{int}(\cdot)$. We extend the interpretation of transition relations in the natural way,

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

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

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

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

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 first-order provability, 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 and the new predicates, by $\overline{x} \vdash T$. □

5.2 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,

$$\sigma \in [\![\Sigma]\!] \text{ and } [\![a]\!]\rho\sigma = (v, \sigma') \quad \Rightarrow \quad \exists\, \Sigma' \geqslant \Sigma \text{ s.t. } \sigma' \in [\![\Sigma']\!] \tag{7}$$

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 (7) holds are given in the statement of the following lemma, and take the choice functions $\phi \in \mathsf{RSF}$ introduced in Sect. 4 into account. We write SF for the domain of "individual" choice functions,

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

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

Lemma 5.4. *Suppose*

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

(H2) $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$ *is a store specification*

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

Then there exists $\phi \in [\mathsf{St} \times \mathit{RSF} \times \mathit{Spec} \rightharpoonup \mathit{Spec} \times \mathit{RSF}]$ *s.t. for all* $\Sigma' \geqslant \Sigma$ *and for all* $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$*, if* $[\![a]\!]\rho\sigma' = (v, \sigma'')\!\downarrow$ *then the following holds:*

(S1) *there exists* $\Sigma'' \geqslant \Sigma'$ *and* $\phi'' \in \mathit{RSF}$ *s.t.* $\phi(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$

(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, i.e., (object construction) and (field update), we must show explicitly that the resulting store satisfies the store specification, according to Definition 4.7.

We consider cases, depending on the last rule of 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 $\phi \in \mathsf{SF}$ s.t. whenever $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and $[\![a]\!]\rho\sigma' = (v, \sigma')$ then *(S1)-(S4)* hold.

Recall the subsumption rule,

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

so 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 (IH) there exists $\phi \in \mathsf{SF}$ s.t. for all $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ with $[\![a]\!]\rho\sigma' = (v, \sigma')$,

(S1) there exists $\Sigma'' \geqslant \Sigma'$, $\phi'' \in \mathsf{RSF}$ s.t. $\phi(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$

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

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

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

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

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

It remains to show

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

Note that by the subtyping rules, $A \equiv Bool$ if and only if $A' \equiv Bool$. 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. Further, assume

(H2) Σ is a store specification

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

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

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

Now suppose $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and $[\![a]\!]\rho\sigma' = (v, \sigma'')$ Then, by the variable rule, we find that a is necessarily a variable x. Further we obtain $x{:}A$ in Γ, $T \equiv T_{\mathsf{res}}(x)$, and the semantics gives $(v, \sigma'') = [\![a]\!]\rho\sigma' = (\rho(x), \sigma')$, i.e.,

$$v = \rho(x) \text{ and } \sigma'' = \sigma'$$

By definition of ϕ above,

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

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

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

(S4) $(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![[\Gamma] \vdash T_{\mathsf{res}}(x)]\!]\rho$, by the definition of $[\![[\Gamma] \vdash T]\!]$ in Tab. 4, and $\sigma'' = \sigma'$ and $v = \rho(x)$.

as required.

- **Const**
  Similar to the previous case: Suppose *(H1)*: $\Gamma \vdash a : A :: T$ has been derived by an application of the rule for `true`, i.e., a is `true`, A is *Bool* and T is $T_{\mathsf{res}}(\mathsf{true})$. Now assume

(H2) Σ is a store specification

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

and define $\phi \in \mathsf{SF}$ by

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

We must show *(S1)-(S4)*. So let $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and suppose $[\![a]\!]\rho\sigma' = (v, \sigma'')$. By definition of the denotational semantics, $(v, \sigma'') = [\![a]\!]\rho\sigma' = (\mathit{true}, \sigma')$. Hence, by definition of ϕ, *(S1)* $\phi(\sigma', \phi', \Sigma') = (\Sigma', \phi')$

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

(S3) $v = \mathit{true} \in \mathsf{BVal} = \|A\|_{\Sigma'}$

(S4) $(\pi_{\mathrm{Val}}(\sigma'), \mathit{true}, \pi_{\mathrm{Val}}(\sigma'')) \in [\![[\Gamma] \vdash T_{\mathsf{res}}(\mathit{true})]\!]\rho$ by definition

as required. The case where a is `false` is analoguous.

- **Conditional**
  By a 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) Σ is a store specification, and

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

Now recall the rule for this case,

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

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 induction hypothesis applied to *(H1')* there is $\phi_1 \in \mathsf{SF}$ s.t. for all $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ with $[\![a_1]\!]\rho\sigma' = (\hat{v}, \hat{\sigma})$, the conclusions of the lemma hold:

(S1') there exists $\hat{\Sigma} \geqslant \Sigma'$ and $\hat{\phi} \in \mathsf{RSF}$ s.t. $\phi(\sigma', \phi', \Sigma') = (\hat{\Sigma}, \hat{\phi})$

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

(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 *(S3')* and Lemma 4.5,

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

Therefore, by induction hypothesis applied to *(H1")* there is $\phi_{\hat{v}} \in \mathsf{SF}$ s.t. for all $\Sigma' \geqslant \hat{\Sigma}$ and all $(\hat{\sigma}, \hat{\phi}) \in \mathit{fix}(\Phi)_{\hat{\Sigma}}$ with $[\![a_2]\!]\rho[x := \hat{v}]\hat{\sigma} = (v, \sigma'')$, the following holds.

(S1") there exists $\Sigma'' \succcurlyeq \hat{\Sigma}$ and $\phi'' \in \mathsf{RSF}$ s.t. $\phi_{\hat{v}}(\hat{\sigma}, \hat{\phi}, \hat{\Sigma}) = (\Sigma'', \phi'')$

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

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

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

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

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

which is continuous due to the flatness of Val.

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

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

which shows

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

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

From the definition of ϕ, and the considerations above, it follows that

(S1) there is $\Sigma'' \succcurlyeq \hat{\Sigma} \succcurlyeq \Sigma'$ s.t. $\phi(\sigma', \phi', \Sigma') = \phi_{\hat{v}}(\hat{\sigma}, \hat{\phi}, \hat{\Sigma}) = (\Sigma'', \phi'')$, where $\phi_1(\sigma', \phi', \Sigma') = (\hat{\Sigma}, \hat{\phi})$, by *(S1')* and *(S1")*

(S2) $(\sigma'', \phi'') \in \mathit{fix}(\Phi)_{\Sigma''}$, by *(S2')* and *(S2")*

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

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

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

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

$$A[\rho[x := v]/(\Gamma, x{:}A_1)] \equiv A[\rho/\Gamma] \tag{8}$$

Moreover, *(C4')*, *(C4")*, Lemma 5.3 and

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

proves

$$(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![\Gamma \vdash T]\!]\rho \tag{9}$$

We therefore obtain

(S3) $v \in \|A[\rho/\Gamma]\|_{\Sigma'}$, by *(C3)* and (8)

(S4) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![\Gamma \vdash T]\!]\rho$, by (9)

as required.

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

(H2) Σ is a store specification

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

We recall the object introduction rule,

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

from which 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 $1 \leq i \leq n$

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

We have to show that there is $\phi \in \mathsf{SF}$ s.t. for all $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ with $[\![a]\!]\rho\sigma' = (v, \sigma'')$, *(S1)-(S4)* hold.

From *(H3)* and Lemma 4.5 we know that for all $\hat{\Sigma} \succcurlyeq \Sigma$ and $l_0 \notin \mathsf{dom}(\hat{\Sigma})$,

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

Hence by induction hypothesis on *(H1")*, there is $\phi^j_{l_0} \in \mathsf{SF}$ for all $1 \leq j \leq m$ s.t. for all $\Sigma_1 \succcurlyeq (\hat{\Sigma}, l_0{:}A[\rho/\Gamma])$, for all $(\sigma_1, \phi_1) \in \mathit{fix}(\Phi)_{\hat{\Sigma}, l_0:A[\rho/\Gamma]}$ with $[\![b_j]\!]\rho[y_j := l_0]\sigma_1 = (v_2, \sigma_2)\downarrow$, we obtain the conclusions *(S1)-(S4)* of the lemma, i.e.,

(S1') there exists $\Sigma_2 \succcurlyeq \Sigma_1$ and $\phi_2 \in \mathsf{RSF}$ s.t.$\phi^j_{l_0}(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$

(S2') $(\sigma_2, \phi_2) \in \mathit{fix}(\Phi)_{\Sigma_2}$

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

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

We have $\{\!|l_0 = \{\!|\mathsf{m}_j = \phi^j_{l_0}|\!\}^{j=1...m}|\!\} \in \mathsf{RSF}$, therefore we can define $\phi \in \mathsf{SF}$ by

$$\phi(\sigma', \phi', \Sigma') = \begin{cases} ((\Sigma', l_0{:}A[\rho/\Gamma]), \phi' + \{\!|l_0 = \{\!|\mathsf{m}_j = \phi^j_{l_0}|\!\}|\!\}) \\ \qquad\qquad \text{if } \Sigma' \succcurlyeq \Sigma \text{ and } [\![a]\!]\rho\sigma' = (l_0, \sigma'') \\ \text{undefined otherwise} \end{cases} \tag{10}$$

We show that *(S1)–(S4)* hold. Let $\Sigma' \geqslant \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and suppose $[\![a]\!]\rho\sigma' = (v, \sigma'')$. By definition of the semantics, and the fact that *(H1')* entails $\rho(x_i)\downarrow$ for $1 \leq i \leq n$, 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

$$\sigma'' = \sigma' + \{\!|l_0 = \{\!|\mathsf{f}_i = \rho(x_i), \mathsf{m}_j = \lambda\sigma.[\![b_j]\!]\rho[y_j := l_0]\sigma|\!\}|\!\} \tag{11}$$

We obtain that there exists $\phi'' \in \mathsf{RSF}$ s.t.

(S1) $\phi(\sigma', \phi', \Sigma') = ((\Sigma', l_0{:}A[\rho/\Gamma]), \phi'')$, by construction of ϕ in equation (10)

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

(S4) $(\pi_{\mathsf{Val}}(\sigma'), v, \pi_{\mathsf{Val}}(\sigma'')) \in [\![\Gamma \vdash T_{\mathsf{obj}}(\mathsf{f}_1 = x_1 \ldots \mathsf{f}_n = x_n)]\!]\rho$, which is easily checked from the definition of $T_{\mathsf{obj}}(\ldots)$, the semantics in Tab. 4 and equation (11).

All that remains to be shown is *(S2)*: $(\sigma'', \phi'') \in \mathit{fix}(\Phi)_{\Sigma''}$, where Σ'' is $\Sigma', l_0{:}A[\rho/\Gamma]$. By the construction of ϕ in (10),

$$\phi'' = \phi' + \{\!|l_0 {=} \{\!|\mathsf{m}_j {=} \phi^j_{l_0}|\!\}|\!\}$$

and we show *(S2)* according to Definition 4.7:

As for (1), by assumption the domains of ϕ' and Σ' agree, and by construction of ϕ, also $\mathsf{dom}(\phi''.l_0) = \{\mathsf{m}_1, \ldots, \mathsf{m}_m\} = \mathsf{dom}(\pi_2(\Sigma''.l_0))$. For (2), suppose $l \in \mathsf{dom}(\Sigma'')$. We distinguish two cases:

– $l \neq l_0$: Then

$$\Sigma''.l = \Sigma'.l = [\mathsf{g}_i{:}A'^{\,i=1\ldots p}_i, \mathsf{n}_j{:}\varsigma(y_j)B'_j :: T'^{\,1\ldots q}_j]$$

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

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

(M) Let $1 \leq j \leq q$, let $\Sigma_1 \geqslant \Sigma''$, let $(\sigma_1, \phi_1) \in \mathit{fix}(\Phi)_{\Sigma_1}$ 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 $\Sigma_1 \geqslant \Sigma'$, the assumption $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and the construction of ϕ'' yield

* $\phi''.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma_1) = \phi'.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$
* $(\sigma_2, \phi_2) \in \mathit{fix}(\Phi)_{\Sigma_2}$
* $v_2 \in \left\|B'_j[l/y_j]\right\|_{\Sigma_2}$
* $(\pi_{\mathsf{Val}}(\sigma_1), v_2, \pi_{\mathsf{Val}}(\sigma_2)) \in [\![T'_j[l/y_j]]\!]$

– $l = l_0$:

(F) By assumption *(H1')* and $\rho \in \|\Gamma\|_\Sigma$ we know that there is $\vdash A_i' <: A_i$ for all $1 \leq i \leq n$ s.t. $x_i{:}A_i'$ in Γ. Hence,

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

(M) Let $1 \leq j \leq m$. Suppose $\Sigma_1 \succcurlyeq \Sigma''$, let $(\sigma_1, \phi_1) \in \mathit{fix}(\Phi)_{\Sigma_1}$ and suppose $\sigma''.l_0.\mathsf{m}_j(\sigma_1) = (v_2, \sigma_2)$. Since $\sigma''.l_0.\mathsf{m}_j = [\![b_j]\!]\rho[y_j := l_0]\sigma_1$ and $\Sigma_1 \succcurlyeq \Sigma'$, the assumption $(\sigma_1, \phi_1) \in \mathit{fix}(\Phi)_{\Sigma'}$ and the construction of ϕ'' give Σ_2 and ϕ_2 s.t.

- $\phi''.l_0.\mathsf{m}_j(\sigma_1, \phi_1, \Sigma_1) = \phi^j_{l_0}(\sigma_1, \phi_1, \Sigma_1) = (\Sigma_2, \phi_2)$, by *(S1')*
- $(\sigma_2, \phi_2) \in \mathit{fix}(\Phi)_{\Sigma_2}$, by *(S2')*
- $v_2 \in \left\|B_j[\rho[y_j := l_0]/\Gamma, y_j{:}A]\right\|_{\Sigma_2} = \left\|B_j[\rho/\Gamma][l_0/y_j]\right\|_{\Sigma_2}$, by *(S3')* and the substitution lemma, Lemma 4.2
- $(\pi_{\mathsf{Val}}(\sigma_1), v_2, \pi_{\mathsf{Val}}(\sigma_2)) \in [\![[\Gamma, y_j{:}A] \vdash T_j]\!]\rho[y_j := l_0]$ which equals $[\![T_j[\rho/\Gamma][l_0/y_j]]\!]$, by *(S4')* and the substitution lemma

Thus we have shown $(\sigma'', \phi'') \in \mathit{fix}(\Phi)_{\Sigma''}$, i.e., *(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' s.t. $A \equiv A'[x/y]$ 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,

$$\phi(\sigma', \phi', \Sigma') = \phi'.\rho(x).\mathsf{m}(\sigma', \phi', \Sigma') \tag{12}$$

Now let $\Sigma' \succcurlyeq \Sigma$, $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ 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,

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

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

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

i.e., 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

$$y \vdash A'' <: A'[\rho/\Gamma] \text{ and } \vdash_{\mathsf{fo}} T'' \to T'[\rho/\Gamma] \qquad (13)$$

Now assumption $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ with equation (12) implies that there are Σ'', ϕ'' s.t.

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

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

(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 transitivity of $<:$, equation (13), Lemma 4.2 and *(S3')*

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

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 (13) and *(S4')*,

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

which was to show.

- **Field Selection**
  Similar. 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 the (field update) rule,

(H2) Σ is a store specification

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

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

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

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

$$\sigma'' = \sigma'[v := \sigma'.v[\mathsf{f}_k := \rho(y)]] \tag{14}$$

We show that *(S1)-(S4)* hold.

By *(H3)*, $\rho(x) \in \|A[\rho/\Gamma]\|_\Sigma \subseteq \|A[\rho/\Gamma]\|_{\Sigma'}$. Then by construction of ϕ, and (14),

(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 Tab. 4

It remains to show *(S2)*, $(\sigma'', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$.

By assumption $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$, condition (1) of Definition 4.7 is satisfied. As for condition (2), suppose $l \in \mathsf{dom}(\Sigma')$ s.t.

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

(F) We distinguish two cases:

- Case $l = \rho(x)$ and $\mathsf{g}_i = \mathsf{f}_k$. Then, by (14), $\sigma''.l.\mathsf{g}_i = \rho(y)$. 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, $A_k' \equiv A_k[\rho/\Gamma]$. Note that *invariance of subspecification* in the field components is needed to conclude this. Now again by *(H3)*,

  $$\rho(y) \in \|A_k[\rho/\Gamma]\|_\Sigma \subseteq \|A_k[\rho/\Gamma]\|_{\Sigma'} = \left\|A_k'\right\|_{\Sigma'}$$

  Hence, $\sigma''.l.\mathsf{g}_i \in \left\|A_i'\right\|_{\Sigma'}$ 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 (14). Hence, by assumption $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$, we have $\sigma''.l.\mathsf{g}_i \in \left\|A_i'\right\|_{\Sigma'}$.

(M) Let $\Sigma'' \geqslant \Sigma'$, let $(\sigma_1, \phi_1) \in \mathit{fix}(\Phi)_{\Sigma''}$ and suppose $\sigma''.l.\mathsf{n}_j(\sigma_1) = (v_2, \sigma_2)$. Then, by assumption $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$ and the fact that $\sigma''.l.\mathsf{n}_j = \sigma'.l.\mathsf{n}_j$ by (14), we obtain that $\phi'.l.\mathsf{n}_j(\sigma_1, \phi_1, \Sigma'') = (\Sigma_2, \phi_2)$ s.t. $\Sigma_2 \geqslant \Sigma''$ and

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

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

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

as required.

which concludes the proof. □

5.3 Soundness Theorem

With Lemma 5.1 and Lemma 5.4, proved in Subsections 5.1 and 5.2, it is now easy to establish our main result:

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

Proof. Suppose $\Gamma \vdash a : A :: T$, and let $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(\mathit{Spec})$ be a store specification and suppose $\rho \in \mathsf{Env}$ s.t. $\rho \in \|\Gamma\|_\Sigma$. Let $\sigma \in [\![\Sigma]\!]$, so by definition there exists $\phi \in \mathsf{RSF}$ s.t. $(\sigma, \phi) \in \mathit{fix}(\Phi)_\Sigma$. Next suppose

$$[\![a]\!]\rho\sigma = (v, \sigma')$$

By Lemma 5.4 there exists $\phi_a \in \mathsf{RSF}$ s.t. $\phi(\sigma, \phi, \Sigma) = (\Sigma', \phi')$ where $\Sigma' \geqslant \Sigma$ and $(\sigma', \phi') \in \mathit{fix}(\Phi)_{\Sigma'}$, i.e., $\sigma' \in [\![\Sigma']\!]$ follows. Moreover,

- $v \in \|A[\rho/\Gamma]\|_{\Sigma'}$, and
- $(\pi_{\mathsf{Val}}(\sigma), v, \pi_{\mathsf{Val}}(\sigma')) \in [\![[\Gamma] \vdash T]\!]\rho$

Now in the case where A is *Bool* we obtain $(v, \sigma') \in [\![[\Gamma] \vdash A]\!]\rho$ from $\|\mathit{Bool}\|_{\Sigma'} = \mathsf{BVal}$. Otherwise A is an object specification, and we must have $\vdash \Sigma'.v <: A[\rho/\Gamma]$ by definition of $\|A[\rho/\Gamma]\|_{\Sigma'}$. Hence, by Lemma 5.1,

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

where the last equality is by the the substitution lemma, Lemma 4.2. $\square$

In particular, if $\vdash a : A :: T$ and $[\![a]\!]\sigma = (v, \sigma')$ then $(v, \sigma') \in [\![A]\!]$ and so $v \neq \mathsf{error}$ by Lemma 3.2.

6 Recursive Specifications

In this section we investigate an extension of the logic with recursive specifications which are important when reasoning about implementations of datatypes such as lists and trees in object-oriented languages. For instance, referring back to the example of the account manager in Fig. 1, if $A_{\mathtt{Manager}}$ should include a list of accounts, we would need a recursive specification $\mu X.\,[\mathsf{head} : A_{\mathtt{Account}}, \mathsf{tail} : X]$.

Below we discuss in more detail how recursive specifications can be dealt with in the logic.

Syntax and Proof Rules To accommodate reasoning about elements of recursive types such as lists of accounts above, 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".

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

where X ranges over an infinite set *TyVar* 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 $\Gamma \vdash \ldots$ by $\Gamma; \Delta \vdash \ldots$, and the definitions of well-formed specifications and well-formed specification contexts are extended, similar to the case of recursive types [1].

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

and

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

and we often write Δ, X for $\Delta, X <: \top$.

Subspecifications for recursive specifications are obtained by the "usual" recursive subtyping rule [3], and $\top$ is the greatest specification,

$$\frac{\Gamma; \Delta, Y <: \top, X <: Y \vdash A <: B}{\Gamma; \Delta \vdash \mu X.A <: \mu Y.B} \qquad \frac{\Gamma; \Delta \vdash A}{\Gamma; \Delta \vdash A <: \top}$$

As will be seen from the semantics below, in our model a recursive specification and its unfolding are not just isomorphic but equal, i.e., $[\![\mu X.A]\!] = [\![A[(\mu X.A)/X]]\!]$. 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,

$$\text{fold}\ \frac{\Gamma; \Delta \vdash \mu X.A}{\Gamma; \Delta \vdash A[(\mu X.A)/X] <: \mu X.A} \qquad \text{unfold}\ \frac{\Gamma; \Delta \vdash \mu X.A}{\Gamma; \Delta \vdash \mu X.A <: A[(\mu X.A)/X]}$$

We will prove their soundness below.

6.1 Existence of Store Specifications

Next, we adapt our notion of store specification to recursive specifications. The existence proof is very similar to the one given in Section 4, however, for completeness we spell it out in detail below.

Definition 6.1. *A store specification is a record $\Sigma \in \mathsf{Rec}_{\mathsf{Loc}}(Spec)$ such that for each $l \in dom(\Sigma)$,*

$$\Sigma.l = \mu(X)[f_i{:}\, A_i{}^{i=1\ldots n},\ m_j{:}\, \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$$

is a closed (recursive) object specification.

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 4.7 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 $\mathcal{P} = \mathcal{P}(\mathsf{St} \times \mathsf{RSF})^{\mathsf{Rec}_{\mathsf{Loc}}(Spec)}$ denote the collection of families of subsets of $\mathsf{St} \times \mathsf{RSF}$, indexed by store specifications (in the sense of Definition 6.1). We define a functional $\Phi : \mathcal{P}^{op} \times \mathcal{P} \to \mathcal{P}$ as follows.*

$(\sigma, \phi) \in \Phi(R, S)_\Sigma :\Leftrightarrow$
(1) $\mathsf{dom}(\Sigma) = \mathsf{dom}(\phi)$ and $\forall l \in \mathsf{dom}(\Sigma).\ \mathsf{dom}(\pi_2(\Sigma.l)) = \mathsf{dom}(\phi.l)$, and
(2) $\forall l \in \mathsf{dom}(\Sigma)$ where $\Sigma.l = \mu(X)[\mathsf{f}_i\colon A_i{}^{i=1\ldots n},\ \mathsf{m}_j\colon \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$:
(F) *$\sigma.l.\mathsf{f}_i \in \|A_i[\Sigma.l/X]\|_\Sigma$, for all $1 \le i \le n$, and*
$\forall \Sigma' \succcurlyeq \Sigma\ \forall (\sigma', \phi') \in R_{\Sigma'}$. if $\sigma.l.\mathsf{m}_j(\sigma') = (v, \sigma'')\!\downarrow$
then $\phi.l.\mathsf{m}_j(\sigma', \phi', \Sigma') = (\Sigma'', \phi'')$ s.t. $\Sigma'' \succcurlyeq \Sigma'$ and
(M1) $(\pi_{\mathrm{Val}}(\sigma'), v, \pi_{\mathrm{Val}}(\sigma'')) \in [\![T_j[l/y_j]]\!]$
(M2) $(\sigma'', \phi'') \in S_{\Sigma''}$
(M3) $v \in \left\|B_j[\Sigma.l/X][l/y_j]\right\|_{\Sigma''}$
for all $1 \le j \le m$

The proof of Lemma 4.8 can be easily adapted to show that this functional also has a unique fixed point, and as before we write $\sigma \in [\![\Sigma]\!]$ if there is some $\phi \in \mathsf{RSF}$ s.t. $(\sigma, \phi) \in \mathit{fix}(\Phi)_\Sigma$.

Lemma 6.3. *Functional Φ, defined in Def. 6.2 has a unique fixpoint $\mathit{fix}(\Phi)$.*

6.2 Semantics of Recursive Specifications

Definition 6.4. *We extend the interpretation of specifications to the new cases, where η maps type 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, $\eta(X) \subseteq \eta(Y)$.
We briefly observe the following facts, most of which are standard.

- By Tarski's Fixed Point Theorem, every monotonic map $f : L \to L$ on a complete lattice $(L, \le)$ has a greatest fixed-point $\mathsf{gfp}(f)$ which is in fact the greatest *post*-fixed point, i.e. whenever $x \le f(x)$ then $x \le \mathsf{gfp}(f)$.
- If $f : L \to L$ additionally preserves meets of decreasing chains $x_0 \ge x_1 \ge \ldots$,

i.e., $f(\bigwedge_i x_i) = \bigwedge_i f(x_i)$, the greatest fixed point can be obtained as

$$\mathsf{gfp}(f) = \bigwedge\{f^n(\top) \mid n \in \mathbb{N}\} \tag{15}$$

where $\top$ is the greatest element of L: Writing $\alpha = \bigwedge\{f^n(\top) \mid n \in \mathbb{N}\}$ it is immediate that $f(\alpha) = \bigwedge\{f^{n+1}(\top) \mid n \in \mathbb{N}\} = \alpha$ is a fixed point of f, and by induction on n, if x is any (post-) fixed point of f then $x \leq f^n(\top)$ for all $n \in \mathbb{N}$, hence $\alpha \geq x$ which shows $\alpha = \mathsf{gfp}(f)$.

- For a complete lattice $(L, \leq)$ and any set A, the set of maps $A \to L$ forms a complete lattice when ordered pointwise,

  $$f_1 \leq f_2 :\Leftrightarrow \text{ for all } a \in A.\ f_1(a) \leq f_2(a)$$

  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: Suppose f, g are monotonic maps in the lattice $L \to L$ with $f \leq g$. Then, by the pointwise ordering,

  $$\mathsf{gfp}(f) = f(\mathsf{gfp}(f)) \leq g(\mathsf{gfp}(f))$$

  which shows $\mathsf{gfp}(f)$ is a post-fixed point of g. This entails $\mathsf{gfp}(f) \leq \mathsf{gfp}(g)$ as the latter is the greatest post-fixed point of g.

- If $f_0 \geq f_1 \geq \ldots$ and $g_0 \geq g_1 \geq \ldots$ are decreasing chains of maps in $L \to L$ s.t. every f_i and g_j is monotonic and preserves meets of descending chains then

  $$\bigwedge_i f_i \circ \bigwedge_j g_j = \bigwedge_n (f_n \circ g_n) \tag{16}$$

  This can be seen by observing $\bigwedge_i f_i \circ \bigwedge_j g_j = \bigwedge_{i,j} f_i \circ g_j$ and the fact that for every $i, j \in \mathbb{N}$ there is $n \in \mathbb{N}$ such that $f_i \geq f_n$ and $g_j \geq g_n$.

  Equation (16) implies $(\bigwedge_i f_i)^n = \bigwedge_i f_i^n$ for all $n \in \mathbb{N}$, and we obtain

  $$\mathsf{gfp}(\bigwedge_i f_i) = \bigwedge_n (\bigwedge_i f_i)^n(\top) = \bigwedge_n \bigwedge_i f_i^n(\top) = \bigwedge_i \mathsf{gfp}(f_i) \tag{17}$$

  i.e., in this case the greatest fixed point operator also preserves meets of descending chains.

The set of admissible subsets of $\mathsf{Val} \times \mathsf{St}$, $\mathcal{A}dm(\mathsf{Val} \times \mathsf{St})$, is closed under arbitrary intersections, hence forms a complete lattice when ordered by set inclusion. Specification environments $\eta : \mathit{TyVar} \to \mathcal{A}dm(\mathsf{Val} \times \mathsf{St})$ 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,

Lemma 6.5 (Well-definedness).

Monotonicity. $[\![\Gamma; \Delta \vdash A]\!]$ *is monotonic:*

$$\eta_1 \leq \eta_2 \ \Rightarrow\ [\![\Gamma; \Delta \vdash A]\!]\rho\eta_1 \subseteq [\![\Gamma; \Delta \vdash A]\!]\rho\eta_2$$

Preservation of Meets. $[\![\Gamma;\Delta \vdash A]\!]$ *preserves meets of descending chains:*

$$\eta_0 \geq \eta_1 \geq \ldots \;\Rightarrow\; [\![\Gamma;\Delta \vdash A]\!]\rho(\textstyle\bigwedge_i \eta_i) = \bigcap_i [\![\Gamma;\Delta \vdash A]\!]\rho\eta_i$$

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

Proof. We can show both properties simultaneously by induction on the structure of A. The only interesting case is where A is $\mu(X)B$.

To show the first part, **Monotonicity**, note that the assumption $\eta_1 \leq \eta_2$ entails

$$\eta_1[X = \chi_1] \leq \eta_2[X = \chi_2] \quad \text{for all } \chi_1 \subseteq \chi_2 \in \mathcal{A}dm(\mathsf{Val} \times \mathsf{St})$$

So for $f_i : \mathcal{A}dm(\mathsf{Val} \times \mathsf{St}) \to \mathcal{A}dm(\mathsf{Val} \times \mathsf{St})$ defined by

$$f_i(\chi) = [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta_i[X = \chi], \qquad i = 1, 2$$

we obtain from the induction hypothesis on B that f_i is monotonic, preserves meets, and $f_1 \leq f_2$. By the observations made above, gfp is monotonic which yields $\mathsf{gfp}(f_1) \subseteq \mathsf{gfp}(f_2)$. Thus

$$[\![\Gamma;\Delta \vdash \mu(X)B]\!]\rho\eta_1 = \mathsf{gfp}(f_1) \subseteq \mathsf{gfp}(f_2) = [\![\Gamma;\Delta \vdash \mu(X)B]\!]\rho\eta_2$$

which shows monotonicity of $[\![\Gamma;\Delta \vdash \mu(X)B]\!]$.

For the second part, **Preservation of Meets**, suppose $\eta_0 \geq \eta_1 \geq \ldots$. If we let $f_i : \mathcal{A}dm(\mathsf{Val} \times \mathsf{St}) \to \mathcal{A}dm(\mathsf{Val} \times \mathsf{St})$,

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

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

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

the induction hypothesis (Preservation of Meets) shows that each f_i preserves meets:

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

We obtain

$$\begin{aligned} [\![\Gamma;\Delta \vdash A]\!]\rho(\textstyle\bigwedge_i \eta_i) &= \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_i \eta_i)[X = \chi]) && \text{by definition} \\ &= \mathsf{gfp}(\lambda\chi.[\![\Gamma;\Delta, X \vdash B]\!]\rho(\textstyle\bigwedge_i \eta_i[X = \chi])) && \text{pointwise meet} \\ &= \mathsf{gfp}(\lambda\chi.\textstyle\bigcap_i [\![\Gamma;\Delta, X \vdash B]\!]\rho\eta_i[X = \chi]) && \text{by induction} \\ &= \mathsf{gfp}(\textstyle\bigwedge_i f_i) && \text{pointwise meet} \\ &= \textstyle\bigcap_i \mathsf{gfp}(f_i) && \text{by (17)} \\ &= \textstyle\bigcap_i [\![\Gamma;\Delta \vdash A]\!]\rho\eta_i && \text{by definition} \end{aligned}$$

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. By induction on A. ◻

6.3 Syntactic Approximations

Recall the statement of Lemma 5.1, one of the key lemmas in the proof of the soundness theorem:

$$\text{for all } \sigma, \Sigma, l \text{ and } \vdash A, \text{ if } \sigma \in [\![\Sigma]\!] \text{ and } \vdash \Sigma.l <: A \text{ then } (l, \sigma) \in [\![\vdash A]\!] \qquad (18)$$

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.

Instead, we consider finite approximations as in [3], 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* $k \in \mathbb{N}$, *we define* $A|^k$ *as*

- $A|^0 = \top$
- $\mu(X)A|^{k+1} = A[\mu(X)A/X]|^{k+1}$
- $[f_i{:}\,A_i{}^{i=1\ldots n},\ m_j{:}\,\varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]|^{k+1} =$
  $[(f_i : A_i|^k)^{i=1\ldots n}, m_j : \varsigma(y_j)B_j|^k :: T_j{}^{j=1\ldots m}]$
- $\top|^{k+1} = \top$
- $X|^{k+1} = X$
- $Bool|^{k+1} = Bool$

Note that, as in [3], 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 already ruled out troublesome cases like $\mu(X)X$.

Properties of Approximations

Unfortunately, approximations $A|^k$ as defined above are not in fact approximating A with respect to the subspecification relation $<:$, the reason being the

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

Table 6. The generalised object subspecification rule

invariance in field specifications. For example, if $A \equiv [\mathsf{f}_1 : X, \mathsf{f}_2 : Bool]$ then

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

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

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

for appropriate Γ and Δ. But subspecifications of object specifications can only be derived for equal components f_1 with the rules of Sect. 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 6.

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 then there exists $n \in \mathbb{N}$ such that for all $k \geq n$, $A \equiv A|^k$.*

Proof. The proofs are by induction on the lexicographic order on k and the number of μ in head position, then considering cases for the specification A. We only show the first claim, the others are similar.

Suppose $k = 0$, then the results follow immediately from $A|^0 = \top$. For $k > 0$, the proof is by a case distinction on the shape of A:

- A is $\top$. Then $A|^k = \top$ and the required subtyping follows from transitivity.
- A is X or $Bool$. Similarly, from $A|^k = X$ and $A|^k = Bool$, resp.

- A is $\mu(X)B$. Then, by induction hypothesis,

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

By definition of approximations, the latter equals $A|^k$. Moreover,

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

by the (unfold) rule, and transitivity then yields $\Gamma; \Delta \vdash A <:^* A|^k$.

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

$$A|^k = [\mathsf{f}_i : A_i|^{k-1}, \mathsf{m}_j : B_j|^{k-1}]$$

By induction hypothesis we obtain that $\Gamma; \Delta \vdash A_i <:^* A_i|^{k-1}$ and that $\Gamma, y_j; \Delta \vdash B_j <:^* B_j|^{k-1}$ which entails

$$\Gamma; \Delta \vdash [\mathsf{f}_i : A_i, \mathsf{m}_j : B_j :: T_j] <:^* [\mathsf{f}_i : A_i, \mathsf{m}_j : B_j :: T_j]|^k$$

by the (modified) subspecification rule, as required.

□

Soundness of Subspecification

Soundness of subspecification is easily established:

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

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

- (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,

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

- For the case of (Object), we must have

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

and

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

such that $\Gamma; \Delta \vdash A_i <:^* A_i'$ and $\Gamma, y_j; \Delta \vdash B_j <:^* B_j'$ and $\vdash_{\mathsf{fo}} T_j \to T_j'$. By induction hypothesis,

$$[\![\Gamma; \Delta \vdash A_i]\!]\rho\eta \subseteq [\![\Gamma; \Delta \vdash A_i']\!]\rho\eta$$

and

$$[\![\Gamma, y_j; \Delta \vdash B_j]\!](\rho[y_j := l])\eta \subseteq [\![\Gamma, y_j; \Delta \vdash B'_j]\!](\rho[y_j := l])\eta$$

for all $1 \leq i \leq n$, $1 \leq j \leq m$ and $l \in \mathsf{Loc}$. Moreover, by soundness of $\vdash_{\mathsf{fo}}$ we know

$$[\![[\Gamma], y_j \vdash T_j]\!](\rho[y_j := l]) \subseteq [\![[\Gamma], y_j \vdash T'_j]\!](\rho[y_j := l])$$

So by definition of $[\![\Gamma; \Delta \vdash A]\!]$, $[\![\Gamma; \Delta \vdash B]\!]$,

$$(l, \sigma) \in [\![\Gamma; \Delta \vdash A]\!]\rho\eta \text{ implies } (l, \sigma) \in [\![\Gamma; \Delta \vdash B]\!]\rho\eta$$

- Finally, for the (Rec) 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

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

which shows 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.

□

Relating Semantics and Syntactic Approximations

Lemma 6.9, in combination with Lemma 6.8(1), shows $[\![\Gamma; \Delta \vdash A]\!]\rho\eta \subseteq [\![\Gamma; \Delta \vdash A|^k]\!]\rho\eta$ for all $\eta \models \Delta$ and $k \in \mathbb{N}$, i.e.,

$$[\![\vdash A]\!]\eta \subseteq \textstyle\bigcap_{k \in \mathbb{N}} [\![\vdash A|^k]\!]\eta \tag{19}$$

For the reverse inclusion, we use the characterisation of greatest fixed points as 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. By 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. Then $\Gamma; \Delta \vdash A[B/X]|^l = B|^l <:^* B|^k|^l = A[B|^k/X]|^l$.
  - A is $\top, Bool$ or $Y \neq X$. Then $\Gamma; \Delta \vdash A[B/X]|^l = A|^l <:^* A|^l = A[B|^k/X]|^l$.
  - A is $[\mathsf{f}_i : A_i{}^{i=1...n}, \mathsf{m}_j : \varsigma(y_j)B_j :: T_j{}^{j=1...m}]$. Then, by induction hypothesis,
  $$\Gamma; \Delta \vdash A_i[B/X]|^{l-1} <:^* A_i[B|^k/X]|^{l-1}$$
  and
  $$\Gamma, y_j{:}A; \Delta \vdash B_j[B/X]|^{l-1} <:^* B_j[B|^k/X]|^{l-1}$$
  for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Hence,
  $$\Gamma; \Delta \vdash A[B/X]|^l <:^* [\mathsf{f}_i : A_i[B|^k/X], \mathsf{m}_j : B_j[B|^k/X]]|^l = A[B|^k/X]|^l$$
  - A is $\mu(Y)C$, without loss of generality Y not free in B. Then by induction hypothesis we find $\Gamma; \Delta \vdash C[A/Y][B/X]|^l <:^* C[A/Y][B|^k/X]|^l$. Using properties of syntactic substitutions, we calculate
  $$\begin{aligned} A[B/X]|^l &= \mu(Y)(C[B/X])|^l \\ &= C[B/X][(\mu(Y)(C[B/X]))/Y]|^l \\ &= C[B/X][(A[B/X])/Y]|^l \\ &= C[A/Y][B/X]|^l \end{aligned}$$
  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* $\eta \models \Delta$,

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

Proof. By (19), all that remains to show 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 $k \geq n$, $A|^k = A$, and so in fact

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

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

- A is $\top, X$ or $Bool$. Then as above, there exists $n \in \mathbb{N}$ such that for all $k \geq n$, $A|^k = A$ and we are done.
- A is $[\mathsf{f}_i\colon A_i{}^{i=1\ldots n},\ \mathsf{m}_j\colon \varsigma(y_j)B_j{::}T_j{}^{j=1\ldots m}]$. Then, by 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 $1 \leq i \leq n$ and $1 \leq j \leq m$. 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

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

  forms a descending chain of environments. Hence we can calculate

  $$\begin{array}{rll}
  f_A(\alpha) & = [\![\Gamma; \Delta, X \vdash B]\!]\rho\eta[X = \alpha] & \text{def. of } f_A \\
  & = [\![\Gamma; \Delta, X \vdash B]\!]\rho(\bigwedge_k \eta[X = [\![\Gamma; \Delta \vdash A|^k]\!]\rho\eta]) & \text{def. of } \alpha \text{ and meets} \\
  & = \bigcap_{k\in\mathbb{N}}[\![\Gamma; \Delta, X \vdash B]\!]\rho\eta[X = [\![\Gamma; \Delta \vdash A|^k]\!]\rho\eta] & \text{Lemma 6.5, meets} \\
  & = \bigcap_{k\in\mathbb{N}}[\![\Gamma; \Delta \vdash B[A|^k/X]]\!]\rho\eta & \text{Lemma 6.6} \\
  & \supseteq \bigcap_{k\in\mathbb{N}}\bigcap_{l\in\mathbb{N}}[\![\Gamma; \Delta \vdash B[A|^k/X]|^l]\!]\rho\eta & \text{Induction Hyp.} \\
  & \supseteq \bigcap_{m\in\mathbb{N}}[\![\Gamma; \Delta \vdash B[A/X]|^m]\!]\rho\eta & \text{Lemma 6.10} \\
  & = \bigcap_{k\in\mathbb{N}}[\![\Gamma; \Delta \vdash A|^k]\!]\rho\eta & \text{Def. of } \mu(X)A|^k
  \end{array}$$

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

This concludes the proof. □

6.4 Soundness

After the technical development in the preceding subsection we can now prove (18). From this result the soundness proof of the logic extended with recursive specifications then follows, along the lines of the proof presented in Section 5 for finite specifications.

Lemma 6.12. *For all* σ, Σ, l *and* $\vdash A$, *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 5.1. 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

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

by Lemma 6.11. □

7 Conclusion

Based on 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 wrt. an operational semantics, our techniques allowed us to distinguish the notions of derivability and validity. Further, we used the denotational framework to extend the logic to recursive object specifications. In comparison to a similar logic presented in [8] our notion of subspecification is structural rather than nominal.

Although our proof is very much 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 Γ cannot reflect the dynamic aspect of the store (which is growing) one uses store specifications Σ. They do not show up in the Abadi-Leino logic as they are automatically preserved by programs. This is shown as part of the soundness proof rather than being a proof obligation on the level of derivations. By contrast to [2], we can view store specifications as predicates on stores which 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 subsumption (nor weakening). This led us to distinguish store specifications from the specifications of individual objects.

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 $\Sigma' \geqslant \Sigma$. This will account 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), specification syntax appears in the definition of $\sigma \in [\![\Sigma]\!]$ (Def. 4.7). 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 subspecification relation instead.

The proof of Theorem 4.8, establishing the existence of store predicates, provides an explanation 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.

Abadi and Leino's logic is peculiar in that verified programs need to preserve store specifications. Put differently, only properties which are in fact preserved can be expressed in the logic. In particular, specifications of field values are limited such that properties like e.g. $\mathsf{self.hd} \leq \mathsf{self.tail.hd}$, stating that a list is sorted, cannot be expressed. In future work we thus plan to investigate how a logic can be set up where

- methods are specified by pre-/post-conditions that explicitly state invariance properties during execution of the method code.
- methods can be specified by pre-/post-conditions that can refer to other methods. This is important for simulating methods that act like higher-order functions (e.g. the map function for lists).
- methods can have additional parameters.
- method update is allowed. In the setting of Abadi and Leino this would require that the new method body satisfies the old specification (in order to establish invariance). More useful would be a "behavioural" update where result and transition specifications of the overriding method are subspecifications of the original method.

The results established in this paper pave the way for the above line of research.

Acknowledgement We wish to thank Thomas Streicher for discussions and comments.

References

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

[2] M. Abadi and K. R. M. Leino. A logic of object-oriented programs. In N. Dershowitz, editor, *Verification: Theory and Practice*, pages 11–41. Springer, 2004.

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

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

[5] F. S. de Boer. A WP-calculus for OO. In W. Thomas, editor, *Foundations of Software Science and Computation Structures*, volume 1578 of *Lecture Notes in Computer Science*, pages 135–149, 1999.

[6] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object-oriented languages: Logical models and tools. In C. Hankin, editor, *Programming Languages and Systems—ESOP'98, 7th European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 105–121, Mar. 1998.

[7] C. A. R. Hoare. An Axiomatic Basis of Computer Programming. *Communications of the ACM*, 12:576–580, 1969.

[8] K. R. M. Leino. Recursive object types in a logic of object-oriented programs. In C. Hankin, editor, *7th European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 170–184, Mar. 1998.

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

[10] L. C. Paulson. *Logic and Computation : Interactive proof with Cambridge LCF*, volume 2 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1987.

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

[12] A. Poetzsch-Heffter and P. Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *European Symposium on Programming*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176, 1999.

[13] U. S. Reddy. Objects and classes in algol-like languages. *Information and Computation*, 172(1):63–97, January 2002.

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

[15] B. Reus. Modular semantics and logics of classes. In M. Baatz and J. A. Makowsky, editors, *Computer Science Logic*, volume 2803 of *Lecture Notes in Computer Science*, pages 456–469. Springer Verlag, 2003.

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

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

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

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

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