EXPRESS 97 (1997) 15 pages

Relating Semantic Models for the Object Calculus

Preliminary Report

Luca Aceto Hans Hüttel Anna Ingólfsdóttir Josva Kleist

Address: Dep. of Computer Science, Aalborg University, Fredrik Bajersvej 7, 9220 Aalborg, Denmark. Telephone: +45 96 35 80 80. Fax: +45 98 15 98 89. Email: {luca,hans,annai,kleist}@cs.auc.dk

Abstract

Abadi and Cardelli have investigated several versions of the ς-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 $\mathbf{Ob}_{1<:\mu}$. 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 $\mathbf{Ob}_{1<:\mu}$-context.

1 Introduction

In [AC96] Abadi and Cardelli present and investigate several versions of the ς-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 λ-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

This preliminary version has been submitted to Express 97.

program manipulation systems. Program equivalences are typically defined according to the following paradigm:

(i) A collection of terms that are considered to be directly executable and observable are designated as programs, and their behaviour is defined;

(ii) 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 $\mathbf{Ob}_{1<:\mu}$ 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 $\mathbf{Ob}_{1<:\mu}$ 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 $\mathbf{Ob}_{1<:\mu}$ that soundly model the equational theory underlying the object calculi. 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* [Sto88] 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 $\mathbf{Ob}_{1<:\mu}$. 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. 7.4). 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. 7.2), and that a program of boolean type evaluates to a boolean value v iff its denotation equals that of v (Corollary 7.3). 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 $\mathbf{Ob}_{1<:\mu}$-context. As a byproduct of our results we obtain an alternative proof of the soundness of the equational theory with respect to bisimulation (Propn. 7.5).

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 $\mathbf{Ob}_{1<:\mu}$. In Section 3 we present the type system for

$\mathbf{Ob}_{1<:\mu}$. Section 4 is devoted to the typed equational theory of $\mathbf{Ob}_{1<:\mu}$. 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 $\mathbf{Ob}_{1<:\mu}$ 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 ς-calculus and its reduction semantics

There are various versions of the ς-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, **Obj**, is defined by the the following abstract syntax:

$$
\begin{array}{lll}
a ::= & [l_i = \varsigma(x_i{:}A_i)b_i \;^{i\in I}] & \text{objects} \\
 & \mid x & \text{self variables} \\
 & \mid a.l & \text{method activation} \\
 & \mid a.l \Leftarrow \varsigma(x{:}A)b & \text{method override} \\
 & \mid \mathsf{fold}(A,a) \mid \mathsf{unfold}(a) & \text{recursive fold/unfold} \\
 & \mid \mathsf{if}(a, b_1, b_2) & \\
 & \mid \mathsf{true} \mid \mathsf{false} & \text{booleans}
\end{array}
$$

Here $x_i \in \mathbf{SVar}$ ranges over self variables, $l_i \in \mathbf{MNames}$ ranges over method names and $A_i \in \mathbf{Type}$. A value, denoted by v, is either an object ($[l_i = \varsigma(x_i{:}B_i)b_i \;^{i\in I}]$), a boolean value ($\mathsf{true}, \mathsf{false}$) or a folded value ($\mathsf{fold}(A, v)$).

The presentation of the ς-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

$$
\begin{array}{ll}
a.l_k \;\rightsquigarrow\; b_k\{a/x_k\} & (k \in I) \\[1ex]
a.l_k \Leftarrow \varsigma(x{:}A)b \;\rightsquigarrow\; [l_k = \varsigma(x{:}A)b,\; l_i = \varsigma(x_i{:}A_i)b_i \;^{i\in I\setminus\{k\}}] & (k \in I) \\[1ex]
\mathsf{if}(\mathsf{true}, b_1, b_2) \;\rightsquigarrow\; b_1 \qquad \mathsf{if}(\mathsf{false}, b_1, b_2) \;\rightsquigarrow\; b_2 & \\[1ex]
\mathsf{unfold}(\mathsf{fold}(A, v)) \;\rightsquigarrow\; v &
\end{array}
$$

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):

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

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.

3 Types

One of the main motivation for the ς-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 $\mathbf{Ob}_{1<:\mu}$ 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 $\mathbf{Ob}_{1<:\mu}$ type expressions is defined via the following abstract syntax:

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

Here 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. 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. 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

$\mathbf{Ob}_{1<:\mu}$ 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 Γ, where Γ 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 Γ is empty we shall sometimes just write $a{:}A$ instead of $\emptyset \vdash a{:}A$. Whenever the typing assumptions in Γ 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 Γ).

An object a has type A under the set of assumptions Γ 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 $\mathbf{Ob}_{1<:\mu}$ 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 Γ. Here the typing assumptions in Γ 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.

(SUB REFL) $\dfrac{\Gamma \vdash A}{\Gamma \vdash A <: A}$ (SUB TRANS) $\dfrac{\Gamma \vdash A_1 <: A_2 \quad \Gamma \vdash A_2 <: A_3}{\Gamma \vdash A_1 <: A_3}$

(SUB X) $\dfrac{\Gamma(X) = A}{\Gamma \vdash X <: A}$ (SUB TOP) $\dfrac{\Gamma \vdash A}{\Gamma \vdash A <: \mathsf{Top}}$

(SUB OBJ) $\dfrac{K \subseteq L \quad \Gamma \vdash A_i \quad \forall i \in L}{\Gamma \vdash [l_i{:}A_i\ ^{i \in I}] <: [l_j{:}A_j\ ^{j \in K}]}$

(SUB REC) $\dfrac{\Gamma \vdash \mu(X_1)A_1 \quad \Gamma \vdash \mu(X_2)A_2 \quad \Gamma[X_2 <: \mathsf{Top}, X_1 <: X_2] \vdash A_1 <: A_2}{\Gamma \vdash \mu(X_1)A_1 <: \mu(X_2)A_2}$

Table 1
The subtyping relation

(VAR) $\dfrac{\Gamma(x) = A}{\Gamma \vdash x{:}A}$ (SELECT) $\dfrac{\Gamma \vdash a{:}[l_i{:}B_i\ ^{i \in I}] \qquad j \in I}{\Gamma \vdash a.l_j{:}B_j}$

(OBJECT) $\dfrac{\Gamma[x_i{:}A] \vdash b_i{:}B_i \quad \forall i \in I \qquad A \equiv [l_i{:}B_i\ ^{i \in I}]}{\Gamma \vdash [l_i = \varsigma(x_i{:}A)b_i\ ^{i \in I}] : A}$

(UPDATE) $\dfrac{\Gamma \vdash a{:}A \quad \Gamma[x{:}A] \vdash b{:}B_j \quad j \in I \qquad A \equiv [l_i{:}B_i\ ^{i \in I}]}{\Gamma \vdash a.l_j \Leftarrow \varsigma(x{:}A)b : A}$

(FOLD) $\dfrac{\Gamma \vdash a{:}B\{A/X\} \quad A \equiv \mu(X)B}{\Gamma \vdash \mathsf{fold}(A, a) : A}$ (UNFOLD) $\dfrac{\Gamma \vdash a{:}A \quad A \equiv \mu(X)B}{\Gamma \vdash \mathsf{unfold}(a) : B\{A/X\}}$

(IF) $\dfrac{\Gamma \vdash a{:}\mathsf{Bool} \quad \Gamma \vdash a_1, a_2 : A}{\Gamma \vdash \mathsf{if}(a, a_1, a_2) : A}$ (BOOL) $\dfrac{b \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma \vdash b{:}\mathsf{Bool}}$

(SUBSUMP) $\dfrac{\Gamma \vdash a{:}A_1 \quad \Gamma \vdash A_1 <: A_2}{\Gamma \vdash a : A_2}$

Table 2
Type assignment

Example 3.1 As noted by Abadi and Cardelli, for any type A, there is a divergent object Ω_A definable as $[l = \varsigma(x{:}[l{:}A])x.l].l$.

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 $\mathbf{Ob}_{1<:\mu}$.

All judgments are of the form $\Gamma \vdash a \leftrightarrow b : A$, where Γ 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 Γ 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

(Eq Symm)	(Eq Trans)	(Eq x)
$\dfrac{\Gamma \vdash a \leftrightarrow b : A}{\Gamma \vdash b \leftrightarrow a : A}$	$\dfrac{\Gamma \vdash a \leftrightarrow b : A, \quad b \leftrightarrow c : A}{\Gamma \vdash a \leftrightarrow c : A}$	$\dfrac{\Gamma(x) = A}{\Gamma \vdash x \leftrightarrow x : A}$

Table 3
Equivalence-inducing equational rules

corresponding to the clauses of the reduction semantics. Finally, we have in Table 5 the rules for subtyping.

The most interesting rule is (Eq Sub Object), defined in Table 5 which allows one to prove equalities between objects with different collections of methods.

5 A labelled transition semantics

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

$$Rel = \{(a_A, b_A) \mid a{:}A \text{ and } b{:}A\}.$$

The observable actions, $\alpha \in Act$, take the following forms:

$$\alpha ::= \mathsf{true} \mid \mathsf{false} \mid l \mid l \Leftarrow \varsigma(x)b \mid \mathsf{unfold}.$$

The family $\{\xrightarrow{\alpha} \mid \alpha \in Act\}$ 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 [Par81,Mil89].

Definition 5.1 (Bisimulation) Bisimilarity $\sim$ is the greatest subset of *Rel* that satisfies the following: $a_A \sim b_A$ if and only if

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

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

(EQ OBJECT) where $A \equiv [l_i{:}B_i \ ^{i \in I}]$

$$\frac{\Gamma, x_i{:}A \vdash b_i \leftrightarrow b_i' : B_i \ \forall i \in I}{\Gamma \vdash [l_i = \varsigma(x_i{:}A)b_i \ ^{i \in I}] \leftrightarrow [l_i = \varsigma(x_i{:}A)b_i' \ ^{i \in I}] : A}$$

(EQ SELECT)

$$\frac{\Gamma \vdash a \leftrightarrow b : [l_i{:}B_i \ ^{i \in I}] \quad j \in I}{\Gamma \vdash a.l_j \leftrightarrow b.l_j : B_j}$$

(EQ OVERRIDE) where $A \equiv [l_i{:}B_i \ ^{i \in I}]$

$$\frac{\Gamma \vdash a \leftrightarrow a' \quad \Gamma, x{:}A \vdash b \leftrightarrow b' : B_j \quad j \in I}{\Gamma \vdash a.l_j \Leftarrow \varsigma(x{:}A)b \leftrightarrow a'.l_j \Leftarrow \varsigma(x{:}A)b' : A}$$

(EQ IF)

$$\frac{\Gamma \vdash b \leftrightarrow b' : \mathsf{Bool} \quad \Gamma \vdash b_1 \leftrightarrow b_1' : B, \ b_2 \leftrightarrow b_2' : B}{\Gamma \vdash \mathsf{if}(b, b_1, b_2) \leftrightarrow \mathsf{if}(b', b_1', b_2') : B}$$

(EQ FOLD) where $A \equiv \mu(X)B$

$$\frac{\Gamma \vdash a \leftrightarrow b : B\{A/X\}}{\Gamma \vdash \mathsf{fold}(A, a) \leftrightarrow \mathsf{fold}(A, b) : A}$$

(EQ UNFOLD) where $A \equiv \mu(X)B$

$$\frac{\Gamma \vdash a \leftrightarrow b : A}{\Gamma \vdash \mathsf{unfold}(a) \leftrightarrow \mathsf{unfold}(b) : B\{A/X\}}$$

(EVAL SELECT)

$$\frac{\Gamma \vdash a{:}A \quad j \in I}{\Gamma \vdash a.l_j \leftrightarrow b_j\{a/x_j\} : B_j} \quad \text{where} \quad \begin{array}{l} A \equiv [l_i{:}B_i \ ^{i \in I}] \\ a \equiv [l_i = \varsigma(x_i{:}A)b_i \ ^{i \in I \cup J}] \end{array}$$

(EVAL OVERRIDE) where $A \equiv [l_i{:}B_i \ ^{i \in I}]$

$a \equiv [l_i = \varsigma(x_i{:}A)b_i \ ^{i \in I \cup J}]$

$$\frac{\Gamma \vdash a{:}A \quad \Gamma, x{:}A \vdash b{:}B_j \quad j \in I}{\Gamma \vdash a.l_j \Leftarrow \varsigma(x{:}A)b \leftrightarrow [l_i = \varsigma(x_i{:}A')b_i, \ l_j\varsigma(x{:}A')b_j \ ^{i \in I \setminus \{j\}}] : A}$$

(EVAL FOLD) where $A \equiv \mu(X)B$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{fold}(A, \mathsf{unfold}(a)) \leftrightarrow a : A}$$

(EVAL UNFOLD) where $A \equiv \mu(X)B$

$$\frac{\Gamma \vdash a : B\{A/X\}}{\Gamma \vdash \mathsf{unfold}(\mathsf{fold}(A, a)) \leftrightarrow a : B\{A/X\}}$$

(EVAL IF$_1$)

$$\frac{\Gamma \vdash b_1, b_2 : B}{\Gamma \vdash \mathsf{if}(\mathsf{true}, b_1, b_2) \leftrightarrow b_1 : B}$$

(EVAL IF$_2$)

$$\frac{\Gamma \vdash b_1, b_2 : B}{\Gamma \vdash \mathsf{if}(\mathsf{false}, b_1, b_2) \leftrightarrow b_2 : B}$$

Table 4
Equational rules specific to the calculus

have the same termination behaviour in all contexts of type 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.

$$\text{(EQ SUBSUMP)}\quad \frac{\Gamma \vdash a \leftrightarrow b : A \quad \Gamma \vdash A <: B}{\Gamma \vdash a \leftrightarrow b : B} \qquad \text{(EQ TOP)}\quad \frac{\Gamma \vdash a{:}A,\ b{:}B}{\Gamma \vdash a \leftrightarrow b : \mathsf{Top}}$$

$$\text{(EQ SUB OBJECT) where } A' \equiv [l_i{:}B_i\ ^{i \in I}] \quad A'' \equiv [l_i{:}B_i\ ^{i \in J}] \quad I \subseteq J$$

$$\frac{\Gamma, x_i{:}A' \vdash b_i{:}B_i\ \forall i \in I \quad \Gamma, x_j{:}A'' \vdash b_j{:}B_j\ \forall j \in J \setminus I}{\Gamma \vdash [l_i = \varsigma(x_i{:}A')b_i\ ^{i \in I}] \leftrightarrow [l_i = \varsigma(x_i{:}A'')b_i\ ^{i \in J}] : A'}$$

Table 5
Equational rules for subtyping

$$\text{(TRANS BOOL)}\quad \frac{a{\Downarrow}v \in \{\mathsf{true}, \mathsf{false}\}}{a_{\mathsf{Bool}} \xrightarrow{v} a_{\mathsf{Top}}}$$

$$\text{(TRANS SELECT)}\quad \frac{j \in I \quad [l_i{:}B_i\ ^{i \in I}]}{a_A \xrightarrow{l_j} a.l_j\ {}_{B_j}}$$

$$\text{(TRANS UPDATE)}\quad \frac{x{:}A \vdash b{:}B_j \quad j \in I \quad A \equiv [l_i{:}B_i]_{i \in I}}{a_A \xrightarrow{l_j \Leftarrow \varsigma(x)b} a.l_j \Leftarrow \varsigma(x{:}A)b_A}$$

$$\text{(TRANS UNFOLD)}\quad \frac{A \equiv \mu(X)B \quad C \equiv B\{A/X\}}{a_A \xrightarrow{\mathsf{unfold}} \mathsf{unfold}(a)_C}$$

Table 6
The rules of the labelled transition semantics

Definition 5.2 (Observational congruence) We write $a_B \overset{A}{\simeq} b_B$ 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 $\mathsf{true} \overset{\mathsf{Top}}{\not\simeq} \Omega_{\mathsf{Top}}$, which violates the rule (EQ TOP) expressing that all objects are to be considered equal at type Top. Amongst the relations $\overset{A}{\simeq}$, congruence at type Top, viz. $\overset{\mathsf{Top}}{\simeq}$, is the most discriminating and $\overset{\mathsf{Bool}}{\simeq}$ the least. Rule (EQ TOP) holds for $\overset{\mathsf{Bool}}{\simeq}$ and, for that reason and by analogy with [Plo77], Gordon and Rees choose $\overset{\mathsf{Bool}}{\simeq}$ as the appropriate notion of observational congruence for $\mathbf{Ob}_{1<:\mu}$.

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 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 $(\llbracket a \rrbracket, \llbracket b \rrbracket) \in \llbracket A \rrbracket$, where $\llbracket a \rrbracket, \llbracket b \rrbracket$ and $\llbracket A \rrbracket$ are the corresponding interpretations.

6.1 The untyped model

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

$$D = \{*\}_\perp + \{\mathit{tt}, \mathit{ff}\}_\perp + (D \to D) + (L \to D)_\perp$$

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

$$\begin{aligned} D_0 &= \{\perp\} \\ D_{n+1} &= \{*\}_\perp + \{\mathit{tt}, \mathit{ff}\}_\perp + (D_n \to D_n) + (L_n \to D_n)_\perp \\ &\quad \text{where } L_n = \{l_1, \ldots, l_n\} \end{aligned}$$

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.

We use $\langle\!\langle l_1 = x_1, \ldots, l_n = x_n \rangle\!\rangle$ to denote the function in $L \to D$ that maps l_i to x_i for $i \leq n$ and all other labels to $*$.

The semantic function for terms $\llbracket \cdot \rrbracket : (\mathbf{SVar} \to D) \to (\mathbf{Obj} \to D)$ is defined in Table 7. Ibidem the symbol $\in$ is a strict membership test. Moreover, conditionals and conjunctions are strict and evaluated left to right. If a is closed we write $\llbracket a \rrbracket$ instead of $\llbracket a \rrbracket_\rho$.

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 domain. A binary relation P is *uniform* if xPy implies $p_i(x)Pp_i(y)$ for all i. It is *complete* if $\perp P \perp$ and if whenever $\langle x_i \rangle$ and $\langle y_i \rangle$ are chains where $x_i P y_i$ for all i then $\sqcup x_i P \sqcup y_i$. A *cuper* is a complete uniform per. The set of all cupers is **Cuper** ranged over by R, S, T.

Cuper is a complete metric space with the metric $d : \mathbf{Cuper} \times \mathbf{Cuper} \to \mathbf{R}_+$ defined as

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

A function $F : \mathbf{Cuper} \to \mathbf{Cuper}$ is *contractive* if whenever $R, S \in \mathbf{Cuper}$, $d(F(R), F(S)) \leq 2^{-1} d(R, S)$. Banach's fixed point theorem guarantees that

$$[\![x]\!]_\rho = \rho(x)$$

$$[\![[l_i = \varsigma(x_i{:}A_i)b_i\ ^{i\in I}]]\!]_\rho = \langle\!\langle l_i = \lambda v.[\![b_i]\!]_{\rho\langle x\mapsto v\rangle}\rangle\!\rangle$$

$$[\![a.l]\!]_\rho = \begin{cases} [\![a]\!]_\rho(l)([\![a]\!]_\rho) & \text{if } [\![a]\!]_\rho \in L \to D \text{ and } [\![a]\!]_\rho(l) \in D \to D \\ * & \text{otherwise} \end{cases}$$

$$[\![a.l{\Leftarrow}\varsigma(x{:}A)b]\!]_\rho = \begin{cases} [\![a]\!]_\rho\langle m \mapsto \lambda v.[\![b]\!]_\rho\langle x \mapsto v\rangle\rangle & \text{if } [\![a]\!]_\rho \in L \to D \\ * & \text{otherwise} \end{cases}$$

$$[\![\mathsf{fold}(A,a)]\!]_\rho = \lambda v.[\![A]\!]_\rho$$

$$[\![\mathsf{unfold}(a)]\!]_\rho = [\![a]\!]_\rho(\bot)$$

$$[\![\mathsf{true}]\!]_\rho = tt,\ [\![\mathsf{false}]\!]_\rho = ff$$

$$[\![\mathsf{if}(a,b_1,b_2)]\!]_\rho = \begin{cases} [\![b_1]\!]_\rho & \text{if } [\![a]\!]_\rho = tt, \\ [\![b_2]\!]_\rho & \text{if } [\![a]\!]_\rho = ff \\ \bot & \text{if } [\![a]\!]_\rho = \bot \\ * & \text{otherwise} \end{cases}$$

Table 7
The semantic function for terms

all contractive endofunctions in **Cuper** have a unique fixed point μF.

The following operators over **Cuper** are used to define the semantics of types:

- $\mathbf{Univ} = (D \setminus \{*\}) \times (D \setminus \{*\})$
- $\mathbf{Bool} = \{(\bot,\bot),(tt,tt),(ff,ff)\}$
- $P \to Q = \{(f,g) \in (D \to D) \times (D \to D) \mid \forall x,y.xPy \Rightarrow f(x)Qg(y)\}$
- $\sqcup_{i\in I}P_i = \mathcal{C}(\cup_{i\in I}P_i)$, where $\mathcal{C}(P)$ is the least cuper that contains P
- $\langle\!\langle l_i{:}B_i\ ^{i\in I}\rangle\!\rangle = \{(\bot,\bot)\}\cup\{(o,o') \in (L \to D)\times(L \to D) \mid \forall i \in I.(o(m_i),o'(m_i)) \in B_i\}$

The function $\lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in I}\rangle\!\rangle$ is contractive and therefore has a unique fixed point. We say that $\lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in J}\rangle\!\rangle$ extends $\lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in I}\rangle\!\rangle$, written $\lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in J}\rangle\!\rangle \preceq \lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in I}\rangle\!\rangle$ if $I \subseteq J$. The set of all functions of the form $\lambda S.\langle\!\langle l_i{:}S \to T_i\ ^{i\in I}\rangle\!\rangle$ is called **Gen**. We have the following operator in **Cuper**:

$$\|m_i{:}B_i\ ^{i\in I}\| = \sqcup\{\mu F | F \in \mathbf{Gen}, F \preceq \lambda S.\langle\!\langle l_i{:}S \to B_i\ ^{i\in I}\rangle\!\rangle\}\ .$$

The semantic function for types

$$[\![\cdot]\!] : (\mathbf{TypeVar} \to \mathbf{Cuper}) \to (\mathbf{Type} \to \mathbf{Cuper})$$

is defined as follows:

$$
\begin{aligned}
\llbracket X \rrbracket_\eta &= \eta(X) \\
\llbracket \mathsf{Top} \rrbracket_\eta &= \mathbf{Univ} \\
\llbracket [l_i{:}B_i\ ^{i\in I}] \rrbracket_\eta &= \| m_i{:}B_i\ ^{i\in I} \| \\
\llbracket \mu(X)A \rrbracket_\eta &= \mu T.(\mathbf{Univ} \to \llbracket A \rrbracket_{\eta\langle X \mapsto T\rangle}) \\
\llbracket \mathsf{Bool} \rrbracket_\eta &= \mathbf{Bool}
\end{aligned}
$$

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 6.1 *If* $\{(x, y), (x', y')\} \subseteq \| m_i{:}T_i\ ^{i\in I} \|$ *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 Γ, η and (ρ, ρ') are *consistent* if

- whenever $X <: A$ is in Γ then $\eta(X) \subseteq \llbracket A \rrbracket_\eta$ and
- whenever $x{:}A$ is in Γ then $(\rho(x), \rho'(x)) \in \llbracket A \rrbracket_\eta$.

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

$$
\begin{aligned}
&\Gamma \models_{\eta,(\rho,\rho')} A && \text{iff } \llbracket A \rrbracket\eta \in \mathbf{Cuper} \\
&\Gamma \models_{\eta,(\rho,\rho')} A <: B && \text{iff } \llbracket A \rrbracket\eta \subseteq \llbracket B \rrbracket\eta \\
&\Gamma \models_{\eta,(\rho,\rho')} e{:}A && \text{iff } (\llbracket e \rrbracket\rho, \llbracket e \rrbracket\rho') \in \llbracket A \rrbracket\eta \\
&\Gamma \models_{\eta,(\rho,\rho')} e \leftrightarrow e' : A && \text{iff } (\llbracket e \rrbracket\rho, \llbracket e' \rrbracket\rho') \in \llbracket A \rrbracket\eta
\end{aligned}
$$

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

$$\Gamma \models \gamma \text{ iff } \forall(\eta, (\rho, \rho')) \in \mathbf{cons}(\Gamma).\Gamma \models_{\eta,(\rho,\rho')} \gamma.$$

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

Theorem 6.2 ([AC96]) *The relation* $\models$ *is preserved by the rules in Tables 1–5. Therefore, for all* Γ *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 7.1 *For every program a and value v, if $a \Downarrow v$ then $[\![a]\!] = [\![v]\!]$.*

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 $[l = \varsigma(x{:}[l{:}\mathsf{Bool}])\mathsf{true}]$ and $[l = \varsigma(x{:}[l{:}\mathsf{Bool}])\mathsf{if}(\mathsf{true}, \mathsf{true}, \mathsf{true})]$ have the same denotation, but are different values. However, if a program has a denotation different from $\perp$, then it reduces to some value. In particular, at the observed type Bool a program evaluates to a value v if and only if its denotation is $[\![v]\!]$. This property is usually referred to as *computational adequacy* [Mey88], and is the essential connection between a denotational and an operationally based semantics.

Theorem 7.2 (Computational Adequacy) *Let $a{:}A$ be such that $[\![a]\!] \neq \perp$. Then $a \Downarrow v$ for some value v.*

Proof. The proof is based on an adaptation of a strategy due to Plotkin [Plo77]. We begin by defining a formal approximation relation $\lhd$ between elements of the domain D and programs with the following properties:

For any $d \in D$ and program a, $d \lhd a$ iff
(i) $d = \perp$, or
(ii) $a \Downarrow v$ for some value v such that $d \lhd v$, where
(a) $\mathit{tt} \lhd \mathsf{true}$ and $\mathit{ff} \lhd \mathsf{false}$,
(b) $\langle\!\langle l_i = d_i \ ^{i \in I} \rangle\!\rangle \lhd [l_i = \varsigma(x_i{:}[l_i{:}A_i \ ^{i \in I}])e_i \ ^{i \in I}]$ iff for every d' such that $(d', d') \in [\![[l_i{:}A_i \ ^{i \in I}]]\!]$, and $a'{:}[l_i{:}A_i \ ^{i \in I}]$, $d' \lhd a'$ implies $d_i(d') \lhd e_i\{^{a'}/_{x_i}\}$ for every $i \in I$,
(c) $\lambda u.d \lhd \mathsf{fold}(\mu(X)A, v)$ iff $d \lhd v$.

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

To complete the proof of the theorem, we then prove the following technical statement by induction on the depth of the proof of the type assignment $x_1{:}A_1, \ldots, x_n{:}A_n \vdash e{:}A$:

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 $d_i \lhd a_i$, for every $i \in \{1, \ldots, n\}$. Then

$$[\![e]\!]\langle x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\rangle \lhd e\{^{a_i}/_{x_i}\}_{i=1}^{n} \ .$$

The claim now follows immediately by the above statement and the definition of the formal approximation relation $\lhd$. □

Corollary 7.3 *Let $a{:}\mathsf{Bool}$. Then $a \Downarrow v$ iff $[\![a]\!] = [\![v]\!]$.*

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

$$([\![a]\!], [\![b]\!]) \in [\![A]\!] \textit{ implies } a_A \overset{\mathsf{Bool}}{\simeq} b_A \ .$$

Proof. Assume that $A \in$ **Type**, $a, b{:}A$ and $([\![a]\!], [\![b]\!]) \in [\![A]\!])$. In light of [GR96, Thm. 2], to prove that $a_A \overset{\mathsf{Bool}}{\simeq} b_A$ it is sufficient to show that $a_A \sim b_A$ holds. Let $X = \{(a_A, b_A) \mid ([\![a]\!], [\![b]\!]) \in [\![A]\!]\}$. 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 $a_A \xrightarrow{\alpha} a'_{A'}$. Below, we limit ourselves to considering two of the possible cases.

(Trans Bool) Then $\alpha = v$ where $a \Downarrow v \in \{\mathsf{true}, \mathsf{false}\}$, $A \equiv \mathsf{Bool}$, $A' \equiv \mathsf{Top}$ and $a' \equiv a$. Recall that $[\![\mathsf{Bool}]\!] = \{(\bot, \bot), (\mathit{tt}, \mathit{tt}), (\mathit{ff}, \mathit{ff})\}$ and that, for all programs $a{:}\mathsf{Bool}$, $[\![a]\!] = [\![v]\!]$ iff $a \Downarrow v$ (Corollary 7.3). As $([\![a]\!], [\![b]\!]) \in [\![\mathsf{Bool}]\!]$ this implies that $[\![b]\!] = [\![v]\!]$. Again by Corollary 7.3, it follows that $b \Downarrow v$, and therefore that $b_{\mathsf{Bool}} \xrightarrow{v} b_{\mathsf{Top}}$. Furthermore $a{:}\mathsf{Top}$, $b{:}\mathsf{Top}$ and $([\![a]\!], [\![b]\!]) \in [\![\mathsf{Top}]\!]$, *i.e.* $(a_{\mathsf{Top}}, b_{\mathsf{Top}}) \in X$.

(Trans Update) In this case $A \equiv A' \equiv [l_i{:}B_i{}^{\,i \in I}]$, $x{:}A \vdash e{:}B_j$, $\alpha = l_j \Leftarrow \varsigma(x)e$ and $a' \equiv a.l_j \Leftarrow \varsigma(x{:}A)e$. Also $b_A \xrightarrow{l_j \Leftarrow \varsigma(x)e} b'_A$ where $b' \equiv b.l_j \Leftarrow \varsigma(x{:}A)e$. 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. 6.2), we infer that $([\![a']\!], [\![b']\!]) \in [\![A]\!]$. This proves that $(a'_A, b'_A) \in X$.

□

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

$$a = [l_1 = \mathsf{true},\ l_2 = \mathsf{true}] \qquad b = [l_1 = \mathsf{true},\ l_2 = \varsigma(x{:}[l_1{:}\mathsf{Bool},\ l_2{:}\mathsf{Bool}])x.l_1]$$

where we have omitted the ς-binder in the methods that do not use self.

We shall now argue that $([\![a]\!], [\![b]\!]) \notin [\![B]\!]$. The denotations of a and b are:

$$[\![a]\!] = \langle\!\langle l_1 = \lambda(v)\mathit{tt},\ l_2 = \lambda(v)\mathit{tt} \rangle\!\rangle$$

and

$$[\![b]\!] = \langle\!\langle l_1 = \lambda(v)\mathit{tt},\ l_2 = \lambda(v)v(l_1)v \rangle\!\rangle$$

Let $b^* = [l_1 = \mathsf{false},\ l_2 = \mathsf{true}]$. As b^* is a program of type B, Thm. 6.2 yields that $([\![b^*]\!], [\![b^*]\!]) \in [\![B]\!]$. If $([\![a]\!], [\![b]\!]) \in [\![B]\!]$, by Lemma 6.1 we would then be able to infer that

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

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

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

As a corollary of Thm. 7.4, we obtain an alternative proof of the following result, due to Gordon and Rees:

Proposition 7.5 *If* $\emptyset \vdash a \leftrightarrow b : A$, *then* $a_A \sim b_A$.

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 ς-calculus, we can express the complement of the halting problem for any given object a by the equation $a \leftrightarrow \Omega$ where Ω 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 λ-calculus [Bar84]. In particular, we shall need an interpretation of types.

8.2 Other models of the ς-calculus

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

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

Acknowledgement

We thank Martín Abadi for his enlightening suggestions.

References

[AC96] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[Bar84] H.P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.

[GR96] Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Programming Languages*, 1996.

[Mey88] A.R. Meyer. Semantical paradigms: Notes for an invited lecture. In *Proceedings* 3^{th} *Annual Symposium on Logic in Computer Science,* Edinburgh, pages 236–242. IEEE Computer Society Press, 1988.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings of 5th GI Conference LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[Pit94] A. M. Pitts. Computational adequacy via 'mixed' inductive definitions. In *Mathematical Foundations of Programming Semantics, Proc. 9th Int. Conf., New Orleans, LA, USA, April 1993*, volume 802 of *Lecture Notes in Computer Science*, pages 72–82. Springer-Verlag, Berlin, 1994.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, December 1977.

[San95] D. Sangiorgi. Lazy functions and mobile processes. Rapports de Recherche RR-2515, INRIA Sophia-Antipolis, 1995.

[Sto88] A. Stoughton. *Fully abstract models of programming languages.* Research Notes in Theoretical Computer Science. Pitman, London, 1988.