A Note on Logical Relations Between Semantics and Syntax

ANDREW M. PITTS, *Cambridge University Computer Laboratory, Cambridge CB2 3QG, United Kingdom.*
E-mail: Andrew.Pitts@cl.cam.ac.uk

Abstract

This note gives a new proof of the 'operational extensionality' property of Abramsky's lazy lambda calculus—namely the coincidence of contextual equivalence with a co-inductively defined notion of 'applicative bisimilarity'. This purely syntactic result is here proved using a logical relation (due to Plotkin) between the syntax and its denotational semantics. The proof exploits a mixed inductive/co-inductive characterisation of the logical relation recently discovered by the author.[1]

Keywords: denotational semantics, logical relation, lambda calculus, contextual equivalence

1 Introduction

The recent categorical analysis of recursively defined domains by Freyd [7] has led to simplified techniques for denotational semantics involving such domains (see [15] and [6], for example). On the other hand, work on co-inductively defined 'applicative bisimilarities' has provided quite powerful theories of equivalence of functional programs, defined in terms of their operational semantics (see [1], [10], [5], and [8], for example). This note ties together these two developments via a 'formal approximation' relation, $d \lhd P$, between the elements, d, of some domain of denotations of programs and the programs, P, themselves. This instance of a so-called 'logical' relation was introduced by Plotkin [19] to prove computational adequacy results. A new proof of existence of this kind of relation was given recently by the author [15]. It was shown that such relations are characterised by a certain 'universal' property of mixed inductive/co-inductive character. Here we show that this universal property can be used to give a simple proof of the congruence property of applicative bisimilarity and hence of the coincidence of applicative bisimilarity with contextual equivalence. The proof hinges upon the observation that a program P is applicatively similar to a program Q if and only if the denotation of P formally approximates Q (see Theorem 5.3 below). This observation is not new (and is quite easy to prove) for the simply typed programming language PCF [18], where it has been used in unpublished proofs of the Context Lemma of Milner [13].[2] However, the fact that it holds for recursively typed and untyped languages is apparently new (and is not so easy to prove). We show that this characterisation of applicative bisimilarity in terms of the formal approximation relation, apart from yielding context lemmas, also gives a very quick proof of

[1] Full version of an invited paper presented at the *3rd Workshop on Logic, Language, Information and Computation* (*WoLLIC'96*, May 8–10, 1996, Salvador (Bahia), Brazil, organised by UFPE and UFBA, and sponsored by IGPL, FoLLI, ASL.

[2] By Achim Jung and Allen Stoughton at least.

the completeness and continuity properties of applicative similarity with respect to chains of finite approximations of fixpoint combinators.

2 Operational extensionality

Recall that two phrases of a programming language are *contextually equivalent* if occurrences of them in any complete program can be interchanged without affecting the observable results of executing the program. This is a reasonable notion of what it means for two phrases to be semantically 'equal'.[3] In particular, the form of the definition, with its quantification over all possible contexts (uses of a phrase in a complete program), guarantees that contextual equivalence is a *congruence*. In other words it is an equivalence relation that is respected by the various phrase-forming constructs of the language; and thus it supports the usual rules of equational reasoning. But that quantification over all contexts also makes it hard to establish further properties of contextual equivalence. Therefore, one seeks indirect means to develop its properties. One such means, at least for programming languages based on the lambda calculus, is the notion of *applicative bisimilarity* first introduced by Abramsky [1]. Roughly speaking, applicative bisimilarity is the greatest binary relation between phrases satisfying certain extensionality properties, such as equating two function expressions if they are equal argumentwise. As shown by Abramsky and later by Howe [10] in greater generality, applicative bisimilarity often coincides with contextual equivalence. Not only does this establish useful extensionality properties of contextual equivalence, but also the co-inductive form of the definition of applicative bisimilarity furnishes useful co-induction principles for contextual equivalence. (See [9] and [16] for example.)

The key to proving the coincidence of contextual equivalence with applicative bisimilarity is to show that the latter is a congruence. Abramsky's original proof of this fact falls out of his 'domain theory in logical form' [2]—a penetrating analysis of the compact elements of (recursively defined, 2/3 SFP) domains which is useful for far more than this congruence property. On the other hand, congruence of applicative bisimilarity is a purely syntactic property and as Howe [10] shows, it can be proved with syntactical methods based on the language's operational semantics. Here we present a new proof, one which falls somewhere between the previous two approaches. The proof makes use of denotational semantics, but only of a very simple kind: we merely need some quite standard considerations to do with computational adequacy and, crucially, some 'relational' properties of recursively defined domains established in previous work by the author [15].[4] On the other hand, modulo this somewhat minimal domain-theoretic machinery, the details of the new proof are perhaps conceptually simpler than Howe's method. Readers must judge for themselves. At least one can say that the material which follows presents an intriguing interplay between syntax and semantics and provides a new characterisation of applicative bisimulation (Theorem 5.3).

[3] Or rather, it is a reasonable family of such notions. For even if we insist that the definition of a programming language must specify not only what are the well-formed programs (the language's syntax), but also how to execute them (the language's 'operational semantics'), still there may be several different flavours of contextual equivalence corresponding to different choices of what constitutes a 'complete program', or what should be the 'observable results' of program execution.

[4] In particular, consideration of the compact elements of domains is not needed; so the approach may be useful for situations outside the scope of Abramsky's duality theory.

Before going any further, let us fix the syntax and operational semantics of an example language. For simplicity I will stick with the Abramsky-Ong 'lazy' lambda calculus [1, 4], although similar considerations apply to syntactically richer functional languages. So the phrases of the language are λ-terms

$$\begin{array}{rcll} M & ::= & x & \text{variables} \\ & \mid & \lambda x.M & \text{abstractions} \\ & \mid & MM & \text{applications} \end{array}$$

and the programs are the closed terms. Execution of programs can be specified by the relation of call-by-name evaluation of closed terms to lambda abstractions, inductively generated by the following axiom and rule:

$$\lambda x.M \Downarrow \lambda x.M \qquad \frac{P \Downarrow \lambda x.M \quad M[Q/x] \Downarrow V}{PQ \Downarrow V}$$

where as usual $M[Q/x]$ indicates the result of substituting Q for all free occurrences of x in M (modulo renaming of bound variables to avoid capture). We take convergence to some value (i.e. to some lambda abstraction) as the observable result of program execution, writing $P{\Downarrow}$ to mean that $P \Downarrow V$ holds for some V. Finally a *context* $C[-]$ is a λ-term with some of its subterms replaced by occurrences of a place-holder, or 'hole', written as $-$; and $C[M]$ indicates the result of replacing all occurrences of this place-holder with the term M (an operation which may well result in capture of free variables in M by λ-binders in $C[-]$).

With these preliminaries, we can define the notions of program equivalence and program ordering with which we will be concerned.

Definition 2.1 (Contextual equivalence) *The* contextual preorder *between (possibly open) λ-terms M and M' is written $M \leq^{\mathrm{ctx}} M'$ and is defined to hold if for all contexts $C[-]$ for which $C[M]$ and $C[M']$ are closed, if $C[M]{\Downarrow}$, then $C[M']{\Downarrow}$. The terms are* contextually equivalent*, written $M \cong^{\mathrm{ctx}} M'$, if both $M \leq^{\mathrm{ctx}} M'$ and $M' \leq^{\mathrm{ctx}} M$.*

Definition 2.2 (Applicative bisimilarity) *An* applicative simulation *is a binary relation $\mathcal{S}$ between closed λ-terms satisfying: if $P \mathrel{\mathcal{S}} P'$ and $P{\Downarrow}\lambda x.M$, then $P'{\Downarrow}\lambda x.M'$ for some M' such that for all closed Q, $M[Q/x] \mathrel{\mathcal{S}} M'[Q/x]$. Two closed terms P and P' are* applicatively similar*, written $P \leq^{\mathrm{app}} P'$, if $P \mathrel{\mathcal{S}} P'$ holds for some applicative simulation $\mathcal{S}$. We extend this relation to open terms via closed instantiations: for two open terms M and M', we write $M \leq^{\mathrm{app}} M'$ to mean that $M[\vec{Q}/\vec{x}] \leq^{\mathrm{app}} M'[\vec{Q}/\vec{x}]$ holds for all substitutions of closed terms $\vec{Q}$ for the free variables $\vec{x}$ of M and M'. Finally, M and M' are* applicatively bisimilar*, written $M \cong^{\mathrm{app}} M'$, if both $M \leq^{\mathrm{app}} M'$ and $M' \leq^{\mathrm{app}} M$.*

It is clear from the form of the first definition that $\leq^{\mathrm{ctx}}$ is reflexive and transitive, and hence that $\cong^{\mathrm{ctx}}$ is an equivalence relation. The same is true for $\leq^{\mathrm{app}}$ and $\cong^{\mathrm{app}}$, because one can easily check that the identity relation is an applicative simulation (hence $\leq^{\mathrm{app}}$ is reflexive) and that the relational composition of two applicative simulations is another such (hence $\leq^{\mathrm{app}}$ is transitive). Indeed the possibility of constructing and manipulating applicative simulations means that one can rapidly establish a number of useful properties of applicative (bi)similarity, such as:

Theorem 2.3 ('Kleene' equivalences) *Closed terms which evaluate to the same abstraction are applicatively bisimilar; and more generally, we have:*

$$\forall V(P \Downarrow V \Rightarrow P' \Downarrow V) \Rightarrow P \leq^{\mathrm{app}} P'.$$

In particular, β-conversion holds up to applicative bisimilarity:

$$(\lambda x.M)Q \cong^{\mathrm{app}} M[Q/x].$$

These properties hold because $\{(P, P') \mid \forall V(P \Downarrow V \;\Rightarrow\; P' \Downarrow V)\}$ is an applicative simulation.

Theorem 2.4 (Extensionality) *(i) For closed terms: $P \cong^{\mathrm{app}} P'$ if (and only if)*

$$(P{\Uparrow} \;\&\; P'{\Uparrow}) \vee \exists V, V'(P \Downarrow V \;\&\; P' \Downarrow V' \;\&\; V \cong^{\mathrm{app}} V')$$

(where $P{\Uparrow}$ means $P \Downarrow V$ does not hold for any V); and moreover

$$V \cong^{\mathrm{app}} V' \Leftrightarrow \forall Q(VQ \cong^{\mathrm{app}} V'Q).$$

(ii) For open terms: $M \cong^{\mathrm{app}} M'$ if (and only if) the closed terms $\lambda\vec{x}.M$ and $\lambda\vec{x}.M'$ are applicatively bisimilar (where $\vec{x}$ is a list of variables containing those free in M and M').

These properties hold simply because $\leq^{\mathrm{app}}$ is itself an applicative simulation (together with the validity of β-conversion, noted above).

The fact that these properties of applicative bisimilarity also hold for contextual equivalence is not at all easy to see merely from the definition of $\cong^{\mathrm{ctx}}$. However, that definition makes it clear that $\cong^{\mathrm{ctx}}$ does satisfy

Theorem 2.5 (Congruence) *For any context $C[-]$, if $M \cong^{\mathrm{ctx}} M'$, then $C[M] \cong^{\mathrm{ctx}} C[M']$.*

On the other hand, it is not at all clear from the definition of $\cong^{\mathrm{app}}$ that it is a congruence. We would like the best of both worlds, in which all these properties hold of our notion of program equivalence, and the following theorem tells us that we have this.

Theorem 2.6 (Abramsky's Operational Extensionality Theorem) *For all λ-terms M and M'*

$$M \leq^{\mathrm{ctx}} M' \Leftrightarrow M \leq^{\mathrm{app}} M'$$

and hence contextual equivalence and applicative bisimilarity coincide.

The key to proving this theorem is to establish the precongruence property for applicative similarity; that is, for all terms M and M', and for all contexts $C[-]$

$$M \leq^{\mathrm{app}} M' \Rightarrow C[M] \leq^{\mathrm{app}} C[M']. \tag{2.1}$$

Once we have (2.1), we certainly have the right-to-left implication of the theorem. For if $M \leq^{\mathrm{app}} M'$ and $C[M]{\Downarrow}$, then since $C[M] \leq^{\mathrm{app}} C[M']$, it is also the case that $C[M']{\Downarrow}$ (using the applicative simulation property of $\leq^{\mathrm{app}}$): therefore $M \leq^{\mathrm{ctx}} M'$.

For the converse, left-to-right implication, first note that $\leq^{\mathrm{ctx}}$ restricted to closed terms is an applicative simulation: for if $P \leq^{\mathrm{ctx}} P'$ and $P \Downarrow \lambda x.M$, then since $P\Downarrow$, it follows that $P'\Downarrow$, that is $P' \Downarrow \lambda x.M'$ holds for some M'. Since P evaluates to $\lambda x.M$, it is Kleene equivalent to it, hence applicatively bisimilar to it, and hence by the previous paragraph, is contextually equivalent to it; similarly for P' and $\lambda x.M'$. Therefore $\lambda x.M \cong^{\mathrm{ctx}} P \leq^{\mathrm{ctx}} P' \cong^{\mathrm{ctx}} \lambda x.M'$, and hence for any Q, $(\lambda x.M)Q \leq^{\mathrm{ctx}} (\lambda x.M')Q$. We noted above that β-conversion holds up to applicative bisimilarity and we now know that that implies contextual equivalence. So $M[Q/x] \cong^{\mathrm{ctx}} (\lambda x.M)Q \leq^{\mathrm{ctx}} (\lambda x.M')Q \cong^{\mathrm{ctx}} M'[Q/x]$. This verifies the applicative simulation condition for $\leq^{\mathrm{ctx}}$ and therefore

$$P \leq^{\mathrm{ctx}} P' \Rightarrow P \leq^{\mathrm{app}} P'.$$

Finally, this implication can be extended from closed to open terms using the closure of $\leq^{\mathrm{ctx}}$ under taking λ-abstractions and applications (immediate from the definition), the validity of β-conversion up to $\cong^{\mathrm{ctx}}$ (which we have already established), and the way that $\leq^{\mathrm{app}}$ is defined on open terms via closed instantiations.

So to prove the Operational Extensionality Theorem one just has to prove the precongruence property (2.1) of applicative similarity. The rest of this note describes a novel proof of this via a detour into denotational semantics and logical relations.

3 Denotational semantics

Lazy lambda calculus contextual equivalence can be modelled (adequately, but not 'fully abstractly') by the domain D recursively defined by $D \cong (D \to D)_\perp$. For our purposes it suffices to take 'domain' to mean a set equipped with a partial order ($\sqsubseteq$), possessing a least element ($\perp$) and least upper bounds for all countable ascending chains; and a function between domains is 'continuous' if it is monotone and preserves these least upper bounds. We refer the reader to the usual sources (such as Smyth [21]) giving constructions of recursively defined domains such as D. All we need to know about D are the following properties, which in fact characterise it uniquely up to isomorphism.

(D1) There is a continuous, order-reflecting function $\mathit{fun} : (D \to D) \longrightarrow D$ whose image is $\{d \in D \mid d \neq \perp\}$. (Here $D \to D$ is the usual domain of continuous functions from D to itself.)

(D2) Each $d \in D$ is the least upper bound of the ascending chain $\pi_0(d) \sqsubseteq \pi_1(d) \sqsubseteq \pi_3(d) \sqsubseteq \ldots$, where the continuous functions $\pi_n \in (D \to D)$ are defined by:

$$\pi_0(d) = \perp$$
$$\pi_{n+1}(d) = \begin{cases} \mathit{fun}(\pi_n \circ f \circ \pi_n) & \text{if } d = \mathit{fun}(f), \\ \perp & \text{if } d = \perp. \end{cases}$$

Property (D1) is equivalent to the existence of an isomorphism between D and the lifted function domain $(D \to D)_\perp$. As is well known, such an isomorphism permits one to give denotations to λ-terms M as elements $[\![M]\!]\rho$ of D, given an *environment* ρ, i.e. a finite partial function ρ mapping (at least) the free variables of M to elements of D. For completeness we give the definition, which proceeds by induction on the structure of M as follows:

$$\begin{aligned}
[\![x]\!]\rho &\overset{\text{def}}{=} \rho(x),\\
[\![\lambda x.M]\!]\rho &\overset{\text{def}}{=} \mathit{fun}(\lambda d \in D.[\![M]\!]\rho[x \mapsto d]),\\
[\![MM']\!]\rho &\overset{\text{def}}{=} \begin{cases} f([\![M']\!]\rho) & \text{if } [\![M]\!]\rho = \mathit{fun}(f),\\ \bot & \text{otherwise}\end{cases}
\end{aligned}$$

(where $\rho[x \mapsto d]$ is the environment mapping x to d and otherwise acting like ρ).

In case P is closed, $[\![P]\!]\rho$ is independent of ρ and we simply write $[\![P]\!]$ for this element of D.

We recall some key properties of the denotational semantics.

Theorem 3.1 (Compositionality) *For all λ-terms M and M' and contexts $C[-]$*

$$\forall \rho([\![M]\!]\rho \sqsubseteq [\![M']\!]\rho) \Rightarrow \forall \rho([\![C[M]]\!]\rho \sqsubseteq [\![C[M']]\!]\rho). \tag{3.1}$$

Theorem 3.2 (Substitution property) *For any λ-terms M and P with P closed, and any environment ρ*

$$[\![M[P/x]]\!]\rho = [\![M]\!](\rho[x \mapsto [\![P]\!]]). \tag{3.2}$$

Theorem 3.3 (Correctness for $\Downarrow$) *For any closed P and V*

$$P \Downarrow V \Rightarrow [\![P]\!] = [\![V]\!]. \tag{3.3}$$

These properties are easily proved from the definition of $[\![-]\!]$ by induction on the structure of $C[-]$, M, and the proof of $P \Downarrow V$ respectively. They only rely upon property (D1) of D. Property (D2) of D is the so-called *minimal invariant* property of this recursively defined domain and is needed to get a denotational characterisation of the operationally defined notion of observation we are using (i.e. convergence to some value).

Theorem 3.4 (Computational Adequacy) *For all closed λ-terms P, $P\Downarrow$ holds if and only if $[\![P]\!] \neq \bot$.*

The 'only if' direction follows immediately from (3.3) together with the fact that the denotation of any value (closed lambda abstraction) is of the form $\mathit{fun}(f)$ and hence is not equal to $\bot$. However, the 'if' direction is not so easy to prove. One method, due to Plotkin [19], makes use of a certain logical relation between domain elements and programs. Before going on to describe that relation in the next section, let us note that the Computational Adequacy property is important because it implies

Theorem 3.5 (Soundness) *If two λ-terms M have equal denotations, then they are contextually equivalent. More generally*

$$\forall \rho([\![M]\!]\rho \sqsubseteq [\![M']\!]\rho) \Rightarrow M \leq^{\text{ctx}} M'. \tag{3.4}$$

For suppose $\forall\rho(\llbracket M\rrbracket\rho \sqsubseteq \llbracket M'\rrbracket\rho)$ holds. For any context $C[-]$, if $C[M]\Downarrow$, say $C[M]\Downarrow V$, then by (3.1) and (3.3) one has

$$\bot \neq \llbracket V\rrbracket = \llbracket C[M]\rrbracket \sqsubseteq \llbracket C[M']\rrbracket$$

and hence $\llbracket C[M']\rrbracket \neq \bot$. Therefore by Computational Adequacy $C[M']\Downarrow$—as required for $M \leq^{\mathrm{ctx}} M'$.

The converse of (3.4) does not hold—D does not provide a 'fully abstract' model of λ-terms modulo the contextual preorder (see [4]). Abramsky and McCusker [3] have recently given a fully abstract model based on game semantics.

4 Formal approximation

Before giving the definition of our version of Plotkin's logical relation, we need some auxiliary definitions to do with relations between domain elements and closed λ-terms.

Definition 4.1 *Let $Rel(D, \Lambda^c)$ denote the set of binary relations between D and the set of closed λ-terms. If $\mathcal{R} \in Rel(D, \Lambda^c)$, we sometimes write $d\ \mathcal{R}\ P$ instead of $(d, P) \in \mathcal{R}$. We say that such a relation $\mathcal{R}$ is* admissible *if for each P, the subset $\{d \mid d\ \mathcal{R}\ P\}$ of D contains $\bot$ and is closed under taking least upper bounds of countable ascending chains. Partially ordered by set-theoretic inclusion, $Rel(D, \Lambda^c)$ is a complete lattice. Note that the intersection of any number of admissible relations is again admissible; therefore the collection of all such relations, partially ordered by inclusion, is again a complete lattice, which we will denote by $Rel_{adm}(D, \Lambda^c)$.*

Definition 4.2 (Formal approximation relation) *Given $\mathcal{R}^-, \mathcal{R}^+ \in Rel(D, \Lambda^c)$, define $\Delta(\mathcal{R}^-, \mathcal{R}^+) \in Rel(D, \Lambda^c)$ by:*

$$\begin{aligned}\Delta(\mathcal{R}^-, \mathcal{R}^+) \overset{\text{def}}{=} \{(d, P) \mid\ & d = \bot \vee \exists f, \lambda x.M \,.\, d = \mathit{fun}(f)\ \&\ P \Downarrow \lambda x.M \\ & \&\ \forall (d', P') \in \mathcal{R}^- \,.\, (f(d'), M[P'/x]) \in \mathcal{R}^+\}.\end{aligned}$$

$\lhd \in Rel(D, \Lambda^c)$ *is a* formal approximation relation *if it is admissible and satisfies:*

(FA1) $\lhd = \Delta(\lhd, \lhd)$*;*

(FA2) *for all $\mathcal{R}^- \in Rel(D, \Lambda^c)$ and $\mathcal{R}^+ \in Rel_{adm}(D, \Lambda^c)$, if $\mathcal{R}^- \subseteq \Delta(\mathcal{R}^+, \mathcal{R}^-)$ and $\Delta(\mathcal{R}^-, \mathcal{R}^+) \subseteq \mathcal{R}^+$, then $\mathcal{R}^- \subseteq \lhd \subseteq \mathcal{R}^+$.*

This definition of $\lhd$ arises out of the method of construction given in [15, Section 5]; (see also [14]). It differs from that given by Plotkin [19] in several respects.[5] Plotkin specifies some requirements on the relation $\lhd$ needed to prove computational adequacy, but which do not characterise it uniquely. These requirements are (FA1) together with the condition that $\lhd$ be downwards-closed in its first argument:

$$d \sqsubseteq d' \lhd P \Rightarrow d \lhd P.$$

We shall see below (Lemma 5.2) that this condition is a consequence of (FA2). Moreover, (FA2) ensures that $\lhd$ is unique if it exists: indeed if $\lhd'$ is any other admissible relation satisfying $\lhd' = \Delta(\lhd', \lhd')$, then property (FA2) implies that $\lhd' \subseteq \lhd \subseteq \lhd'$.

[5]Plotkin treats the case of a rather expressive, recursively typed language rather than the simple, untyped language we are using here; but it is relatively straightforward to extend our approach to such a language.

The existence of a relation satisfying (FA1) and (FA2) is not immediately obvious. For one thing, the function $\mathcal{R} \mapsto \Delta(\mathcal{R}, \mathcal{R})$ is not monotone, so one cannot just appeal to the Tarski fixed point theorem to satisfy (FA1). In fact $\lhd$ is an example of what is termed in [15] an *invariant relation* on the recursively defined domain D. As shown there, the existence of such relations can be deduced by combining the minimal invariant property of the recursively defined domain (property (D2) in the case of $D \cong (D \to D)_\perp$) with the well-known fact that every monotone function on a complete lattice possesses a least pre-fixed point. We will sketch the proof: for more details see *loc. cit.* (or [14] for a more concrete presentation of an example similar to the one we are considering here).

First, it is not hard to see that $\Delta(-,-)$ is order-reversing in its first argument and order-preserving in its second; and moreover that $\Delta(\mathcal{R}^-, \mathcal{R}^+)$ is admissible when $\mathcal{R}^+$ is.[6] Therefore

$$(\mathcal{R}^-, \mathcal{R}^+) \mapsto (\Delta(\mathcal{R}^+, \mathcal{R}^-), \Delta(\mathcal{R}^-, \mathcal{R}^+)) \tag{4.1}$$

determines a monotone function from the complete lattice

$$Rel_{adm}(D, \Lambda^c)^{op} \times Rel_{adm}(D, \Lambda^c)$$

to itself, which therefore has a least pre-fixed point, $(\lhd^-, \lhd^+)$ say. It suffices to prove that $\lhd^- = \lhd^+$: for then (FA1) will hold because any least pre-fixed point is always a fixed point; and (FA2) will hold when $\mathcal{R}^-$ is admissible by virtue of the least pre-fixed point property, and then will hold for general $\mathcal{R}^-$ by simple considerations involving the admissible closure of $\mathcal{R}^-$ (i.e. the intersection of all admissible relations containing $\mathcal{R}^-$).

To show that $\lhd^-$ and $\lhd^+$ are equal, it suffices to show that $\lhd^- \subseteq \lhd^+$, since the reverse inclusion is automatic from the defining property of $(\lhd^-, \lhd^+)$ and the symmetry of the function (4.1). Now the definition of Δ allows one to prove

$$\forall n (d \lhd^- P \Rightarrow \pi_n(d) \lhd^+ P)$$

by induction on n. Then $\lhd^- \subseteq \lhd^+$ follows from this using (D2) and the admissibility of $\lhd^+$.

So there is an element $\lhd$ of $Rel_{adm}(D, \Lambda^c)$ satisfying (FA1) and (FA2), and it is unique.

5 Applications of $\lhd$

Property (FA1) of $\lhd$ alone is sufficient to ensure that it enjoys a 'fundamental property of logical relations' of the following form, which can be proved by induction on the structure of M.

Theorem 5.1 (Fundamental Property of $\lhd$) *Given finite partial functions ρ and σ mapping variables to elements of D and to closed λ-terms respectively, write $\rho \lhd \sigma$*

[6] However, preservation of admissibility does seem to rely upon the determinacy of the evaluation relation, i.e. the fact that for each P there is at most one V satisfying $P{\Downarrow}V$. See the remarks at the end of Section 6 for the significance of this.

to mean that $dom(\rho) = dom(\sigma) = V$ say, and that $\forall x \in V(\rho(x) \lhd \sigma(x))$. Then for any λ-term M whose free variables are contained in V

$$\rho \lhd \sigma \Rightarrow [\![M]\!]\rho \lhd M[\sigma] \tag{5.1}$$

where $M[\sigma]$ indicates the result of simultaneously substituting $\sigma(x)$ for each $x \in V$.

Specialising this Fundamental Property to the case of a closed term P and with ρ and σ the empty partial functions (which vacuously satisfy $\rho \lhd \sigma$) we get

$$[\![P]\!] \lhd P. \tag{5.2}$$

In particular, if $[\![P]\!] \neq \bot$, then by (FA1) $P\Downarrow$. Therefore the existence of a relation $\lhd$ satisfying (FA1) is enough to complete the proof of the Computational Adequacy property.

What about property (FA2)? It determines $\lhd$ uniquely, but what else is it good for? As [15, Section 6] shows, this property of invariant relations can be made to yield various induction and co-induction principles for recursively defined domains, by varying the notion of relation (and associated action on relations, Δ). Here we will apply it to give a rather slick proof of the congruence property of applicative bisimilarity, and hence of the Operational Extensionality Theorem. The following is the key lemma.

Lemma 5.2

$$d \sqsubseteq d' \lhd P' \leq^{\text{app}} P \;\Rightarrow\; d \lhd P.$$

PROOF. Consider

$$\mathcal{R} \overset{\text{def}}{=} \{(d, P) \mid \exists d', P'(d \sqsubseteq d' \lhd P' \leq^{\text{app}} P)\}.$$

We have to show that $\mathcal{R} \subseteq \lhd$. By (FA2) with $\mathcal{R}^- = \mathcal{R}$ and $\mathcal{R}^+ = \lhd$, it suffices to check that $\mathcal{R} \subseteq \Delta(\lhd, \mathcal{R})$ and $\Delta(\mathcal{R}, \lhd) \subseteq \lhd$. For the second of these inclusions, note that since clearly $\lhd \subseteq \mathcal{R}$, we have $\Delta(\mathcal{R}, \lhd) \subseteq \Delta(\lhd, \lhd) = \lhd$, by (FA1). So it just remains to prove that if $d \mathrel{\mathcal{R}} P$, say $d \sqsubseteq d' \lhd P' \leq^{\text{app}} P$, then $(d, P) \in \Delta(\lhd, \mathcal{R})$.

By definition of Δ (Definition 4.2), if $d = \bot$ then there is nothing to prove. So we may assume that $d = \mathit{fun}(f)$ for some $f \in (D \to D)$. We have to show that $P \Downarrow \lambda x.M$ for some M satisfying

$$\forall d'', P''(d'' \lhd P'' \Rightarrow f(d'') \mathrel{\mathcal{R}} M[P''/x]). \tag{5.3}$$

But since $\mathit{fun}(f) = d \sqsubseteq d'$, it must be the case that $d' = \mathit{fun}(f')$ for some f' with $f \sqsubseteq f'$ (by property (D1) of D). Now $\mathit{fun}(f') = d' \lhd P'$, so since $\lhd = \Delta(\lhd, \lhd)$ it must be the case that $P' \Downarrow \lambda x.M'$ for some M' satisfying

$$\forall d'', P''(d'' \lhd P'' \Rightarrow f'(d'') \lhd M'[P''/x]).$$

Moreover, since $P' \Downarrow \lambda x.M'$ and $P' \leq^{\text{app}} P$, it must be that $P \Downarrow \lambda x.M$ for some M satisfying $M'[P''/x] \leq^{\text{app}} M[P''/x]$, for all P''. Thus for any d'' and P'', if $d'' \lhd P''$, then

$$f(d'') \sqsubseteq f'(d'') \lhd M'[P''/x] \leq^{\text{app}} M[P''/x]$$

which is to say that $f(d'') \mathrel{\mathcal{R}} M[P''/x]$. Therefore (5.3) holds, as required. ∎

Theorem 5.3 *For all closed λ-terms P and P'*

$$P \leq^{\mathrm{app}} P' \;\Leftrightarrow\; [\![P]\!] \lhd P'.$$

PROOF. If $P \leq^{\mathrm{app}} P'$ then by (5.2), $[\![P]\!] \lhd P \leq^{\mathrm{app}} P'$, so $[\![P]\!] \lhd P'$ holds by Lemma 5.2.

For the converse implication it suffices to show that

$$\mathcal{S} \overset{\mathrm{def}}{=} \{(P, P') \mid [\![P]\!] \lhd P'\}$$

is an applicative simulation: for then it is contained in the greatest such, $\leq^{\mathrm{app}}$. So suppose $[\![P]\!] \lhd P'$ and that $P \Downarrow \lambda x.M$. We have to show that $P' \Downarrow \lambda x.M'$ for some M' such that $(M[Q/x], M'[Q/x]) \in \mathcal{S}$ for all closed Q.

But if $P \Downarrow \lambda x.M$, then by the correctness property (3.3) of $[\![-]\!]$, $[\![P]\!] = \mathit{fun} f$ with $f = \lambda d \in D.[\![M]\!](x \mapsto d)$. Since $[\![P]\!] \lhd P'$, by (FA1) $P' \Downarrow \lambda x.M'$ for some M' such that

$$\forall d'', P''(d'' \lhd P'' \Rightarrow f(d'') \lhd M'[P''/x]).$$

So for any closed Q, since $[\![Q]\!] \lhd Q$, we have $f([\![Q]\!]) \lhd M'[Q/x]$. But by definition of f and by the substitution property (3.2) of $[\![-]\!]$

$$f([\![Q]\!]) = [\![M]\!](x \mapsto [\![Q]\!]) = [\![M[Q/x]]\!].$$

Therefore $[\![M[Q/x]]\!] \lhd M'[Q/x]$ holds for all closed Q, as required. ∎

Armed with this theorem we can give our proof of the Operational Extensionality Theorem 2.6.

PROOF. As noted at the end of Section 2, we just have to prove the precongruence property (2.1) of applicative similarity. Equivalently, we must prove

$$M \leq^{\mathrm{app}} M' \Rightarrow \lambda x.M \leq^{\mathrm{app}} \lambda x.M' \tag{5.4}$$

and

$$(M_1 \leq^{\mathrm{app}} M_1' \;\&\; M_2 \leq^{\mathrm{app}} M_2') \Rightarrow M_1 M_2 \leq^{\mathrm{app}} M_1' M_2'. \tag{5.5}$$

The first of these implications follows immediately from the fact that $\leq^{\mathrm{app}}$ is an applicative simulation, together with the way that $\leq^{\mathrm{app}}$ is extended from closed to open terms by taking instantiations. Similarly, it suffices to prove (5.5) just for closed terms; and using transitivity of $\leq^{\mathrm{app}}$, we can deduce this case from the slightly more general statement that for all closed P, P', and $\lambda x.M$

$$P \leq^{\mathrm{app}} P' \Rightarrow M[P/x] \leq^{\mathrm{app}} M[P'/x]. \tag{5.6}$$

But if $P \leq^{\mathrm{app}} P'$, then by Theorem 5.3 $[\![P]\!] \lhd P'$, so

$$\begin{aligned} [\![M[P/x]]\!] &= [\![M]\!](x \mapsto [\![P]\!]) && \text{by the substitution property (3.2) of } [\![-]\!] \\ &\lhd M[P'/x] && \text{by the fundamental property (5.1) of } \lhd \end{aligned}$$

and hence $M[P/x] \leq^{\mathrm{app}} M[P'/x]$ by the theorem again. ∎

Remark 5.4 ('Rational' completeness properties of $\leq^{\mathrm{app}}$) *It is perhaps worth mentioning another corollary of Theorem 5.3, to do with properties of fixpoints up to applicative similarity. Consider the usual fixpoint combinator*

$$Y \stackrel{\text{def}}{=} \lambda x.(\lambda y.x(yy))(\lambda y.x(yy))$$

and its finite approximations

$$Y_n \stackrel{\text{def}}{=} \begin{cases} \lambda x.\Omega & \text{if } n = 0 \\ \lambda x.x(Y_{n-1}x) & \text{if } n > 0 \end{cases}$$

where

$$\Omega \stackrel{\text{def}}{=} (\lambda x.xx)(\lambda x.xx).$$

Using the evaluation behaviour of these terms and the precongruence property (5.6)*, it is easy enough to show that for any closed P*

$$Y_0 P \leq^{\mathrm{app}} Y_1 P \leq^{\mathrm{app}} \ldots \leq^{\mathrm{app}} YP \cong^{\mathrm{app}} P(YP).$$

In fact YP is not just an upper bound of the chain $(Y_n P \mid n < \omega)$, it is the least such (with respect to $\leq^{\mathrm{app}}$) and this least upper bound is preserved by the various term-forming constructs:

$$\forall n(M[Y_n P/x] \leq^{\mathrm{app}} Q) \Rightarrow M[YP/x] \leq^{\mathrm{app}} Q. \tag{5.7}$$

Using the admissibility property of $\lhd$ built into its definition and the substitution property (3.2) *of $[\![-]\!]$,* (5.7) *follows immediately from Theorem 5.3 together with the well-known fact that $[\![YP]\!]$ is the least upper bound in the domain D of the elements $[\![Y_n P]\!]$ (for a nice proof of this, due to Plotkin, using just the minimal invariant property (D2) of D, see Example 3.6 of [15]).*

Property (5.7) *can serve as the basis of syntactic versions of various kinds of fixpoint induction arguments for $\leq^{\mathrm{app}}$ and hence for $\leq^{\mathrm{ctx}}$. The proof of it we have just sketched is not as compelling an application of Theorem 5.3 as the proof of Operational Extensionality, because once we know that $\leq^{\mathrm{app}}$ coincides with $\leq^{\mathrm{ctx}}$,* (5.7) *can be proved more or less directly from the definition of the contextual preorder using the computational adequacy of the denotational semantics. However, for some languages there are useful notions of applicative similarity contained in, but not coinciding with $\leq^{\mathrm{ctx}}$, for which one can still use this method of establishing* (5.7) *via a formal approximation relation between semantics and syntax. (The fragment of ML with references and the notion of applicative similarity considered in [20] provides an example of such a potential application.)*

6 Conclusion

In this note we have indicated how to prove the congruence property of applicative bisimulation, a purely syntactic result, using a logical relation between semantics and syntax that possesses a 'universal' property of mixed inductive/co-inductive character.

The domain theory needed for this is quite 'classical'. So the moral, if the reader wants one, is that there is still something new to be squeezed out of the traditional domain-theoretic toolkit even if exciting new methods (such as in [3], [11], and [12]) have arrived.

We confined attention to the lazy λ-calculus because it minimises syntactic complications without trivialising the difficulty of proving the congruence property of the associated notion of applicative bisimilarity. However, it should be possible to extend the method to more expressive languages—namely ones whose denotational semantics involves the kind of recursive domains whose relational properties are analysed in [15]. We claim that this is possible for recursively typed, functional languages like Plotkin's FPC [6], with either call-by-name or call-by-value evaluation: the details will appear elsewhere. It should also be possible to employ this approach to proving congruence of applicative bisimilarity for some imperative languages with local state. However, language features that introduce (countable) non-determinism seem to be outside the scope of this method. There is a technical reason for this: the fact that the 'action' $(\mathcal{R}^-, \mathcal{R}^+) \mapsto \Delta(\mathcal{R}^-, \mathcal{R}^+)$ associated with the recursively defined domain D preserves admissibility in its second argument is crucial to the proof method; but, as the reader can check, the proof of this fact seems to rely heavily on the determinacy of the evaluation relation $\Downarrow$ used in the definition of Δ. By contrast, the purely operational method of Howe [10] for proving congruence already includes the non-deterministic case (see also [17]).

Acknowledgements

The idea of using my mixed induction/co-induction property of the formal approximation relation to prove congruence properties of applicative bisimilarity came to me after listening to a talk by Achim Jung on 'A rational reconstruction of K. Mulmuley's thesis' at the *Workshop on Full Abstraction of PCF and Related Languages* that took place at the BRICS Institute, Aarhus, in April 1995. I would like to thank Furio Honsell, Marina Lenisa and Guy McCusker for their comments on this work.

This note is an expanded version of an invited talk given at the *3rd Workshop on Logic, Language, Information and Computation* (*WoLLIC'96*), Salvador, Brazil, in May 1996. I wish to thank the organisers in general, and Ruy de Queiroz in particular, for organising such a nice meeting.

Research partially supported by the EU HCM Research Network on 'Lambda Calcul Typé'.

References

[1] Abramsky, S. The lazy λ-calculus. In D. A. Turner (Ed.), *Research Topics in Functional Programming*, Chapter 4, pp. 65–117. Addison Wesley, 1990.

[2] Abramsky, S. Domain theory in logical form. *Annals of Pure and Applied Logic 51*, 1–77, 1991.

[3] Abramsky, S. and G. McCusker. Games and full abstraction for the lazy λ-calculus. In *10th Annual Symposium on Logic in Computer Science*, pp. 234–243. IEEE Computer Society Press, Washington, 1995.

[4] Abramsky, S. and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation 105*, 159–267, 1993.

[5] Egidi, L., F. Honsell, and S. R. della Rocca. Operational, denotational and logical descriptions:

a case study. *Fundamenta Informaticae 26*, 149–169, 1992.

[6] Fiore, M. and G. D. Plotkin. An axiomatization of computationally adequate domain theoretic models of FPC. In *9th Annual Symposium on Logic in Computer Science*, pp. 92–102. IEEE Computer Society Press, Washington, 1994.

[7] Freyd, P. J. Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts (Eds.), *Applications of Categories in Computer Science, Proceedings LMS Symposium, Durham, UK, 1991*, Volume 177 of *LMS Lecture Note Series*, pp. 95–106. Cambridge University Press, 1992.

[8] Gordon, A. D. *Functional Programming and Input/Output*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994.

[9] Gordon, A. D. Bisimilarity as a theory of functional programming. In *Eleventh Conference on the Mathematical Foundations of Programming Semantics, New Orleans, 1995*, Volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1995.

[10] Howe, D. J. Proving congruence of bisimulation in functional programming languages. *Information and Computation 124*(2), 103–112, February 1996.

[11] Hyland, J. M. E. and C.-H. L. Ong. On full abstraction for PCF. Submitted for publication, 1996.

[12] McCusker, G. Games and full abstraction for FPC. In *11th Annual Symposium on Logic in Computer Science*, pp. 174–183. IEEE Computer Society Press, Washington, 1996.

[13] Milner, R. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science 4*, 1–22, 1977.

[14] Pitts, A. M. 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*, pp. 72–82. Springer-Verlag, Berlin, 1994.

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

[16] Pitts, A. M. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts (Eds.), *Semantics and Logics of Computation*, series *Publications of the Newton Institute*, Cambridge University Press, 1997, pp. 241–298.

[17] Pitts, A. M. and J. R. X. Ross. Process calculus based upon evaluation to committed form. *Theoretical Computer Science*, to appear. A preliminary version appeared in U. Montanari and V. Sassone (Eds.), *CONCUR'96: Concurrency Theory, Proc. 7th Int. Conf. Pisa, 1996.*, Volume 1119 of *Lecture Notes in Computer Science*, pp. 18–33. Springer-Verlag, Berlin, 1996.

[18] Plotkin, G. D. LCF considered as a programming language. *Theoretical Computer Science 5*, 223–255, 1977.

[19] Plotkin, G. D. Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.

[20] Ritter, E. and A. M. Pitts. A fully abstract translation between a λ-calculus with reference types and Standard ML. In *2nd Int. Conf. on Typed Lambda Calculus and Applications, Edinburgh, 1995*, Volume 902 of *Lecture Notes in Computer Science*, pp. 397–413. Springer-Verlag, Berlin, 1995.

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

Received 31 August 1996. Revised 26 November 1996