Bisimulations for Untyped Imperative Objects

Vasileios Koutavas and Mitchell Wand
Northeastern University
Proving Equivalence

e e′
Proving Equivalence

∀C.

\[ \frac{C[e]}{C[e']} \]

\[ n \sim n' \]

**Standard Formulation of Equivalence**

- Put the terms in all possible program contexts
- check if the final answers are equal:

\[ n = n' \]
Proving Equivalence

∀C.
C[e]  C[e']

Standard Formulation of Equivalence
• Put the terms in all possible program contexts
• check if the final answers are equal:
  n = n'

Hard to use this formulation
  – must consider all possible contexts and their final answers
  – no clue about induction!

Bisimulations for Untyped Imperative Objects
Proving Equivalence

Some techniques:

- CIU theorems [Gordon, Hankin & Lassen]
- Logical Relations [Pitts & Stark]
- Bisimulations [Gordon & Rees, Sumii & Pierce]
- etc.
Proving Equivalence

Some techniques:
- CIU theorems [Gordon, Hankin & Lassen]
- Logical Relations [Pitts & Stark]
- Bisimulations [Gordon & Rees, Sumii & Pierce]
- etc.

Each of them has shortcomings that make it hard to reason about some equivalences.
- E.g. Some of the Meyer & Sieber [1988] equivalences
Contributions of Our Technique

- Gives a *sound and complete* proof method of equivalence for the Untyped Imperative Object Calculus of Abadi & Cardelli

- Extends the bisimulation method of Sumii & Pierce
  - We give an *easier set of conditions* in the definition
  - Uses an “up-to store” technique to reduce the size of bisimulations
  - Proves equivalences which an adaptation of S&P’s method is not able to prove.
Contributions of Our Technique

• Proves equivalences between objects that invoke callbacks (higher-order) and objects with different local store manipulation
  – e.g. all the Meyer & Sieber equivalences [1988], where other approaches fail
    • Denotational methods, CIU theorems, Logical Relations, Bisimulations.

• It is applicable to many calculi
  – CBV/CBN lambda calculus
  – Imperative CBV lambda calculus [POPL’06]
  – Imperative Object calculus
Technical Difficulties

- Expression language is impoverished
- Structure is spread through the environment and store
- Requires some additional tools (e.g. an “up-to store” technique)
Language

Abadi & Cardelli’s **impç:**

- \[ l = \varsigma(x)e, \ldots \] (object creation)
- \( e.l \) (method invocation)
- \( e.l \leftarrow \varsigma(x)e \) (method update)
- \( \text{clone}(e) \) (object duplication)
- \( \text{let } x = e \text{ in } e \) (let-binding)

The language is **untyped**

The semantics are **big-step environment semantics** of the form:

\[ s; \rho \vdash e \downarrow v; s' \]

- Values are objects \[ l = v, \ldots \]
- Environments map identifiers to **values**
- Store maps locations \( i \) to **method closures** \( <\varsigma(x)b, \rho> \)
Example 1

```java
class Cell {
    private Object y;

    Cell (Object x) {
        y = x;
    }

    public void set (Object z) {
        y = z;
    }

    public Object get () {
        return y;
    }
}
```

```java
class Cell {
    private Object y1, y2;
    private int c;

    Cell (Object x) {
        c = 0; y1 = x; y2 = x;
    }

    public void set (Object z) {
        c = c+1; y1 = z; y2 = z;
    }

    public Object get () {
        if ((c % 2)==0)
            then return y1;
        else return y2;
    }
}
```
Example 1

class Cell {
    private Object y;

    Cell (Object x) {
        y = x;
    }

    public void set (Object z) {
        y = z;
    }

    public Object get () {
        return y;
    }
}

class Cell {
    private Object y1, y2;
    private int c;

    Cell (Object x) {
        c = 0; y1 = x; y2 = x;
    }

    public void set (Object z) {
        c = c+1; y1 = z; y2 = z;
    }

    public Object get () {
        if ((c % 2)==0)
            then return y1;
        else return y2;
    }
}
Example 1

```java
class Cell {
    private Object y;

    Cell (Object x) {
        y = x;
    }

    public void set (Object z) {
        y = z;
    }

    public Object get () {
        return y;
    }
}
```

```java
class Cell {
    private Object y1, y2;
    private int c;

    Cell (Object x) {
        c = 0; y1 = x; y2 = x;
    }

    public void set (Object z) {
        c = c+1; y1 = z; y2 = z;
    }

    public Object get () {
        if ((c % 2)==0)
            then return y1;
        else return y2;
    }
}
```

Partial store invariant
Example 2 [Meyer & Sieber '88]

```java
class EvenCounter {
    private int c;
    EvenCounter () { c = 0; }
    public void inc() { c = c+2; }
    public bool test (Callback o) {
        o.eval(this);
        return ((c % 2) == 0);
    }
}
```

Example 2 [Meyer & Sieber ‘88]

class EvenCounter {
    private int c;
    EvenCounter () { c = 0; }
    public void inc() { c = c+2; }
    public bool test (Callback o) {
        o.eval(this);
        return ((c % 2) == 0);
    }
}

class EvenCounter {
    EvenCounter () { }
    public void inc () { }
    public bool test (Callback o) {
        o.eval(this);
        return true;
    }
}
Example 2 [Meyer & Sieber ‘88]

```java
class EvenCounter {
    private int c;
    EvenCounter () { c = 0; }
    public void inc() { c = c+2; }
    public boolean test (Callback o) {
        o.eval(this);
        return ((c % 2) == 0);
    }
}
```

```java
class EvenCounter {
    EvenCounter () { }
    public void inc () { }
    public boolean test (Callback o) {
        o.eval(this);
        return true;
    }
}
```

Unknown callback
Example 2 [Meyer & Sieber `88]

```java
class EvenCounter {
    private int c;
    EvenCounter () { c = 0; }
    public void inc() { c = c+2; }
    public bool test (Callback o) {
        o.eval(this);
        return ((c % 2) == 0);
    }
}
```

2n

Partial store invariant

```java
EvenCounter () {
    public void inc () {
        public bool test (Callback o) {
            o.eval(this);
            return true;
        }
    }
}
```
Our Technique

A Relational Approach:

\[ e \quad e' \quad (e, e') \in X \]
Our Technique

\[ \forall C. \]

\[
\begin{align*}
C[e] & \sim C[e'] \\
\downarrow & \downarrow \\
n & \sim n'
\end{align*}
\]

A Relational Approach:

- \((e, e') \in X\)
- **Adequacy:** \((e, e') \in X \implies n = n'\)
Our Technique

∀C.

C[e] C[e']

∈ M

G

A Relational Approach:

• (e, e') ∈ X

• Adequacy: (e, e') ∈ X ⇒ n = n'

Question 1: given X how do we show adequacy?

Question 2: given (e, e'), how do we construct an appropriate X?
Our Technique

**Question:** Given an X, how do we show it is adequate?
Our Technique

**Question:** Given an X, how do we show it is adequate?

- Consider *any set X*
Our Technique

**Question:** Given an X, how do we show it is adequate?

- Consider **any set X**
- Formulate an **IH**
  - such that $\forall k. \text{IH}_x(k) \Rightarrow X$ is adequate
Our Technique

**Question:** Given an X, how do we show it is adequate?

- Consider **any set X**
- Formulate an **IH**
  - such that \( \forall k. \text{IH}_X(k) \Rightarrow X \text{ is adequate} \)
- **Analyze the inductive proof** \( \text{IH}_X(k) \Rightarrow \text{IH}_X(k+1) \)
  - This leaves **proof obligations** on X
Our Technique

**Question:** Given an \( X \), how do we show it is adequate?

- Consider **any set** \( X \)
- Formulate an **IH**
  - such that \( \forall k. \text{IH}_X(k) \Rightarrow X \text{ is adequate} \)
- **Analyze the inductive proof** \( \text{IH}_X(k) \Rightarrow \text{IH}_X(k+1) \)
  - This leaves **proof obligations** on \( X \)
- **Def:** \( X \) is a **bisimulation** iff it satisfies the proof obligations
Our Technique

- **Thm**: $X$ is a bisimulation $\Rightarrow X$ is adequate (**theoretical soundness**)
Our Technique

- **Thm**: $X$ is a bisimulation $\Rightarrow X$ is adequate (*theoretical soundness*)
- **Thm**: $e \equiv e' \Rightarrow \exists$ a bisimulation $X$ s.t. $(e, e') \in X$ (*theoretical completeness*)
Our Technique

- **Thm:** $X$ is a bisimulation $\Rightarrow X$ is adequate (**theoretical soundness**)
- **Thm:** $e \equiv e' \Rightarrow \exists$ a bisimulation $X$ s.t. $(e, e') \in X$ (**theoretical completeness**)
- **Experience fact:** For cases of interest we can write down a bisimulation that does the job (**practical completeness** 😊).
Technical Roadmap

• Consider **any set $X$**
• Formulate an **appropriate IH**
  – Such that: $\forall k. \text{IH}_X(k) \Rightarrow X$ is adequate
• **Analyze** the inductive proof $\text{IH}_X(k) \Rightarrow \text{IH}_X(k+1)$
  – Identify the **proof obligations** of $X$
• Define any $X$ that satisfies the proof obligations is a **bisimulation**
  – Remove some proof obligations by an “**up-to store**” method
Structure of Sets

The sets we consider will need:

• To keep track of the related objects:

  \textbf{Value Relations (R)}
  
  \[ \left( [ l_1 = \nu_1, \ldots ], [ l'_1 = \nu'_1, \ldots ] \right) \in R \]
Structure of Sets

The sets we consider will need:

- To keep track of the related objects:
  
  \[
  \text{{Value Relations (R)}}
  \]
  
  \[
  ( [ l_1 = \nu_1, \ldots ], [ l'_1 = \nu'_1, \ldots ] ) \in R
  \]

- To keep track of the changes in stores:
  
  Sets will contain the stores \( s, s' \) of the two sides
Structure of Sets

The sets we consider will need:

- To keep track of the related objects:
  
  **Value Relations (R)**
  
  \[(l_1 = \iota_1, \ldots, l'_1 = \iota'_1, \ldots) \in R\]

- To keep track of the changes in stores:
  
  Sets will contain the stores \(s, s'\) of the two sides

Thus we will consider **sets of states** \(<s, s', R>\)
Structure of Sets

The sets we consider will need:

- To keep track of the related objects:
  \[ \text{Value Relations (R)} \\]
  \[ ( [ l_1 = v_1, \ldots ], [ l'_1 = v'_1, \ldots ] ) \in R \]

- To keep track of the changes in stores:
  Sets will contain the stores \( s, s' \) of the two sides

Thus we will consider **sets of states** \( <s, s', R> \)

Stores will satisfy the **invariant** of the equivalence
Structure of Sets

The sets we consider will need:

- To keep track of the related objects:
  
  **Value Relations (R)**

  \[( [ l_1 = \nu_1, \ldots ], [ l'_1 = \nu'_1, \ldots ] ) \in R\]

- To keep track of the changes in stores:
  
  Sets will contain the stores \(s, s'\) of the two sides

Thus we will consider **sets of states** \(<s, s', R>\)
Technical Roadmap

- Consider any set $X$
- Formulate an appropriate IH
  - Such that: $\forall k. \text{IH}_x(k) \Rightarrow X$ is adequate
- Analyze the inductive proof $\text{IH}_x(k) \Rightarrow \text{IH}_x(k+1)$
  - Identify the proof obligations of $X$
- Define any $X$ that satisfies the proof obligations is a bisimulation
  - Remove some proof obligations by an “up-to store” method
Towards an IH for Adequacy

To define the IH we will also need to **extend value relations to environments**:

- **Environment Relations** ($R^\varepsilon$) to keep all related environments (same domain, related bindings)

  $$R^\varepsilon = \{ ((x_1 = o_1, \ldots), (x_1 = o'_1, \ldots)) \mid (o_1, o'_1) \in R \ldots \}$$
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:

for any $<s, s', R> \in X$

for any $(\rho, \rho') \in R^\epsilon$

for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$

$s; \rho \models e \downarrow$ iff $s'; \rho' \models e \downarrow$
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:

for any $<s, s', R> \in X$

for any $(\rho, \rho') \in R^\varepsilon$

for any expression $e$ with $FV(e) \subseteq \text{Dom}(\rho)$

$s; \rho \Downarrow e$ iff $s'; \rho' \Downarrow e$

Represents $\forall C[-]$

- Free Variables = Holes
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:
for any $<s, s', R> \in X$
for any $(\rho, \rho') \in R^\epsilon$
for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$
$s; \rho \Downarrow e \iff s'; \rho' \Downarrow e$

related values in the holes of the contexts
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:
for any $<s, s', R> \in X$
for any $(\rho, \rho') \in R^\varepsilon$
for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$

$s; \rho \Downarrow e$ iff $s'; \rho' \Downarrow e$

related instead of equal stores
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:

for any $<s, s', R> \in X$

for any $(\rho, \rho') \in R^\varepsilon$

for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$

$$s; \rho \Downarrow e \iff s'; \rho' \Downarrow e$$
Towards an IH for Adequacy

A set \( X \) of states \(<s, s', R>\) is adequate iff:

for any \(<s, s', R> \in X\)

for any \((\rho, \rho') \in R^\varepsilon\)

for any expression \(e\) with \(\text{FV}(e) \subseteq \text{Dom}(\rho)\)

\[ s; \rho \vdash e \downarrow \quad \Rightarrow \quad s'; \rho' \vdash e \downarrow \]

...and vice versa
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:

for any $<s, s', R> \in X$

for any $(\rho, \rho') \in R^\varepsilon$

for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$

\[
\text{\color{blue}{s;\ \rho \models e \downarrow^k \Rightarrow s';\ \rho' \models e \downarrow}}
\]

...and vice versa
Towards an IH for Adequacy

A set $X$ of states $<s, s', R>$ is adequate iff:

1. for any $<s, s', R> \in X$
2. for any $(\rho, \rho') \in R^\varepsilon$
3. for any expression $e$ with $\text{FV}(e) \subseteq \text{Dom}(\rho)$

$$s; \rho \vdash e \Downarrow^k \Rightarrow s'; \rho' \vdash e \Downarrow$$

...and vice versa
Technical Roadmap

• Consider any set $X$
• Formulate an appropriate IH
  – Such that: $\forall k. \text{IH}_X(k) \Rightarrow X$ is adequate
• Analyze the inductive proof $\text{IH}_X(k) \Rightarrow \text{IH}_X(k+1)$
  – Identify the proof obligations of $X$
• Define any $X$ that satisfies the proof obligations is a bisimulation
  – Remove some proof obligations by an “up-to store” method
Analysis of the Inductive Proof

IH_x(k) \Rightarrow IH_x(k+1)

- Proceed by cases on the syntax
- Most of the proof goes through directly by the IH
- Get about 1 condition on the set X for every syntactic form of the language (Proof obligations)
Technical Roadmap

- Consider **any set** $X$
- Formulate an **appropriate IH**
  - Such that: $\forall k. \text{IH}_x(k) \Rightarrow X$ is adequate
- **Analyze** the inductive proof $\text{IH}_x(k) \Rightarrow \text{IH}_x(k+1)$
  - Identify the **proof obligations** of $X$
- Define any $X$ that satisfies the proof obligations is a **bisimulation**
  - Remove some proof obligations by an “**up-to store**” method
Bisimulations

X is a bisimulation iff for all \(<s, s', R> \in X:\)
- \([(l_1 = \nu_1, ...), (l'_1 = \nu'_1, ...)] \in R \implies (l_1 = l'_1) \ldots\)
Bisimulations

\(X\) is a bisimulation iff for all \(<s, s', R> \in X:\)

- \(((l_1 = \nu_1, \ldots), (l'_1 = \nu'_1, \ldots)) \in R \Rightarrow (l_1 = l'_1) \ldots\)
- \(((l_1 = \nu_1, \ldots), (l'_1 = \nu'_1, \ldots)) \in R \Rightarrow \forall l_i, (\rho, \rho') \in R^\epsilon, \exists Q \supseteq R:\)
  \(<(s.l_i=<\zeta(x)e, \rho>), (s'.l'_i=<\zeta(x)e, \rho'>), Q> \in X\)
Bisimulations

X is a bisimulation iff for all \(<s, s', R> \in X:\n
- \((\langle l_1 = i_1, \ldots \rangle, \langle l'_1 = i'_1, \ldots \rangle) \in R \Rightarrow (l_1 = l'_1) \ldots\)\n- “X is closed under method update of known objects”
Bisimulations

X is a bisimulation iff for all \(<s, s', R> \in X>:

- \(([l_1=v_1, ...], [l'_1=v'_1, ...]) \in R \Rightarrow (l_1 = l'_1)\) ...
- "X is closed under method update of known objects"
- "X is closed under creation of new objects"
Bisimulations

X is a bisimulation iff for all <s, s’, R> ∈ X:

• ([l_1=\nu_1, ...], [l'_1=\nu'_1, ...]) ∈ R ⇒ (l_1 = l_1') ...
• “X is closed under method update of known objects”
• “X is closed under creation of new objects”
• “X is closed under cloning of objects”
Bisimulations

X is a bisimulation iff for all \( <s, s', R> \in X \):

- \( ([l_1=v_1, \ldots], [l'_1=v'_1, \ldots]) \in R \implies (l_1 = l'_1) \ldots \)
- “X is closed under method update of known objects”
- “X is closed under creation of new objects”
- “X is closed under cloning of objects”
- \( (o, o') \in R \implies \forall l \in \text{Methods}(o):
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
  \!
Bisimulations

X is a bisimulation iff for all \( <s, s', R> \in X \):

- \( ([l_1 = i_1, \ldots], [l'_1 = i'_1, \ldots]) \in R \Rightarrow (l_1 = l'_1) \ldots \)
- "X is closed under method update of known objects"
- "X is closed under creation of new objects"
- "X is closed under cloning of objects"
- \( (o, o') \in R \Rightarrow \forall I \in \text{Methods}(o): \)
  \[
  \text{IH}_X(k) \Rightarrow \text{IH}_X(k+1) [ o.I, o'.I ]
  \]

helps to reason about invocations of unknown callbacks
Bisimulations

X is a bisimulation iff for all \( \langle s, s', R \rangle \in X \):

- \( ([\ell_1 = \ell_1', \ldots], [\ell'_1 = \ell'_1', \ldots]) \in R \Rightarrow (\ell_1 = \ell_1') \ldots \)
- “X is closed under method update of known objects”
- “X is closed under creation of new objects”
- “X is closed under cloning of objects”
- \( (o, o') \in R \Rightarrow \forall \ell \in \text{Methods}(o) : \)
  \[ \text{IH}_X(k) \Rightarrow \text{IH}_X(k+1) [ o.\ell, o'.\ell ] \]

sets that satisfy these conditions are complicated
Up-to Store Technique

We close the sets up to method update and creation of new objects with an “up-to store” operator on sets

\[ X^* = \{ \langle s, s', R \rangle \mid \exists \langle s_0, s'_0, R_0 \rangle \in X: \langle s_0, s'_0, R_0 \rangle \sqsubseteq \langle s, s', R \rangle \} \]

up-to store extension of states
Bisimulations Up-to Store

X is a **bisimulation up-to store** iff for all \(<s, s', R> \in X\):

- \(((l_1=\iota_1, ...), [l'_1=\iota'_1, ...]) \in R \Rightarrow (l_1 = l'_1) \ldots\)
- “X is closed under cloning of objects”
- \((o, o') \in R \Rightarrow \forall l \in \text{Methods}(o):
  \quad \text{IH}_{X^*}(k) \Rightarrow \text{IH}_{X^*}(k+1) [\; o.l, o'.l \; ]\)

**Theorem:** If X is a bisimulation up-to store then X* is a bisimulation
Using Bisimulations

**Question:** given \((e, e')\), how do we construct an appropriate \(X\)?

- \(X\) should relate \((e, e')\)
- **Encode the invariant** of the equivalence in the construction of \(X\)
- The rest of the construction will be guided by the conditions of the definition of bisimulations
- Show that \(X\) is a bisimulation.
Example 1 Revisited

\[ \text{Cell} = \]
\[ \begin{align*}
&\text{let } o = [y=0] \\
&\text{in} \\
&\quad [ \text{set} = \text{setMethod}_M \\
&\quad \text{get} = \text{getMethod}_M ]
\end{align*} \]

\[ \text{Cell}’ = \]
\[ \begin{align*}
&\text{let } o = [y1=0, y2=0, c=0] \\
&\text{in} \\
&\quad [ \text{set} = \text{setMethod}_N \\
&\quad \text{get} = \text{getMethod}_N ]
\end{align*} \]

\[ X = \{ <s, s’, R> | \} \]
Example 1 Revisited

Cell =
  let o = [ y=0 ]
  in
  [ set = setMethod_M
    get = getMethod_M ]

Cell' =
  let o = [ y1=0, y2=0, c=0 ]
  in
  [ set = setMethod_N
    get = getMethod_N ]

X = { <s, s', R> | 
  R = { ([set=ι_s, get=ι_g], [set=ι_s', get=ι_g']), ... } }
Example 1 Revisited

Cell =
  let o = [y=0]
in
  [ set = setMethod_M
    get = getMethod_M ]

Cell' =
  let o = [y1=0, y2=0, c=0]
in
  [ set = setMethod_N
    get = getMethod_N ]

X = { <s, s', R> | 
  R = { ([set=₁_s, get=₁_g], [set=₁'_s, get=₁'_g]), ... } 
  s = (₁_s=setMethClosure_M, ₁_g=getMethClosure_M, ...)
  s' = (₁'_s=setMethClosure_N, ₁'_g=getMethClosure_N, ...) 
}

Bisimulations for Untyped Imperative Objects
Example 1 Revisited

Cell =
let o = [y=0]
in
[ set = setMethod_M
get = getMethod_M ]

Cell' =
let o = [y1=0, y2=0, c=0]
in
[ set = setMethod_N
get = getMethod_N ]

X = \{ <s, s', R> \mid
\begin{align*}
R &= \{ ([set=\iota_s, get=\iota_g], [set=\iota'_s, get=\iota'_g]), \ldots \} \\
s &= (\iota_s=set\text{MethClosure}_M, \iota_g=get\text{MethClosure}_M, \\
&\quad \iota_y=v, \ldots) \\
s' &= (\iota'_s=set\text{MethClosure}_N, \iota'_g=get\text{MethClosure}_N, \\
&\quad \iota_{y1}=v', \iota_{y2}=v', \iota_c=n, \ldots) \\
\end{align*} \}

Bisimulations for Untyped Imperative Objects ESOP’06 62
Example 1 Revisited

Cell =
  let o = [y=0]
in
  [ set = setMethod_{\mathcal{M}}
    get = getMethod_{\mathcal{M}} ]

Cell' =
  let o = [y1=0, y2=0, c=0]
in
  [ set = setMethod_{\mathcal{N}}
    get = getMethod_{\mathcal{N}} ]

X = \{ <s, s', R> | 
    R = \{ ([set=\iota_s, get=\iota_g], [set=\iota'_s, get=\iota'_g]), ... \} 
    s = (\iota_s=setMethClosure_{\mathcal{M}}, \iota_g=getMethClosure_{\mathcal{M}},
         \iota_y=v, ...)
    s' = (\iota'_s=setMethClosure_{\mathcal{N}}, \iota'_g=getMethClosure_{\mathcal{N}},
           \iota_{y1}=v', \iota_{y2}=v', \iota_c=n, ...)
    (v,v') \in R \}
Contributions of our Method

- It gives a **sound and complete** proof method of equivalence for the untyped, imperative object calculus of Abadi & Cardelli
- It is a **constructive** bisimulation method
  - Uses the **IH** to deal with unknown callbacks
  - Uses an **“up-to store” technique** to reduce the size of bisimulations.
- It proves equivalences where other methods are hard to use.
Future Work

- **Better notation** for specifying bisimulations (especially in the presence of store)
  - Separation Logic [Reynolds ’02]

- **Scaling up**: We applied our method to a number of calculi, but can we handle other language features?
  - classes, inheritance
  - exceptions, control operators.
Thank you

www.ccs.neu.edu/home/vkoutav/