Relating Semantic Models for the Object Calculus
Preliminary Report

Luca Aceto    Hans Hüttel    Anna Ingólfsdóttir
    Josva Kleist *

Abstract

Abadi and Cardelli have investigated several versions of the object calculus, a calculus for describing central features of object-oriented programs, with particular emphasis on various type systems. In this paper we study the properties of a denotational semantics due to Abadi and Cardelli vis-à-vis the notion of observational congruence for the calculus $\text{Ob}_{1<\mu}$. In particular, we prove that the denotational semantics based on partial equivalence relations is correct with respect to observational congruence. By means of a counter-example, we argue that the denotational model is not fully abstract with respect to observational congruence. In fact, the model is able to distinguish objects that have the same behaviour in every $\text{Ob}_{1<\mu}$-context.

1 Introduction

In [AC96] Abadi and Cardelli present and investigate several versions of the $\varsigma$-calculus, a calculus for describing central features of object-oriented programs, with particular emphasis on various type systems. These object calculi formalize key aspects of object-oriented programming languages, such as method update and object subsumption, without recourse to complex encodings of these features into general theories of types or various kinds of

*Address: Dep. of Computer Science, Aalborg University, Fredrik Bajersvej 7, 9220 Aalborg, Denmark. Telephone: +45 9635 8080. Fax: +45 9815 9889. Email: {luca,hans,annai,kleist}@cs.auc.dk
\(\lambda\)-calculi. Their simplicity, together with their clearly object-oriented flavour, has made these calculi an important area of research in the field of the semantics of object-oriented languages. As a natural step in the development of the theory of their object calculi, Abadi and Cardelli have developed equational theories that can be used to prove certain equalities between objects in a purely syntactic way [AC96]. The equational theories are sound with respect to a denotational semantics based on partial equivalence relations [AC96, Chapter 14].

Notions of program equivalence are central to the theory and practice of programming languages. They form the basis for program optimization, and can be used to justify correctness preserving transformations performed by program manipulation systems. Program equivalences are typically defined according to the following paradigm:

1. A collection of terms that are considered to be directly executable and observable are designated as programs, and their behaviour is defined;
2. Two arbitrary terms are defined to be equivalent iff they have the same behaviour in every program context.

The resulting notion of program equivalence is usually referred to as observational congruence [Mey88]. Observational congruence for the first order object calculus with subtyping \(\text{Ob}_{1<\mu}\) has been defined in [GR96] thus: Two programs are observationally congruent iff they have the same termination behaviour in all contexts of type boolean. Following earlier work on functional languages, in op. cit. the calculus \(\text{Ob}_{1<\mu}\) is equipped with a labelled transition system semantics, and its associated notion of bisimulation equivalence is proven to coincide with observational congruence. Like the denotational model presented in [AC96, Chapter 14], observational congruence soundly models Abadi and Cardelli’s equational theory for objects (cf. [GR96, Thm. 3]).

The results discussed so far provide one with two different semantic models for the calculus \(\text{Ob}_{1<\mu}\) that soundly model the equational theory underlying that version of the object calculus. However, the acid test for the goodness of any denotational model for programming languages is the nature of the connection between the mathematical meaning it assigns to programs and their computational behaviour. In particular, a denotational model should be correct [Sto88a] in the sense that it identifies only terms that are related by observational congruence. Models with the ideal property of identifying
exactly those terms that are observationally congruent are called fully abstract. Perhaps surprisingly, the literature on the object calculi lacks a study of the relationship between Abadi and Cardelli’s denotational semantics and observational congruence, as studied by Gordon and Rees. This is the aim of this study.

In this paper we study Abadi and Cardelli’s denotational semantics vis-à-vis observational congruence over the calculus $\text{Ob}_{1<\mu}$. In particular, we prove that the denotational semantics based on partial equivalence relations of [AC96, Chapter 14] is correct with respect to observational congruence of objects (Thm. 11). As an important stepping stone towards this correctness result, we show that the denotational semantics is computationally adequate with respect to the reduction semantics (Thm. 8), and that a program of boolean type evaluates to a boolean value $v$ iff its denotation equals that of $v$ (Corollary 10). By means of a counter-example, we argue that the denotational model is not fully abstract with respect to observational congruence. In fact, the model is able to distinguish objects that have the same behaviour in every $\text{Ob}_{1<\mu}$-context. As a byproduct of our results we obtain an alternative proof of the soundness of the equational theory with respect to bisimulation (Propn. 12).

We end this introduction with a brief road-map to the contents of the paper. Section 2 introduces the abstract syntax and reduction semantics of the object calculus $\text{Ob}_{1<\mu}$. In Section 3 we present the type system for $\text{Ob}_{1<\mu}$. Section 4 is devoted to the typed equational theory of $\text{Ob}_{1<\mu}$. The labelled transition semantics of the calculus and the notion of bisimulation equivalence are introduced in Section 5. Section 6 gives a brief overview of the denotational model of $\text{Ob}_{1<\mu}$ and its types. Finally, Section 7 presents our main result, viz. that the denotational model is correct, but not fully abstract. Directions for further work are discussed in Section 8.

2 The $\varsigma$-calculus and its reduction semantics

There are various versions of the $\varsigma$-calculus. In this paper we shall consider what is essentially the first order object calculus with recursive types of [AC96, Chapter 9] with booleans added. Our presentation will closely follow [GR96], and the reader is referred to op. cit. for more details. The set of object terms, $\text{Obj}$, is defined by the following abstract syntax:
\[a, b, c, e ::= \begin{array}{ll}
l_i = \varsigma(x_i : A_i)b_i & \text{objects} \\
x & \text{self variables} \\
a.l & \text{method activation} \\
l_i = \varsigma(x : A)b_i & \text{method override} \\
fold(A, a) & \text{recursive fold/unfold} \\
if(a, b_1, b_2) & \text{booleans} \\
true & \text{booleans} \\
false & \text{booleans}
\end{array}\]

Here \(x_i \in S\text{Var}\) ranges over self variables, \(l_i \in M\text{Names}\) ranges over a countable collection of method names and \(A_i\) is a type (cf. Section 3 for information on types). We use the standard notions of free and bound variables in terms, with \(\varsigma(x : A)\) as the binding construct. A (closed) substitution \(\sigma\) is a map from self variables to (closed) object terms. For a term \(a\) and substitution \(\sigma\), we write \(a\sigma\) for the result of substituting the term \(\sigma(x)\) for each free occurrence of \(x\) in \(a\). We refer to, e.g., [Sto88b] for the standard details of such an operation in the presence of binders. In what follows, the notation \(\{\vec{a}/\vec{x}\}\), where \(\vec{a}\) and \(\vec{x}\) are lists of terms and distinct variables of the same length, will be used to denote the substitution mapping each variable \(x_i\) in \(\vec{x}\) to the term \(a_i\), and acting like the identity on all the other variables. A value, denoted by \(v\), is either an object \([l_i = \varsigma(x_i : A_i)b_i \ i \in I]\), a boolean value \((\text{true}, \text{false})\) or a folded value \((\text{fold}(A, v))\).

The presentation of the \(\varsigma\)-calculus given in [GR96] uses a small-step reduction semantics, which is also used in the definition of the labelled transition semantics in Section 5. This we now proceed to present.

Let \(a = [l_i = \varsigma(x_i : A_i)b_i \ i \in I]\). The reduction rules are given by

\[
\begin{align*}
  a.l_k & \leadsto b_k \{a/x_k\} \quad (k \in I) \\
  a.l_k & \leftarrow \varsigma(x : A)b \leadsto [l_k = \varsigma(x : A)b_i \ i \in I \setminus \{k\}] \quad (k \in I) \\
  \text{if}(\text{true}, b_1, b_2) & \leadsto b_1 \\
  \text{if}(\text{false}, b_1, b_2) & \leadsto b_2 \\
  \text{unfold}(\text{fold}(A, v)) & \leadsto v
\end{align*}
\]

The activation of the method \(l_k\) of object \(a\) results in the method body being activated with the self variable being bound to the original object. Method override results in an object with the overridden method replaced by the new method.

The reduction order is leftmost; this is expressed via evaluation contexts \((C[-])\) which have the following abstract syntax (with \([-]\) denoting the hole
of the context):

\[ C[-] ::= [-].l \mid [-].l \leftarrow \varsigma(x:A)b \mid \text{unfold}([-]) \mid \text{fold}(A, [-]) \mid \text{if}([-), a_1, a_2) \]

and an evaluation strategy given by the reduction rule

\[ \frac{a \leadsto b}{C[a] \leadsto C[b]} \]

We write \( a \downarrow v \) ("\( a \) converges to the value \( v \)") if there is a terminating reduction sequence \( a \leadsto a_1 \leadsto \cdots v \), and \( a \downarrow \) if \( a \downarrow v \) for some \( v \). As the reduction relation is deterministic [GR96, Lemma 6], for every term \( a \) there is at most one value \( v \) for which \( a \downarrow v \) holds.

3 Types

One of the main motivations for the \( \varsigma \)-calculus is that of studying various type systems of object-oriented programming languages within a unified framework. In this paper we shall consider the type system \( \text{Ob}_{1:<\mu} \) from [AC96, Chapter 9] as presented in [GR96]; this is a first-order type system with recursion and subtyping.

3.1 The type language

The set of \( \text{Ob}_{1:<\mu} \) type expressions is defined via the following abstract syntax:

\[ A ::= \text{Bool} \mid \{l_i : A_i \}_{i \in I} \mid \text{Top} \mid \mu(X)A \mid X \]

Here \( \text{Bool} \) denotes the only ground type, namely that of truth values. The type \( \{l_i : A_i \}_{i \in I} \) denotes an object record type, where the method \( l_i \) has type \( A_i \). \( \text{Top} \) denotes the most general or unspecified type, \( \mu(X)A \) is a recursive type and \( X \) ranges over TypeVar, the set of type variables. A type expression \( A \) is closed if every occurrence of a type variable \( X \) in \( A \) is within the scope of a \( \mu(X) \) binder. We write Type for the collection of closed, well-formed type expressions (cf. [GR96, Section 2] for details). Elements of Type will be referred to as types.
3.2 Assigning types to objects

\( \text{Ob}_{1<:\mu} \) has two kinds of judgments: Type judgments and subtyping judgments. Type judgments are of the form \( \Gamma \vdash a{:}A \) and state that the object \( a \) has type \( A \) under the assumptions in \( \Gamma \), where \( \Gamma \) describes typing assumptions for free self variables. For instance, \( \Gamma(x) = A \) states that we assume that the free self variable \( x \) has type \( A \). If \( \Gamma \) is empty we shall sometimes just write \( a{:}A \) instead of \( \emptyset \vdash a{:}A \). Whenever the typing assumptions in \( \Gamma \) are extended with the additional assumption \( x{:}A \), we write this as \( \Gamma, x{:}A \) (assuming here that no assumption about the type of \( x \) occurs in \( \Gamma \)).

An object \( a \) has type \( A \) under the set of assumptions \( \Gamma \) if \( \Gamma \vdash a{:}A \) can be inferred from the type assignment rules in Table 2. An object term \( a \) is said to be a program of type \( A \) if we can infer that \( \emptyset \vdash a{:}A \).

The type system \( \text{Ob}_{1<:\mu} \) also incorporates a notion of subtyping, which intuitively captures the idea that some types are more general than others. The expression \( A <: B \) denotes that \( A \) is a subtype of \( B \) and thus that objects of type \( A \) may be used in lieu of objects of type \( B \).

Subtyping judgments \( \Gamma \vdash A <: B \) state that the type \( A \) is a subtype of \( B \), given the subtyping assumptions in \( \Gamma \). Here the typing assumptions in \( \Gamma \) describe subtyping constraints on type variables. \( \Gamma(X) = A \) states that we assume \( X <: A \).

The subtyping relation is defined by the inference rules of Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Sub Refl)</td>
<td>( \Gamma \vdash A )</td>
<td>( \Gamma \vdash A &lt;: A )</td>
</tr>
<tr>
<td>(Sub Trans)</td>
<td>( \Gamma \vdash A_1 &lt;: A_2 ), ( \Gamma \vdash A_2 &lt;: A_3 )</td>
<td>( \Gamma \vdash A_1 &lt;: A_3 )</td>
</tr>
<tr>
<td>(Sub X)</td>
<td>( \Gamma(X) = A )</td>
<td>( \Gamma \vdash X &lt;: A )</td>
</tr>
<tr>
<td>(Sub Top)</td>
<td>( \Gamma \vdash A )</td>
<td>( \Gamma \vdash A &lt;: \text{Top} )</td>
</tr>
<tr>
<td>(Sub Obj)</td>
<td>( J \subseteq I ), ( \Gamma \vdash A_i \forall i \in I )</td>
<td>( \Gamma \vdash [i{:}A_i\ i \in I] &lt;: [i{:}A_j\ j \in J] )</td>
</tr>
<tr>
<td>(Sub Rec)</td>
<td>( \Gamma \vdash \mu(X_1)A_1 ), ( \Gamma \vdash \mu(X_2)A_2 ), ( \Gamma, X_1 &lt;: \text{Top}, X_2 &lt;: \text{Top}, X_1 &lt;: X_2 \vdash A_1 &lt;: A_2 )</td>
<td>( \Gamma \vdash \mu(X_1)A_1 &lt;: \mu(X_2)A_2 )</td>
</tr>
</tbody>
</table>

Table 1: The subtyping relation
\( \text{(VAR)} \quad \Gamma(x) = A \quad \frac{\Gamma \vdash x : A}{\Gamma} \)  

\( \text{(SELECT)} \quad \frac{\Gamma \vdash a : [l_i : B_i \ i \in I]}{\Gamma \vdash a.l_j : B_j \ j \in I} \)  

\( \text{(OBJECT)} \quad \frac{\Gamma, x_i : A \vdash b_i : B_i \ \forall i \in I \quad \exists i \in I \quad A \equiv [l_i : B_i \ i \in I]}{\Gamma \vdash [x_i : A]b_i : [l_i : B_i \ i \in I] : A} \)  

\( \text{(UPDATE)} \quad \frac{\Gamma \vdash a : A \quad \Gamma, x : A \vdash b : B_j \ j \in I \quad \forall i \in I \quad A \equiv [l_i : B_i \ i \in I]}{\Gamma \vdash a.l_j : [l_i : B_i \ i \in I] \vdash \varsigma(x : A)b : A} \)  

\( \text{(FOLD)} \quad \frac{\Gamma \vdash a : B\{A/X\} \quad A \equiv \mu(X)B}{\Gamma \vdash \text{fold}(A, a) : A} \)  

\( \text{(UNFOLD)} \quad \frac{\Gamma \vdash a : A \quad A \equiv \mu(X)B}{\Gamma \vdash \text{unfold}(a) : B\{A/X\}} \)  

\( \text{(IF)} \quad \frac{\Gamma \vdash a : \text{Bool} \quad \Gamma \vdash a_1, a_2 : A}{\Gamma \vdash \text{if}(a, a_1, a_2) : A} \)  

\( \text{(BOOL)} \quad \frac{b \in \{\text{true, false}\}}{\Gamma \vdash b : \text{Bool}} \)  

\( \text{(SUBSUMP)} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash A_1 < : A_2}{\Gamma \vdash a : A_2} \)  

\text{Table 2: Type assignment}
**Example 1** As noted by Abadi and Cardelli, for any type $A$, there is a divergent object $\Omega_A$ definable as $[l = \varsigma(x : [l : A]) x x l].l$. Note that $\Omega_A \leadsto \Omega_A$, and thus this object term is indeed divergent.

The following lemma collects some basic properties of the type assignment, and of its interaction with the reduction relation.

**Lemma 1 ([GR96])**

1. If $\Gamma \vdash e : A$, then the free variables in $e$ are contained in the domain of $\Gamma$.

2. If $a : A$ and $a \leadsto b$, then $b : A$.

## 4 Equational theory

Equational theories allow us to prove certain equalities between objects in a purely syntactic way. In this section we present the equational theory for $\text{Ob}_{1<:A}$.

All judgments are of the form $\Gamma \vdash a \leftrightarrow b : A$, where $\Gamma$ is a type environment mapping self variables to types, $a$ and $b$ are objects and $A$ is a type. The intended interpretation of this judgment is that, under the assumptions in $\Gamma$ about the free variables in $a$ and $b$, the expressions $a$ and $b$ are considered equal as objects of type $A$.

The rules in Table 3 establish symmetry and transitivity, plus a limited form of reflexivity; a general rule for reflexivity is not needed, as it follows as a derived rule. Table 4 collects congruence rules for objects and rules $(\text{Eq Symm})$

<table>
<thead>
<tr>
<th>$\Gamma \vdash a \leftrightarrow b : A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash b \leftrightarrow a : A$</td>
</tr>
</tbody>
</table>

$(\text{Eq Trans})$

<table>
<thead>
<tr>
<th>$\Gamma \vdash a \leftrightarrow b : A$, $b \leftrightarrow c : A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash a \leftrightarrow c : A$</td>
</tr>
</tbody>
</table>

$(\text{Eq x})$

<table>
<thead>
<tr>
<th>$\Gamma(x) = A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash x \leftrightarrow x : A$</td>
</tr>
</tbody>
</table>

Table 3: Equivalence-inducing equational rules

corresponding to the clauses of the reduction semantics. Finally, we have in Table 5 the rules for subtyping.
Table 4: Equational rules specific to the calculus

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eq Object</td>
<td>$A \equiv [l_i:B_i^{i\in I}]$</td>
</tr>
<tr>
<td>Eq Select</td>
<td>$\Gamma \vdash a \leftrightarrow b : [l_i:B_i^{i\in I}]$  \quad $j \in I$</td>
</tr>
<tr>
<td>Eq If</td>
<td>$\Gamma \vdash b \leftrightarrow b' : \text{Bool}$  \quad $\Gamma \vdash b_1 \leftrightarrow b'_1 : B$, $b_2 \leftrightarrow b'_2 : B$</td>
</tr>
<tr>
<td>Eq Fold</td>
<td>$\Gamma \vdash \text{fold}(A,a) \leftrightarrow \text{fold}(A,b) : A$</td>
</tr>
<tr>
<td>Eq Override</td>
<td>$\Gamma \vdash a \leftrightarrow b : \mu(X)B$  \quad $\Gamma \vdash a \leftrightarrow a' : A$  \quad $\Gamma \vdash x:A \vdash b \leftrightarrow b' : B_j$  \quad $j \in I$</td>
</tr>
<tr>
<td>Eq Unfold</td>
<td>$\Gamma \vdash \text{unfold}(a) \leftrightarrow \text{unfold}(b) : B{A/X}$</td>
</tr>
<tr>
<td>Eval Select</td>
<td>$\Gamma \vdash a : A$  \quad $A \equiv [l_i:B_i^{i\in I}]$  \quad $a \equiv [l_i=\varsigma(x_i:A)b_i^{i\in I\cup J}]$</td>
</tr>
<tr>
<td>Eval Override</td>
<td>$\Gamma \vdash a : A$  \quad $\Gamma \vdash x:A \vdash b : B_j$  \quad $j \in I$</td>
</tr>
<tr>
<td>Eval If1</td>
<td>$\Gamma \vdash \text{if}(\text{true},b_1,b_2) \leftrightarrow b_1 : B$</td>
</tr>
<tr>
<td>Eval If2</td>
<td>$\Gamma \vdash \text{if}(\text{false},b_1,b_2) \leftrightarrow b_2 : B$</td>
</tr>
</tbody>
</table>
The most interesting rule is \((\text{EQ SUB OBJECT})\), defined in Table 5 which allows one to prove equalities between objects with different collections of methods.

\section{A labelled transition semantics}

In this section we shall give a short review of the labelled transition semantics proposed by Gordon and Rees in \cite{GR96}. In \textit{op. cit}. only terms of matching types are considered to be related semantically. This is formalized by introducing the notion of \textit{proved programs}, i.e. elements of the form \(a_A\) where \(a\) is a program of type \(A\). Let \(\text{Rel}\) be the universal relation on proved programs of the same type, i.e.

\[\text{Rel} = \{(a_A, b_A) | a:A\text{ and } b:A\}\].

The observable actions, \(\alpha \in \text{Act}\), take the following forms:

\[\alpha ::= \text{true} | \text{false} | l | l \leftarrow \varsigma(x)b | \text{unfold}.
\]

The family \(\{\alpha \rightarrow | \alpha \in \text{Act}\}\) of transition relations over proved programs is defined as the set of the least relations satisfying the rules in Table 6. The definition of bisimulation equivalence over proved programs is then basically standard \cite{Par81, Mil89}.

\textbf{Definition 2 (Bisimulation)} Bisimilarity \(\sim\) is the greatest subset of \(\text{Rel}\) that satisfies the following: \(a_A \sim b_A\) if and only if
\( a \downarrow v \in \{\text{true, false}\} \)

\[ a_{\text{Bool}} \xrightarrow{v} a_{\text{Top}} \]

(i) \( a_A \xrightarrow{\alpha} a'_{A'} \Rightarrow \exists b'_{A'} . (b_A \xrightarrow{\alpha} b'_{A'} \land a'_{A'} \sim b'_{A'}) \) and

(ii) \( b_A \xrightarrow{\alpha} b'_{A'} \Rightarrow \exists a'_{A'} . (a_A \xrightarrow{\alpha} a'_{A'} \land a'_{A'} \sim b'_{A'}) \).

If \( a_A \sim b_A \) we say that \( a_A \) and \( b_A \) are bisimilar.

A natural notion of equivalence for the object calculus is that of observational congruence [Mey88] where two terms are considered equivalent if they have the same termination behaviour in all contexts of type \( \text{Bool} \). We shall only consider well typed contexts and we write \(-:B \vdash C[-]:A\) if the context \( C \) has type \( A \) under the assumption that the hole has type \( B \).

**Definition 3 (Observational congruence)** We write \( a_B \overset{A}{\sim} b_B \) iff for all contexts satisfying \(-:B \vdash C[-]:A\) we have \( C[a] \downarrow \) iff \( C[b] \downarrow \).

Intuitively, contexts should be considered as the possible tests that an object can be subjected to. One should note that the naturalness of the notion of observational congruence crucially depends upon the choice of observable types. For instance, it is easy to see that \( \text{true} \not\sim \Omega_{\text{Top}} \), which violates the rule (Eq \( \text{Top} \)) expressing that all objects are to be considered equal at type \( \text{Top} \). Amongst the relations \( \overset{A}{\sim} \), congruence at type \( \text{Top} \), viz. \( \overset{\text{Top}}{\sim} \), is the most
discriminating and $\models_{\text{Bool}}$ the least. Rule (EQ Top) holds for $\models_{\text{Bool}}$ and, for that reason and by analogy with [Pl67], Gordon and Rees choose $\models_{\text{Bool}}$ as the appropriate notion of observational congruence for $\text{Ob}_{\leq \mu}$.

In [GR96] Gordon and Rees show that bisimulation coincides with observational congruence and that these relations validate the equational theory of Tables 3–5.

6  The denotational semantics

In this section we shall give a short description of the denotational semantics given in [AC96, Chapter 14]. The interested reader is referred ibidem for more details.

The denotational semantics is based on a two-level approach. The first level consists of a standard cpo model for interpreting untyped objects. Types are then interpreted as certain kinds of partial equivalence relations (pers) over the object domain. In this two-level semantics the objects $a$ and $b$ are considered equal in the type $A$ if $([a], [b]) \in [\langle A \rangle]$, where $[a]$, $[b]$ and $[\langle A \rangle]$ are the corresponding interpretations.

6.1  The untyped model

The untyped model is a cpo obtained as a solution to the domain equation

$$D = \{\bot\} + \{\#, \text{ff}\} + (D \to D) + (\text{MNames} \to D)$$

where $\text{MNames} = \{l_1, l_2 \ldots\}$ is a countable set of method names, $D \to D$ and $(\text{MNames} \to D)\bot$ have the usual meaning and $+$ is coalesced sum. The solution is obtained as the limit of the following sequence of iterates:

\[
\begin{align*}
D_0 &= \{\bot\} \\
D_{n+1} &= \{\ast\} + \{\#, \text{ff}\} + (D_n \to D_n) + (\text{MNames}_n \to D_n)\bot
\end{align*}
\]

where $\text{MNames}_n = \{l_1, \ldots, l_n\}$

We consider $D_i$ as being a subset of $D$.

There is an increasing sequence, $p_n : D \to D_n$, of projections related to the model with the identity map as its least upper bound. If $x \in D$ and $p_n(x) = x$ for some $n$, then $x$ is finite. The rank of a finite element $x$ is the least $n$ such that $p_n(x) = x$. 

12
We use \( \langle \langle l_1 = x_1, \ldots, l_n = x_n \rangle \rangle \) to denote the function in \( \text{MNames} \to D \) that maps \( l_i \) to \( x_i \) for \( i \leq n \) and all other labels to \( * \). The notation \( \langle \langle l_1 = x_1, \ldots, l_n = x_n \rangle \rangle \langle l \mapsto x \rangle \) will stand for the function in \( \text{MNames} \to D \) that maps \( l \) to \( x \), and agrees with \( \langle \langle l_1 = x_1, \ldots, l_n = x_n \rangle \rangle \) on all the other inputs.

The semantic function for terms \( \langle \cdot \rangle : (\text{SVar} \to D) \to (\text{Obj} \to D) \) is defined in Table 7. Ibidem the symbol \( \in \) is a strict membership test, and we use \( \rho \) to stand for an environment, i.e., a mapping from \( \text{SVar} \) to \( D \). Moreover, conditionals and conjunctions are strict and evaluated left to right. If \( a \) is closed we write \( \langle a \rangle \) instead of \( \langle a \rangle \rho \).

The following standard result will be useful in the remainder of the paper. (Cf. Lemma C.4–6 on page 356 of [AC96].)

**Lemma 4 (Substitution Lemma)** For object terms \( a, b \) and variable \( x \),

\[
\langle a \rangle \rho \langle x \mapsto [b] \rangle = \langle a \{ b / x \} \rangle \rho .
\]

### 6.2 Introducing types into the model

Types are modelled as certain binary relations over \( D \). A *per* is a symmetric, transitive, binary relation on \( D \) that (by convention) does not have \( * \) in its
A binary relation \( P \) is uniform if \( xPy \) implies \( p_i(x)Pp_i(y) \) for all \( i \). It is complete if \( \bot P \bot \) and if whenever \( \langle x_i \rangle \) and \( \langle y_i \rangle \) are chains where \( x_iP y_i \) for all \( i \) then \( \sqcup x_iP \sqcup y_i \). A super is a complete uniform per. The set of all cupers is \( \text{Cuper} \) ranged over by \( R, S, T \).

We use \( p_r(R) \) to denote the restriction of the super \( R \) to those pairs whose rank is no greater than \( r \). \( \text{Cuper} \) can be given the structure of a complete metric space with the metric \( d : \text{Cuper} \times \text{Cuper} \to \mathbb{R}_+ \) defined as

\[
d(R, T) = \max\{0 \cup \{2^{-r} \mid p_r(R) \neq p_r(T)\}\}.
\]

A function \( F : \text{Cuper} \to \text{Cuper} \) is contractive if whenever \( R, S \in \text{Cuper} \),
\( d(F(R), F(S)) \leq 2^{-1}d(R, S) \). Banach’s fixed point theorem guarantees that all contractive endofunctions in \( \text{Cuper} \) have a unique fixed point \( \mu F \).

The following operators over \( \text{Cuper} \) are used to define the semantics of types:

- **Univ** = \( (D \setminus \{\ast\}) \times (D \setminus \{\ast\}) \)
- **Bool** = \{\((\bot, \bot), (\#\#, \#\#), (ff, ff)\)\}
- \( P \to Q = \{(f, g) \in (D \to D) \times (D \to D) \mid \forall x, y . xPy \Rightarrow f(x)Qg(y)\} \)
- \( \sqcup_{i \in I} P_i = \mathcal{C}(\sqcup_{i \in I} P_i) \), where \( \mathcal{C}(P) \) is the least super that contains \( P \)
- \( \llangle l_i : B_i \to T_i \rrangle \) = \{\((\bot, \bot)\)\} \cup \{\((o, o') \in (\text{MNames} \to D) \times (\text{MNames} \to D) \mid \forall i \in I . (o(l_i), o'(l_i)) \in B_i\} \)

The function \( \lambda S.\llangle l_i : S \to T_i \rrangle \) is contractive and therefore has a unique fixed point. We say that \( \lambda S.\llangle l_i : S \to T_i \rrangle \) extends \( \lambda S.\llangle l_i : S \to T_i \rrangle \), written \( \lambda S.\llangle l_i : S \to T_i \rrangle \preceq \lambda S.\llangle l_i : S \to T_i \rrangle \) if \( I \subseteq J \). The set of all functions of the form \( \lambda S.\llangle l_i : S \to T_i \rrangle \) is called \( \text{Gen} \). We have the following operator in \( \text{Cuper} \):

\[
||l_i : B_i \rrangle = \sqcup\{\mu F \mid F \in \text{Gen}, F \preceq \lambda S.\llangle l_i : S \to B_i \rrangle\}.
\]

The semantic function for types

\[
\llbracket \cdot \rrbracket : (\text{TypeVar} \to \text{Cuper}) \to (\text{Type} \to \text{Cuper})
\]

is defined as follows:
\[ \eta = \eta(X) \]
\[ \text{Top} = \text{Univ} \]
\[ \llbracket \{ l_i : B_i \mid i \in I \} \rrbracket \eta = \llbracket \{ l_i : B_i \mid i \in I \} \rrbracket \]
\[ \llbracket \mu(X)A \rrbracket \eta = \mu T. (\text{Univ} \rightarrow \llbracket A \rrbracket \eta \rightarrow T) \]
\[ \llbracket \text{Bool} \rrbracket \eta = \text{Bool} \]

where \( \eta \) denotes a type environment, i.e., a mapping from \textbf{TypeVar} to \textbf{Cuper}. Again we write \( \llbracket A \rrbracket \) instead of \( \llbracket A \rrbracket \eta \) for closed type expressions.

In later developments, we shall need the following result.

**Lemma 5** If \( \{(x, y), (x', y')\} \subseteq \llbracket \{ l_i : T_i \mid i \in I \} \rrbracket \) then \( (x(m_i)x', y(m_i)y') \in T_i \) for all \( i \in I \).

**Proof.** Similar to the proof of Proposition C.4-4 in [AC96]. \( \square \)

### 6.3 Soundness of the type and equational theory

We can now define the semantic counterparts of type and subtyping judgments. In order to do this, we shall need a notion of consistency. We say that \( \Gamma, \eta \) and \( (\rho, \rho') \) are consistent if

- whenever \( X <: A \) is in \( \Gamma \) then \( \eta(X) \subseteq \llbracket A \rrbracket \eta \) and
- whenever \( x:A \) is in \( \Gamma \) then \( (\rho(x), \rho'(x)) \in \llbracket A \rrbracket \eta \).

Now for any consistent \( \Gamma \) and \( \eta \), \( (\rho, \rho') \) and any \( A, B, e, e' \) we define

\[ \Gamma \models \eta, (\rho, \rho') A \quad \text{iff} \quad \llbracket A \rrbracket \eta \in \text{Cuper} \]
\[ \Gamma \models \eta, (\rho, \rho') A <: B \quad \text{iff} \quad \llbracket A \rrbracket \eta \subseteq \llbracket B \rrbracket \eta \]
\[ \Gamma \models \eta, (\rho, \rho') e : A \quad \text{iff} \quad (\llbracket e \rrbracket \rho, \llbracket e' \rrbracket \rho') \in \llbracket A \rrbracket \eta \]
\[ \Gamma \models \eta, (\rho, \rho') e \leftrightarrow e' : A \quad \text{iff} \quad (\llbracket e \rrbracket \rho, \llbracket e' \rrbracket \rho') \in \llbracket A \rrbracket \eta \]

Let \( \text{cons}(\Gamma) = \{ (\eta, (\rho, \rho')) \mid \Gamma \text{ and } (\eta, (\rho, \rho')) \text{ are consistent} \} \). For \( \gamma \in \{ A, A <: B, a:A, a \leftrightarrow b : A \} \) we say that

\[ \Gamma \models \gamma \quad \text{iff} \quad \forall (\eta, (\rho, \rho')) \in \text{cons}(\Gamma) \ . \ \Gamma \models \eta, (\rho, \rho') \gamma. \]

The soundness of the type and equational theory can now be stated as follows.

**Theorem 6** ([AC96]) The relation \( \models \) is preserved by the rules in Tables 1-3. Therefore, for all \( \Gamma \) and \( \gamma \in \{ A, A <: B, a:A, a \leftrightarrow b : A \} \), \( \Gamma \vdash \gamma \) implies \( \Gamma \models \gamma \).
7 Correctness of the denotational model

We shall now investigate the relationship between the equivalence on programs induced by Abadi and Cardelli’s denotational semantics, and observational congruence. More precisely, we prove that the denotational semantics presented in Sect. 6 is correct with respect to observational congruence, i.e., that it identifies only terms that are related by observational congruence. By means of an example, we shall also argue that the denotational semantics is not fully abstract.

The proof of correctness of the denotational semantics will be delivered in three steps. We begin by showing a soundness result for the reduction relation with respect to the denotational semantics.

Proposition 7 For every program $a$ and value $v$, if $a \Downarrow v$ then $\llbracket a \rrbracket = \llbracket v \rrbracket$.

Proof. We begin by proving that if $a \sim b$, then $\llbracket a \rrbracket = \llbracket b \rrbracket$. In light of Lemma 4, it is sufficient to establish the above claim for the basic reduction rules. We confine ourselves to examining two of these rules.

- Assume that $a = a'.l_k \leadsto b_k \{a'/x_k \} = b$, where $a' = [l_i = \varsigma(x_i:A_i)b_i \ i \in I]$ and $k \in I$. First of all, note that

  $\llbracket a' \rrbracket = \llbracket \lambda u.\llbracket b_k \rrbracket_{\rho(x_k \mapsto u)} \rrbracket$.

  Using the definition of the semantic function given in Table 7 and Lemma 4, we can now prove that $\llbracket a \rrbracket = \llbracket b \rrbracket$ thus:

  $\llbracket a \rrbracket = \llbracket a' \rrbracket(l_k)(\llbracket a' \rrbracket) = (\lambda u.\llbracket b_k \rrbracket_{\rho(x_k \mapsto u)})(\llbracket a' \rrbracket) = \llbracket b_k \rrbracket_{\rho(x_k \mapsto a')} = \llbracket b_k \{a'/x_k \} \rrbracket = \llbracket b \rrbracket$.

- Assume that $a = \text{unfold}(\text{fold}(A,v)) \leadsto v = b$. Using the definition of the semantic function given in Table 7, we can now prove that $\llbracket a \rrbracket = \llbracket b \rrbracket$ thus: $\llbracket a \rrbracket = (\lambda u.\llbracket v \rrbracket)(\bot) = \llbracket v \rrbracket = \llbracket b \rrbracket$.

The statement now follows easily by induction on the length of the (unique) sequence of reductions leading from $a$ to $v$. \qed
Of course, one cannot expect the converse of this soundness property to hold because objects are values whether or not the bodies of their methods are fully evaluated. For example, the objects \[ l = \varsigma(x:[l: \text{Bool}]) \text{true} \] and \[ l = \varsigma(x:[l: \text{Bool}]) \text{if}(\text{true}, \text{true}, \text{true}) \] have the same denotation, but are different values. However, if a program has a denotation different from \( \bot \), then it reduces to some value. In particular, at the observed type \( \text{Bool} \) a program evaluates to a value \( v \) if and only if its denotation is \([v]_\text{Bool}\). This property is usually referred to as computational adequacy \[ \text{Mey88} \], and is the essential connection between a denotational and an operationally based semantics.

**Theorem 8 (Computational Adequacy)** Let \( a:A \) be such that \([a]_\text{Bool} \neq \bot \). Then \( a \Downarrow v \) for some value \( v \).

The proof of the above result is based on an adaptation of a strategy due to Plotkin \[ \text{Plo77} \]. We begin by defining a formal approximation relation \( \llangle \) between elements of the domain \( D \) and programs with the following properties:

For any \( d \in D \) and program \( a \), \( d \llangle a \) iff

1. \( d = \bot \), or
2. \( a \Downarrow v \) for some value \( v \) such that \( d \llangle v \), where
   (a) \# \llangle \text{true} \) and \( \text{false} \llangle \text{false} \),
   (b) \([l_i = d_i : i \in I] \llangle [l_i = \varsigma(x_i : [l_i : A_i]) e_i : i \in I] \) iff for every \( d' \) such that \((d', d') \in \[ [l_i : A_i] : i \in I] \) and \( a':[l_i : A_i] : i \in I] \), \( d' \llangle a' \) implies \( d_i(d') \llangle e_i\{a'//x_i\} \) for every \( i \in I \),
   (c) \( \lambda u.d \llangle \text{fold}(\mu(X)A, \upsilon) \) iff \( d \llangle \upsilon \).

The existence of a relation with these properties may be shown following the developments in \[ \text{Pit94} \].

The key to the proof of Theorem 8 is the following technical result.

**Lemma 9** Assume that \( x_1:A_1, \ldots , x_n:A_n \vdash e:A \). Let \( d_1, \ldots , d_n \) and \( a_1, \ldots , a_n \) be such that \( (d_i, d_i) \in \[ [A_i] \] \), \( a_i:A_i \) and \( d_i \llangle a_i \), for every \( i \in \{1, \ldots , n\} \). Then

\[
\[e\]_{(x_1:=d_1, \ldots , x_n:=d_n)} \llangle e \{a_i//x_i\} \}_{i=1}^{n} .
\]

**Proof.** We prove the claim by induction on the depth of the proof of the type assignment \( x_1:A_1, \ldots , x_n:A_n \vdash e:A \). For the sake of conciseness,
throughout the proof we shall use $\rho$ to stand for the environment $\langle x_1 \mapsto d_1, \ldots, x_n \mapsto d_n \rangle$, and $\sigma$ to denote the substitution $\{a_i/x_i\}_{i=1}^n$. The list of type assumptions $x_1:A_1, \ldots, x_n:A_n$ will be referred to as $\Gamma$. We proceed by a case analysis on the last rule used in the proof of the type assignment $\Gamma \vdash e:A$, and only detail the proof for the nontrivial cases. (The reader is referred to Table 2 for the list of the typing rules.)

• (If) Assume that $\Gamma \vdash e:A$ because $e = \text{if}(e_1, e_2, e_3)$ and, by shorter inferences,

$$
\Gamma \vdash e_1:\text{Bool}, \quad \Gamma \vdash e_i:A, \quad i = 1, 2.
$$

The induction hypothesis now yields that, for $i \in\{1, 2, 3\}$,

$$
[e_i]_\rho \prec e_i \sigma . \quad (1)
$$

We proceed with the proof by distinguishing three cases, depending on whether $[e_1]_\rho$ is equal to $\bot$, $\top$ or $\text{ff}$.

- If $[e_1]_\rho = \bot$, then $[e]_\rho = \bot$. By the definition of $\prec$, it follows immediately that $[e]_\rho = \bot \prec e \sigma$, which was to be shown.

- If $[e_1]_\rho = \top$, then $[e]_\rho = [e_2]_\rho$. In case $[e_2]_\rho = \bot$, it follows immediately that $[e]_\rho = \bot \prec e \sigma$.

Assume therefore that $[e_2]_\rho \neq \bot$. In light of (1) and the definition of $\prec$, it follows that $e_1 \sigma \Downarrow \text{true}$, and that $e_2 \sigma \Downarrow v$ for some value $v$ such that $[e_2]_\rho \prec v$. Collecting the above information, we now derive that

$$
eq \text{if}(e_1 \sigma, e_2 \sigma, e_3 \sigma) \leadsto^* \text{if}(\text{true}, e_2 \sigma, e_3 \sigma) \leadsto e_2 \sigma \leadsto^* v .
$$

Thus $e \sigma \Downarrow v$ and $[e]_\rho = [e_2]_\rho \prec v$. By the definition of the relation $\prec$, we may now infer that $[e]_\rho \prec e \sigma$, which was to be shown.

- The case $[e_1]_\rho = \text{ff}$ is similar to the one above.

• (Object) Assume that $\Gamma \vdash e:A$ because $e = [l_i = \varsigma(y_i; A) e_i]_{i \in I}$, $A \equiv [l_i; B_i]_{i \in I}$ and, by shorter inferences,

$$
\Gamma, y_i:A \vdash b_i:B_i \quad (2)
$$
for every $i \in I$. Using the definition of the denotational semantics we find that:

$$\llbracket e \rrbracket_\rho = \llbracket l_i = \lambda v. \llbracket b_i \rrbracket_{\rho(y_i \mapsto v)} \rrbracket$$

As the list $\vec{a}$ only contains closed terms and $y_i$ is different from all the variables in $\{x_1, \ldots, x_n\}$ by (2), we obtain that

$$e_\sigma = [l_i = \varsigma(y_i; A) b_i \sigma \ i \in I] .$$

Applying the inductive hypothesis to (2), we may now infer that, for every $i \in I$, $d' \in D$ such that $(d', d') \in \llbracket A \rrbracket$ and $a':A$ with $d' \smallfrown a'$,

$$\llbracket b_i \rrbracket_{\rho(y_i \mapsto d')} \smallfrown b_i \sigma [y_i \mapsto d'] .$$

By the definition of the relation $\smallfrown$, we finally conclude that

$$\llbracket e \rrbracket_\rho = \llbracket l_i = \varsigma(y_i; A) e_i \sigma \ i \in I \rrbracket \smallfrown [l_i = \varsigma(y_i; A) b_i \sigma]$$

which was to be shown.

- (SELECT) Assume that $\Gamma \vdash e:B_j$ because $e = e'.l_j$ and, by a shorter inference,

  $$\Gamma \vdash e':[l_i: B_i \ i \in I] \quad (j \in I) . \quad (3)$$

If $\llbracket e' \rrbracket_\rho = \bot$, then $\llbracket e' \rrbracket_\rho \smallfrown e_\sigma$ is immediate from the definition of the relation $\smallfrown$. Assume therefore that $\llbracket e' \rrbracket_\rho \neq \bot$. Applying the induction hypothesis to (3), we obtain that $\llbracket e' \rrbracket_\rho \smallfrown e_\sigma$. Note now that, as $\llbracket e' \rrbracket_\rho \neq \bot$, it must be the case that $\llbracket e' \rrbracket_\rho \neq \bot$. Thus, by the definition of the relation $\smallfrown$, there is a value $v$ of type $B = [l_i: B_i \ i \in I]$ such that $e_\sigma \Downarrow v$ and $\llbracket e' \rrbracket_\rho \smallfrown v$. Again using the definition of $\smallfrown$, it must be the case that:

- $\llbracket e' \rrbracket_\rho = \llbracket l_k = d_k \ k \in K \rrbracket$, for some superset $K$ of $I$,

- $v = [l_k = \varsigma(y_k; B) b_k \ k \in K]$, and

- for every $\hat{d} \in D$ and closed term $\hat{a}$ such that $(\hat{d}, \hat{d}) \in \llbracket \hat{B} \rrbracket$, $\hat{a}:B$ and $\hat{d} \smallfrown \hat{a}$, it holds that $d_k(\hat{d}) \smallfrown b_k{\hat{a}/y_k}$ for every $k \in K$. 

19
As \( e' : B \) and \( e' \downarrow v \), it follows that \( v : B \) (Lemma 1(2)). In light of (3), Theorem 6 yields that \((\llbracket e' \rrbracket_\rho, \llbracket e' \rrbracket_\rho) \in \llbracket B \rrbracket \). Using the fact that \( \llbracket e' \rrbracket_\rho \triangleleft e' \sigma \), we may therefore derive that
\[
\llbracket e \rrbracket_\rho = \llbracket e' \rrbracket_\rho(l_j)(\llbracket e' \rrbracket_\rho) \\
= d_j(\llbracket e' \rrbracket_\rho) \\
\triangleleft b_j\{v/y_k\} .
\]

As \( e\sigma = (e'\sigma).l_j \leadsto^* v.l_j \leadsto b_j\{v/y_k\} \) and \( \llbracket e \rrbracket_\rho \triangleleft b_j\{v/y_k\} \), we may finally conclude that \( \llbracket e \rrbracket_\rho \triangleleft e\sigma \), which was to be shown.

• (UPDATE) Assume that \( \Gamma \vdash e : A \) because \( e = e'.l_j \triangleleft\varsigma(x:A)b : A, A \equiv [l_i:B_i \ i \in I], j \in I \) and, by shorter inferences,
\[
\begin{align*}
\Gamma & \vdash e' : A \\
\Gamma, x:A & \vdash b:B_j 
\end{align*}
\]

Using the definition of the denotational semantics, we obtain that
\[
\llbracket e \rrbracket_\rho = \llbracket e' \rrbracket_\rho(l_j \mapsto \lambda u.\llbracket b \rrbracket_\rho(x \mapsto u)) .
\]

As the terms in \( \vec{a} \) are closed, and \( x \) does not occur in \( \vec{x} \), we infer that
\[
e\sigma = (e'\sigma).l_j \triangleleft\varsigma(x:A)b\sigma .
\]

Applying induction to (4), we derive that
\[
\llbracket e' \rrbracket_\rho \triangleleft e'\sigma .
\]

If \( \llbracket e' \rrbracket_\rho = \bot \), then \( \llbracket e \rrbracket_\rho \) is also equal to \( \bot \), and the claim follows immediately by the definition of \( \triangleleft \). Assume therefore that \( \llbracket e' \rrbracket_\rho \neq \bot \). As \( \llbracket e' \rrbracket_\rho \triangleleft e'\sigma \), it must be the case that \( e'\sigma \downarrow v \) for some value \( v \) of type \( A \) such that \( \llbracket e' \rrbracket_\rho \triangleleft v \). As \( A \equiv [l_i:B_i \ i \in I] \) and \( \llbracket e' \rrbracket_\rho \triangleleft v \), we obtain that

1. \( \llbracket e' \rrbracket_\rho = \llbracket l_k = d_k^{k \in K} \rrbracket \), for some superset \( K \) of \( I \),
2. \( v = [l_k = \varsigma(y_k:A)b_k^{k \in K}] \), and
3. for every \( \hat{d} \in D \) and closed term \( \hat{a} \) such that \( (\hat{d}, \hat{d}) \in \llbracket A \rrbracket \), \( \hat{a} : A \) and \( \hat{d} \triangleleft \hat{a} \), it holds that \( d_k(\hat{d}) \triangleleft b_k\{\hat{a}/y_k \} \) for every \( k \in K \).
Applying the inductive hypothesis to (5), we have that, whenever \((d', d') \in \llbracket A \rrbracket\), \(a' : A\) and \(d' \triangleleft a'\),
\[
\llbracket b \rrbracket_{\rho(x \mapsto a')} \triangleleft b_\sigma[x \mapsto a'].
\]
This yields, together with 3 above, that
\[
\llbracket e \rrbracket_{\rho} = \llbracket e' \rrbracket_{\rho} \triangleleft e_\sigma.
\]
Thus we may finally conclude that \([e]_{\rho} \triangleleft e_\sigma\), which was to be shown.

**(FOLD)** Assume that \(\Gamma \vdash e : A\) because \(e = \text{fold}(A, e')\), \(A \equiv \mu(X)E\) and, by a shorter inference,

\[
\Gamma \vdash e' : E\{A/X\}.
\]

(6)

By the definition of the denotational semantics, we derive that
\[
\llbracket e \rrbracket_{\rho} = \lambda u. \llbracket e' \rrbracket_{\rho}.
\]
The inductive hypothesis, applied to (6), yields
\[
\llbracket e' \rrbracket_{\rho} \triangleleft e'_\sigma.
\]

If \([e']_{\rho} = \perp\), then \([e]_{\rho} = \lambda u. \perp\). This is the least element of \(D \rightarrow D\), and, by the definition on \(D\), is identified with \(\perp\). In this case, \([e]_{\rho} \triangleleft e_\sigma\) follows immediately. Assume therefore that \([e']_{\rho} \neq \perp\). As \([e']_{\rho} \triangleleft e'_\sigma\), it must be the case that \(e'_\sigma \downarrow v\) for some \(v\) such that \([e']_{\rho} \triangleleft v\). As \(e'_\sigma \downarrow v\), it follows immediately that \(e_\sigma = \text{fold}(A, e'_\sigma) \downarrow \text{fold}(A, v)\). Moreover, using clause (ii)(c) of the definition of \(\triangleleft\) and (7), we conclude that \([e]_{\rho} \triangleleft \text{fold}(A, v)\), which was to be shown.

**(UNFOLD)** Assume that \(\Gamma \vdash e : A\) because \(e = \text{unfold}(e')\), \(A \equiv E\{\mu(X)E/X\}\) and, by a shorter inference, \(\Gamma \vdash e' : \mu(X)E\). The definition of the denotational semantics yields
\[
\llbracket e \rrbracket_{\rho} = \llbracket e' \rrbracket_{\rho}(\perp).
\]
If $[e]_\rho = \perp$, then the claim is immediate. Assume therefore that $[e]_\rho \neq \perp$. By induction, we may derive that

$$[e']_\rho \triangleleft e' \sigma.$$  

(8)

Note, furthermore, that $[e']_\rho \neq \perp$. Thus (8) yields that $e' \sigma \Downarrow v$ for some value $v$ of type $\mu(X)E$ such that $[e']_\rho \triangleleft v$. The value $v$ must be of the form $\text{fold}(\mu(X)E, v')$. By the definition of $\triangleleft$, if $[e']_\rho \triangleleft \text{fold}(\mu(X)E, v')$ then $[e']_\rho$ is of the form $\lambda u.d'$ for some $d' \triangleleft v'$. It is now easy to see that $e \sigma \Downarrow v'$ and that

$$[e]_\rho = [e']_\rho(\perp) = \lambda u.d'(\perp) = d'.$$

As $d' \triangleleft v'$, it follows that $[e]_\rho \triangleleft e \sigma$, which was to be shown.

- (Subsump) Assume that $\Gamma \vdash e : A$ because, by shorter inferences, $\Gamma \vdash e:B$ and $\Gamma \vdash B <: A$. Then the inductive hypothesis immediately yields that $[e]_\rho \triangleleft e \sigma$.

This completes the proof of this statement. $\square$

Theorem 8 now follows immediately by the above statement and the definition of the formal approximation relation $\triangleleft$.

The following consequence of Proposition 7 and Theorem 8 will be useful in the remainder of this section.

**Corollary 10** Let $a: \text{Bool}$. Then $a \Downarrow v$ iff $[a] = [v]$.

**Proof.** The “only if” implication is just Proposition 7. To establish the “if” implication, assume that $[a] = [v]$ and $a: \text{Bool}$. As $[a] \neq \perp$, Theorem 8 yields that $a \Downarrow v'$ for some value $v'$ of type $\text{Bool}$. By Proposition 7 it follows that $[v] = [v']$. As $v$ and $v'$ are of type $\text{Bool}$, this entails that $v = v'$ and thus that $a \Downarrow v$. $\square$

We are now in a position to prove the main result of this paper, viz. that the denotational semantics is correct with respect to observational congruence.

**Theorem 11** Let $A \in \text{Type}$ and $a,b:A$. Then

$$([a], [b]) \in [A] \text{ implies } a_A^\text{Bool} \simeq b_A.$$
Proof. Assume that $A \in \textbf{Type}$, $a, b : A$ and $(\llbracket a \rrbracket, \llbracket b \rrbracket) \in \llbracket A \rrbracket$. In light of [GR96, Thm. 2], to prove that $a_A \simeq b_A$ it is sufficient to show that $a_A \sim b_A$ holds. Let $X = \{(a_A, b_A) \mid (\llbracket a \rrbracket, \llbracket b \rrbracket) \in \llbracket A \rrbracket\}$. We prove that $X$ is a bisimulation. To this end, assume that $(a_A, b_A) \in X$ and $a_A \xrightarrow{\alpha} a'_A$. By symmetry it is enough to prove that $b_A \xrightarrow{\alpha} b'_A$ for some $b' : A'$ such that $(a'_A, b'_A) \in X$. The proof of this claim proceeds by case analysis of the transition rule used in inferring the transition $a_A \xrightarrow{\alpha} a'_A$.

(Trans Bool) Then $\alpha = v$ where $a \downarrow v \in \{\text{true}, \text{false}\}$, $A \equiv \text{Bool}$, $A' \equiv \text{Top}$ and $a' \equiv a$. Recall that $\llbracket \text{Bool} \rrbracket = \{\bot, \top\}$ and that, for all programs $a : \text{Bool}$, $\llbracket a \rrbracket = \llbracket v \rrbracket$ if $a \downarrow v$ (Corollary 10). As $(\llbracket a \rrbracket, \llbracket b \rrbracket) \in \llbracket \text{Bool} \rrbracket$ this implies that $\llbracket b \rrbracket = \llbracket v \rrbracket$. Again by Corollary 10, it follows that $b \downarrow v$, and therefore that $b \xrightarrow{v} b_{\text{Top}}$. Furthermore $a : \text{Top}, b : \text{Top}$ and $(\llbracket a \rrbracket, \llbracket b \rrbracket) \in \llbracket \text{Top} \rrbracket$, i.e. $(a_{\text{Top}}, b_{\text{Top}}) \in X$.

(Trans Select) In this case $A \equiv \llbracket l_i ; B_i \mid i \in I \rrbracket$, $A' \equiv \llbracket l_j \rrbracket$, $a' \equiv a.l_j$ and $A' \equiv B_j$ for some $j \in I$. As $b : A$ we also have that $b_A \xrightarrow{l_j} b.l_j_{B_j}$. By the type assignment rule (Select), $a.l_j : B_j$ and $b.l_j : B_j$. Furthermore, by the equational rule (Eq Select) and the soundness of the equational theory, $(\llbracket a.l_j \rrbracket, \llbracket b.l_j \rrbracket) \in \llbracket B_j \rrbracket$. This proves that $(a.l_j_{B_j}, b.l_j_{B_j}) \in X$.

(Trans Update) In this case $A \equiv A' \equiv \llbracket l_i : B_i \mid i \in I \rrbracket$, $x : A \vdash e : B_j$, $\alpha = l_j \leftarrow \zeta(x) e$ and $a' \equiv a.l_j \leftarrow \zeta(x : A) e$. Also $b_A \xrightarrow{l_j \leftarrow \zeta(x) e} b'_A$ where $b' \equiv b.l_j \leftarrow \zeta(x : A) e$. By the type assignment rule (Update), $a' : A$ and $b' : A$. By the equational theory $x : A \vdash e : B_j$ implies $x : A \vdash e \leftrightarrow e : B_j$. Therefore, using the equational rule (Eq Override) and the soundness of the equational theory with respect to the model (Thm. 3), we infer that $(\llbracket a' \rrbracket, \llbracket b' \rrbracket) \in \llbracket A \rrbracket$. This proves that $(a'_A, b'_A) \in X$.

(Trans Unfold) Then $A \equiv \mu X B$, $C \equiv B \{\bar{A} / X\}$, $\alpha \equiv \text{unfold}$, $a' \equiv \text{unfold}(a)_C$. By assumption $a : A$ and therefore the type assignment rule (Fold) implies that $\text{unfold}(a) : B \{\bar{A} / X\}$ and $\text{unfold}(b) : B \{\bar{A} / X\}$. Furthermore by the equational rule (Eq Unfold) and soundness of the equational theory, $(\llbracket \text{unfold}(a) \rrbracket, \llbracket \text{unfold}(b) \rrbracket) \in \llbracket C \rrbracket$. This proves that $(\text{unfold}(a)_C, \text{unfold}(b)_C) \in X$.

\qed

23
To see that the denotational model is not fully abstract, consider the following two objects (from [AC96]) of type $B = \left[ l_2 : \text{Bool} \right]$:

$$a = [l_1 = \text{true}, l_2 = \text{true}] \quad b = [l_1 = \text{true}, l_2 = \zeta(x : [l_1 : \text{Bool}, l_2 : \text{Bool}])x.l_1]$$

where we have omitted the $\zeta$-binder in the methods that do not use self.

We shall now argue that $([a], [b]) \not\in [B]$. The denotations of $a$ and $b$ are:

$$[a] = \langle \langle l_1 = \lambda(v)v, l_2 = \lambda(v)v \rangle \rangle$$

and

$$[b] = \langle \langle l_1 = \lambda(v)v, l_2 = \lambda(v)v(l_1)v \rangle \rangle$$

Let $b^* = [l_1 = \text{false}, l_2 = \text{true}]$. As $b^*$ is a program of type $B$, Thm. 6 yields that $([a], [b^*]) \in [B]$. If $([a], [b]) \in [B]$, by Lemma 5 we would then be able to infer that

$$([a](l_2)[b^*], [b](l_2)[b^*]) \in [\text{Bool}]$$.

However, this is obviously not the case, because the denotation of $b^*$ is $\langle \langle l_1 = \lambda(v)v, l_2 = \lambda(v)v \rangle \rangle$ and therefore

$$[a](l_2)[b^*] = \# \text{ and } [b](l_2)[b^*] = \text{ff}$$.

As a corollary of Thm. 11, we obtain an alternative proof of the weak soundness of the equational theory for $\text{Ob}_{1<\mu}$ w.r.t. bisimulation equivalence, a result originally due to Gordon and Rees.

**Proposition 12** If $\emptyset \vdash a \leftrightarrow b : A$, then $a_A \sim b_A$.

**Proof.** Suppose $\emptyset \vdash a \leftrightarrow b : A$. The soundness of the equational theory in the denotational model implies that $([a], [b]) \in [A]$ and Thm. 11 in turn implies that $a_A \simeq b_A$. As $\simeq$ and $\sim$ coincide [GR96], the result now follows. \qed

## 8 Conclusion and directions for further work

In this paper we have shown that the denotational model proposed by Abadi and Cardelli [AC96] is correct, but not fully abstract with respect to the reduction semantics. This is just a first step in the study of the connections between the denotational and operational theories of objects, and much remains to be done.
8.1 Incompleteness of equational theories

It is no surprise that the equational theory is sound but incomplete in the untyped case. As we can express all computable functions within the $\varsigma$-calculus, we can express the complement of the halting problem for any given object $a$ by the equation $a \leftrightarrow \Omega$ where $\Omega$ is the divergent object. The set of such equations is clearly not recursively enumerable. However, the set of provable equalities is a recursively enumerable set, so if the model can adequately capture simple nontermination properties, some equalities will not be provable. However, one would like a systematic approach that will shed more light on the model under consideration.

In a forthcoming paper we shall show the incompleteness of certain equational theories by establishing a result on soundness, namely that Abadi and Cardelli’s equational theory is ‘sound in all models’.

In order to achieve this latter result, we need to make precise the notion of an object model along the lines of the familiar notion of a model for the $\lambda$-calculus [Bar84]. In particular, we shall need an interpretation of types.

8.2 Other models of the $\varsigma$-calculus

As an important by-product, the notion of a model of the $\varsigma$-calculus lets us compare various interpretations already in existence. Ideally, the translation of the untyped $\varsigma$-calculus into the $\pi$-calculus should provide us with another example of a $\varsigma$-model, just as Sangiorgi [San95] has shown that a translation of the $\lambda$-calculus into the $\pi$-calculus gives rise to a $\lambda$-model. Whether this is indeed the case, is a topic for future investigation.

We are also interested in determining whether the translation of $\text{Ob}_{1<\mu}$-types into the modal mu-calculus together with a suitably quotiented term model gives rise to a typed $\varsigma$-model.

Acknowledgement

We thank Martín Abadi for his enlightening suggestions.
References


