Bisimulations for Untyped Imperative Objects

Vasileios Koutavas and Mitchell Wand

Northeastern University
vkoutav@ccs.neu.edu wand@ccs.neu.edu

Abstract. We present a sound and complete method for reasoning about contextual equivalence in the untyped, imperative object calculus of Abadi and Cardelli [1]. Our method is based on bisimulations, following the work of Sumii and Pierce [25, 26] and our own [14]. Using our method we were able to prove equivalence in more complex examples than the ones of Gordon, Hankin and Lassen [7] and Gordon and Rees [8]. We can also write bisimulations in closed form in cases where similar bisimulation methods [26] require an inductive specification. To derive our bisimulations we follow the same technique as we did in [14], thus indicating the extensibility of this method.

1 Introduction

Contextual equivalence, attributed to Morris in 1968 [19], is the standard relation used to prove that two terms are operationally identical. Terms a and a' are contextually equivalent if and only if for any program context C, $C[a]$ and $C[a']$ co-terminate. *CIU theorems* [16] try to ease the quantification over all contexts by examining only a subset of them (usually the reduction contexts).

In the presence of a store, though, an inductive proof of equivalence must reason not only about the possible contexts under empty, or even equal stores, but also under *related* stores. This is something that neither the standard definition of contextual equivalence, nor CIU theorems take into account. Thus using them to prove the equivalence of expressions that manipulate the store in a sufficiently different way can become cumbersome.

Denotational approaches addressed this problem by translating terms to more structured mathematical models (e.g. [18]). If two terms have the same denotations in some model then they are equivalent. Unfortunately even some evident equivalences do not hold in naive models [17], and finding fully-abstract models, in which all contextual equivalences hold, is generally difficult. For example consider the two different implementations of a cell class shown in Figure 1. The first is the usual implementation; the second stores the object in two fields and its `get` method returns one of them, depending on the value of a counter. These two programs have different denotations in most models, and thus can't be proven equivalent by a straightforward denotational method.

A more appropriate method to deal with such equivalences is by using *bisimulations*. Bisimulations were introduced in process calculi by Hennessy and Milner [9, 10], and adapted later to sequential calculi by Abramsky [2]. They are relations between entire program configurations, and thus the main difficulty of

```
class Cell {                   class Cell {
  private Object y;              private Object y1, y2;
                                 private int p;
  Cell (Object x)                Cell (Object x)
    {y = x;}                       {p = 0; y1 = x; y2 = x;}
  public void set (Object z)     public void set (Object z)
    {y = z;}                       {p = p+1; y1 = z; y2 = z;}
  public int get ()              public int get ()
    {return y;}                    {if ((p % 2) == 0) then return y1;
                                                      else return y2;}
}                              }
```

Fig. 1. Cell Example

using them as equivalence relations is to apply them to terms and show that they are a congruence. Moreover bisimulations are often hard to write down explicitly because they are usually infinite sets with little structure.

Sumii and Pierce in [25, 26] greatly simplified the use of bisimulations in sequential calculi. Their main innovation was to group the related pairs of configurations according to their conditions of knowledge (e.g. the type environment). In this way they gave more structure to their bisimulations, thus making their concrete definition easier. Similar ideas have also been used in process calculi (eg. [5]).

In [14] we improved their method and provided a proof technique for equivalence in an imperative, untyped λ-calculus. Our improvements aimed mainly to reduce the size of bisimulations, and make possible a constructive proof even in the presence of higher-order procedures and a general store, where previous methods had shown limitations [3, 20, 26]. We achieved this by applying "up-to" techniques usually used in process calculi [22, 21, 23], and by analyzing a direct proof of equivalence to unveil weaker conditions for our bisimulations.

In this paper we follow the same methodology to create a bisimulation proof technique for the imperative, untyped object calculus of Abadi and Cardelli [1]. In contrast to [14], the values of this calculus do not have significant structure, and thus an "up-to context" technique is not useful here. Instead we use an "up-to store" technique to deal with the complex structure of the store. Using our framework we are able to construct bisimulations to prove known examples [7, 8], as well as more complex ones, which are hard to prove using the method in [7].

The rest of the paper is structured as follows: In Section 2 we review the untyped, imperative object calculus impς. In Section 3 we define a notion of contextual equivalence for values, and we connect it with the standard notion of contextual equivalence. In Section 4 we attempt a proof of a set being included in contextual equivalence, from which we derive the necessary conditions for the elements of that set. In Section 5 we gather these conditions into a definition for a bisimulation which we simplify in Section 6 by introducing an "up-to store"

$$
\begin{array}{rrcll}
\textsc{Expressions:} & a, b & ::= & x & \text{variables} \\
 & & \mid & [\overline{l = \varsigma(x)b}] & \text{Objects} \\
 & & \mid & a.l & \text{Method Invocation} \\
 & & \mid & a.l \Leftarrow \varsigma(x)b & \text{Method update} \\
 & & \mid & \texttt{clone}(a) & \text{Cloning} \\
 & & \mid & \texttt{let}\, x = a \,\texttt{in}\, b & \text{Let} \\
\textsc{Locations:} & \iota & & & \\
\textsc{Values:} & v, u & ::= & [\overline{l = \iota}] & \\
\textsc{Environments:} & \rho & ::= & (\overline{x \mapsto v}) & \\
\textsc{Stores:} & \sigma & ::= & (\overline{\iota \mapsto \langle \varsigma(x)b, \rho \rangle}) &
\end{array}
$$

Fig. 2. The impς Language

closure on sets. In Section 7 we use bisimulations to prove the equivalence of some examples. Sections 8 and 9 summarize the related work and conclusions.

2 The language impς

We develop our theory based on impς [1], an untyped, imperative object calculus. The syntactic domains and the big-step environment semantics of the language are shown in Fig. 2 and 3, respectively.

As in [14] we use an overbar notation to denote a syntactic sequence:

$$\overline{s} = s_1, \ldots, s_n$$

where s is a syntax fragment and s_i is the same fragment with i-subscripts on all meta-identifiers it contains. Thus we write $[\overline{l = \varsigma(x)b}]$, instead of $[l_1 = \varsigma(x_1)b_1, \ldots, l_n = \varsigma(x_n)b_n]$, and $\overline{(v, v')} \in R$ instead of $(v_1, v_1'), \ldots, (v_n, v_n') \in R$. The size of the sequence in the overbar notation is arbitrary, or implicitly defined by the context.

In impς, objects are defined by the syntactic construct $[\overline{l = \varsigma(x)b}]$, where l_i and b_i are the label and the body of the i-th method, respectively. The variable x_i is bound to the entire object when b_i is evaluated. During the evaluation bindings are kept in environments (called "stacks" in [1]).

Values in impς are objects that map method names to locations in the store; they are denoted by $[\overline{l = \iota}]$. Locations range over an infinite, countable set. The store maps locations to method closures, which consist of a method and the appropriate environment, e.g. $\langle \varsigma(x_i)b_i, \rho_i \rangle$.

We use the operational semantics of impς given in [1], with the addition of an extra condition in the Env x rule which guarantees that there are no dangling pointers in an environment. Furthermore, there is an implicit α-conversion in the Red Select and Red Let rule, so that the identifiers added to the environments are unique.

$$\boxed{\sigma \vdash_{\mathrm{wf}} \diamond}$$

$$\frac{}{\emptyset \vdash_{\mathrm{wf}} \diamond}\ \textsc{Store}\ \emptyset \qquad \frac{\sigma; \rho \vdash_{\mathrm{wf}} \diamond \qquad \iota \notin \mathit{Dom}(\sigma) \qquad \mathit{FV}(b) \subseteq \mathit{Dom}(\rho) \cup \{x\}}{(\sigma, \iota {\mapsto} \langle \varsigma(x)b, \rho \rangle) \vdash_{\mathrm{wf}} \diamond}\ \textsc{Store}\ \iota$$

$$\boxed{\sigma; \rho \vdash_{\mathrm{wf}} \diamond}$$

$$\frac{\sigma \vdash_{\mathrm{wf}} \diamond}{\sigma; \emptyset \vdash_{\mathrm{wf}} \diamond}\ \textsc{Env}\ \emptyset \qquad \frac{\sigma; \rho \vdash_{\mathrm{wf}} \diamond \qquad x \notin \mathit{Dom}(\rho) \qquad \{\overline{\iota}\} \subseteq \mathit{Dom}(\sigma)}{\sigma; (\rho, x {\mapsto} [\overline{l = \iota}]) \vdash_{\mathrm{wf}} \diamond}\ \textsc{Env}\ x$$

$$\boxed{\sigma; \rho \vdash a \Downarrow v; \sigma'}$$

$$\frac{\sigma; (\rho', x {\mapsto} v, \rho'') \vdash_{\mathrm{wf}} \diamond}{\sigma; (\rho', x {\mapsto} v, \rho'') \vdash x \Downarrow v; \sigma}\ \textsc{Red x}$$

$$\frac{\sigma; \rho \vdash_{\mathrm{wf}} \diamond \qquad \{\overline{\iota}\} \cap \mathit{Dom}(\sigma) = \emptyset}{\sigma; \rho \vdash [\overline{l = \varsigma(x)b}] \Downarrow [\overline{l = \iota}]; (\sigma, \overline{\iota {\mapsto} \langle \varsigma(x)b, \rho \rangle})}\ \textsc{Red Object}$$

$$\frac{\begin{array}{c}\sigma; \rho \vdash a \Downarrow [\overline{l = \iota}]; \sigma' \qquad l_j \in \{\overline{l}\} \qquad \sigma'(\iota_j) = \langle \varsigma(x_j)b_j, \rho' \rangle \\ x_j \notin \mathit{Dom}(\rho') \qquad \sigma'; (\rho', x_j {\mapsto} [\overline{l = \iota}]) \vdash b_j \Downarrow v; \sigma''\end{array}}{\sigma; \rho \vdash a.l_j \Downarrow v; \sigma''}\ \textsc{Red Select}$$

$$\frac{\sigma; \rho \vdash a \Downarrow [\overline{l = \iota}]; \sigma' \qquad l_j \in \{\overline{l}\}}{\sigma; \rho \vdash a.l_j \Leftarrow \varsigma(x)b \Downarrow [\overline{l = \iota}]; (\sigma'.\iota_j {\leftarrow} \langle \varsigma(x)b, \rho \rangle)}\ \textsc{Red Update}$$

$$\frac{\sigma; \rho \vdash a \Downarrow [\overline{l = \iota}]; \sigma' \qquad \{\overline{\iota'}\} \notin \mathit{Dom}(\sigma')}{\sigma; \rho \vdash \texttt{clone}(a) \Downarrow [\overline{l = \iota'}]; (\sigma', \overline{\iota' {\mapsto} \sigma'(\iota)})}\ \textsc{Red Clone}$$

$$\frac{\sigma; \rho \vdash a \Downarrow v'; \sigma' \qquad x \notin \mathit{Dom}(\rho) \qquad \sigma'; (\rho, x {\mapsto} v') \vdash b \Downarrow v''; \sigma''}{\sigma; \rho \vdash \texttt{let}\, x \,{=}\, a \,\texttt{in}\, b \Downarrow v''; \sigma''}\ \textsc{Red Let}$$

Fig. 3. Operational Semantics

Judgments of the form $\sigma; \rho \vdash a \Downarrow v; \sigma_1$ represent a big-step evaluation, where expression a, under store σ and environment ρ, evaluates to value v and a new store σ_1. We also use the form $\sigma; \rho \vdash a \Downarrow^{<k} v; \sigma_1$ to denote that the evaluation tree has height less than k.

Store and environment extensions are written as $(\sigma, \overline{\iota {\mapsto} \langle \varsigma(x)b, \rho \rangle})$ and $(\rho, \overline{x {\mapsto} \iota})$, respectively. Store update is written as $(\sigma.\iota {\leftarrow} \langle \varsigma(x)b, \rho \rangle)$; the contents of the location ι of store σ is given by $\sigma(\iota)$. The domain of a store or environment is given by $\mathit{Dom}(\sigma)$ or $\mathit{Dom}(\rho)$, and the locations of a value v are given by $\mathit{Locs}(v)$.

Judgments of the form $\sigma \vdash_{\mathrm{wf}} \diamond$ define the *well-formed stores*, while judgments $\sigma; \rho \vdash_{\mathrm{wf}} \diamond$ define the *well-formed environments* under some store. We also call the form $\sigma; \rho \vdash a$ a *configuration*, and define the notion of a *well-formed configuration* as follows:

Definition 1. *We say that $\sigma; \rho \vdash a$ is a* well-formed configuration*, and we write $\sigma; \rho \vdash_{\mathrm{wf}} a$, iff:*

$$\sigma; \rho \vdash_{\mathrm{wf}} \diamond \quad \wedge \quad FV(a) \subseteq \mathit{Dom}(\rho)$$

Lemma 1. *If $\sigma; \rho \vdash_{\mathrm{wf}} a$ and $\sigma; \rho \vdash a \Downarrow v; \sigma_1$, then for any environment ρ_1, and identifier $x \notin \mathit{Dom}(\rho_1)$, such that $\sigma_1; \rho_1 \vdash_{\mathrm{wf}} \diamond$, we have $\sigma_1; (\rho_1, x \mapsto v) \vdash_{\mathrm{wf}} \diamond$.*

Proof. By straightforward induction on the height of $\sigma; \rho \vdash a \Downarrow v; \sigma_1$.

3 Contextual Equivalence

The first relation we define is the standard contextual equivalence for this calculus.

Definition 2 (Standard Contextual Equivalence ($\equiv_{\mathsf{std}}$)). $(a, a') \in \equiv_{\mathsf{std}}$ *if and only if for all contexts C such that $\emptyset; \emptyset \vdash_{\mathrm{wf}} C[a]$ and $\emptyset; \emptyset \vdash_{\mathrm{wf}} C[a']$, we have:*

$$\emptyset; \emptyset \vdash C[a] \Downarrow \iff \emptyset; \emptyset \vdash C[a'] \Downarrow$$

Proving equivalence of two expressions using this definition is hard because carrying out the proof will require us to reason about equivalent expressions not just under empty or even equal stores and environments, but also under *equivalent* stores and environments.

To address this complication we first define a different notion of contextual equivalence ($\equiv$) as a set of *bisimulation states*. A bisimulation state consists of a pair of stores, and a set of pairs of values that are to be considered equivalent in these states. Then we extend this equivalence to any expression and we show that it coincides with standard contextual equivalence.

We reach this new definition of contextual equivalence by building on the following two definitions of simpler relations.

Definition 3 (Value Relation). *A* value relation *R is a set of pairs of values.*

Value relations hold the related values, and implicitly the related store locations, in some state of the equivalence. These objects and locations are also the only ones that a context can access.

Definition 4 (Environment Relation (R^{ϵ})). *If R is a value relation, then R^{ϵ} is a relation on environments, defined by:*

$$R^{\epsilon} = \{(\rho, \rho') \mid \mathit{Dom}(\rho) = \mathit{Dom}(\rho') \ \&\ \forall x \in \mathit{Dom}(\rho) \,.\, (\rho(x), \rho'(x)) \in R\}$$

Since environments related by R^{ϵ} contain objects from R, the locations in the environments are in the domain of the stores of a state iff R contains locations only in the domain of these stores. Therefore we define the notion of well-formed bisimulation states:

Definition 5 (Well-Formed State). *We call a state* (σ, σ', R) well-formed, *and we write* $\sigma, \sigma' \vdash_{\text{wf}} R$, *iff the following holds:*

$$\begin{aligned} \forall (v, v') \in R \ . \ & (\mathit{Locs}(v) \subseteq \mathit{Dom}(\sigma)) \ \ \wedge (\sigma \vdash_{\text{wf}} \diamond) \\ \wedge \ & (\mathit{Locs}(v') \subseteq \mathit{Dom}(\sigma')) \wedge (\sigma' \vdash_{\text{wf}} \diamond) \end{aligned}$$

We now give the definition of contextual equivalence:

Definition 6 (Contextual Equivalence ($\equiv$)). Contextual equivalence *is the set of all states* (σ, σ', R) *such that* σ, σ' *are stores and* R *is a value relation, and:*

1. $\sigma, \sigma' \vdash_{\text{wf}} R$,
2. *if* $(\rho, \rho') \in R^{\epsilon}$, *and* a *is any expression such that* $FV(a) \subseteq \mathit{Dom}(\rho)$ *then:*

$$\sigma; \rho \vdash a \Downarrow \quad \textit{iff} \quad \sigma'; \rho' \vdash a \Downarrow$$

To extend $\equiv$ to any pair of expressions, we create a one-method object for each expression, such that the expression is the body of the method, and we relate these objects in an appropriate state of $\equiv$.

Definition 7 (Extension of ($\equiv$) to any expression). *Two expressions* a, a' *are related by* $\equiv$, *and we write* $a \equiv a'$, *iff:*

$$\begin{aligned} \forall \sigma, \rho \,.\, \exists l, \iota \colon \textit{if} \quad & \quad \sigma_1 = (\sigma, \iota \mapsto \langle \varsigma(\_) a, \rho \rangle) \\ & \wedge \sigma_1' = (\sigma, \iota \mapsto \langle \varsigma(\_) a', \rho \rangle) \\ \textit{then} \quad & \quad \sigma_1, \sigma_1' \vdash_{\text{wf}} \mathit{Id}(\sigma) \\ & \wedge (\sigma_1, \sigma_1', \mathit{Id}(\sigma)) \in \equiv \end{aligned}$$

where:

$$\mathit{Id}(\sigma) = \{([\overline{l = \iota}], [\overline{l = \iota}]) \mid \bar{\iota} \in \mathit{Dom}(\sigma)\}$$

The extended $\equiv$ coincides with $\equiv_{\mathsf{std}}$:

Theorem 1. $a \equiv a'$ *if and only if* $a \equiv_{\mathsf{std}} a'$.

4 Deriving Obligations for an Equivalence Proof

As in [14] we attempt to prove that a set of triples (σ, σ', R), namely $\mathcal{X}$, is included in $\equiv$. In this proof we encounter sub-cases where $\mathcal{X}$ must satisfy specific conditions in order for the proof to go through. These conditions will become the necessary conditions of our bisimulations in the next section.

For $\mathcal{X}$ to be included in $\equiv$, one must prove that for all $(\sigma, \sigma', R) \in \mathcal{X}$:

1. $\sigma, \sigma' \vdash_{\text{wf}} R$,

2. if $(\rho, \rho') \in R^\epsilon$ and a is an expression such that $FV(a) \subseteq Dom(\rho)$ then:

$$\sigma; \rho \vdash a \Downarrow \quad \text{iff} \quad \sigma'; \rho' \vdash a \Downarrow$$

The proof of the first condition is straightforward. It suffices to inspect that the environments of all stored methods and the values in R refer to locations in the domains of σ and σ'. This condition will become the first condition of our bisimulations.

Proving the second part requires an induction on the height of the derivations of $\sigma; \rho \vdash a \Downarrow$ in the forward direction, and $\sigma'; \rho' \vdash a \Downarrow$ in the reverse direction. We show only the forward direction; the other is symmetric.

To carry out the induction we strengthen the induction hypothesis by relating the final configurations under some state of $\mathcal{X}$.

$$\begin{aligned} IH(k) = \forall \sigma, \sigma', R, \rho, \rho', a, v, \sigma_1 \,.& \\ &((\sigma, \sigma', R) \in \mathcal{X}) \wedge ((\rho, \rho') \in R^\epsilon) \wedge (FV(a) \subseteq Dom(\rho)) \\ &\wedge (\sigma; \rho \vdash a \Downarrow^{<k} v; \sigma_1) \\ &\implies \exists v', \sigma_1', Q : \ (\sigma'; \rho' \vdash a \Downarrow v'; \sigma_1') \\ &\qquad\qquad \wedge \ ((v, v') \in Q) \wedge (Q \supseteq R) \\ &\qquad\qquad \wedge \ ((\sigma_1, \sigma_1', Q) \in \mathcal{X}) \end{aligned} \tag{1}$$

Using (1) as the induction hypothesis we proceed by induction on k, considering the cases of a. Most of the cases follow immediately by the induction hypothesis; the rest will become the proof obligations for $\mathcal{X}$, and furthermore the conditions in the definition of bisimulations.

To demonstrate this we consider the case of method invocation ($a = a_1.l$). We assume (1) for k and we prove it for $k+1$.

Let $(\sigma, \sigma', R) \in \mathcal{X}$, $(\rho, \rho') \in R^\epsilon$, $FV(a) = FV(a_1) \subseteq Dom(\rho)$, and $\sigma; \rho \vdash a_1.l_j \Downarrow^{<k+1} v; \sigma_2$. We have to show that:

$$\begin{aligned} \exists v', \sigma_1', Q : \ & (\sigma'; \rho' \vdash a_1.l \Downarrow v'; \sigma_1') \\ \wedge \ & ((v, v') \in Q) \wedge (Q \supseteq R) \\ \wedge \ & ((\sigma_1, \sigma_1', Q) \in \mathcal{X}) \end{aligned}$$

The Red Select evaluation rule for the left-hand side gives:

$$\frac{\begin{array}{c} \sigma; \rho \vdash a_1 \Downarrow^{<k} [\overline{l = \iota}]; \sigma_1 \qquad l_j \in \{\bar{l}\} \\ \sigma_1(\iota_j) = \langle \varsigma(x) b_j, \rho_1 \rangle \qquad x \notin Dom(\rho_1) \qquad \sigma_1; (\rho_1, x \mapsto [\overline{l = \iota}]) \vdash b_j \Downarrow^{<k} v; \sigma_2 \end{array}}{\sigma; \rho \vdash a_1.l_j \Downarrow^{<k+1} v; \sigma_2}$$

To show that $\sigma'; \rho' \vdash a_1.l \Downarrow v'; \sigma_1'$ we must establish the premises of Red Select for the right-hand side as well.

By the induction hypothesis at a_1 we get that there exist σ_1', $\overline{l', \iota'}$, Q, such that:

$$\sigma'; \rho' \vdash a_1 \Downarrow [\overline{l' = \iota'}]; \sigma_1', \quad Q \supseteq R, \quad ([\overline{l = \iota}], [\overline{l' = \iota'}]) \in Q, \quad (\sigma_1, \sigma_1', Q) \in \mathcal{X} \tag{2}$$

We also need to show that $\{\bar{l'}\} \subseteq \{\bar{l}\}$. This does not follow from the induction hypothesis and therefore we formulate it as a condition on $\mathcal{X}$:

$\boxed{\textit{If } (\sigma, \sigma', R) \in \mathcal{X} \textit{ and } ([\overline{l = \iota}], [\overline{l' = \iota'}]) \in R, \textit{ then } \{\bar{l}\} \subseteq \{\overline{l'}\}.}$

By Lemma 1 and the first formula of (2) we get that $\sigma'_1; (x \mapsto [\overline{l' = \iota'}]) \vdash_{\text{wf}} \diamond$, and therefore there exist b'_j, ρ'_j, such that $\sigma'_1(\iota'_j) = \langle \varsigma(x) b'_j, \rho'_j \rangle$.

Finally, to show that the right-hand side terminates and prove the inductive step, we require the following condition to hold for $\mathcal{X}$:

If $(\sigma, \sigma', R) \in \mathcal{X}$ *and* $([\overline{l = \iota}], [\overline{l = \iota'}]) \in R$, *then for all* $l_j \in \{\bar{l}\}$, *with* $\sigma(\iota_j) = \langle \varsigma(x) b, \rho \rangle$ *and* $\sigma'(\iota'_j) = \langle \varsigma(x') b', \rho' \rangle$, *the following must be true:*

a) $x \notin \rho$, *and* $x' \notin \rho'$.
b) If $IH(k)$ *holds, then:*

$$\begin{aligned}
&\sigma; (\rho, x \mapsto [\overline{l = \iota}]) \vdash b \Downarrow^{<k} v; \sigma_1 \\
&\quad \Longrightarrow \exists v', \sigma'_1, Q : \ (\sigma'; (\rho', x' \mapsto [\overline{l = \iota'}]) \vdash b' \Downarrow v'; \sigma'_1) \\
&\qquad\qquad \wedge \ ((v, v') \in Q) \wedge (Q \supseteq R) \\
&\qquad\qquad \wedge \ ((\sigma_1, \sigma'_1, Q) \in \mathcal{X})
\end{aligned}$$

With a similar treatment for the rest of the cases we discover all the proof obligations of $\mathcal{X}$, which we will formulate as the conditions of bisimulations in the following section.

5 Small Bisimulations

We first define some new notation to make the transfer of the induction hypothesis from the direct proof into the definition of bisimulations easier.

Definition 8 (k-approximation). *We write* $(\sigma, \sigma', R) \vdash_{\mathcal{X}} a|_{\rho} \sqsubseteq_k a'|_{\rho'}$ *to mean:*

$$\begin{aligned}
&\forall v, \sigma_1 .\ \sigma; \rho \vdash a \Downarrow^{<k} v; \sigma_1 \\
&\quad \Longrightarrow \exists v', \sigma'_1, Q : \ \sigma'; \rho' \vdash a' \Downarrow v'; \sigma'_1 \\
&\qquad\qquad \wedge \ ((v, v') \in Q) \wedge (Q \supseteq R) \\
&\qquad\qquad \wedge \ ((\sigma_1, \sigma'_1, Q) \in \mathcal{X})
\end{aligned}$$

Similarly, in the other direction, we write $(\sigma, \sigma', R) \vdash_{\mathcal{X}} a|_{\rho} \sqsupseteq_k a'|_{\rho'}$ *to mean:*

$$\begin{aligned}
&\forall v', \sigma'_1 .\ \sigma'; \rho' \vdash a' \Downarrow^{<k} v'; \sigma'_1 \\
&\quad \Longrightarrow \exists v, \sigma_1, Q : \ \sigma; \rho \vdash a \Downarrow v; \sigma_1 \\
&\qquad\qquad \wedge \ ((v, v') \in Q) \wedge (Q \supseteq R) \\
&\qquad\qquad \wedge \ ((\sigma_1, \sigma'_1, Q) \in \mathcal{X})
\end{aligned}$$

Note that the two directions are not converse, since they both contain $(\sigma_1, \sigma'_1, Q) \in \mathcal{X}$ and $(v, v') \in Q$. Similarly we give two versions of the full induction hypothesis, one for each direction:

Definition 9 (Induction Hypotheses).

$$IH^L_{\mathcal{X}}(k) \triangleq \forall(\sigma,\sigma',R) \in \mathcal{X}\,.\ \forall(\rho,\rho') \in R^\epsilon\,.\ \forall a : FV(a) \subseteq Dom(\rho)\,.\ (\sigma,\sigma',R) \vdash_{\mathcal{X}} a|_\rho \sqsubseteq_k a|_{\rho'}$$

$$IH^R_{\mathcal{X}}(k) \triangleq \forall(\sigma,\sigma',R) \in \mathcal{X}\,.\ \forall(\rho,\rho') \in R^\epsilon\,.\ \forall a : FV(a) \subseteq Dom(\rho)\,.\ (\sigma,\sigma',R) \vdash_{\mathcal{X}} a|_\rho \sqsupseteq_k a|_{\rho'}$$

The induction hypotheses involve the k-approximation of the same terms a under related stores and environments. The definition of bisimulations follows:

Definition 10 (Bisimulation). *A set $\mathcal{X}$ of states (σ,σ',R) is called a* bisimulation *if and only if, for any $(\sigma,\sigma',R) \in \mathcal{X}$, the following conditions are satisfied:*

1. $\sigma,\sigma' \vdash_{\text{wf}} R$
2. *For all $\bar{\iota} \notin Dom(\sigma)$, $\bar{\iota'} \notin Dom(\sigma')$, labels $\bar{l}$, $(\rho,\rho') \in R^\epsilon$, $x \notin Dom(\rho)$, and expressions b such that $FV(b) \subseteq \rho \cup \{x\}$ there exists $Q \supseteq R$ with*

$$([\overline{l=\iota}], [\overline{l=\iota'}]) \in Q \quad \textit{and} \quad ((\sigma, \overline{\iota \mapsto \langle\varsigma(x)b,\rho\rangle}), (\sigma', \overline{\iota' \mapsto \langle\varsigma(x)b,\rho'\rangle}), Q) \in \mathcal{X}$$

3. *If $([\overline{l=\iota}], [\overline{l'=\iota'}]) \in R$, then $\{\bar{l}\} = \{\bar{l'}\}$.*
4. *If $([\overline{l=\iota}], [\overline{l=\iota'}]) \in R$, then, for all $l_j \in \{\bar{l}\}$, $(\rho,\rho') \in R^\epsilon$, $x \notin Dom(\rho)$, and expressions b such that $FV(b) \subseteq Dom(\rho) \cup \{x\}$, there exists $Q \supseteq R$, with*

$$((\sigma.\iota_j \leftarrow \langle\varsigma(x)b,\rho\rangle), (\sigma'.\iota'_j \leftarrow \langle\varsigma(x)b,\rho'\rangle), Q) \in \mathcal{X}$$

5. *If $([\overline{l=\iota}], [\overline{l=\iota'}]) \in R$, then, for all $\overline{\iota_1} \notin Dom(\sigma)$, $\overline{\iota'_1} \notin Dom(\sigma')$, there exists $Q \supseteq R$ with*

$$([\overline{l=\iota_1}], [\overline{l=\iota'_1}]) \in Q \quad \textit{and} \quad ((\sigma, \overline{\iota_1 \mapsto \sigma(\iota)}), (\sigma', \overline{\iota'_1 \mapsto \sigma'(\iota')}), Q) \in \mathcal{X}$$

6. *If $([\overline{l=\iota}], [\overline{l=\iota'}]) \in R$, then, for any $l_j \in \{\bar{l}\}$ with $\sigma(\iota_j) = \langle\varsigma(x)b,\rho\rangle$ and $\sigma'(\iota'_j) = \langle\varsigma(x)b',\rho'\rangle$, let $\rho_1 = (\rho, x \mapsto [\overline{l=\iota}])$ and $\rho'_1 = (\rho', x \mapsto [\overline{l=\iota'}])$, we have:*

$$\begin{aligned} IH^L_{\mathcal{X}}(k) &\Longrightarrow (\sigma,\sigma',R) \vdash_{\mathcal{X}} b|_{\rho_1} \sqsubseteq_k b'|_{\rho'_1} \\ IH^R_{\mathcal{X}}(k) &\Longrightarrow (\sigma,\sigma',R) \vdash_{\mathcal{X}} b|_{\rho_1} \sqsupseteq_k b'|_{\rho'_1} \end{aligned}$$

The first condition of the definition allows only well-formed states in the bisimulations. The second condition addresses the proof obligation for the case of evaluating an object. Conditions 3 and 6 are the proof obligations for the case of method invocation, as explained in Section 4. Conditions 4 and 5 address the proof obligations for the cases of method update and cloning, respectively. The case of `let` follows immediately from the induction hypothesis and generates no condition for the bisimulations.

Next we show that our method is sound and complete, and that a maximal bisimulation exists.

Theorem 2 (Completeness). *Contextual equivalence is a bisimulation.*

$$\frac{}{(\sigma, \sigma', R) \sqsubseteq (\sigma, \sigma', R)}$$

$$\frac{(\sigma, \sigma', R) \sqsubseteq (\sigma_1, \sigma'_1, R_1) \qquad \overline{\iota_e, \iota'_e} \text{ fresh} \qquad (\rho, \rho') \in R^{\epsilon}}{(\sigma, \sigma', R) \sqsubseteq ((\sigma_1, \overline{\iota_e {\mapsto} \langle\varsigma(x)b, \rho\rangle}), (\sigma'_1, \overline{\iota'_e {\mapsto} \langle\varsigma(x)b, \rho'\rangle}), R_1 \cup \{([\overline{l = \iota_e}], [\overline{l = \iota'_e}])\})}$$

$$\frac{\begin{array}{c}(\sigma, \sigma', R) \sqsubseteq (\sigma_1, \sigma'_1, R_1) \\ ([\overline{l = \iota}], [\overline{l = \iota'}]) \in R_1 \qquad (\rho, \rho') \in R^{\epsilon} \qquad \iota_u \in \{\overline{\iota}\} \qquad \iota'_u \in \{\overline{\iota'}\}\end{array}}{(\sigma, \sigma', R) \sqsubseteq ((\sigma_1.\iota_u {\leftarrow} \langle\varsigma(x)b, \rho\rangle), (\sigma'_1.\iota'_u {\leftarrow} \langle\varsigma(x)b, \rho'\rangle), R_1)}$$

Fig. 4. Up-to Store Extension of States

Theorem 3 (Soundness). *Any bisimulation $\mathcal{X}$ is included in Contextual Equivalence.*

Proof. This proof recapitulates the derivation of Section 4.

Theorem 4 (Bisimilarity). *A maximal bisimulation, called Bisimilarity ($\sim$), exists and coincides with Contextual Equivalence.*

Proof. By the definition of ($\equiv$) and Theorems 2 and 3, we get that ($\equiv$) is itself the largest sound bisimulation. Thus bisimilarity coincides with contextual equivalence.

6 Up-to Store Closure

Conditions 2 and 4 of Definition 10 close bisimulations under any possible extensions of the store with new object methods and any possible update of existing methods. Writing down sets to satisfy these conditions can become cumbersome.

To eliminate the need of satisfying these conditions when one writes a bisimulation in closed form, we introduce an *up-to store* closure operator on sets. Then we give a new set of necessary conditions on the (smaller) sets, such that their up-to store closure is a bisimulation.

Definition 11 (Up-to Store Extension of States). *The state $(\sigma_1, \sigma'_1, R_1)$ is an* up to store extension *of state $(\sigma_0, \sigma'_0, R_0)$, written $(\sigma_0, \sigma'_0, R_0) \sqsubseteq (\sigma_1, \sigma'_1, R_1)$, iff it satisfies the rules of Figure 4.*

The second rule of Figure 4 states that extending a bisimulation state with arbitrary new pairs of related objects, and also extending the stores accordingly to keep their methods, is a valid up-to store extension. The third rule states that updating some known locations with related methods is also a valid up-to store extension.

We now give the definition of an up-to store closure operator on sets, and the necessary conditions for a set $\mathcal{X}$ such that its up-to store closure is a bisimulation:

Definition 12 (Up-to Store Closure of Sets).

$$\mathcal{X}^* = \{(\sigma, \sigma', R) \mid \exists (\sigma_0, \sigma_0', R_0) \in \mathcal{X} : (\sigma_0, \sigma_0', R_0) \sqsubseteq (\sigma, \sigma', R)\}$$

Theorem 5. *$\mathcal{X}^*$ is a bisimulation if for all $(\sigma, \sigma', R) \in \mathcal{X}$, we have:*

1. $\sigma, \sigma' \vdash_{\text{wf}} R$
2. *If* $([\overline{l = \iota}], [\overline{l' = \iota'}]) \in R$*, then* $\{\bar{l}\} = \{\bar{l'}\}$*.*
3. *If* $([\overline{l = \iota}], [\overline{l = \iota'}]) \in R$*, then, for all* $\overline{\iota_1} \notin \mathit{Dom}(\sigma)$*,* $\overline{\iota_1'} \notin \mathit{Dom}(\sigma')$*, there exists* $Q \supseteq R$ *with*
$$([\overline{l = \iota_1}], [\overline{l = \iota_1'}]) \in Q \quad \textit{and} \quad ((\sigma, \overline{\iota_1 \mapsto \sigma(\iota)}), (\sigma', \overline{\iota_1' \mapsto \sigma'(\iota')}), Q) \in \mathcal{X}^*$$
4. *If* $([\overline{l = \iota}], [\overline{l = \iota'}]) \in R$*, then, for any* $l_j \in \{\bar{l}\}$ *with* $\sigma(\iota_j) = \langle \varsigma(x)b, \rho \rangle$ *and* $\sigma'(\iota_j') = \langle \varsigma(x')b', \rho' \rangle$*, and for all* $(\sigma_1, \sigma_1', R_1)$ *with* $(\sigma, \sigma', R) \sqsubseteq (\sigma_1, \sigma_1', R_1)$*, let* $\rho_1 = (\rho, x \mapsto [\overline{l = \iota}])$ *and* $\rho_1' = (\rho', x \mapsto [\overline{l = \iota'}])$*, we have:*
$$IH^L_{\mathcal{X}^*}(k) \Longrightarrow (\sigma_1, \sigma_1', R_1) \vdash_{\mathcal{X}^*} b|_{\rho_1} \sqsubseteq_k b'|_{\rho_1'}$$
$$IH^R_{\mathcal{X}^*}(k) \Longrightarrow (\sigma_1, \sigma_1', R_1) \vdash_{\mathcal{X}^*} b|_{\rho_1} \sqsupseteq_k b'|_{\rho_1'}$$

Proof. Straightforward by inspecting that $\mathcal{X}^*$ satisfies the conditions of Definition 10.

7 Example

Using our bisimulations and Theorem 5 we were able to prove the equivalences in [7] and the untyped, imperative equivalent of the example in [8]. Due to space limitations, we omit these examples here. Instead we prove equivalence in a more interesting example that demonstrates the ability of our method to deal with hidden imperative fields, different store manipulation, and higher-order methods.

Consider the two classes shown in Figure 1. Objects of these classes are indistinguishable in any context (in the absence of reflection). To prove this we encode the objects in impς. We simplify the encoding by extending impς in the usual way with integers, arithmetic operators, and a conditional statement, and we encode methods containing λ-abstractions as follows:

$$[\cdots, f = \varsigma(s)\lambda y.e, \cdots] \triangleq [\mathit{arg} = \varsigma(s)s.\mathit{arg}, \cdots, f = \varsigma(s)e[s.\mathit{arg}/y], \cdots]$$

The context passes an argument to the body of method f by updating the *arg* label and then selecting f. Because the *arg* label may be updated with an arbitrary method, every argument is potentially arbitrary complicated. The induction hypothesis in the last condition of Definition 10 and Theorem 5 is crucial for reasoning about these arguments.

The objects of the `Cell` classes are encoded as:

$$\begin{aligned} M = \texttt{let}\, o &= [y = \varsigma(\_)0] \\ &\texttt{in}\, [\mathit{arg} = \varsigma(s)s.\mathit{arg}, \mathit{set} = \mathit{set}_M, \mathit{get} = \mathit{get}_M] \end{aligned}$$

$$\begin{aligned} N = \texttt{let}\, o &= [y_1 = \varsigma(\_)0, y_2 = \varsigma(\_)0, c = \varsigma(\_)0] \\ &\texttt{in}\, [\mathit{arg} = \varsigma(s)s.\mathit{arg}, \mathit{set} = \mathit{set}_N, \mathit{get} = \mathit{get}_N] \end{aligned}$$

where:

$$
\begin{array}{llll}
\mathit{set}_M \triangleq \varsigma(s)\,\mathtt{let}\, z = s.\mathit{arg} & & \mathit{get}_M \triangleq \varsigma(\_)\,o.y \\
\qquad\qquad \mathtt{in}\, o.y \Leftarrow \varsigma(\_)z & & \\
\mathit{set}_N \triangleq \varsigma(s)\,\mathtt{let}\, n = o.c + 1 & & \mathit{get}_N \triangleq \varsigma(\_)\,\mathtt{let}\, x = \mathit{even?}(o.c) \\
\qquad\qquad\quad z = s.\mathit{arg} & & \qquad\qquad \mathtt{in}\,\mathtt{if}\, x\, \mathtt{then}\, o.y_1\, \mathtt{else}\, o.y_2 \\
\qquad\qquad \mathtt{in}\,(o.c \Leftarrow \varsigma(\_)n; & & \\
\qquad\qquad\quad o.y_1 \Leftarrow \varsigma(\_)z; & & \\
\qquad\qquad\quad o.y_2 \Leftarrow \varsigma(\_)z) & &
\end{array}
$$

By Theorem 1, to prove $M \equiv_{\mathsf{std}} N$ it is sufficient to show that $M \equiv N$. Thus we have to construct a bisimulation that contains $((\sigma, \iota_0 \mapsto \langle \varsigma(\_)M, \rho\rangle), (\sigma, \iota_0 \mapsto \langle \varsigma(\_)N, \rho\rangle), R)$, for all σ, ρ, and for some ι_0, l_0 and R, such that $([l_0 = \iota_0], [l_0 = \iota_0]) \in R$.

To do this we define the parameterized value relation:

$$
\begin{aligned}
& Q(\overline{\iota_s, \iota_g, \iota_a, \iota'_s, \iota'_g, \iota'_a}, \overline{\iota_0}) \\
& \qquad = \overline{\{([\mathit{arg} = \iota_a, \mathit{set} = \iota_s, \mathit{get} = \iota_g], [\mathit{arg} = \iota'_a, \mathit{set} = \iota'_s, \mathit{get} = \iota'_g]),} \\
& \qquad\quad \overline{([l_0 = \iota_0], [l_0 = \iota_0])\}}
\end{aligned}
$$

the parameterized stores:

$$
\begin{aligned}
& \sigma(\overline{\iota_s, \iota_g, \iota_a, \iota_y, \rho_M, \rho_1}, \overline{\iota_0, \rho_0}) \\
& \qquad = (\overline{\iota_0 \mapsto \langle \varsigma(\_)M, \rho_0\rangle}, \overline{\iota_a \mapsto \langle \varsigma(s)s.\arg, \rho_M\rangle, \iota_s \mapsto \langle \mathit{set}_M, \rho_M\rangle, \iota_g \mapsto \langle \mathit{get}_M, \rho_M\rangle,} \\
& \qquad\quad \overline{\iota_y \mapsto \langle \varsigma(\_)x, \rho_1\rangle})
\end{aligned}
$$

$$
\begin{aligned}
& \sigma'(\overline{\iota'_s, \iota'_g, \iota'_a, \iota_{y_1}, \iota_{y_2}, \iota_c, \rho_N, \rho'_1, n}, \overline{\iota'_0, \rho'_0}) \\
& \qquad = (\overline{\iota_0 \mapsto \langle \varsigma(\_)N, \rho'_0\rangle}, \overline{\iota'_a \mapsto \langle \varsigma(s)s.\arg, \rho_N\rangle, \iota'_s \mapsto \langle \mathit{set}_N, \rho_N\rangle, \iota'_g \mapsto \langle \mathit{get}_N, \rho_N\rangle,} \\
& \qquad\quad \overline{\iota_{y_1} \mapsto \langle \varsigma(\_)x, \rho'_1\rangle, \iota_{y_2} \mapsto \langle \varsigma(\_)x, \rho'_1\rangle, \iota_c \mapsto \langle \varsigma(\_)n, \rho'_1\rangle})
\end{aligned}
$$

and the set:

$$
\begin{aligned}
\mathcal{X} = \big\{(\sigma, \sigma', R) \,|\, \exists & \overline{\iota_s, \iota_g, \iota_a, \iota'_s, \iota'_g, \iota'_a, \iota_y, \iota_{y_1}, \iota_{y_2}, \iota_c, \rho_M, \rho_N, \rho_1, \rho'_1, n}, \overline{\iota_0, \iota'_0, \rho_0, \rho'_0} \\
: \;& R = Q(\overline{\iota_s, \iota_g, \iota_a, \iota'_s, \iota'_g, \iota'_a}, \overline{\iota_0}) \\
\wedge \;& \sigma = \sigma(\overline{\iota_s, \iota_g, \iota_a, \iota_y, \rho_M, \rho_1}, \overline{\iota_0, \rho_0}) \\
\wedge \;& \sigma' = \sigma'(\overline{\iota'_s, \iota'_g, \iota'_a, \iota_{y_1}, \iota_{y_2}, \iota_c, \rho_N, \rho'_1, n}, \overline{\iota'_0, \rho'_0}) \\
\wedge \;& \overline{((\rho_0, \rho'_0) \in R^\epsilon)} \wedge \overline{((\rho_1, \rho'_1) \in R^\epsilon)} \\
\wedge \;& \overline{\rho_M = (\rho_1, y \mapsto \iota_y)} \\
\wedge \;& \overline{\rho_N = (\rho'_1, y_1 \mapsto \iota_{y_1}, y_2 \mapsto \iota_{y_2}, c \mapsto \iota_c)}\big\}
\end{aligned}
$$

We have to show that $\mathcal{X}$ satisfies the conditions of Theorem 5, and thus $\mathcal{X}^*$ is a bisimulation. It is easy to check that conditions 1, 2, and 3 are satisfied. It remains to prove Condition 4 for $(\iota_{0_k}, \iota'_{0_k})$, $(\iota_{s_k}, \iota'_{s_k})$, and $(\iota_{g_k}, \iota'_{g_k})$, for any k.

We consider $(\iota_{s_k}, \iota'_{s_k})$. Let $(\sigma_1, \sigma'_1, R_1) \in \mathcal{X}^*$, and:

$$
([\mathit{arg}{=}\iota_{a_k}, \mathit{set}{=}\iota_{s_k}, \mathit{get}{=}\iota_{g_k}], [\mathit{arg}{=}\iota'_{a_k}, \mathit{set}{=}\iota'_{s_k}, \mathit{get}{=}\iota'_{g_k}]) \in R_1
$$

By the definition of up-to store extension for states, we observe that some of the labels of the above objects may have been updated by the context. Thus we have two cases:

Case 1. The labels ι_{s_k} and ι'_{s_k} have been updated: $\sigma_1(\iota_{s_k}) = \langle\varsigma(x)b, \rho'\rangle$, $\sigma'_1(\iota'_{s_k}) = \langle\varsigma(x)b, \rho'\rangle$, $(\rho, \rho') \in R^\epsilon$.

We have to show that if $\rho_1 = (\rho, x{\mapsto}[\mathit{arg}{=}\iota_{a_k}, \mathit{set}{=}\iota_{s_k}, \mathit{get}{=}\iota_{g_k}])$ and $\rho'_1 = (\rho', x{\mapsto}[\mathit{arg}{=}\iota'_{a_k}, \mathit{set}{=}\iota'_{s_k}, \mathit{get}{=}\iota'_{g_k}])$, then:

$$\begin{aligned} IH^L_{\mathcal{X}^*}(k) &\Longrightarrow (\sigma_1, \sigma'_1, R_1) \vdash_{\mathcal{X}^*} b|_{\rho_1} \sqsubseteq_k b|_{\rho'_1} \\ IH^R_{\mathcal{X}^*}(k) &\Longrightarrow (\sigma_1, \sigma'_1, R_1) \vdash_{\mathcal{X}^*} b|_{\rho_1} \sqsupseteq_k b|_{\rho'_1} \end{aligned}$$

But these are immediately satisfied by $IH^L_{\mathcal{X}^*}(k)$ and $IH^R_{\mathcal{X}^*}(k)$, since $(\rho_1, \rho'_1) \in R^\epsilon$.

Case 2. The labels ι_{s_k} and ι'_{s_k} have not been updated: $\sigma_1(\iota_{s_k}) = \langle\varsigma(\_)\mathit{set}_M, \rho_{M_k}\rangle$ $\sigma'_1(\iota'_{s_k}) = \langle\varsigma(\_)\mathit{set}_N, \rho_{N_k}\rangle$. We will show only the forward direction:

$$IH^L_{\mathcal{X}^*}(k) \Longrightarrow (\sigma_1, \sigma'_1, R_1) \vdash_{\mathcal{X}^*} \mathit{set}_M|_{\rho_{M_k}} \sqsubseteq_k \mathit{set}_N|_{\rho_{N_k}}$$

Let:

$$\sigma_1; \rho_{M_k} \vdash \mathit{set}_M \Downarrow^{<k} [\mathit{arg} = \iota_{a_k}, \mathit{set} = \iota_{s_k}, \mathit{get} = \iota_{g_k}]; \sigma_2$$

This implies that $(\sigma_1; \rho_{M_k} \vdash s.\mathit{arg} \Downarrow^{<k-1} v; \sigma_3)$, $\sigma_2 = (\sigma_3.\iota_{y_k} \leftarrow \langle\varsigma(\_)z, \rho_1\rangle)$, and $\rho_1(z) = v$. From $IH^L_{\mathcal{X}^*}(k)$ we get:

$$\begin{aligned} \exists v', \sigma'_3, R_3 \ :\ & \sigma'_1; \rho_{N_k} \vdash s.\mathit{arg} \Downarrow v'; \sigma'_3 \\ \wedge\ & ((v, v') \in R_3) \wedge (R_3 \supseteq R_1) \\ \wedge\ & ((\sigma_3, \sigma'_3, R_3) \in \mathcal{X}^*) \end{aligned}$$

thus:

$$\begin{aligned} &\sigma'_1; \rho_{N_k} \vdash \mathit{set}_N \Downarrow [\mathit{arg} = \iota'_{a_k}, \mathit{set} = \iota'_{s_k}, \mathit{get} = \iota'_{g_k}]; \sigma'_2 \\ &\sigma'_2 = (\sigma'_3.\iota_{y_{1_k}} \leftarrow \langle\varsigma(\_)z, \rho'_1\rangle, \iota_{y_{2_k}} \leftarrow \langle\varsigma(\_)z, \rho'_1\rangle, \iota_{c_k} \leftarrow \langle\varsigma(\_)m, \rho'_1\rangle) \end{aligned}$$

and by the definition of $\mathcal{X}$ and $\mathcal{X}^*$ we get that there exists $R_2 \supseteq R_1$ such that $(\sigma_2, \sigma'_2, R_2) \in \mathcal{X}^*$.

Similarly we prove Condition 4 for $(\iota_{g_k}, \iota'_{g_k})$, and $(\iota_{0_k}, \iota'_{0_k})$.

8 Related Work

Gordon, Hankin and Lassen in [7] gave an *operational equivalence* for the same calculus that we consider here, and they showed that it coincides with contextual equivalence. This equivalence is a CIU theorem for this language. Their relation does not provide a technique to prove difficult equivalences. For example, proving the `Cell` example with their CIU theorem would require an induction over all reduction contexts. Such an induction is not obvious because the stores and the environments of the two sides may change in a different way in some of the cases. Such a proof would be at least as difficult as the induction discussed in Section 4.

On the other hand, using our bisimulations we were able to prove equivalence for all of their examples, as well as more complex ones, by a constructive proof.

Gordon and Rees in [8] proved for one of the stateless, typed, object calculi of Abadi and Cardelli that bisimilarity coincides with contextual equivalence. This was the first study of contextual equivalence in an object calculus. Their method was quite different than ours, being closer to the original operational bisimulations of Abramsky [2] and Howe's proof of congruence [11]. Reasoning about higher-order programs is easier with our method because of the use of the induction hypotheses in the definition of bisimulation.

A different approach to studying equivalence in object calculi is by translating them to π-calculus. Kleist and Sangiorgi have done this in [13] for the typed version of the calculus we study here, and Sangiorgi in [24] for the functional version of this calculus. Similar work has been done for parallel object oriented languages (e.g. [12, 15, 27]). This approach is in essence denotational and all of these translations were not fully abstract, so none of these methods is complete.

9 Conclusions and Future Work

We have presented a method of deriving bisimulations for the untyped, imperative object calculus of Abadi and Cardelli. To our knowledge this is the first sound and complete method that uses bisimulations to prove equivalence for this calculus, and successfully handles complex examples.

We hope to use our method to investigate contextual equivalence in more realistic imperative object languages [4, 6]. We also plan to to investigate better ways to express stylized bisimulations, like those in the example of Section 7.

References

1. Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, Berlin, Heidelberg, and New York, 1996.
2. Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.
3. Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings*, volume 3461 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2005.
4. Gavin Bierman, Matthew Parkinson, and Andrew Pitts. MJ: An imperative core calculus for Java and Java with effects. Technical Report 563, Cambridge University Computer Laboratory, April 2003.
5. Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 445–456. Springer-Verlag, 2004.
6. Matthew Flatt, Shriram Krishnamurthi, and Matthias Felleisen. A programmer's reduction semantics for classes and mixins. In *Formal Syntax and Semantics of Java*, pages 241–269, London, UK, 1999. Springer-Verlag.
7. Andrew D. Gordon, Paul D. Hankin, and Søren B. Lassen. Compilation and equivalence of imperative objects. In *Proceedings of the 17th Conference on Foundations*

of Software Technology and Theoretical Computer Science, pages 74–87, London, UK, 1997. Springer-Verlag.

8. Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings 23rd Annual ACM Symposium on Principles of Programming Languages*, pages 386–395, New York, NY, USA, 1996. ACM Press.
9. Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *ICALP*, pages 299–309, 1980.
10. Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.
11. Douglas J. Howe. Equality in lazy computation systems. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 198–203, 1989.
12. Cliff B. Jones. A pi-calculus semantics for an object-based design notation. In Eike Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1993.
13. Josva Kleist and Davide Sangiorgi. Imperative objects as mobile processes. *Sci. Comput. Program.*, 44(3):293–342, 2002.
14. Vasileios Koutavas and Mitchell Wand. Smaller bisimulations for reasoning about higher-order imperative programs. In *Proceedings 33rd Annual ACM Symposium on Principles of Programming Languages*, New York, NY, USA, 2006. ACM Press.
15. Xinxin Liu and David Walker. Partial confluence of processes and systems of objects. *Theor. Comput. Sci.*, 206(1-2):127–162, 1998.
16. Ian A. Mason and Carolyn L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.
17. Albert R. Meyer and Kurt Sieber. Towards fully abstract semantics for local variables: Preliminary report. In *Proceedings 15th Annual ACM Symposium on Principles of Programming Languages*, pages 191–203, 1988.
18. Robert Milne and Christopher Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, 1976. Also Wiley, New York.
19. James H. Morris, Jr. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, Cambridge, MA, 1968.
20. Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In Andrew Gordon and Andrew Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Publications of the Newton Institute, Cambridge University Press, 1998.
21. D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.
22. D. Sangiorgi. On the bisimulation proof method. *Mathematical. Structures in Comp. Sci.*, 8(5):447–479, 1998.
23. D. Sangiorgi and R. Milner. The problem of “Weak Bisimulation up to”. In W.R. Cleveland, editor, *Proc. CONCUR ’92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.
24. Davide Sangiorgi. An interpretation of typed objects into typed π-calculus. *Inf. Comput.*, 143(1):34–73, 1998.
25. Eijiro Sumii and Benjamin C. Pierce. A bisimulation for dynamic sealing. In *Proceedings 31st Annual ACM Symposium on Principles of Programming Languages*, pages 161–172, New York, NY, USA, 2004. ACM Press.
26. Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. In *Proceedings 32nd Annual ACM Symposium on Principles of Programming Languages*, pages 63–74, New York, NY, USA, 2005. ACM Press.
27. David Walker. Objects in the π-calculus. *Inf. Comput.*, 116(2):253–271, 1995.